# Coordinated En-Route Web Caching*

Xueyan Tang & Samuel T. Chanson
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
E-mail: {tangxy, chanson}@cs.ust.hk

**Abstract**

Web caching is an important technique for reducing Internet access latency, network traffic, and server load. This paper investigates cache management strategies for en-route web caching environment where caches are associated with routing nodes in the network. We propose a novel caching scheme that integrates both object placement and replacement policies, and which makes caching decisions on all candidate sites in a coordinated fashion. In our 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 low-cost dynamic programming algorithm. Extensive simulation experiments have been performed to evaluate the proposed scheme in terms of a wide range of performance metrics. The results show that the proposed scheme significantly outperforms existing algorithms which consider either object placement or replacement at individual caches only.

**Index Terms:** web caching, web cache management, web object placement, transparent web cache, dynamic programming, performance evaluation, World Wide Web.

## 1 Introduction

The explosive growth in popularity of the World Wide Web is leading to a number of performance problems. The Internet is becoming increasingly congested and popular web sites are suffering from overload conditions due to large number of simultaneous accesses. As a consequence, considerable latency is often experienced in retrieving web objects from the Internet.

*The work described in this paper was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. HKUST6066/00E).

Caching web objects at various components in the network (such as servers, proxies and clients) is an important approach for enhancing web content delivery [1, 2, 3]. With caching, copies of popular objects are stored closer to the users. This has the benefits of reducing network traffic and alleviating server load, thereby decreasing access latency. Web caching is different from traditional memory caching supported by operating systems. Unlike pages, web objects typically have different sizes and very different access frequencies. Web requests to servers and objects are found to exhibit a Zipf-like distribution [4]. In addition, as web servers are geographically located all over the Internet, the performance gain of caching an object depends very much on the network distance from the content server of the object. The issues of object sizes, access frequencies and distances from the servers in the object replacement strategy of a single web cache have been addressed in recent research [5, 6, 7, 8, 9, 10].

To obtain the full benefits of caching, multiple web caches are often deployed which cooperate with each other in serving client requests. Examples include hierarchical caching and distributed caching [11, 12, 13]. Recent advances on transparent web caches [14] has enabled the development of a new caching architecture called *en-route web caching* [15, 16, 17]. In this approach, web caches are associated with routing nodes in the network and are referred to as *en-route caches*. An en-route cache intercepts any client request that passes through the associated routing node. If the requested object is in the cache, the object is sent to the client and the request will not be propagated further upstream. Otherwise, the routing node forwards the request along the regular routing path towards the content server. If no en-route cache is found to contain the target object, the request is eventually serviced by the content server. En-route web caching can be implemented by a number of light-weight techniques such as extending the standard TCP or HTTP protocol in existing IP networks [18, 19] or using an active network where the routers can manipulate the messages flowing through them and perform customized computations [20]. En-route web caching has a number of advantages. First, it is transparent to both content servers and clients. Second, since no request is detoured off the regular routing path, the additional bandwidth consumption and network delay for cache miss are minimized. Moreover, it eliminates the extra overhead of locating the objects such as sending broadcast queries [21] and maintaining directories [22, 23]. Therefore, en-route caching is easy to manage and has good system scalability.

The performance of en-route web caching depends on the locations of the caches and how the cache contents are managed. While the first issue has been studied in recent years [17, 24], the second issue has received little attention. In this paper, we focus on the second issue and explore effective content management techniques for en-route web caches, including object placement and replacement algorithms. Dynamically determining the appropriate number of object copies and placing them in suitable en-route caches are challenging tasks. When a new object is placed in a cache, other objects may need to be removed in order to create room. The interaction effect between object placement and replacement in the set of candidate caches further complicates the problem. Existing caching schemes consider either object placement or replacement at individual caches only and are

not optimal. In this paper, we propose a novel caching scheme that incorporates both object placement and replacement strategies. The scheme dynamically places the object in the caches on the path from the server to the client in a coordinated fashion. We formulate the object placement problem as an optimization problem and obtain the optimal locations for caching objects using a dynamic programming algorithm. The effectiveness of the proposed scheme has been tested by extensive simulation experiments. The results show that our scheme signifi cantly outperforms existing schemes which consider either object placement or replacement at individual caches only.

The rest of the paper is organized as follows. Section 2 summarizes the related work. The proposed coordinated en-route caching scheme is presented in Section 3. Section 4 describes the simulation model, and the experimental results are discussed in Section 5. Finally, Section 6 concludes the paper.

## 2 Related Work

Cache collaboration is an important technique that improves caching performance. Cooperation among caches can be performed in two orthogonal dimensions: horizontal and vertical. Horizontal cooperation is performed by caches that are geographically clustered and have similar distances to the content servers (e.g., [25, 26, 9]). Vertical cooperation is performed by caches that are geographically distributed and have different distances to the content servers (e.g., [15, 27]). Vertical cooperation is more commonly used in en-route caching because the caches are often located throughout the global network instead of being deployed within the user organization only. Therefore, our work has focused on cooperation among caches in the vertical dimension, i.e., the caches on the routing paths of requests. Bhattacharjee *et al.* [15] studied the benefi ts of associating caches with switching nodes throughout the network. Their proposed caching schemes were simple but did not take object size, access frequency and distance from the server into account. Yu *et al.* [27] discussed collaboration between parent and child proxies for hierarchical caching. However, there was no analytical modeling. Hierarchical caching can be viewed as a special case of en-route caching in that the requests are only generated at the leaf nodes of the hierarchy and are routed towards a single root cache. Our model is general and is applicable to hierarchical caching also.

Early work on single web cache management had investigated simple extensions of traditional page replacement algorithms such as LRU and LFU [28]. More recent work has focused on cost-based cache replacement algorithms [5, 29]. The idea is to design a cost function for cache replacement policy that incorporates multiple factors such as object size, access frequency and retrieval cost from the server. A typical example is the LNC-R algorithm proposed by Scheuermann *et al.* [29]. They further proved that the suggested function is optimal (i.e., maximizing delay reduction for a single cache) in a simplifi ed model [8]. However, the scheme automatically places a newly referenced object in the cache without evaluating whether it is benefi cial to do so. This may re-

sult in ineffective use of cache space when there are a large number of caches in the network such as in en-route caching. The approach also incurs high workload on the caches and hence has poor system scalability. Aggarwal *et al.* [7] provided a Size-Adjusted LRU scheme and used an admission control policy to decide whether or not caching an object is worthwhile. However, the authors did not further consider cooperation among different caches.

The cache location problem (i.e., where to place the caches in the network) for en-route caching was studied by Krishnan [17] and Li [24]. This problem is different from the one we are addressing in this paper, namely, where to place web objects given the set of caches, and which objects should be replaced if necessary. In fact, our work and theirs may be integrated in some situations.

# 3 Coordinated En-Route Caching

We model the network as a graph $G = (V, E)$, where $V$ is the set of nodes (routers) in the network, each associated with an en-route cache[1], and $E$ is the set of network links. Every content server or client is attached to a node in $V$. Clients issue requests for web objects maintained by content servers. Each web object is served by exactly one server. Usually, the client and the server of a request are not located at the same site, and there are a lot more clients than servers. For each object $O$, a non-negative cost $c(u, v, O)$ is associated with each link $(u, v) \in E$. It represents the cost of sending a request for object $O$ and the associated response over the link $(u, v)$. If a request travels through multiple network links before obtaining the target object, the access cost of the request is simply the sum of the corresponding costs on all these links. The term "cost" in our analysis is used in a general sense. It can be interpreted as different performance measures such as network latency, bandwidth requirement and processing cost at the cache, or a combination of these measures.

