INT J COMPUT COMMUN, ISSN 1841-9836
8(4):635-646, August, 2013.

Knapsack-model-based Schemes and WLB Algorithm for the Capacity and Efficiency Management of Web Cache

A.B. Xiang, L. Xu, B.Z. Niu

Anbo Xiang
Development Research Center of the State Council, Beijing, 100010, China
abxiang@drc.gov.cn

Liang Xu
Department of Logistics Management, School of Business Administration,
Southwestern University of Finance and Economics, Chengdu, 611130, China
arecxuliang1112@gmail.com

Baozhuang Niu*
Lingnan College, Sun Yat-sen University, Guangzhou, 510275, China
niubzh@mail.sysu.edu.cn
*Corresponding author

Abstract: Web cache refers to the temporary storage of web files/documents. In reality, a set of caches can be grouped into a cluster to improve the server system's performance. In this paper, to achieve the overall cluster efficiency, we propose a weighted load balancing (WLB) routing algorithm by considering both the cache capability and the content property to determine how to direct an arrival request to the right node. Based on Knapsack models, we characterize three new placement/replacement schemes for Web contents caching and then conduct the comparison based on WLB algorithm. We also compare WLB algorithm with two other widely used algorithms: Pure load balancing (PLB) algorithm and Round-Robin (RR) algorithm. Extensive simulation results show that the WLB algorithm works well under the examined cluster content placement/replacement schemes. It generally results in shorter response time and higher cache hit ratio, especially when the cache cluster capacity is scarce.
Keywords: web cache placement/replacement scheme, Knapsack model, load balancing and routing algorithm, performance analysis.

1 Introduction

In recent years, with the rapid growth of the size of contents delivered and the number of internet users, the World Wide Web (WWW) is better known as "World Wide Wait" [1]. Although various information technologies are being developed rapidly, which makes the internet becomes faster and faster, according to [2]'s observation, "the trend of increasing traffic on the Internet is likely to continue". It is reported that if a web page can not be loaded within eight seconds, then users are likely to give up or load the link in a second browser [3]. As a result, a loss of revenue will be suffered, which is called the *Zona effect* [1]. For example, the possible revenue loss by increasing a millisecond to execute automated trades, according to [3], can be as high as $100 million. Therefore, there exist plenty of incentives to reduce the web response time when the server capacity is constrained (It may cost a lot to enlarge the server capacity.). In practice, an effective approach to alleviate the user-waiting problem and to optimize the Web resource utilization is *caching*, in which the Web content is generated once and kept for a period of time for the future use. When a user requests a previously requested content, the content can be accessed directly from the cache. Recently, content caching has been a very active research area.

Generally speaking, a good caching system must address the following issues:

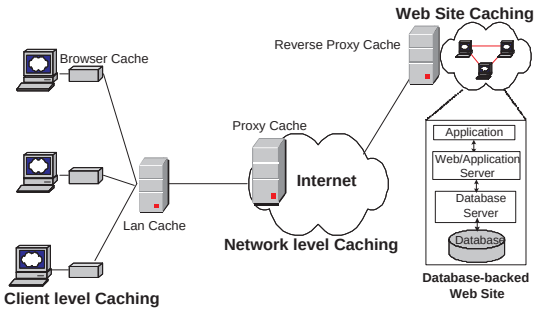


Figure 1: Overview of Caching

(1) What to cache. Cacheable elements include DNS mapping, connection, and content.

(2) Where to cache. A cache usually can reside in one of the three locations: (a) client level, at the client browser and LAN, (b) network level, at a proxy somewhere in between the client and the Web site, or (c) at the Web site itself.

(3) When and how to cache. When are elements are placed in cache and when are elements are evicted from cache.

Industrial experts have already found clear answers/rules to address the first two issues [4], however, more analysis and/or solving methods are needed to address the third issue [5]. Therefore, in this paper, we mainly focus on the third issue for the Web server with content caching cluster at the web site, i.e., reverse proxy cache. Reverse proxy cache, which is a web accelerator, can reduce the workload of a busy Web application server that provides both static and dynamic contents. The static contents can be cached on the reverse proxy while the Web application server is freed up to better handle the dynamic contents.

