Cache Placement Optimization in Hierarchical Networks: Analysis and Performance Evaluation

Wenzhong Li[1,2], Edward Chan[2], Yilin Wang[1,2], Daoxu Chen[1], and Sanglu Lu[1]
[1]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu, 210093, China
[2]Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong
Email: lwz@dislab.nju.edu.cn

Abstract. Caching popular content in the Internet has been recognized as one of the effective solution to alleviate network congestion and accelerate user information access. Sharing and coordinating in cache data placement provide an opportunity to improve system performance. This paper studies cache placement strategies and their performance in hierarchical network environments. A theoretical model is introduced to analyze the access cost of placing a set of object copies in the cache hierarchy, under which the object placement problem is formulated as an optimization problem. The problem is proved to be divided into subproblems, and a dynamic programming algorithm is proposed to obtain the optimal solution. Performance of different caching strategies is evaluated using simulations. It is shown that the proposed algorithm outperforms other cache placement strategies in hierarchical caching systems.
Keywords: cooperative caching, hierarchical caching system, cache placement and replacement

1 Introduction

Data caching is a widely used technique to improve system performance, i.e. reduce network traffic, alleviate server load and decrease access latency. Danzig et. al. first showed that by providing caches in a hierarchical arrangement, the network bandwidth consumed for file transfer could be significantly reduced [4]. Following these findings, a hierarchical caching system was implemented in the Harvest system [2].

Coordinating object placement is one of the important issues in hierarchical caching systems. Given a set of caches, their network distances and access frequencies, cache placement and replacement decides which objects should be stored in the caching system and which object should be removed in order to minimize the total access cost [5].

In this paper, we study the performance of placement strategies for hierarchical caching using both analytical model and simulations. We first setup a theoretical model to analyze the accumulative access cost of cache placement

strategies. Then we formulate the object placement problem as an optimization problem. The problem is further proved to be divided into subproblems, thus optimal solution using dynamic programming is proposed. We use simulation to evaluate the performance of different hierarchical caching algorithms. Experimental results show that the proposed strategy outperforms most existing hierarchical caching algorithms.

2 Related Work

Hierarchical web caching was first proposed in the context of the Harvest [2] project, where a series of caches hierarchically are arranged in a tree-like structure. Requests are first received at the leaf caches and are routed upwards until they reach a cache (or the content server) that stores a copy of the requested document. And then, the object is sent back on the reverse path to the client, each cache on this path gets to store a copy of the object. However, caching redundant object copies may not be an efficient approach and recent work [6, 7, 3] has shown that the scheme can be improved.

Che et. al. [3] analyzed an uncooperative two-level hierarchical caching system where the LRU algorithm is locally run at each cache. On the basis of this analysis, a new algorithm referred to as *Filter* is proposed. A cache can be viewed roughly as a low pass filter with its cutoff frequency equal to the inverse of the characteristic time. Documents with access frequencies lower than this cutoff frequency will have good chances to pass through the cache without storing a local copy. However, Filter incurs additional complexity as it needs to estimate the request frequency of each requested document in all clients and disseminate this information to all the caches.

Laoutaris et. al. studied several meta algorithms for hierarchical web caching, namely Prob, LCD (Leave Copy Down) and MCD (Move Copy Down) [6]. Comparison of the three algorithms with the LCE and Filter in [7] showed that LCD performs the best under all the studied scenarios. It even appears to perform better than Filter, despite the fact that it is much simpler.

Korupolu et. al. investigated the placement problem for hierarchical cooperative caching, that is, how to fill the available cache space with copies of objects in such a way that average access cost is minimized [5]. Both exact and approximate polynomial-time algorithms were presented. The exact algorithm was based on a reduction to a min-cost flow problem. However, it does not appear to be practical for large problem size.

Tang et. al. proposed a coordinated en-route web caching scheme [11]. In this scheme, cache status information along the routing path of a request is used in dynamically determining where to cache the requested object and what to replace if there is not enough space. The object placement problem is formulated as an optimization problem and the optimal locations to cache the object are obtained using a dynamic programming algorithm. Similar solution can be found in [8]. Their work are the basis of our analysis.

