

Coordinated Placement and Replacement for Grid-Based Hierarchical Web Caches

Wenzhong Li, Kun Wu, Xu Ping, Ye Tao, Sanglu Lu, and Daoxu Chen

State Key Laboratory for Novel Software Technology,
Department of Computer Science and Technology, Nanjing University,
P.R. China 210093
lwz@dislab.nju.edu.cn

Abstract. Web caching has been well accepted as a viable method for saving network bandwidth and reducing user access latency. To provide cache sharing on a large scale, hierarchical web caching has been widely deployed to improve the scalability of content dissemination through the World Wide Web. In this paper, we present GHC, a grid-based hierarchical web caching architecture, for constructing efficient cache hierarchy in Grid environment. Basing on the GHC architecture, we also proposed HCPR algorithm, which is a novel coordinated placement and replacement algorithm for hierarchical web caching scheme. The principle of HCPR algorithm is to improve cache usage, to cache hot document close to clients so as to minimize the number of hops to locate and access Web documents. The simulation study shows that the HCPR algorithm has higher cache hit ratio and lower access latency compared with the traditional hierarchical caching algorithms.

1 Introduction

Cooperative web caching has been recognized as one of the effective solutions to alleviate web service bottlenecks, reduce bandwidth consumption, access latency, as well as server load. There are basically two types of cooperative architecture—Hierarchical and Distributive. With hierarchical caching [1], caches are placed at multiple levels of the network. With distributed caching, caches are only placed at the bottom levels of the network and there are no intermediate caches [2]. There are two important issues to cooperative caching: object location and object placement/replacement. Object location focuses on finding nearby copies of objects [3]. Object placement/replacement attempts to solve the problem of making storage decisions and coordinating object replacement. Efficient coordinated object placement and replacement algorithms have been widely studied in [2].

Grid computing has emerged as the next major distributed computing platform [4]. Data grid addresses the problem of storage and data management, network-intensive data transfers and data caching and replication, while maintaining high reliability and availability of the data. Web caching Grid [5] uses grid caches to achieve load balancing among multiple Web servers, in which remote servers in a Grid may function as caches when request rates are high and quality of service (QoS) for response times are in danger of degradation. GCaching [6], a Grid-based cooperative caching system, organizes a set of distributed caching proxy servers to form a “caching pool” for large-scale distributed caching data sharing.

This paper addresses the issue of coordinated placement and replacement for grid-based hierarchical web caches. The remainder of this paper is organized as follows. Section 2 present the grid-based hierarchical web caching architecture called GHC. In Section 3, we introduce HCPR algorithm. In Section 4, we evaluate the performance of HCPR algorithm by means of synthetic simulation. Finally, we conclude this paper in Section 5.

2 GHC: A Grid-Based Hierarchical Web Caching Architecture

Web caching sharing was first proposed in the context of the Harvest [1] project. The basic idea of Harvest system is to have a series of caches hierarchically arranged in a tree-like structure and to allow those caches to leverage from each other when an object request arrives and the receiving cache do not have that object. Hierarchical architecture is widely deployed for large-scale web caching sharing.

Web caches need to be arranged in a hierarchical structure due to the hierarchical nature of the Internet. In the Grid context, Hierarchical discovery scheme is used for resource registration, querying and sharing among multiple virtual organizations (VOs). A Grid-based hierarchical Web caching architecture called GHC is presented in Fig. 1. In this architecture, web caches are arranged in a hierarchical tree-like form, which allow caches sharing and coordinating among VOs. The leaf nodes correspond to the end users. 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 web server containing the document.

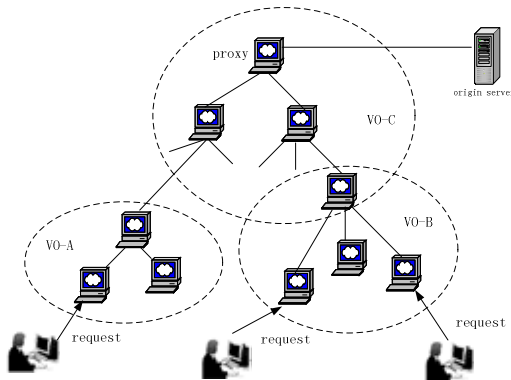


Fig. 1. The GHC Architecture

3 HCPR: A Hierarchical Web Caching Placement and Replacement Algorithm

In the traditional hierarchical web caching, requests are first received at the leaf caches and are routed upwards until they reach a cache that stores a copy of the requested document. If the document is not found at any cache level, the root cache contacts directly the origin server. When the requested document is found, it is sent

on the reverse path to the client, and each cache on this path gets to store a local copy of the document so as to service future requests. Such traditional hierarchical web caching algorithm is known as LCE (Leaving Copies Everywhere) algorithm, which has been considered as a de facto behavior.