As shown in Figure 1, a request goes along a routing path from the client (denoted by $c$) to the server (denoted by $s$). For any pair of nodes $v_1$ and $v_2$ on the path, we say $v_2$ is at a higher level than $v_1$ if $v_2$ is closer to $s$. In en-route caching, the request is satisfied by the node at the lowest level (denoted by $w$) containing the target object. Notice that $w$ is the same as $s$ if the requested object is not in any cache along the path between $c$ and $s$. It is known that most web objects are relatively static i.e., the access frequency is much higher than the update frequency [6, 30]. We shall assume the objects stored in the caches are up-to-date (e.g., by using a cache coherency protocol [31] if necessary). After the request reaches $w$, the target object (denoted by $R$) is sent along the same path back to the client. Routing paths from all nodes to a given server are represented by a tree topology [17, 24]. For simplicity, symmetric routing is assumed in our analytical model. However, since this assumption may not be valid in some situations [32], we have also modified the proposed coordinated caching scheme to handle routing asymmetry and studied its performance by simulation experiments (see Section 5.3).

[1]For simplicity, we assume that every node is associated with an en-route cache. The following analysis can be easily extended to the case where en-route caches are associated with certain subset of nodes by only including the nodes with caches in the graph.

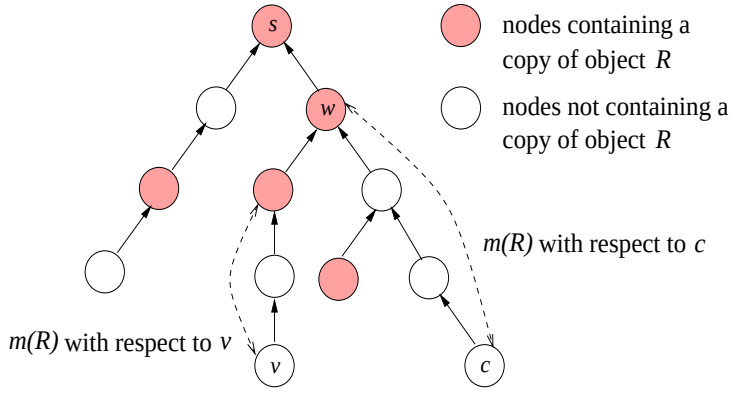


Figure 1: En-route Caching

To reduce the cost of future accesses to object $R$, a copy of $R$ can be dynamically placed in some of the caches along the path between $w$ and $c$ as $R$ is being sent to the client. The issues to be investigated include: (i) which nodes should $R$ be placed (object placement problem); and (ii) which objects to be removed from a cache if there is not enough free space (object replacement problem).

### 3.1 Problem Formulation

The object placement problem is trivial if cache sizes are infi nite, in which case objects can be stored in every cache to minimize total access cost. However, due to limited cache space, one or more objects may need to be removed from the cache when a new object is inserted. Removing an object increases its access cost (referred to as *cost loss*) while inserting an object decreases its access cost (referred to as *cost saving*). The object placement problem for en-route caching is further complicated by *caching dependencies*, i.e., a placement decision at one node in the network affects the performance gain of caching the same object at other nodes. The optimal locations to cache an object depends on the cost losses and cost savings at all the nodes along the routing path. Our objective is to minimize the total access cost of all objects in the network.

We start by computing the cost saving and the cost loss of caching an object at individual nodes. Consider a node $v$. Let $f(O)$ be the *access frequency* of object $O$ observed by node $v$, i.e., the rate of requests passing through $v$ and targeting for $O$. Let $m(O)$ be the *miss penalty* of object $O$ with respect to $v$. The miss penalty is defi ned as the additional cost of accessing the object if it is not in the cache at $v$. In our en-route caching model, $m(O)$ is given by

$$m(O) = \sum_{(u_1,u_2)\in PATH(v,v')} c(u_1, u_2, O),$$

where $v'$ is the nearest higher level node of $v$ that caches $O$, and $PATH(v, v')$ is the set of network links on the path between $v$ and $v'$ (see Figure 1).

Let $R$ be the requested object. Clearly, the cost saving of caching $R$ at $v$ is

$$f(R) \cdot m(R).$$

Computing the cost loss of caching $R$ at $v$ is a bit more complicated. Let $O_1, O_2, \ldots, O_k$ be the objects currently cached at $v$. The cost loss introduced by removing object $O_i$ from the cache is

$$f(O_i) \cdot m(O_i).$$

Obviously, the purged objects should introduce the least total cost loss while creating enough space to accommodate $R$. This is equivalent to the knapsack problem and is NP-hard. The following greedy heuristic can be used to select replacement candidates. Notice that the normalized cost loss (NCL, i.e., the cost loss introduced by creating one unit of free space) of purging $O_i$ is

$$\frac{f(O_i) \cdot m(O_i)}{s(O_i)},$$

where $s(O_i)$ is the size of object $O_i$. The objects in the cache are ordered by their NCLs, and are selected sequentially starting from the object with the smallest NCL until suffi cient space is created for $R$. The cost loss of caching $R$ at $v$ is calculated by summing the cost losses introduced by all the selected objects.

We now formulate the object placement problem on the path between $w$ and $c$. Consider the snapshot when a request for object $R$ is being served (see Figure 2). Let $A_0 = w$ be the content server or the high level node satisfying the object request, $A_n = c$ be the client issuing the request, and $A_1, A_2, \ldots, A_{n-1}$ are the nodes on the routing path from $A_0$ to $A_n$.

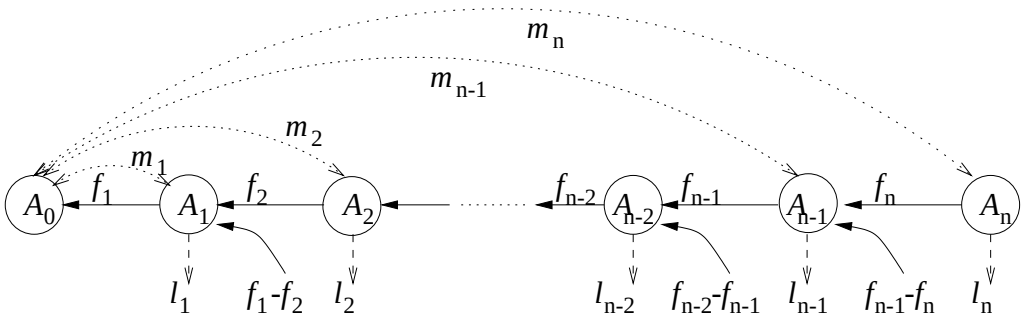


Figure 2: System Model for Coordinated En-Route Caching

Let $m_i$ represent the miss penalty of object $R$ with respect to $A_i$, then

$$m_i = \sum_{j=1}^{i} c(A_{j-1}, A_j, R).$$

Let $f_i$ be the access frequency of object $R$ at $A_i$. Since requests for $R$ that go through $A_i$ must also pass through $A_{i-1}, A_{i-2}, \ldots, A_1$, we have $f_1 \geq f_2 \geq \cdots \geq f_n$. Suppose $R$ is cached in $r$ intermediate nodes

$A_{v_1}, A_{v_2}, \ldots, A_{v_r}$ where $r \geq 0$ and $1 \leq v_1 < v_2 < \cdots < v_r \leq n$. Taking into consideration the caching dependencies of $R$ along the path, the total cost saving is given by

$$\sum_{i=1}^{r}((f_{v_i} - f_{v_{(i+1)}}) \cdot m_{v_i}),$$

where $f_{v_{(r+1)}}$ is set to 0. Now let $l_i$ be the cost loss of evicting objects at node $A_i$ to create enough space for $R$, then the total cost loss is

$$\sum_{i=1}^{r} l_{v_i}.$$

Therefore, the reduction of total access cost in the network is given by

$$\sum_{i=1}^{r}((f_{v_i} - f_{v_{(i+1)}}) \cdot m_{v_i} - l_{v_i}). \quad (1)$$

Our objective is to place object $R$ in a subset of the nodes $\{A_1, A_2, \ldots, A_n\}$ that maximizes the cost reduction (1), thereby minimizing the total access cost.

### 3.2 Dynamic Programming Solution

For the purpose of analysis, we first provide a generalized definition of the problem.