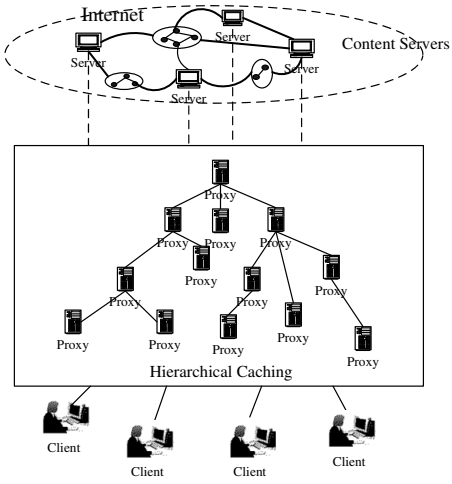


Fig. 1. Hierarchical caching architecture

3 Hierarchical Caching System

A hierarchical caching architecture is illustrated in Figure 1. In this architecture, a set of cache servers (proxies) are arranged in a hierarchical tree-like form, with the leaf nodes corresponding to the lowest level caches closest to the end users and the root nodes corresponding to the highest level caches. The caches in the higher level can be shared by the caching proxy servers in the lower level, forming a highly scalable caching hierarchy.

User requests travel from a given leaf node towards the root node, until the requested document is found. If the requested document cannot be found even at the root level, the request is redirected to the content server containing the document. When the document is found, it is passed down through the reverse path to the client. Each cache on the reverse path decides to cache the document or not according to a chosen strategy.

We address an important issue in designing hierarchical caching systems: how to appropriately deploy cache objects so that the total user access cost in minimized. In the following sections, we will present a theoretical model to investigate this problem.

4 Hierarchical cache placement problem

Figure 2 depicts a scenario of data access in hierarchical caching. A leaf proxy v initiates a query for object O. The query is forwarded to the higher level of the hierarchy, until it is served at a proxy u. Let node *0* be proxy u, node n be proxy v, and *node* n-1,n-2, $\cdots$, 1 denote the proxies on the path from v to u.

Let f_i be the access frequency of object O observed by node i, i.e. the number of requests for object O passing through v to the number of total requests. When

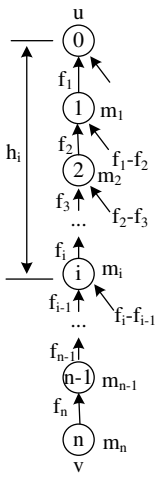


Fig. 2. Data access model for hierarchical caching

object O is passed down from node *0* to node *n*, a copy of O can be dynamically placed in some of the proxies along the path. The question is which nodes should cache a copy of object O, so that the total access cost is minimized.

4.1 Cache Placement

Cache placement assigns copies of objects to the nodes along the path subject to the cache capacity constraints. The set of nodes which are chosen to cache the object is called a *placement* of cache copies. We first give a formal definition of placement for hierarchical caching.

Definition 1 (HiePlacement). *Suppose object O is cached in K intermediate nodes* $d_1, d_2, \cdots, d_K$*, where* $0 \leq K \leq n$*, and* $1 \leq d_1 < d_2 < \cdots < d_K \leq n$*. The set* $D_K = \{d_1, d_2, \cdots, d_K\}$ *is a subset of the nodes along the path. We call* D_K *a HiePlacement (short for hierarchical placement) of the set* $\{1,2,\cdots,n\}$*.*

4.2 Access cost

The cost that a node fetches the requested data object from a near proxy is called *access cost* of the object. The access cost can be evaluated by different metrics, i.e. the network delay to fetch the object, the total bandwidth consumed and the communication overhead caused. In this paper, we use network latency to measure the access cost of an object. Please notice that the strategy proposed in the following sections can be also apply to other cost metrics. Assume the access delay is calculated by the hops that the object travels through to serve a query. The access cost of object i depends on: the access frequency to the object,

denoted by f_i, and the distance to fetch the object, denoted by $dist_i$. Formally the access cost can be expressed as:

$$cost_i = f_i \cdot dist_i. \tag{1}$$

4.3 Replacement Cost

As a proxy has limited cache space, when a new object is inserted, one or more objects should be removed from the cache to make room for the new object. If an object is removed, the access cost to it is increased, which is referred to as *replacement cost*. The replacement cost of an object equals to the total access cost of the evicted objects.