Towards the cache placement/replacement schemes, in practice, the Least-recently-used (LRU) and least-frequently-used (LFU) are commonly used due to their simplicity. However, with no or limited considerations of content properties or request patterns, these schemes could not provide the best performance. Many other schemes have been proposed, which take into account the retrieval latency of contents, their sizes, the popularity of references and temporal locality of requested contents, to solve the problem in a systematic manner (e.g., [6]).

From a theoretic perspective, the focused problem in this paper is closely related to the *Knapsack problem* that maximizes the benefit under a constraint capacity. Different approaches have been proposed based on the Knapsack model in caching or similar applications, for example, [7] and [8]. Most of these works utilize the cost-benefit optimizing idea of Knapsack model to develop cache management schemes.

It is worth noting that, unlike hard disk storage space, cache's internal main memory is still a limited resource that must be managed wisely [9]. Accordingly, there is another trend in caching technology which provides larger cache capacity (hereafter, cache capacity refers to its internal main memory) to support more concurrent users: cluster-based design for web application server (e.g., [10], [11], [12]). With this design, a set of caches are grouped into a loosely coupled cluster to solve some common issues such as capacity, availability, and performance. When clustered, cache's capacity increases linearly and the number of cacheable missed requests sent to the Web application server is reduced accordingly. The impact of an individual cache node failure in the cluster on the site availability and performance is also reduced considerably. This Web cache cluster architecture is represented in Figure 2, in which a load balancer utilizes the inherent locality of the requests and an adaptive scheme to tune the load allocated to each node in the cluster based on that node's capability. Some leading companies also propose similar commercial

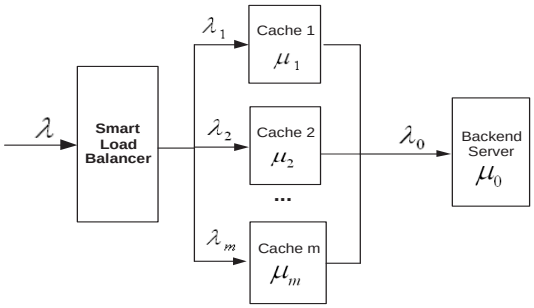


Figure 2: Model of Web Cache Cluster

solutions, such as Microsoft Internet Security and Acceleration Server with Cache Array Routing Protocol, IBM Web Traffic Express proxy servers with Network Dispatcher, Oracle9iAS Web Cache Cluster, Cisco Cache/Content Engine, and so on.

In short, our study was motivated by problems arising in practice and trends in cluster caching. Most of the previous works required or assumed that all nodes in the cluster are symmetric with an equal chance to serve any incoming request. We would like to relax these assumptions and propose a more general solution. Also we want to consider the nonhomogeneity of the content properties together with the request pattern so as to maximize the utilization of the valuable cache capacity.

We first study how to efficiently place/replace cache contents by accurately monitoring the properties of contents, i.e., whether to cache a content, and if we do that, which cluster node to place it so that the system performance is optimized? Furthermore, based on the prior knowledge about the requests arrival pattern, contents properties, and the cache contents placement/replacement scheme, we develop a routing algorithm to determine how to direct an arrival request by efficiently coordinating the cluster nodes.

The remainder of the paper is organized as follows. In the next section, we introduce our Web cache cluster model. The content placment/replacement schemes are discussed based on the Knapsack model in section 3. In Section 4, we propose a weighted load balancing algorithm, by which we can design a smart load-balancer to determine how to direct an arrival request to a cluster node. Extensive simulations have been carried out to evaluate the performance of the algorithms. These simulation results are presented and discussed in section 5. Section 6 summarizes the paper.

2 Model Description

As a reverse proxy, our cache cluster is dedicated to a single or a set of original Web server systems. The model consists of a set of cacheable contents $N = \{1, 2, ..., n\}$, and a set of caches $M = \{1, 2, ..., m\}$, as illustrated in Figure 2. For convenience, we introduce some notation below. More detailed descriptions will be presented when they are used.

C_j capacity of cache j, $j \in M$,

λ total system requests arrival rate,

μ_0 service rate of back-end server,

λ_0 requests arrival rate of the back-end server,

μ_j service rate of the cache j, $j \in M$,

λ_j requests arrival rate of the cache j, we have $\lambda = \sum_{j \in M} \lambda_j$,

O_j set of all contents currently in the cache j, $j \in M$,

w forwarding cost per unit of time in the cache cluster,

R_i retrieval cost of the requested content i from the back-end server,

S_i size of the content i, $i \in N$,

p_i "profit" from serving the content i, $i \in N$,

$P_n(i)$ probability that an arriving request is made for the content i, $i \in N$, given that there are n contents in N.