**Definition 1** Given $f_1, f_2, \ldots, f_n, f_{n+1}$; $m_1, m_2, \ldots, m_n$; and $l_1, l_2, \ldots, l_n$ ($n > 0$), where $f_1 \geq f_2 \geq \cdots \geq f_n \geq f_{n+1} = 0$, $m_i \geq 0$ and $l_i \geq 0$ ($i = 1, 2, \ldots, n$). Let $k$ be an integer such that $0 \leq k \leq n$, and $v_1, v_2, \ldots, v_r$ be a set of $r$ integers such that $1 \leq v_1 < v_2 < \cdots < v_r \leq k$. The objective function $\Delta cost(k : v_1, v_2, \ldots, v_r)$ is defined as

$$\Delta cost(k : v_1, v_2, \ldots, v_r) \quad = \quad \sum_{i=1}^{r}((f_{v_i} - f_{v_{(i+1)}}) \cdot m_{v_i} - l_{v_i}),$$

where $f_{v_{(r+1)}} = f_{k+1}$. If $r = 0$, define $\Delta cost(k : \phi) = 0$. Finding $r$ and $v_1, v_2, \ldots, v_r$ that maximize $\Delta cost(k : v_1, v_2, \ldots, v_r)$ is referred to as the $k$-optimization problem. □

The object placement problem formulated in Section 3.1 is simply an $n$-optimization problem i.e., maximizing $\Delta cost(n : v_1, v_2, \ldots, v_r)$. In the following, we develop a dynamic programming algorithm which is inspired by [33] to solve the problem. Theorem 1 proves that the optimal solution to the problem must contain optimal solutions to some subproblems.

**Theorem 1** Let $1 \leq k \leq n$ and $r > 0$. Suppose $v_1, v_2, \ldots, v_r$ is an optimal solution to the $k$-optimization problem, and $u_1, u_2, \ldots, u_{r'}$ is an optimal solution to the $(v_r - 1)$-optimization problem, then $u_1, u_2, \ldots, u_{r'}, v_r$ is also an optimal solution to the $k$-optimization problem.

**Proof:** By definition,

$$\Delta cost(v_r - 1 : u_1, u_2, \ldots, u_{r'}) \quad \geq \quad \Delta cost(v_r - 1 : v_1, v_2, \ldots, v_{(r-1)}).$$

Therefore,

$$\begin{aligned}
& \Delta cost(k : u_1, u_2, \ldots, u_{r'}, v_r) \\
= \quad & (f_{u_1} - f_{u_2}) \cdot m_{u_1} - l_{u_1} + (f_{u_2} - f_{u_3}) \cdot m_{u_2} - l_{u_2} \\
& + \cdots + (f_{u_{r'}} - f_{v_r}) \cdot m_{u_{r'}} - l_{u_{r'}} + (f_{v_r} - f_{k+1}) \cdot m_{v_r} - l_{v_r} \\
= \quad & \Delta cost(v_r - 1 : u_1, u_2, \ldots, u_{r'}) + (f_{v_r} - f_{k+1}) \cdot m_{v_r} - l_{v_r} \\
\geq \quad & \Delta cost(v_r - 1 : v_1, v_2, \ldots, v_{(r-1)}) + (f_{v_r} - f_{k+1}) \cdot m_{v_r} - l_{v_r} \\
= \quad & (f_{v_1} - f_{v_2}) \cdot m_{v_1} - l_{v_1} + (f_{v_2} - f_{v_3}) \cdot m_{v_2} - l_{v_2} \\
& + \cdots + (f_{v_{(r-1)}} - f_{v_r}) \cdot m_{v_{(r-1)}} - l_{v_{(r-1)}} + (f_{v_r} - f_{k+1}) \cdot m_{v_r} - l_{v_r} \\
= \quad & \Delta cost(k : v_1, v_2, \ldots, v_{(r-1)}, v_r). \qquad (2)
\end{aligned}$$

On the other hand, since $v_1, v_2, \ldots, v_r$ is an optimal solution to the $k$-optimization problem,

$$\Delta cost(k : u_1, u_2, \ldots, u_{r'}, v_r) \quad \leq \quad \Delta cost(k : v_1, v_2, \ldots, v_{(r-1)}, v_r). \qquad (3)$$

Combining (2) and (3), we have

$$\Delta cost(k : u_1, u_2, \ldots, u_{r'}, v_r) \quad = \quad \Delta cost(k : v_1, v_2, \ldots, v_{(r-1)}, v_r).$$

Hence, the theorem is proved. □

We need the following definitions before presenting the recurrences for dynamic programming.

**Definition 2** Let $0 \leq k \leq n$. Define $OPT_k$ to be the maximum value of $\Delta cost(k : v_1, v_2, \ldots, v_r)$ obtained in the $k$-optimization problem, and $L_k$ is the maximum index in the optimal solution. If the optimal solution is an empty set, define $L_k = -1$. □

Obviously, $OPT_0 = 0$ and $L_0 = -1$. From Theorem 1, we know that if $L_k > 0$,

$$OPT_k = OPT_{(L_k - 1)} + (f_{L_k} - f_{k+1}) \cdot m_{L_k} - l_{L_k}.$$

Hence, we can check all possible locations of $L_k$ and select the one that maximizes the objective function when calculating $OPT_k$. Therefore, we have

$$\begin{cases}
OPT_0 \quad = \quad 0 \\
OPT_k \quad = \quad max\{0,\ OPT_{i-1} + (f_i - f_{k+1}) \cdot m_i - l_i \\
\qquad\qquad (i = 1, 2, \ldots, k)\}, \qquad \text{for } 1 \leq k \leq n,
\end{cases}$$

and

$$\begin{cases} L_0 = -1 \\ L_k = \begin{cases} v & \text{if index } v\ (1 \le v \le k) \text{ satisfies } OPT_k \\ & \quad = OPT_{v-1} + (f_v - f_{k+1}) \cdot m_v - l_v, \\ -1 & \text{if } OPT_k = 0. \end{cases} \end{cases}$$

The original object placement problem (i.e., the $n$-optimization problem) can be solved using dynamic programming with these recurrences. After computing $OPT_k$ and $L_k$ ($0 \le k \le n$), we can start from $v_r = L_n$ and obtain all the locations in the optimal solution by setting $v_i = L_{(v_{(i+1)}-1)}$ iteratively for all $1 \le i < r$. Theorem 1 ensures the correctness of this calculation procedure.

### 3.3 Coordinated Caching Scheme

In this subsection, we present our coordinated caching scheme based on the previous analysis.

Every cache maintains some information on the objects in the form of object descriptors. An object descriptor contains the object size, the access frequency and the miss penalty of the object with respect to the associated node. When a request is issued at node $A_n$ for object $R$, each node $A_i$ on the path between $A_0$ and $A_n$ piggybacks the corresponding information $f_i$, $m_i$ and $l_i$ on the request message as it passes through the node. When the request arrives at $A_0$, $A_0$ computes the optimal locations to place the requested object based on the piggybacked information using the dynamic programming algorithm and sends the decision together with the object back to the client node $A_n$. Along the way to $A_n$, the intermediate nodes on the routing path adjust their cache contents according to the caching decision. If the object is instructed to be cached at $A_i$, $A_i$ executes the greedy heuristic given in Section 3.1 to select replacement candidates and updates its cache accordingly.

Since the contents of the caches change over time, the access frequency and miss penalty of an object with respect to a node need to be updated from time to time. The access frequency can be estimated based on recent request history which is locally available (e.g., by using a "sliding window" technique [8]). The miss penalty is updated by the response messages. Specifically, a variable with an initial value of 0 is attached to each object (i.e., the response) sent to the client. At each intermediate node along the way, the variable is increased by the cost of the last link the object has just traversed. This value is then used to update the miss penalty of the object maintained by the associated cache. If the object is inserted into the cache, the node resets the variable to 0 before forwarding the object downstream. In this way, the updated miss penalties of the requested object are disseminated to all caches on the response path. To avoid unnecessary communication overhead, the miss penalty changes of the requested object at caches not along the response path are not updated immediately. The same is true for miss penalty changes caused by object removals that may result from the insertion of the requested object. These changes would be discovered by the related caches upon subsequent object requests/responses. Since no additional message exchange or probing operation is used for information

update, the communication overhead in deploying coordinated caching is small.