We employs a greedy cache replacement strategy in our scheme: each object is associated with a replacement cost value; when cache replacement occurs, the objects with the smallest replacement cost are selected, until sufficient space is created.

4.4 Accumulative Access Cost

We now derive the accumulative access cost under a HiePlacement. Consider the scenario of Figure 2. Assume $D_K = \{d_1, d_2, \cdots, d_K\}$ $(1 \leq d_1 < d_2 < \cdots < d_K \leq n)$ is a HiePlacement. Assume each request is satisfied by the closest copy of the requested object. We use a function $dist(i, a, D_K)(a \leq i)$ to calculate the distance from node i to a closest node holding the requested object under HiePlacement D_K. If no such node exists, it equals to $i-a$. Formally, the function can be defined as:

$$dist(i, a, D_K) = i - \tau \tag{2}$$

where

$$\tau = max\{a, d_j(a \leq d_j \leq i)\}$$

According to equation 1, for node i, the access cost to the object is $f_i \cdot dist(i, a, D_K)$. We now consider the access cost of all nodes in the path. As shown in the data access model in Figure 2, requests for O that go through node i must also pass through nodes i-1,i-2,$\cdots$,1. Let $F_i = f_i - f_{i+1}$. The total access cost to object O under HiePlacement D_K is (Let $f_{n+1} = 0$):

$$\sum_{i=1}^{n} F_i \cdot dist(i, 0, D_K)$$

By placing an object in node i, there is a replacement cost due to cache replacement, denoted by m_i. Define a *penalty* function as:

$$penalty(i, D_K) = \begin{cases} m_i & if\ i \in D_K; \\ 0 & else \end{cases}$$

The accumulative access cost in Figure 2 under HiePlacement D_K can be calculated as:

$$\sum_{i=1}^{n}(F_i \cdot dist(i, 0, D_K) + penalty(i, D_K)) \tag{3}$$

Our objective is to find an optimal HiePlacement, so that the total access cost in equation (3) is minimized. We provide a more general definition of the placement problem as following.

Definition 2 ((a,b)-HiePlacement problem). *Given two integers a,b ($0 \leq a \leq b$), and a set of real number f_a, f_{a+1}, $\cdots$, f_{b-1}, f_b, m_a, m_{a+1}, $\cdots$, m_{b-1}, m_b, where $f_a \geq f_{a+1} \geq ... \geq f_b \geq 0$, and $m_i \geq 0$ (i=a,a+1,...,b), Assume $D_K = \{d_1, d_2, \cdots, d_K\}$ $(a < d_1 < d_2 < \cdots < d_K \leq b)$ is a HiePlacement of the set $\{a+1, a+2, ..., b\}$. Let $F_i = f_i - f_{i+1}$ and $f_{b+1} = 0$. Define an objective function as*

$$C(a, b, D_K) = \sum_{i=a+1}^{b} (F_i \cdot dist(i, a, D_K) + penalty(i, D_K)). \tag{4}$$

The $(a, b) - HiePlacement$ problem is: find K and D_K, so that $C(a, b, D_K)$ is minimized.

According to the definition, the object placement problem in Figure 2 is simply a $(0, n) - HiePlacement$ problem.

5 Solution

Before presenting our solution, we first give the following observation on the objective function.