One remarkable disadvantage of LCE is documents redundancy [7]. Basing on the GHC architecture, we present a history-based hierarchical caching replacement algorithm called HCPR. The principle of HCPR algorithm is to improve cache usage, to cache data close to clients and to minimize the number of hops to locate and access Web documents.

```

Procedure to be invoked on reference to document p at time t:
If p does not exist in the HIST table
then
insert p to the HIST table;
HIST(p,1)=0;
end if
For i=2 to K do HIST(p, i)=HIST(p, i-1) od
HIST(p, 1)=t;
If be a leaf node then //leaf node replacement algorithm
if p is already in the local cache then
return the requested document;
else
request_forward(FatherProxy, p);
if p is fetched from the origin server then
//select replacement document ;
min=t
for all document q in the local cache do
if HIST(q,K)<min then
victim_document=q ;
min=HIST(q,K) ;
end if
od
cache document p, replace victim_document ;
document_upgrade(fatherProxy, victim_document)
end if;
end if
else // non-leaf node replacement algorithm
if p is already in the local cache then
return the requested document;
else
request_forward(FatherProxy, p);
return the requested document;
end if
end if

```

Fig. 2. Pseudo-code of the HCPR Algorithm

HCPR algorithm uses documents reference history information for caching placement and replacement decision. The basic idea of HCPR algorithm is placing the most frequently referenced documents in the leaf node of the hierarchy. The less frequently a document is referenced, the higher level it is placed. HCPR Algorithm tries to avoid redundancy by placing documents in a “bottom-up” manner. Documents are first cached at the bottom level caches (leaf nodes). When the bottom level reaches the cache predetermined limits, documents are replaced according to their reference history. By using a page upgrade process, the replaced document from level m cache is brought to its upper level cache.

Suppose that r_1, r_2, \dots, r_n are the requests for Web documents as logged at the time units t_1, t_2, \dots, t_n , respectively. Basing on the idea of [8], the reference history record of document x is defined as follows:

$$\text{HIST}(x, k) = \begin{cases} t_i & \text{if there are exactly } k - 1 \text{ references between times } t_i \text{ and } t_n \\ 0 & \text{otherwise} \end{cases}$$

Here t_i denotes the time of the first of the last k references to x . $\text{HIST}(x, k)$ is a time metric which defines the time of the past k -th reference to document x .

In the GHC architecture, each caching proxy runs HCPR algorithm. Figure 2 presents the HCPR algorithm in pseudo-code.

When caching replacement occurs, the less frequently referenced document is replaced. The replaced document will be “upgraded” to its father proxy. The pseudo-code of document upgrade process is presented in Figure 3.

```

Procedure to be invoke on receiving a document upgrade request:
p=the document to be upgraded;
If p does not exist in the HIST table then
    insert p to the HIST table;
    HIST(p,1)=0;
end if
For i=2 to K do HIST(p, i)=HIST(p, i-1) od
HIST(p, 1)=t;
if p is already in the local cache then
    return;
else
    //select replacement document
    min=t;
    for all document q in the local cache do
        if HIST(q,K)<min then
            victim_document=q;
            min=HIST(q,K);
        end if
    cache document p, replace victim_document ;
    if not reaches the root level
        document_upgrade(fatherProxy, victim_document) ;
    end if

```

Fig. 3. Pseudo-code of Document Upgrade Process

4 Synthetic Simulations

In order to evaluate the performance of the HCPR algorithm, we conduct simulations using synthetically generated Zipf-like document popularity distributions [9] to model client request. Let N be the total number of web pages in the universe. Let $p(i)$ be the probability of requesting the i th most popular document. We assume that $p(i)$,

defined for $i=1, 2, \dots, N$, has a “cut-off” Zipf-like distribution given by $P(i) = \frac{C}{i^\alpha}$,

where $C = (\sum_{i=1}^N \frac{1}{i^\alpha})^{-1}$.

In our simulation, we follow the network model presented in [10]. We model the network topology as a full Q-ary tree with L levels. An origin server resided outside the hierarchy is considered to be at a conceptual level L+1. A request travels from a leaf cache towards the location of the hit. If a request hits in the l-level cache, the hops of fetching the document are defined to be l-1.

Two metrics are used to evaluate our algorithm: cache hit ratio and Average Access Distance (AAD). Cache hit ratio is defined by the ratio of the number of hits in the cache hierarchy to the total number of user requests. AAD denotes the average hops to fetch a document.

For all the case studies in this paper, we set L=3, Q=4 and N=100000. For the reference history, we set K=2. We assume all documents have the same size. We also assume the total storage capacity is S unit sized documents, equally allocated to the n caches of a hierarchy with each cache taking S/n units of storage. We study the performance of HCPR algorithm under Zipf-like distribution, with $\alpha=0.7, 0.8$ and 0.9 . Figure 4 to Figure 9 present the cache hit ratio and average access distance for the HCPR and LCE algorithms under different α value.