### 3.4 Discussion

The size of an object descriptor is typically a few tens of bytes and is negligible compared to the web object size. Hence, the memory overhead of maintaining the descriptors of cached objects is very small. On the other hand, the descriptors of objects not stored in the cache need to be maintained by the cache as well. Fortunately, it is not necessary to keep all such descriptors in the cache as discussed below. The following theorem presents an important property of the coordinated caching scheme.

**Theorem 2** The optimal locations $v_1, v_2, \ldots, v_r$ computed by the coordinated caching scheme satisfy the following inequalities:

$$f_{v_i} \cdot m_{v_i} \geq l_{v_i}, \quad i = 1, 2, \ldots, r.$$

**Proof:** (Prove by contradiction.) Suppose the coordinated scheme places a copy of the object at node $A_{v_i}$ $(1 \leq i \leq r)$ where $f_{v_i} \cdot m_{v_i} < l_{v_i}$.

If $i = 1$, we have

$$\begin{aligned}
& \Delta cost(n : v_1, v_2, \ldots, v_r) \\
= \; & (f_{v_1} - f_{v_2}) \cdot m_{v_1} - l_{v_1} + \Delta cost(n : v_2, v_3, \ldots, v_r) \\
\leq \; & f_{v_1} \cdot m_{v_1} - l_{v_1} + \Delta cost(n : v_2, v_3, \ldots, v_r) \\
< \; & \Delta cost(n : v_2, v_3, \ldots, v_r).
\end{aligned}$$

If $i > 1$, we have

$$\begin{aligned}
& \Delta cost(n : v_1, v_2, \ldots, v_r) \\
= \; & \Delta cost(v_{(i-1)} - 1 : v_1, v_2, \ldots, v_{(i-2)}) + (f_{v_{(i-1)}} - f_{v_i}) \cdot m_{v_{(i-1)}} - l_{v_{(i-1)}} \\
& + (f_{v_i} - f_{v_{(i+1)}}) \cdot m_{v_i} - l_{v_i} + \Delta cost(n : v_{(i+1)}, v_{(i+2)}, \ldots, v_r) \\
\leq \; & \Delta cost(v_{(i-1)} - 1 : v_1, v_2, \ldots, v_{(i-2)}) + (f_{v_{(i-1)}} - f_{v_i}) \cdot m_{v_{(i-1)}} - l_{v_{(i-1)}} \\
& + (f_{v_i} \cdot m_{v_i} - l_{v_i}) + \Delta cost(n : v_{(i+1)}, v_{(i+2)}, \ldots, v_r) \\
< \; & \Delta cost(v_{(i-1)} - 1 : v_1, v_2, \ldots, v_{(i-2)}) + (f_{v_{(i-1)}} - f_{v_i}) \cdot m_{v_{(i-1)}} - l_{v_{(i-1)}} \\
& + \Delta cost(n : v_{(i+1)}, v_{(i+2)}, \ldots, v_r) \\
\leq \; & \Delta cost(v_{(i-1)} - 1 : v_1, v_2, \ldots, v_{(i-2)}) + (f_{v_{(i-1)}} - f_{v_{(i+1)}}) \cdot m_{v_{(i-1)}} - l_{v_{(i-1)}} \\
& + \Delta cost(n : v_{(i+1)}, v_{(i+2)}, \ldots, v_r) \\
= \; & \Delta cost(n : v_1, v_2, \ldots, v_{(i-1)}, v_{(i+1)}, \ldots, v_r).
\end{aligned}$$

This implies that $A_{v_i}$ can be removed from the solution of the coordinated scheme to obtain a higher cost reduction, which contradicts with the optimality of the coordinated scheme. Hence, the theorem is proved. □

Theorem 2 implies that the coordinated scheme only needs to consider placing objects in the caches where the replacement operation is *locally* beneficial (i.e., the cost saving outweighs the cost loss with respect to the single cache). Since the miss penalty and the cost loss generally increase with the size of the requested object, this suggests that the descriptors of objects with low access frequency are less important for computing the optimal locations. Based on this observation, we propose to allocate a small auxiliary cache (we shall call it the *d-cache*) at each node to store the descriptors of the most frequently accessed objects not in the regular cache. The size of the d-cache is negligible compared to the regular cache which stores the objects. If the requested object is not instructed to be cached at $A_i$ and its descriptor is not in the d-cache, the descriptor is inserted into the d-cache at $A_i$ when the object passes through. Simple LFU replacement policy can be used to manage the object descriptors in the d-caches.

An intermediate node on the routing path that does not have the descriptor of the requested object in its d-cache will indicate this fact by attaching a special tag to the request message. Based on the attached information, $A_0$ removes nodes from the candidate set whose d-caches do not contain the object descriptor. The dynamic programming algorithm is applied to the remaining nodes to compute the optimal locations for caching. The rationale behind this strategy is that the requested object is not frequently accessed at the excluded nodes compared to the other objects, and hence removing them from the candidate set would not affect the optimal placement decision significantly.

It is easy to see that the time complexity of the dynamic programming algorithm is $O(k^2)$, where $k$ is the number of nodes on the path that contain the descriptor of the requested object in their d-caches ($k \leq n$). We expect that $k$ is not very large in practice. This is because $n$ is small for popular objects as they would be cached in the network with high density. Moreover, the descriptors of unpopular objects would not be cached by most intermediate nodes resulting in small $k$. Therefore, the cost of computing the optimal locations to cache the requested object is low.

The overhead in maintaining object descriptors can be kept small by using judicious data structures. For example, descriptors of cached objects can be organized as a heap based on their normalized cost losses. In this way, the time complexity for each adjustment (e.g., insertion and removal) is $O(log\, m)$, where $m$ is the number of cached objects. Object descriptors in the d-cache can be organized in one or more LRU stacks if the access frequencies of objects are estimated using a "sliding window" technique (see Section 4.1). As a result, the time complexity for each insertion and removal in the d-cache is $O(1)$.

# 4 Simulation Model

Extensive simulation experiments have been performed to compare the coordinated caching scheme with existing schemes. The network in our simulation consists of hundreds of routing nodes and content servers. It is difficult to get trace data for a network of this size. To the best of our knowledge, no such trace is available in the open literature. Therefore, our simulation employed synthetic workload generated from empirical models reported in recent publications [34, 35, 4, 30]. We believe that the results are valid for the purpose of comparing the *relative performance* of different en-route caching schemes. This section outlines the system settings and the caching schemes used in our simulation.

## 4.1 System Settings

Table 1 shows the system parameters and their default values used in the experiments.

| Parameter | Default value |
|---|---|
| Total number of nodes | 200 |
| Ratio of WAN nodes to MAN nodes | 1 : 1 |
| Number of network links | 344 |
| Average delay of WAN links | 0.46 second |
| Average delay of MAN links | 0.07 second |
| Number of servers | 100 |
| Number of objects | 1000 objects per server |
| Object size distribution | Body: Lognormal distribution $p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x - \mu)^2/2\sigma^2}$ $(\mu = 9.357, \sigma = 1.318)$. Tail: Pareto distribution $p(x) = \alpha k^\alpha x^{-(\alpha+1)}$ $(k = 8596, \alpha = 1.1)$. Average size: 26 KB |
| Relative cache size per node | 4% |
| D-cache size per node | 8000 object descriptors |
| Average request rate at each MAN node | $U(1, 9)$ requests per second |
| Access frequency distribution | Server: Zipf, $1/i^\alpha,\ \alpha = 0.8$ Object: Zipf, $1/j^\beta,\ \beta = 0.8$ |