Theorem 1. *Assume $D_K = \{d_1, d_2, ..., d_K\}$ $(a < d_1 < d_2 < \cdots < d_K \leq b)$ is a HiePlacement of the set $\{a+1, a+2, ..., b\}$. $C(a, b, D_K)$ is an objective function of an $(a, b) - HiePlacement$ problem. For any $\mu(1 \leq \mu \leq K)$, the following equation should be satisfied:*

$$C(a, b, D_K) = C(a, d_\mu - 1, \{d_1, ..., d_{\mu-1}\}) + m_{d_\mu} + C(d_\mu, b, \{d_{\mu+1}, ..., d_K\}) \tag{5}$$

Proof. For any $\mu(1 \leq \mu \leq K)$, the set of D_K can be divided into three subsets: $P = \{d_1, ..., d_{\mu-1}\}$, $Q = \{d_\mu\}$, $R = \{d_{\mu+1}, ..., d_K\}$.

$$\begin{aligned}
&C(a, d_\mu - 1, P) + m_{d_\mu} + C(d_\mu, b, R) \\
&= \sum_{i=a+1}^{d_\mu - 1} (F_i \cdot dist(i, a, P) + penalty(i, P)) + m_{d_\mu} \\
&+ \sum_{i=d_\mu+1}^{b} (F_i \cdot dist(i, d_\mu, R) + penalty(i, R))
\end{aligned}$$

According to the definition of *dist* and *penalty*, we have:

(a) when $a < i < d_\mu$, $dist(i, a, P) = dist(i, a, D_K)$ and $penalty(i, P) = penalty(i, D_K)$;

(b) when $i = d_\mu$, $dist(i, a, D_K) = 0$ and $penalty(d_\mu, D_K) = m_{d_\mu}$;

(c) when $d_\mu < i \leq b$, $dist(i, d_\mu, R) = dist(i, a, R) = dist(i, a, D_K)$ and $penalty(i, R) = penalty(i, D_K)$.

So,

$$
\begin{aligned}
& C(a, d_\mu - 1, P) + m_{d_\mu} + C(d_\mu, b, R) \\
&= \sum_{i=a+1}^{d_\mu - 1} (F_i \cdot dist(i, a, D_K) + penalty(i, D_K)) + F_{d_\mu} \cdot 0 + penalty(d_\mu, D_K) \\
&+ \sum_{i=d_\mu+1}^{b} (F_i \cdot dist(i, a, D_K) + penalty(i, D_K)) \\
&= \sum_{i=a+1}^{b} (F_i \cdot dist(i, a, D_K) + penalty(i, D_K)) \\
&= C(a, b, D_K)
\end{aligned}
$$

The following Theorem shows that the optimal HiePlacement can be obtained by solving subproblems.

Theorem 2. *Assume a, b, K are integers, $0 \leq a \leq b$ and $0 \leq K \leq b-a$. Suppose $D_K = \{d_1, d_2, ..., d_K\}$ $(a < d_1 < d_2 < \cdots < d_K \leq b)$ and $D_\mu = \{q_1, q_2, ..., q_\mu\}$ $(a < q_1 < ... < q_\mu < d_K)$ are two HiePlacements. If D_K is an optimal placement to the $(a, b) - HiePlacement$ problem, and D_μ is an optimal placement to the $(a, d_K - 1) - HiePlacement$ problem, then $D = D_\mu + \{d_K\}$ is also an optimal placement to the $(a, b) - HiePlacement$ problem.*

Proof. As D_μ is an optimal placement to the $(a, d_K - 1) - HiePlacement$ problem, we have

$$C(a, d_K - 1, D_\mu) \leq C(a, d_K - 1, \{d_1, d_2, ..., d_{K-1}\})$$

According to Theorem 1, we have

$$
\begin{aligned}
C(a, b, D) &= C(a, d_K - 1, D_\mu) + m_{d_K} + C(d_K, b, \emptyset) \\
&\leq C(a, d_K - 1, \{d_1, d_2, ..., d_{K-1}\}) + m_{d_K} + C(d_K, b, \emptyset) \\
&= C(a, b, D_K) \qquad (6)
\end{aligned}
$$

On the other hand, since D_K is an optimal placement to the $(a, b) - HiePlacement$ problem,

$$C(a, b, D_K) \leq C(a, b, D) \qquad (7)$$

Combining (6) and (7), we have

$$C(a, b, D) = C(a, b, D_K).$$

Hence, the theorem is proven.