We formulate the cache cluster content placement/replacement problem as a Multiple-Knapsack problem (MKP) with an objective of maximizing the total "profit" by placing the most "valuable" contents in the available caches:

$$Maximize \sum_{i=1}^{n} \sum_{j=1}^{m} p_i x_{ij} \ , \tag{1}$$

subject to,

$$\sum_{i=1}^{n} S_i x_{ij} \leq C_j \ , \quad j \in M, i \in N, \tag{2}$$

where

$$x_{ij} = \begin{cases} 1, & \text{object } i \text{ is placed in cache } j. \\ 0, & \text{otherwise.} \end{cases}$$

3 Content Placement/Replacement Schemes

The cache cluster content placement/replacement scheme specifies the contents in caches to achieve the specific performance objective. At each decision epoch, the following actions are carried out:

1. initiation, preparing the initial cacheable contents,
2. update the cache contents when cache hit ratio is lower than a preset threshold value,
3. or update the cache contents when a hit miss occurs.

We can achieve different specific objective by specifying appropriate p_i in the equation (1). For example,

1. when $p_i = P_n(i)$, the objective is to maximize the cache hit ratio,
2. when $p_i = P_n(i)S_i$, the objective is to maximize the byte hit ratio,
3. when $p_i = P_n(i)R_i$, the objective is to minimize the retrieval cost.

The Knapsack problem is known to be NP-hard. However, there exist fast heuristics with good performance records. In the following subsections, we discuss how different cache management methodologies are applied in different cases.

3.1 As many contents in cache as possible (AMAP)

In this case, a cache cluster is regarded as a pure Multiple-Knapsack. We follow the principle of placing *as many* contents in caches *as possible* to effectively utilize the capacity of the whole cluster. Consequently we need to add one more constraint in addition to (2):

$$\sum_{j=1}^{m} x_{ij} \leq 1, \quad i \in N. \tag{3}$$

This constraint guarantees that no content would be cached redundantly. It is a reasonable consideration when the contents space is huge while the cluster capacity is limited.

[13] present an approximate dynamic programming (ADP) approach for the multidimensional Knapsack problem that produces near optimal solutions efficiently. We apply their adaptive fixing heuristic to solve our cluster content placement/replacement problem. Our computational evidence suggests that the ADP-based heuristic is an attractive methodology that usually generates good quality solutions in reasonable time.

3.2 Popular contents replication and hit ratio threshold (Threshold value)

In this case, we aim at achieving a high *availability* of the most popular contents on condition that a certain level of hit ratio is guaranteed.

Some prior knowledge of the Web request pattern can facilitate Web cache resource planning and cache hierarchy design, and help us to predict the most popular contents. It has been shown that the Web page request follows a Zipf-like distribution (see [14]). Following this result, we rank all the pages in order of their popularity where page i is the i'th most popular page. Suppose that the number of contents in the system is n, the probability that an arriving request is made for page i is approximated by

$$P_n(i) = \frac{\Omega}{i^{\alpha}}, \tag{4}$$

where $\Omega = \left(\sum_{i=1}^{n} \frac{1}{i^{\alpha}} \right)^{-1}$ and α is the Zipf parameter determined by the system property.

If the top k most popular contents can guarantee a certain level of hit ratio, we only place these top k most popular contents in caches. If there are extra free space, we duplicate these contents in caches sequentially until there are no more free space in the cluster. Let P_{th} be a threshold value for hit ratio, then k is determined by the formula below:

$$k = \arg \inf_{l \in N} \sum_{i=1}^{l} P_n(i) \geq P_{th}.$$

With this replication caching scheme, the more popular contents are cached redundantly and higher hit ratio can be achieved at any single node. At the same time, a high availability of popular contents can still be maintained when any individual node fails. Due to the replication of popular contents, the tradeoff is less effective usage of the combined capacity of the whole cluster.

3.3 Web contents space partitioning (Partition)

In a cache cluster, cache members may not be identical and have different capabilities in dealing with the arrival requests. Capability (capacity, processing power, bandwidth, etc.) represents a member's potential contribution to the cluster.