Table 1: System Parameters

The network topology is randomly generated using the Tiers program [34]. It consists of a wide area

network (WAN) and a number of metropolitan area networks (MAN[2]), in Tiers terminology. We have performed experiments for a wide range of topologies consisting of different numbers of WAN nodes, MAN nodes and network links. The relative performance of the caching schemes was found to be insensitive to the topology. Due to space limitation, only the results of one topology are reported in this paper. The characteristics of this topology and the workload model are listed in Table 1. These values are chosen based on those reported in the open literature and what are considered reasonable. The WAN is regarded as the backbone network, and no content servers or clients are directly attached to the WAN nodes. Each MAN node is assumed to connect to a content server. Every server stores a set of objects, and the object sets associated with different servers are disjoint. The distribution of object sizes is assumed to follow a hybrid model [35] (the probability density functions and the mean value are given in Table 1). An en-route cache is associated with every WAN and MAN node. Similar to other studies [4, 6, 8, 36], the cache size at each node is described relative to the total size of all objects available in the network (we shall call it the *relative cache size*). The d-cache size is measured in terms of the number of object descriptors. By default, the d-cache size is set to twice the average number of objects the regular cache can hold[3]. We have conducted experiments for different d-cache sizes relative to the cache size and found that the results were similar when the cache and d-cache were capable of accommodating the same order of objects and object descriptors respectively. To simulate the requests made by the clients, a continuous request stream is randomly generated at each MAN node. The average request rate of each node is assigned from the distribution $U(1, 9)$ requests per second, where $U(x, y)$ is a uniformly distributed number between $x$ and $y$. The access frequencies of the content servers as well as the objects maintained by a given server both follow a Zipf-like distribution [4, 30]. Specifi cally, the probability of a request for object $O$ in server $s$ is proportional to $1/(i^{\alpha} \cdot j^{\beta})$, where $s$ is the $i$th most popular server and $O$ is the $j$th most popular object in $s$.

Routing paths from all nodes to a given server are set to the shortest-path tree rooted at the server except in Section 5.3 where the caching performance is examined under asymmetric routing. Each network link connects two nodes and represents one hop in the network. The average length of all routing paths in our simulation is about 11 hops, similar to the statistics given in [32]. For simplicity, the delay caused by sending a request and the associated response over a network link is set proportionally to the size of the requested object. This delay includes the propagation delay, the transmission delay, and the look-up delay in the en-route cache. The delays generated by the Tiers program for the network links are taken to be the delays of an average size object. The ratio of the average delays of WAN links and MAN links for the default topology is approximately 7:1 (see Table 1). The cost function is taken to be the delay of the link. This means the generic cost in the analytical model (see Section 3) is interpreted as the access latency in our simulation.

To make the caching schemes less sensitive to transient workload, a "sliding window" technique is em-

[2]The term "MAN" is used in Tiers [34]. Essentially it applies to any high-speed interconnected local area networks.

[3]Note that this ratio is not equal to the ratio of their physical storage capacities. The capacity of the d-cache is much smaller than that of the regular cache.

ployed to dynamically estimate the access frequency of an object [8]. Specifically, for each object, up to $K$ most recent reference times are recorded and the access frequency is computed by $f(O) = \frac{\mathcal{K}}{t - t_{\mathcal{K}}}$, where $\mathcal{K} \leq K$ is the number of references recorded, $t$ is the current time and $t_{\mathcal{K}}$ is the $\mathcal{K}$th most recently referenced time. $K$ is set to 3 in our simulation experiments [8]. To reduce the overhead, the access frequency estimate of an object is only updated when the object is referenced, and at reasonably large intervals (say several minutes[4]) to reflect aging.

### 4.2 Caching Schemes

In addition to the coordinated caching scheme, the following algorithms were also included in our simulation experiments for comparison purposes:

- LRU: This is a standard caching algorithm. The requested object is cached by every node through which the object passes. If there is not enough free space, the cache purges one or more least recently referenced objects to make room for the new object. No d-cache is needed in this scheme.

- MODULO [15]: This is a modified LRU scheme that employs a simple placement optimization. On the path from the cache (or server) to the client, the object is cached at the nodes that are a fixed number (called cache radius) of hops apart. The caches use the LRU policy to remove objects when necessary. The cache radius is set to 3 in our experiments[5] [15]. Similar to LRU, no d-cache is needed in this scheme.

- LNC-R [29]: This is a cost-based caching algorithm shown to be effective in the context of a single proxy server. It optimizes cache replacement by using the function $\frac{f(O) \cdot m(O)}{s(O)}$ (i.e., the normalized cost loss) to select replacement candidates. Similar to LRU, the requested object is cached by all nodes along the routing path, and for each node, the miss penalty of the object is set to the delay of the immediate upstream link. To obtain more accurate access frequency estimates, the descriptors of the most frequently accessed objects not in the regular cache are maintained in the d-cache as in the coordinated scheme.

## 5 Performance Results

### 5.1 Impact of Cache Size

First, we compare the effectiveness of different caching schemes across a wide range of cache sizes, from 0.04% to 12.0%. The relative cache size of 12% is very large in the context of en-routing caching due to the

[4] Our experimental results show that caching performance is practically the same for aging intervals of 1 and 10 minutes.

[5] We have conducted experiments with different radii and found that a cache radius of 3 gives the best overall performance under our experimental settings.

large network under consideration (e.g., that of a regional ISP).

Figure 3 shows the average access latency as a function of the relative cache size at each node. Since the objects have very different sizes in our workload, we also plotted in Figure 4 the average response ratio. The response ratio of a request is defined as the ratio of its access latency to the size of the target object. The lower the average response ratio, the better the performance. This metric is more objective as the effect of object size is eliminated. Moreover, users are likely to expect short delays for small objects and willing to tolerate longer delays for larger objects.

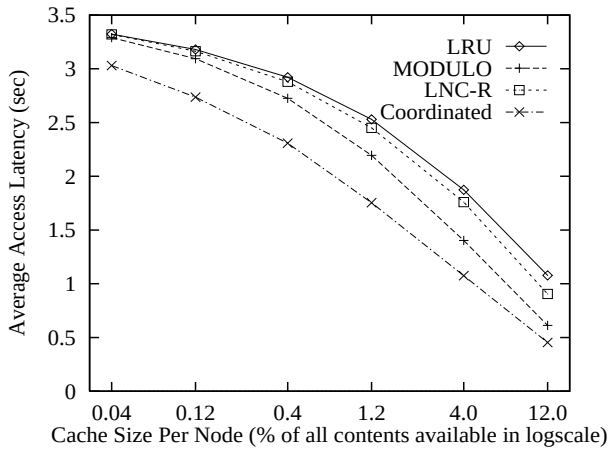


Figure 3: Average Access Latency vs. Cache Size

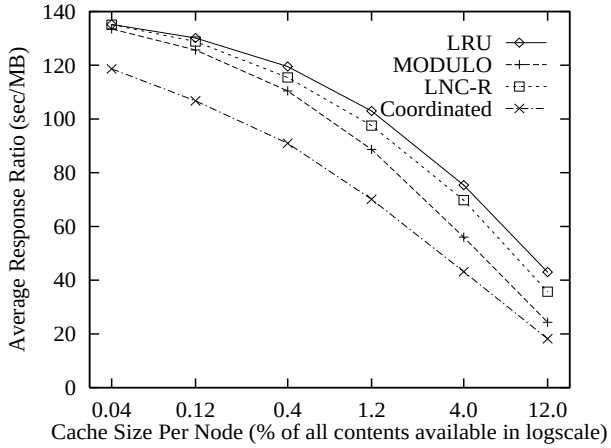


Figure 4: Average Response Ratio vs. Cache Size

As can be seen, all caching schemes provide steady performance improvement as the cache size increases. The coordinated scheme significantly reduces the average access latency and response ratio compared to the other schemes examined. This shows the importance of managing object placement and replacement strategies in an integrated fashion. To achieve the same access latency, the schemes that do not optimize placement decisions (LRU and LNC-R) would require 3 to 10 times the cache space of coordinated caching (note that the