Theorem 2 shows that the problem can be divided into subproblems, thus dynamic programming can be applied to obtain an optimal solution of the problem. Let $OPT(a, b, x)$ be the minimum access cost of the $(a, b) - HiePlacement$ problem and $x(x \leq a)$ be the closest node to a which holds a copy of the requested object. Let $dist(a, x)$ denote the network distance between node a and node x. We can devise a dynamic programming solution to the $(a, b) - HiePlacement$ problem as follows.

$$\begin{cases} OPT(a, a, x) = min\{m_a, f_a \cdot dist(a, x)\}; \\ OPT(a, b, x) = min\{OPT(a + 1, b, a) + m_a, OPT(a + 1, b, x) + F_a \cdot dist(a, x)\} \end{cases}$$
$$(0 \leq a \leq b \leq n) \tag{8}$$

According to section 4.4, the solution of the object placement problem in Figure 2 is $OPT(1, n, 0)$.

6 Performance Evaluation

6.1 The Simulation Environment

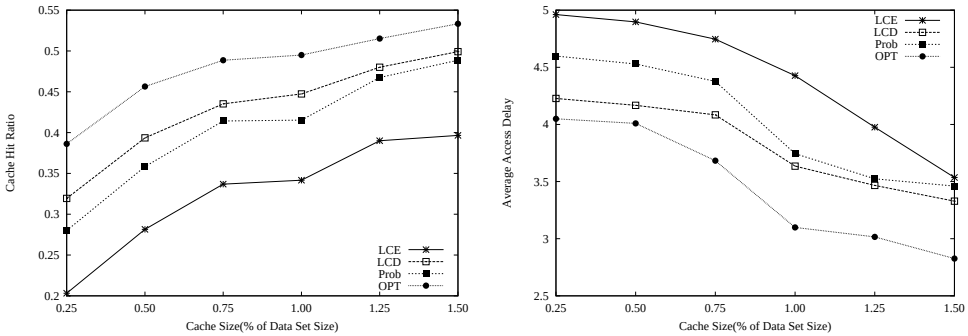
In our simulation, we employ a hierarchical network model similar to [9]. We model the network topology as a tree with L levels. For each node in the tree, if it is not a leaf node, it has a set of children. The number of children of a node is uniformly distributed in the range of [1,M]. A content server resides outside the hierarchy. Assume all queries can be served at the content server.

Each node needs to maintain visit history information. Similar to [10], we estimate the access probability p_i using a "sliding window" of K most reference times as: $p_i = \frac{K}{t - t_K}$, where t is the current time and t_K is the time of the Kth most recent reference. The length of sliding window is set to 1000, and the value of K is set to 3 in our experiments.

We use synthetic workload to evaluate the performance of the proposed cache replacement policies. The synthetic workload simulates a data set with 10000 items, averaging 25 KB per item. Assume the query arrival rate follows a Poisson process with a mean arrival rate of λ. A Zipf-like distribution is used to model the possibility that a data item is requested [1]. In this model, the access probability p_i of data item d_i is given by $p_i = \frac{C}{i^\alpha}$, where $C = (\sum_{k=1}^{N} \frac{1}{k^\alpha})^{-1}$. The parameter $\alpha(0 \leq \alpha \leq 1)$ is the skewness parameter of the Zipf-like distribution, indicating the degree of concentration of requests. We assume homogeneous data access pattern in the mobile clients. That is, all client requests follow the same Zipf pattern.

6.2 Performance Analysis

We employ two widely used metrics in our performance evaluation: *cache hit ratio*, and *average access delay*. Three cache placement algorithms are included for comparison: LCE, Prob and LCD [7]. The proposed algorithm is referred to as "OPT" algorithm in our later analysis.



(a) Cache Hit Ratio. (b) Average Access Delay.