In this case, we follow [10] to partition Web contents space based on members' capability, as shown in Figure 3. Each cache is assigned to a certain part of the Web contents. Consequently,

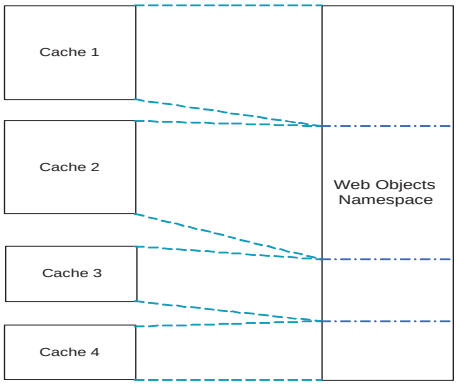


Figure 3: Partitioning Web Objects Namespace based on Caches' Capability

the MKP problem (1) reduces to multiple single-Knapsack problems with cache j being in charge of the contents subset N_j, $N_j \subseteq N$.

A well-known Knapsack problem solution method is the Greedy heuristic, which selects objects for inclusion in the knapsack using the "density" of object as the criterion to be greedy upon. Let $d_{i_j} = p_{i_j}/S_{i_j}$ be the density of content i_j, $i_j \in N_j$. We set $x_{i_j} = 0$ for all i_j that satisfies $S_{i_j} > C_j$, and then arrange the remaining contents in decreasing order of density from top to bottom. Starting from the top, the Greedy heuristic sets $x_{i_j} = 1$ as it goes down until the cache capacity C_j is reached. At each stage, if the next content cannot be included in the cache because its size exceeds the remaining capacity, x_{i_j} is set to 0 for that content and the process continues with the contents below it. The process terminates either when the cache capacity is used up (in this case, $x_{i_j} = 0$ for all contents below the current one), or when all the contents have been examined in this way.

In the simulation section, we will compare the performance of these three schemes: placing as many contents in caches as possible, threshold value for hit ratio and replication for popular contents, and partitioning of Web contents based on caches' capabilities under different parameter settings.

4 Weighted Load Balancing (WLB) Routing Algorithm

Cache clustering introduces a new problem: When a browser requests a particular content, to which cache in the cache cluster should the request be directed? How do we match an incoming request with the cache best able to respond?

Content-aware routing working at application layer certainly can increase the cache hit ratio. But inspecting every incoming HTTP request would increase the system delay, so that the load-balancer may easily become a bottleneck itself and slow down the entire system. Thus, the load-balancer should be kept as "light-weight" and simple as possible to avoid introducing new vulnerabilities into the system (system is only as secure as its weakest component). So in designing our load-balancer, we should consider the system response time and the cache hit ration simultaneously. We want it to work at transport layer and focus on forwarding the data at maximal speed without inspecting every incoming HTTP request, while still guarantees an acceptable cache hit performance.

4.1 The dynamic routing model

When a new request arrives and is assigned to cache j, cache j will be associated with an expected cost function $F(j)$ for handling the request(s). We want to find a suboptimal and easy

implemented routing strategy to determine which cache to serve the new arrival request so that the cluster-wide total expected cost is minimized.

In general, serving a request directly from Web cache cluster, i.e., a cache hit, is significantly faster than forwarding the request to the back-end server for generation. In the cache cluster, we also assume that the network bandwidth between peer caches is large and network latency is low, and thus retrieving a cached content from a peer cache is also significantly faster than getting the same content from the back-end server. We denote as w the forwarding cost per unit time in the cache cluster.

There are three possible ways for an arriving request to be served when routed to a cache: (1) the cache contains the requested content and serves the request immediately; (2) the cache does not contain the requested content, but another cache in the cluster contains the content and serves the request; (3) the content is not cached and the request is eventually routed to the back-end server. For the cases (1) and (3), the cost associated with routing the request to any cache is the same. Thus, the cost difference only lies in the forwarding cost in the cluster when the requested content is not cached in the assigned cache but is cached in the other node(s) in the cluster.

From a request's "viewpoint", the routing decision could be determined by the expected cost of sending the request to a cache. i.e. we should direct each arriving request to the queue with the minimum expected waiting cost. Therefore we need to find $l = arg \min_{j \in M} F(j)$. Such a policy minimizes each arriving request's individual expected waiting cost as well as the long-run system waiting cost [15].