cache size axis is in logscale). Although LNC-R incorporates various cost factors to optimize cache replacement decisions, its performance is similar to that of LRU. This is not surprising because cache contents change very frequently due to frequent replacements if all requested objects are stored by all intermediate nodes on the routing path. Therefore, the miss penalties of cached objects change very frequently for LNC-R in the en-route caching environment and are less helpful in making replacement decisions. The relative improvement of coordinated caching over LRU and LNC-R increases with cache size. Coordinated caching outperforms LRU and LNC-R by more than 50% in average access latency at cache size 12%. On the other hand, even though MODULO utilizes cache space more efficiently than LRU and LNC-R by storing the same object at fewer locations, Figures 3 and 4 show that it still performs much worse than coordinated caching over a wide range of cache sizes. This is because the placement decision of MODULO is not based on object access frequencies so that popular and unpopular objects are cached with the same density in the network. The impact of MODULO's disadvantage is less significant as the cache size increases. Therefore, the absolute difference between coordinated caching and MODULO decreases with increasing cache size. However, the relative improvement of coordinated caching remains significant for large cache sizes. For example, at cache size 12%, the average access latencies for coordinated caching and MODULO are 0.45 and 0.61 second respectively (about 25% improvement).

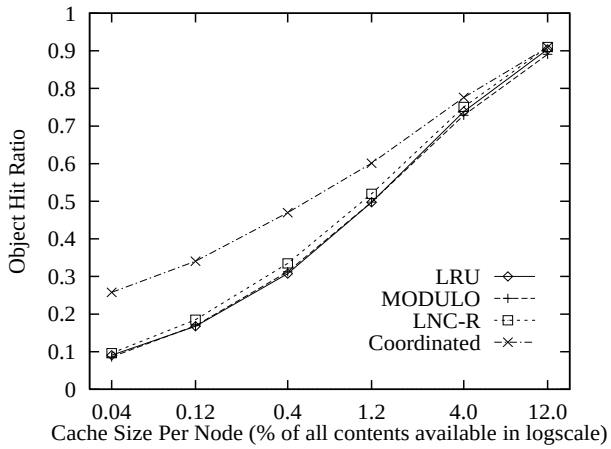


Figure 5: Object Hit Ratio vs. Cache Size

Figure 5 plots the object hit ratio[6] curves as a function of cache size for different caching schemes. To study the system's caching behavior, the object hit ratio is defined as the ratio of the number of requests served by the caches as a whole (as opposed to those served by the content servers) to the total number of requests. By making optimal caching decisions along the routing paths, the coordinated scheme greatly improves the object hit ratio over the other schemes examined especially for smaller cache sizes. This implies substantial load

[6]Similar performance trends have been observed for object hit ratio and byte hit ratio. Due to space limitation, only the results of object hit ratio are shown in this paper.

reduction at the content servers. Figure 6 shows the workload of the *most heavily loaded* content server in the network, where workload is measured in the number of bytes served per second. It can be seen that the highest server load under the coordinated scheme is considerably lower than the other schemes. In contrast, MODULO and LNC-R, which optimize object placement or replacement only, have similar performance to LRU. On the other hand, when the cache size is very large, all schemes are capable of caching popular objects somewhere in the network. Therefore, as shown in Figures 5 and 6, the overall system hit ratios of all the schemes converge. In this case, the major benefi t of coordinated caching is to reduce access latency and response ratio by judicious placement of objects in the caches (see Figures 3 and 4).

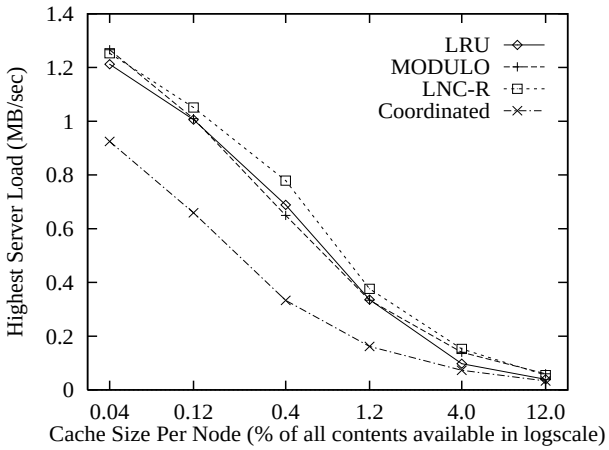


Figure 6: Highest Server Load vs. Cache Size

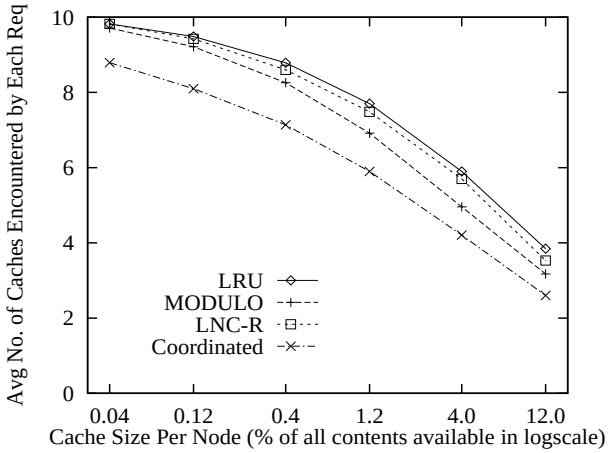


Figure 7: Avg No. of Caches Encountered by Each Req vs. Cache Size

The workload of the caches consists of two parts: (i) looking up the requested objects and forwarding requests in case of cache miss; and (ii) reading/writing objects from/into the cache. It is obvious that the lower the cache load, the more scalable the caching scheme. Everything else being equal, the overhead in look-up and forwarding requests is proportional to the number of caches a request goes through before obtaining the

target object. Figure 7 shows that on average, a request in the coordinated scheme passes through fewer caches than in the other schemes. This implies the coordinated caching scheme stores frequently accessed objects closer to the clients and therefore introduces lower look-up and forwarding load on the caches. Reading and writing objects occur in the following situations. A request causes a read operation on an en-route cache if the request results in a cache hit, and a decision to place an object copy introduces a write operation on the en-route cache. The read load is necessary to serve the requests while the write load represents an overhead for caching. To compare the overall read/write load, we calculated the mean aggregated read and write load (measured in bytes) introduced by each request on all the caches. As shown in Figure 8, coordinated caching has the lowest read/write load among all the schemes studied. By contrast, LRU and LNC-R introduce 3 to 35 times the read/write load of coordinated caching because they do not consider object placement optimization. MODULO has lower read/write load compared to LRU and LNC-R, but its load is still much higher than that of the coordinated scheme. Our data shows the read load takes up 75% to 90% of the overall read/write load in coordinated caching. Since the read load of coordinated caching is higher than the other schemes due to its higher cache hit ratio (see Figure 5), the results presented in Figure 8 suggest that the coordinated scheme involves substantially lower write load (i.e., overhead) on the caches. Thus, coordinated caching has better scalability than the other schemes.

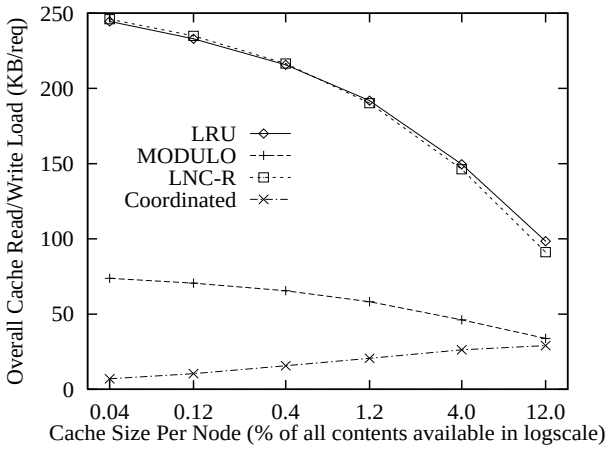


Figure 8: Overall Cache Read/Write Load vs. Cache Size

Finally, we examine the load on the network links. Figures 9 shows the average network traffic (measured in byte·hops) required to satisfy a request, and Figure 10 shows the load (measured in bytes/second) of the most congested link in the network. From these figures, it can be clearly seen that the coordinated caching scheme results in much lower load on the network links than the other schemes. This further demonstrates the effectiveness of coordinated caching in bringing popular objects closer to the clients.