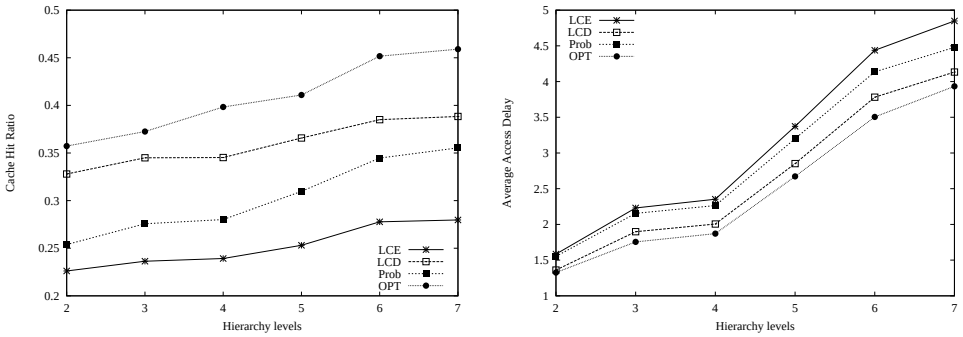
Fig. 3. Performance under various cache size.

Impact of the Cache Size: We first compare the performance of the different cache schemes under different cache size. Each node in the experiment has a relative cache size, which is the percent of the total size of the total data set, and is varied from 0.25 percent to 1.50 percent in our simulation.

Figure 3 compares the cache hit ratio as a function of the relative cache size. As shown in Figure 3(a), the cache hit ratio of all the cache schemes steady improves as the cache size increases. the cache hit ratio of LCE scheme increases from 0.2 to 0.4 (about 100 percent improvement), and the cache hit ratio of OPT scheme increases from 0.37 to 0.53 (about 43 percent improvement). It also can be seen that LCE has the lowest cache hit ratio in all conditions. Prob performs better than LCE, but does not fare as well as LCD. However LCD, which is considered a simple and efficient cache placement algorithm in hierarchical caching, is not the best in our simulation. OPT obtains the highest cache hit ratio, which is about 10 to 25 percent higher than LCD in most conditions.

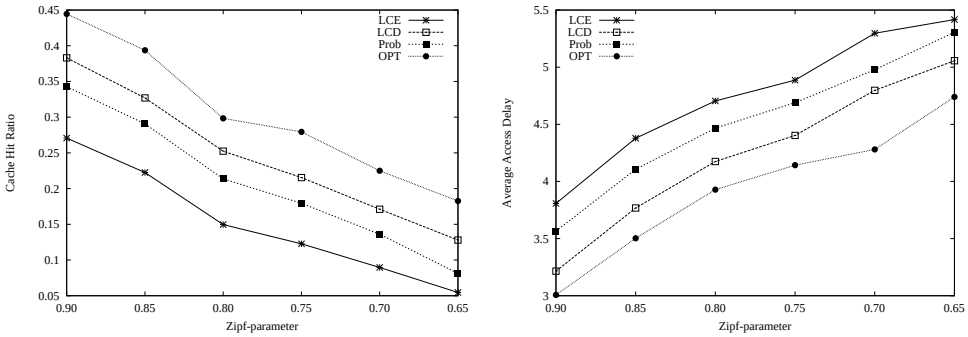
Comparison of average access delay is shown in Figure 3(b). It can be seen that access delay decreases with the increasing cache size. For example, the average access delay of LCE and OPT decrease from 5 down to 3.6 and from 4.1 down to 2.8 accordingly (Figure 3(b)). LCE has the highest access delay in both workloads. Prob is a little better than LCE. OPT has the lowest access delay, which indicates data access in the OPT scheme is much faster, and an improvement of 20-30 percent is observed.

Impact of the Network Hierarchy Level: Network hierarchy level has a significant impact on the performance of a caching system. The larger the network, the more proxies cooperate in the caching, and the more likely that an object will obtain a hit in the cache. On the other hand, if L is large, a query needs to travel more hops before it reaches the content server. We explore the influence of network topology in this section, with the value of L ranging from 2 to 7.



(a) Cache Hit Ratio. (b) Average Access Delay.

Fig. 4. Performance under different hierarchy levels.



(a) Cache Hit Ratio. (b) Average Access Delay.