4.2 WLB routing algorithm

We assume that the request inter-arrival times and service times are all exponential. Let λ_j and μ_j denote the request arrival rate and the service rate at cache j, respectively. Let O_j denote the set of all contents currently in the cache j and k_j denote the number of outstanding requests waiting at the cache j. $p(i \notin O_h, i \in \bigcup_{v=1}^{m} O_v)$ denotes the probability that the request content i is not in the cache h but is in the cluster.

Based on the above notation, assumptions and analysis, we need to compare two arbitrary decisions h and l. We have

$$F(h) - F(l) = \frac{k_h w}{\mu_h - \lambda_h} \sum_{i=1}^{n} p_n(i) p(i \notin O_h, i \in \bigcup_{v=1}^{m} O_v) - \frac{k_l w}{\mu_l - \lambda_l} \sum_{i=1}^{n} p_n(i) p(i \notin O_l, i \in \bigcup_{v=1}^{m} O_v) \ ,$$

where

$$p(i \notin O_h, i \in \bigcup_{v=1}^{m} O_v) = \frac{\sum_{j=1}^{m} \sum_{e \in O_j} p_n(e) - \sum_{e \in O_h} p_n(e)}{\sum_{j=1}^{m} \sum_{e \in O_j} p_n(e)} \ .$$

The two terms in the right-hand side of the equation represent the forwarding costs when the requested content is not cached in the assigned cache h (l) but is cached in the other node(s) in the cluster, respectively. The router should direct the arriving request to cache l if $F(h) - F(l) > 0$, or to cache h otherwise. Thus, we have the following heuristic routing policy:

Rule: when a request arrives, the router directs it to the server l if

$$l = \arg\min_{j} \left\{ \frac{k_j}{\mu_j - \lambda_j} P[\ \textit{the requested file is in another cache instead of cache } j\ ] \right\}.$$

The essence of this WLB algorithm is that the queue length k_j is weighted by the probability that the cache does not have the target content and the service capability. The router directs each arrival request to the cache with the lightest *effective* workload. This WLB routing algorithm makes use of the information related to the cache (it's service capability and workload) and the property of the content (the probability that a request can be satisfied in one cache). Moreover, this algorithm can be easily extended to implement Web server cluster. Since each Web server can satisfy all of the requests, it is natural for us to utilize a pure load balancing policy to guarantee the Web server cluster's performance.

5 Performance Analysis

In this section we compare the performances of different caching schemes. With the WLB arrival request routing algorithm, we first compare performance of LRU scheme with our proposed three new content placement/replacement schemes based on the Knapsack model:

1. placing *as many* contents in caches *as possible* (AMAP scheme, see Section 4.1),
2. placing only contents with popularity values higher than a *threshold value* in caches (Threshold Value scheme, see Section 4.2),
3. partitioning of contents based on caches' capabilities (Partition Scheme, see Section 4.3).

Secondly, we compare the performance of the WLB algorithm with the following two routing algorithms under the *threshold value* content placement/replacement scheme:

1. *Pure Load Balancing* (PLB) Algorithm. With this algorithm, a request is directed to the server with the shortest queue length;
2. *Round-Robin* (RR) Algorithm. A request is directed to the server next to the server which received the previous request.

Using simulation, the average response time (ART), cache hit ratio (CHR), and cache cluster hit ratio (CCHR), the ratio between the total requests and the requests are not served rightly by the assigned caches but by other cache in the cluster, of these algorithms have been compared over following parameters setting:

1. *size rate*, the ratio of cache cluster size vs contents total size;
2. ρ, the ratio of request arrival rate vs the cache cluster service rate. This factor indicates the level of system's workload;
3. *cache types*, we assumed that the caches can be all identical, or be divided into two groups with different capabilities (in terms of service rate, capacity, etc), or all be different in capability.