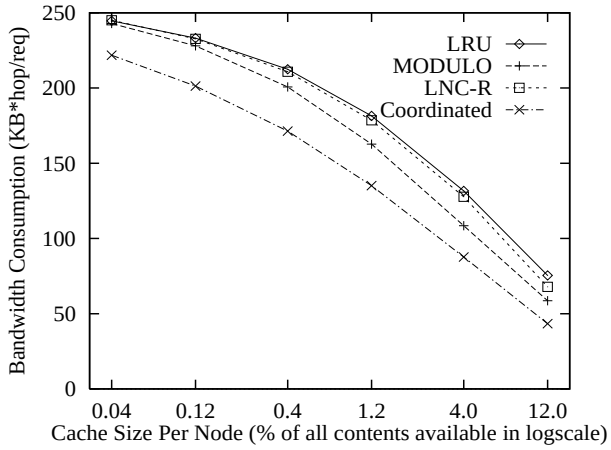


Figure 9: Bandwidth Consumption vs. Cache Size

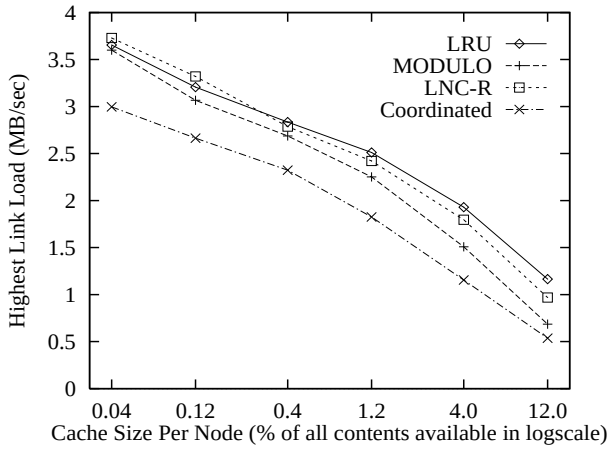


Figure 10: Highest Link Load vs. Cache Size

## 5.2 Impact of Access Frequency Distribution

In this set of experiments, we examine the impact of access frequency distribution on the effectiveness of caching. For simplicity, the Zipf parameters for servers and objects (i.e., $\alpha$ and $\beta$) are assigned equal values. Figures 11, 12 and 13 show the average access latency, the average response ratio and the object hit ratio respectively for values of $\alpha$ and $\beta$ between 0.2 and 1.2.

The coordinated caching scheme consistently provides the best performance over a wide range of access frequency distributions. Under relatively homogeneous reference patterns (i.e., small Zipf parameters, see the left part of the figures), cached objects account for relatively low proportion of the requests since all objects have similar access frequencies. Therefore, the relative performance difference among the caching schemes is not very large. With increasing skewness in the access pattern, the relative improvement of coordinated caching in terms of access latency and response ratio is more substantial. This is because the coordinated scheme caches frequently accessed objects closer to the clients as discussed in Section 5.1. For example, the

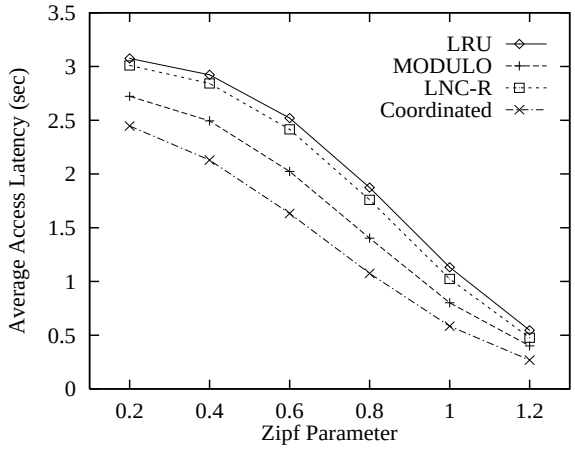


Figure 11: Average Access Latency vs. Zipf Parameter

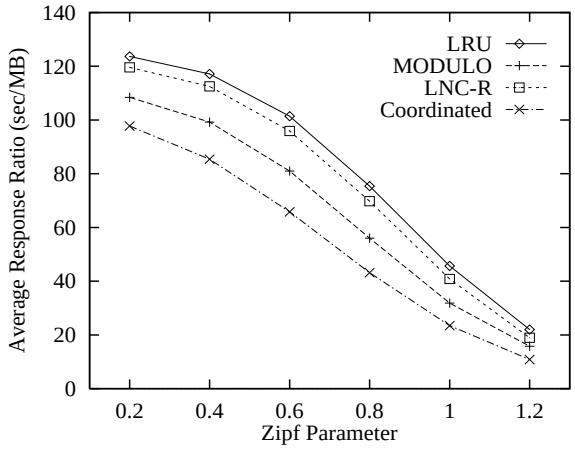


Figure 12: Average Response Ratio vs. Zipf Parameter

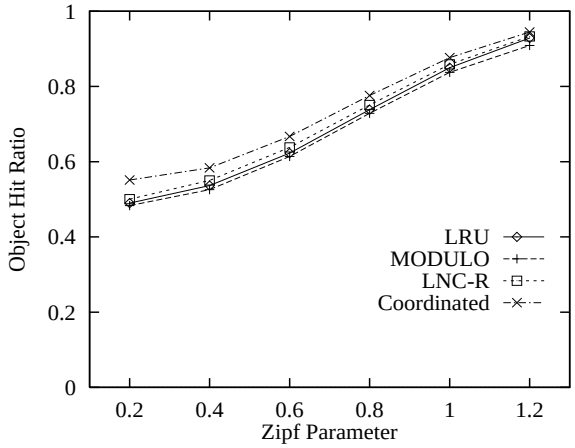


Figure 13: Object Hit Ratio vs. Zipf Parameter

coordinated scheme reduces average response time by 20%, 43% and 51% compared to LRU, and by 10%, 23% and 33% compared to MODULO for Zipf parameters of 0.2, 0.8 and 1.2 respectively. Figure 13 shows that object hit ratio increases with access skewness for all caching schemes. Since the default cache size used here (4%) is fairly large, coordinated caching does not provide significant improvement in object hit ratio as discussed in Section 5.1.

### 5.3 Impact of Routing Asymmetry

We have assumed that the routing paths are symmetric in the analytical model, i.e., the route from node $A$ to node $B$ is the same as the route from $B$ to $A$. Although this is true in many cases[7], there could be some upstream and downstream route pairs in the Internet that do not contain the same set of intermediate nodes [32]. In this section, the coordinated caching scheme is modified to handle routing asymmetry and compared with the existing algorithms.

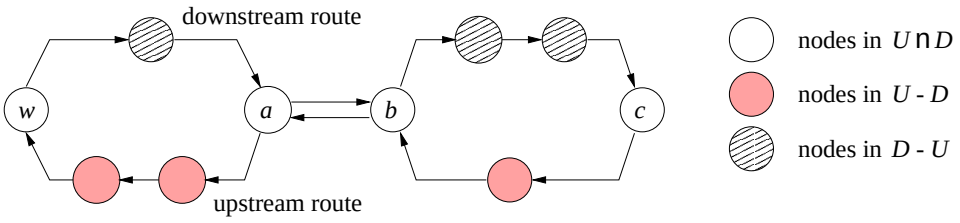


Figure 14: Asymmetric Route Pair

Consider the asymmetric routes of a request and the associated response shown in Figure 14, where $w$ is the node satisfying the request and $c$ is the client node issuing the request. Let $U$ and $D$ be the sets of nodes on the upstream and downstream routes respectively. The union of $U$ and $D$ can be divided into three disjoint subsets: $U \cup D = (D - U) \cup (U - D) \cup (U \cap D)$. The benefits of caching the requested object at the nodes in each subset are discussed below:

- $D - U$: Caching the object at the nodes in $D - U$ is not useful, because subsequent requests from $c$ will not go through these nodes. Therefore, they should not be considered in the computation of the optimal locations to cache the requested object. The revised coordinated caching scheme simply collects status information of the caches along the upstream route (i.e., the request path) and computes the optimal caching locations among them. The nodes in $D - U$ do not update their caches and d-caches when the requested object (i.e., the response) passes through them.