Fig. 5. Performance under various Zipf parameters.

As it is shown in Figure 4(a), cache hit ratio increases with the number of hierarchical levels increasing, but not as remarkable as the influence of increasing cache size. Again, LCE has the lowest cache hit ratio, and OPT outperforms the other cache schemes.

Figure 4(b) presents the experiment result of average access delay. One can see that the access delay also increases as the number of levels in the hierarchy increases. This is because when L becomes larger, a query generated in the leaf node needs to travel more hops to reach the content server, thus the access delay becomes longer if a query is missed in all caches. The OPT algorithm still achieves the lowest access delay among the cache schemes.

Impact of the Data Access Pattern: Data access pattern can affect the performance of a caching system [1]. As mentioned before, the Zipf skewness parameter α indicates the degree of concentration of file accesses. By changing

the value of α, we generate a number of synthetic workloads with different data access patterns to investigate the performance of the various cache replacement strategies.

Figure 5 shows the performance when the Zipf skewness parameter varies from 0.9 to 0.65. We can see from the graph that when the Zipf parameter decreases, the performance decrease accordingly. A larger Zipf parameter value means more queries are focused on a set of "hot" data items. As a result, cache replacement policies can easily identify the frequently accessed data items and keep them in the cache. This explains the performance degradation when the data access interest becomes more sparse. Comparing the performance results shown in Figure 5, we can see that OPT still performs the best under different query patterns.

7 Conclusion

Coordinating data placement in hierarchical caching system helps to reduce user access cost. The novelty of our work is that we study coordinated cache placement strategies using an analytical model and evaluate their performance in hierarchical network environment. The object placement problem is formulated as an optimization problem and the optimal solution is derived using dynamic programming algorithm. Performance of cache placement strategies are evaluated by simulations, which shows that the proposed algorithm performs well in hierarchical caching system.

Acknowledgment

The work described in this paper is partially supported by the National High-Tech Research and Development Program of China (863) under Grant No. 2006AA01Z199; the National Natural Science Foundation of China under Grant No. 90718031,60573106,60721002; the National Basic Research Program of China (973) under Grant No. 2006CB303000.

References

[1] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM'99:Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 126–134, 1999.

[2] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *ATEC'96: Proceedings of the Annual Technical Conference on USENIX 1996 Annual Technical Conference*, pages 13–13, Berkeley, CA, USA, 1996. USENIX Association.

[3] H. Che, Z. Wang, and Y. Tung. Analysis and design of hierarchical web caching systems. In *In Proc. of Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM2001)*, pages 1416–1424. IEEE, 2001.

[4] P. B. Danzig, R. S. Hall, and M. F. Schwartz. A case for caching file objects inside internetworks. *SIGCOMM Comput. Commun. Rev.*, 23(4):239–248, 1993.

[5] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Placement algorithms for hierarchical cooperative caching. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 586–595, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[6] N. Laoutaris, H. Che, and I. Stavrakakis. The lcd interconnection of lru caches and its analysis. *Perform. Eval.*, 63(7):609–634, 2006.

[7] N. Laoutaris, S. Syntila, and I. Stavrakakis. Meta algorithms for hierarchical web caches. In *International Conference on Performance, Computing, and Communications (PCCC2004)*, pages 445– 452. IEEE, 2004.

[8] K. Li, H. Shen, F. Y. L. Chin, and S. Q. Zheng. Optimal methods for coordinated enroute web caching for tree networks. *ACM Transactions on Internet Technology*, 5(3):480–507, 2005.

[9] P. Rodriguez, C. Spanner, and E. W. Biersack. Analysis of web caching architectures: hierarchical and distributed caching. *IEEE/ACM Transactions on Networking*, 9(4):404–418, 2001.

[10] J. Shim, P. Scheuermann, and R. Vingralek. Proxy cache algorithms: Design, implementation, and performance. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):549–562, 1999.

[11] X. Tang and S. T. Chanson. Coordinated en-route web caching. *IEEE Transactions on Computers*, 51(6):595–607, 2002.