5.1 Model-driven simulation

The primary motivation for performing model-driven simulation is to understand the effect of different schemes on cache cluster content management. In the model-driven simulation experiment, the arrival requests follow a poisson stream with rate λ. The target content of the request has Zipf-like frequency distribution with the Zipf parameter α. Generally, without specifying otherwise, the default settings are $\lambda = 0.3$, $\mu_0 = 0.05$, $\mu_j = 0.045$, $m = 10$ (i.e. there

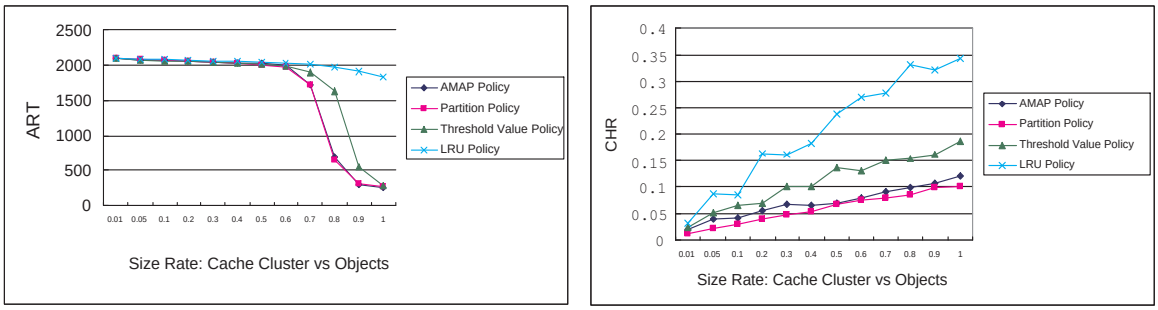
are 10 caches in the cluster). Hence the $\rho = 0.6$ (this means the system is of moderate level of workload). The *size rate* equals 0.5, the Zipf parameter $\alpha = 0.8$. We use 10^3 contents of sizes uniformly distributed between 1 and 1000.

The ART, CHR, and CCHR are explored under different parameters combinations. The results are presented in four sets of figures below. We now discuss the numerical results in detail.

Observation 1*: Increasing cache cluster's capacity can improve the system performances.* We observe the impact of varying the caches' size on the system performances. From Figure 4 and Figure 5, ART is decreasing in cache cluster's size, CHR and CCHR are increasing in cache cluster's size.

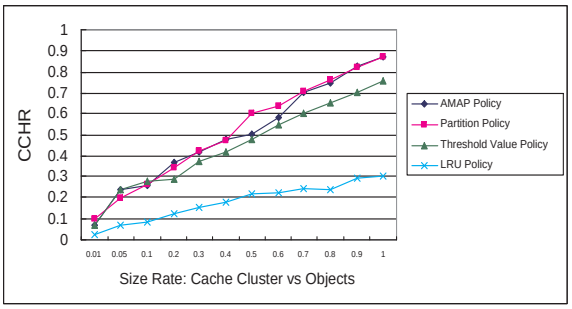
Observation 2*: The threshold value scheme is more efficient when the cache cluster capacity is scarce. The partition scheme is more efficient when cache cluster capacity is sufficient.* We compare the performances of different content placement/replacement schemes under different situations aim to find which scheme is more appropriate for a certain situation. From Figure 4(a), it is noticed that when the cache cluster size is small (size rate < 0.6), the *threshold value* scheme achieves shorter ART than other content placement/replacement schemes.

Observation 3*: Better performance can be achieved by considering both node's capability and content's property.* With the *threshold value* content placement/replacement scheme, we compare the performances of the WLB, PLB, and RR routing algorithms under different situations by varying the setting of *size rate*, ρ, and *cache cluster types*. The simulation results (Figures 5) suggest that the WLB has shorter ART, higher CHR and CCHR than PLB and RR algorithm in most of the situations.