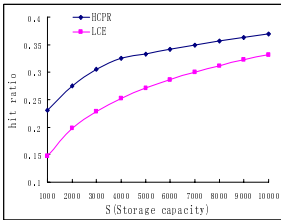


Fig. 4. Cache hit ratio for LCE and HCPR, under Zipf-like (0.9) requests. N=100000, L=3, Q=4.

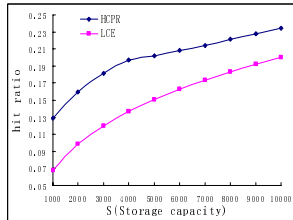


Fig. 5. Cache hit ratio for LCE and HCPR, under Zipf-like (0.8) requests. N=100000, L=3, Q=4.

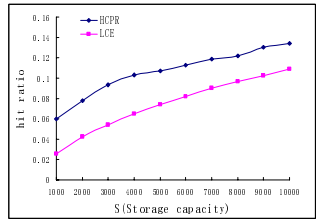


Fig. 6. Cache hit ratio for LCE and HCPR, under Zipf-like (0.7) requests. N=100000, L=3, Q=4.

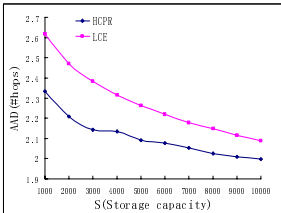


Fig. 7. Average Access Distance for LCE and HCPR, under Zipf-like (0.9) requests. N=100000, L=3, Q=4.

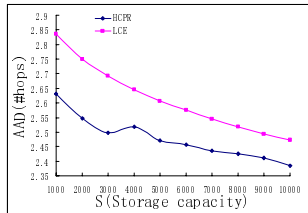


Fig. 8. Average Access Distance for LCE and HCPR, under Zipf-like (0.8) requests. N=100000, L=3, Q=4.

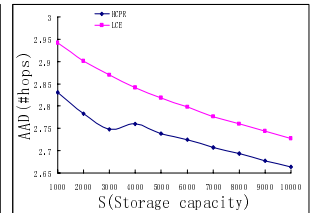


Fig. 9. Average Access Distance for LCE and HCPR, under Zipf-like (0.7) requests. N=100000, L=3, Q=4.

The simulation study shows that the HCPR algorithm has higher cache hit ratio and lower access latency compared with the traditional hierarchical caching algorithm.

5 Conclusions

Hierarchical web caching has been widely deployed to improve the scalability of content dissemination for large scale caches sharing. In this paper, we present a grid-based hierarchical web caching architecture called GHC. Basing on GHC architecture, we present HCPR algorithm which is a novel coordinated placement and replacement algorithm for hierarchical web caching. The simulation study shows that the HCPR algorithm outperforms the traditional LCE algorithm.

Acknowledgement

This work is partially supported by the National Natural Science Foundation of China under Grant No.60402027; the National High-Tech Research and Development Program of China (863) under Grant No.2004AA112090; the National Basic Research Program of China (973) under Grant No.2002CB312002.

References

1. A. Chankhunthod, P.B. Danzig, C.Neerdaels, et al "A Hierarchical Internet Object Cache," Usenix'96, 1996.
2. M.R. Korupolu and M.Dahlin, "Coordinated Placement and Replacement for Large-scale Distributed Caches," In *Proc. of the IEEE Workshop on Internet Applications*, 1999.
3. H. Zhuge, J. Liu, L. Feng, et al "Query Routing in a Peer-to-Peer Semantic Link Network," *Computational Intelligence*, 21(2): pp197-216, 2005.
4. I.Foster, C.Kesselman, "The Grid, BluePrint for a New Computing Infrastructure," San Francisco, Morgan Kaufmann Publisherd Inc, 1999
5. Catherine H.Crawford, Daniel M.Dias, Arun K.Iyengar, et al "Commercial Applications of Grid Computing", IBM Research Division, Thomas J.Watson Research Center, 2003
6. LI Wen-Zhong, GU Tie-Cheng, et al "GCaching: A Grid-Based Cooperative Caching System", *Journal of Computer Research and Development*, 41(12): pp2211-2217, 2004.
7. Nikolaos Laoutaris, Sofia Syntila and Ioannis Stavrakakis, "Meta Algorithms for Hierarchical Web Caches," In *IEEE International Performance Computing and Communications Conference (IEEE IPCCC)*, 2004
8. E.J. O'Neil, and G. Weikum. "The LRU-K Page Replacement Algorithm for Database Disk Buffering," In *Proc. of the 1993 ACM Sigmod International Conference on Management of Data*, pp297-306, 1993.
9. L. Breslau, P. Cao, L. Fan, et al "Web caching and Zipf-like Distributions: Evidence and Implications." In *Proc. of IEEE INFOCOM'99*, pp126-134, New York, USA, 1999.
10. P. Rodriguez, C.Spanner, and E.W.Biersack, "Web Caching Architectures: Hierarchical and Distributed Caching," In *Proc. of WCW*, 1999.