(a) Average Response Time vs Size Rate

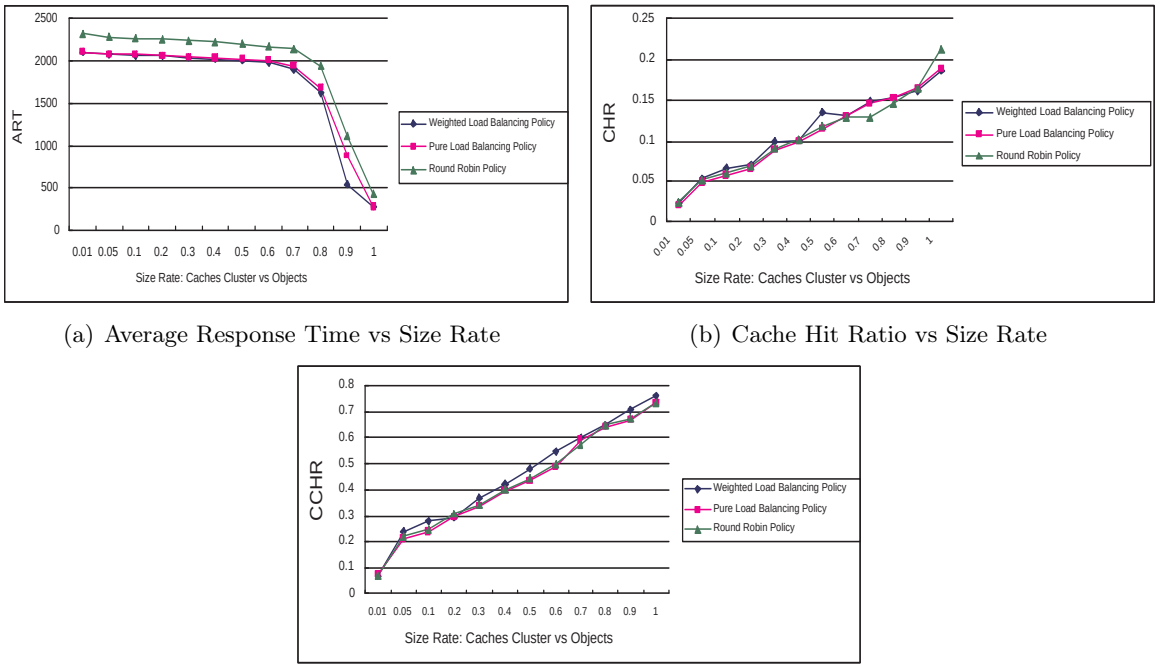
(b) Cache Hit Ratio vs Size Rate



(c) Cache Cluster Hit Ratio vs Size Rate

Figure 4: Performances of content placement/replacement schemes with WLB algorithm

Of all the cases simulated, the average improvement in the ART achieved by the WLB is 2.56% compared with the PLB algorithm, and 13.96% compared with the RR algorithm.



(a) Average Response Time vs Size Rate

(b) Cache Hit Ratio vs Size Rate

(c) Cache Cluster Hit Ratio vs Size Rate

Figure 5: Performances of different routing algorithms with the threshold value scheme

5.2 Trace-driven simulation

We now present results from trace-driven simulations using real Web server's traces, the access log data of a Squid proxy server system which was in operation at HKUST in September 2012. The trace length is 409623 user accesses. We ran the trace-driven simulation for different values of the cache cluster capacity by varying the size rate.

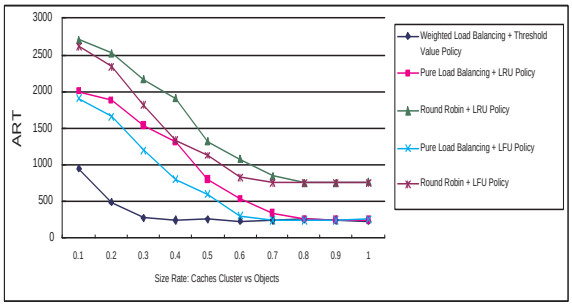
We first compare the performance of the following cache cluster management policies:

1. WLB routing algorithm with threshold value content placement/replacement scheme,
2. PLB routing algorithm with LRU content placement/replacement scheme,
3. RR routing algorithm with the LRU Value content placement/replacement scheme,
4. PLB routing algorithm with LFU content placement/replacement scheme,
5. RR routing algorithm with the LFU content placement/replacement scheme.

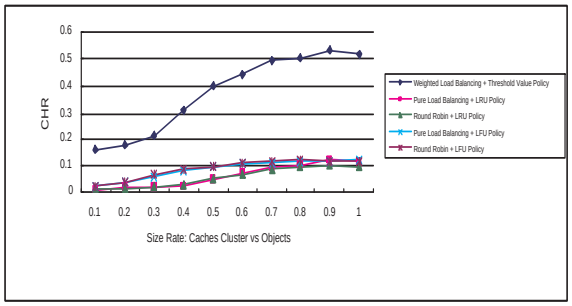
From Figure 6, we find that WLB algorithm with threshold value content placement/replacement scheme achieves better performance than other combination policies significantly, especially when the size rate is smaller. This observation validates the advantage of our Knapsack-based cache cluster management scheme.

6 Conclusion

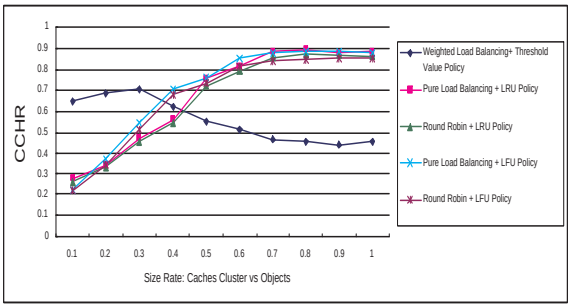
In this paper, based on the Knapsack model, we first propose three efficient placement/replacement schemes for content caching, and then develop a WLB routing algorithm working at transport layer, which considers both of the node's capability and content's property, to determine how to direct an arrival request to the right node by efficiently coordinating the cluster nodes.



(a) Average Response Time vs Size Rate



(b) Cache Hit Ratio vs Size Rate



(c) Cache Cluster Hit Ratio vs Size Rate

Figure 6: Performances of cache cluster management policies under Trace-driven Simulation

Extensive simulation results show that the WLB algorithm with the content replacement scheme leads to satisfactory quality of service (shorter response time and higher cache hit ratio). The simulation results also indicate that content placement/replacement scheme dominates the system performance, and modeling the cache cluster into a multiple Knapsack problem is a good way for us to study the content placement/replacement scheme. Furthermore, the simulation results point out which content placement/replacement scheme is more appropriate under different situations.

In the future study, we plan to integrate the “admission control" and “request priority" mechanisms into our Web cache cluster model with the objective to maximize the system reward. We also expect to investigate the distributed data caching infrastructure over Internet and the distributed resources optimization problems. Moreover, to consider how to organize the cluster (e.g., finding out the optimal number of cache nodes in the cluster) will bring some insight in cache cluster management.

Acknowledgement

This project is supported in part by National Natural Science Foundation of China under Grant 71201175 and National Natural Science Foundation of China under Grant 71201127 and Guangdong Natural Science Foundation under Grant S2012040008081 and S2011040001069.

Bibliography

[1] Datta, A. et al (2003); World Wide Wait: A Study of Internet Scalability and Cache-based Approaches to Alleviate It, *Management Science*, ISSN 0025-1909, 49: 1425-1444.

[2] Kumar, C. ; and Norris, J. (2008); A New Approach for A Proxy-level Web Caching Mechanism, *Decision Support Systems*, ISSN 0167-9236, 46: 52-60.

[3] http://www.economist.com/node/15523761.

[4] http://plone.org/documentation/kb/optimizing-plone/what-to-cache.

[5] Xiang, A. (2006); Essays on information service systemes. PhD Dissertation. Hong Kong University of Science and Technology.

[6] Bahat, O.; Makowski, A. M.; (2003) Optimal Replacement Policies for Non-uniform Cache Objects with Optional Eviction. *IEEE INFOCOM 2003*, San Francisco, California, April.

[7] Wang, B. et al (2002); Proxy-based Distribution of Streaming Video over Unicast/Multicast Connections. *Proceedings of IEEE Infocom*, New York, June.

[8] Otoo, E. et al (2002); Disk Cache Replacement Algorithms for Storage Resource Managers in Data Grids. *The 15th Annual Supercomputer Conference*, Baltimore, Maryland, November.

[9] Rabinovich, M.; Spatscheck, O. (2002); *Web Caching and Replication.* Addison-Wesley, Boston, MA, US.

[10] Feenan,J. et al (2002); Clustering Web Accelerators. *IEEE WECWIS Proceedings*, San Diego, June.

[11] Zhang, X., et al (1999); HACC: An Architecture for Cluster-based Web Servers. *3rd USENIX Windows NT Symposium*, Seattle, Washington, July.

[12] Clevenot, F., et al (2005); Stochastic Fluid Models for Cache Cluster. ISSN 0166-5316, *Performance Evaluation*, 59: 1-18.

[13] Bertsimas, D.; Demir, R. (2002); An Approximate Dynamic Programming Approach to Multidimensional Knapsack Problems. ISSN 0025-1909, *Management Science*, 48: 550-565.

[14] Breslau, L., et al (1999); Web Caching and Zipf-like Distributions: Evidence and Implications. ISSN 0743-166X, *IEEE INFOCOM*, 1, 126-134.

[15] Liu, L. et al (2005); Weighted Load Balancing for Web Server Clusters with Caching. Working paper. Hong Kong University of Science and Technology.