- $U - D$: Placing the object at the nodes in $U - D$ requires additional object transfer since the object does not pass through these nodes when it is being sent to the client node. To avoid extra network traffic

[7] Some transparent web caching approaches guarantee the request and the response paths pass through the same set of caches (e.g., see [18]).

and control overhead, the object is not placed at the nodes in $U - D$ in our scheme. Note that under asymmetric routing, it is possible that some computed optimal caching locations are in $U - D$. In this case, we simply discard these locations.

- $U \cap D$: The nodes in $U \cap D$ are most appropriate to cache the requested object (e.g., the nodes $a$ and $b$ in Figure 14). Each node $v \in (U \cap D)$ takes the same actions as in the case of symmetric routing. Specifically, if $v$ is included in the computed optimal locations, the object is inserted into its cache; otherwise, $v$ updates the miss penalty of the object in its d-cache.

To allow fair comparisons, the three caching schemes described in Section 4.2 were also modified accordingly in this set of experiments. Specifically, LRU and LNC-R place the requested object at every node in $U \cap D$, and MODULO places the requested object at every node in $U \cap D'$, where $D'$ is a subset of $D$ containing nodes that are 3 hops apart on the downstream route.

To study the impact of routing asymmetry on caching performance, we artificially generated some asymmetric route pairs in our experiments. For each node $A$, two different routing trees, an upstream tree and a downstream tree, are built. The messages sent to $A$ use the route in the upstream tree while the messages sent from $A$ make use of the route in the downstream tree. The routing asymmetry degree is characterized by the triple $(x, y, z)$, meaning that $x\%$ of all route pairs are asymmetric, $y\%$ of the asymmetric route pairs differ in at least 2 hops, and the average difference of all asymmetric route pairs is $z$ hops. Obviously, the larger the values of $x$, $y$ and $z$, the higher the degree of routing asymmetry. The caching schemes were evaluated under a variety of asymmetry degrees (listed in increasing order)[8]: (26/27/1.6), (47/43/2.0), (72/68/3.3) and (84/80/4.8). We note that the asymmetry level of (47/43/2.0) is closest to the statistics reported in the literature [32]. Figures 15, 16 and 17 show the average access latency, the average response ratio and the object hit ratio respectively for asymmetric routing with different degrees.

From Figures 15 – 17, we observe that the relative performance of the caching schemes under asymmetric routing is quite similar to that for the symmetric routing case (i.e., the case of (0/0/0.0)). The coordinated caching scheme still consistently outperforms all the other schemes studied significantly. This further verifies the importance of coordinating object placement and replacement strategies. It is interesting to note that the performance of LRU and LNC-R relative to MODULO and the coordinated scheme improves slightly (about 3%) under high level of routing asymmetry. The reason is because LRU and LNC-R will place fewer copies of the same object in the network when the upstream and downstream routes have less nodes in common. This somewhat compensates for their disadvantage of caching unpopular objects (which make up the majority of the objects due to the Zipf-like request distribution) with high density in the network.

[8]Note that the average length of all routes is about 11 hops.

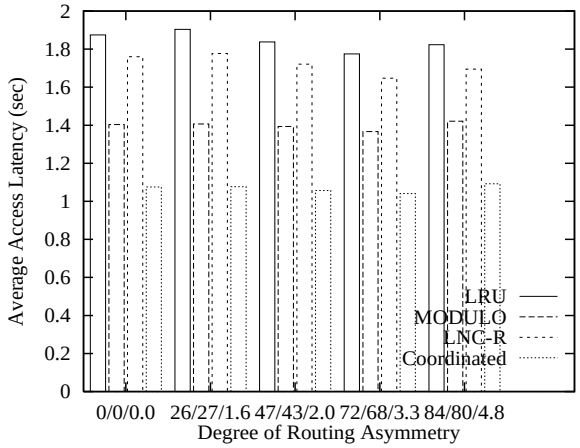


Figure 15: Average Access Latency under Asymmetric Routing

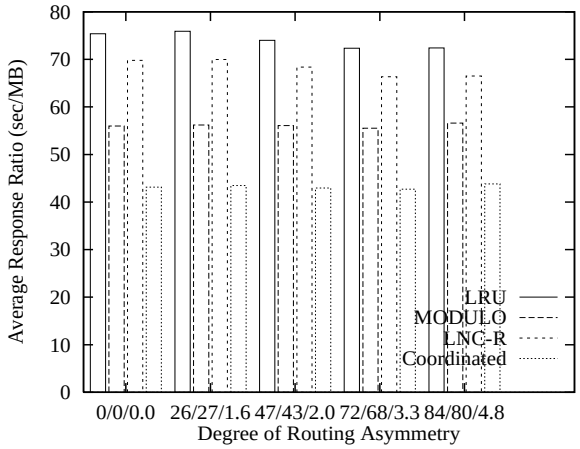


Figure 16: Average Response Ratio under Asymmetric Routing

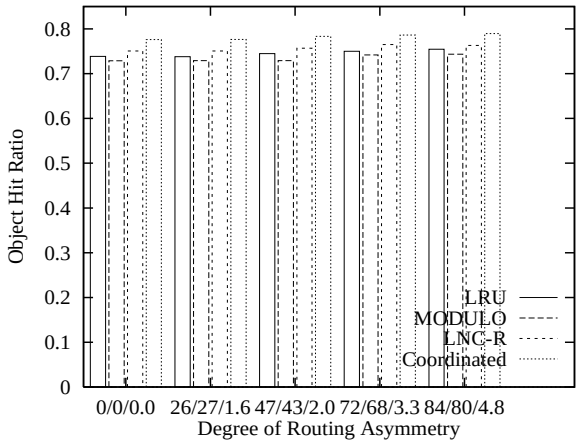


Figure 17: Object Hit Ratio under Asymmetric Routing

# 6 Conclusion

Object placement and replacement are two important issues in en-route web cache management. In this paper, we have presented a coordinated caching scheme where object placement and replacement policies are managed in an integrated fashion. In this scheme, status information of the caches along the routing path of a request is used in dynamically determining where to cache the requested object and what to replace if necessary. The optimal placement decision is obtained using a low-cost dynamic programming algorithm. We have performed extensive simulation experiments to compare the proposed coordinated scheme with a number of existing caching algorithms. The results show that the coordinated caching scheme effectively reduces access latency, server load and network traffi c in both symmetric and asymmetric routing situations, and is more scalable in terms of cache read/write load. The proposed scheme considerably outperforms existing algorithms which consider either placement or replacement at individual caches only.

# References


[1] Steven Glassman. A caching relay for the world wide web. *Computer Networks and ISDN Systems*, 27(2):165–173, November 1994.

[2] Jia Wang. A survey of web caching schemes for the Internet. *ACM SIGCOMM Computer Communication Review*, 29(5):36–46, October 1999.

[3] Brian D. Davison. A web caching primer. *IEEE Internet Computing*, 5(4):38–45, July/August 2001.

[4] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM'99*, pages 126–134, March 1999.

[5] Roland P. Wooster and Marc Abrams. Proxy caching that estimates page load delays. *Computer Networks and ISDN Systems*, 29(8–13):977–986, September 1997.

[6] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1st USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 193–206, December 1997.

[7] Charu Aggarwal, Joel L. Wolf, and Philip S. Yu. Caching on the world wide web. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):94–107, January/February 1999.

[8] Junho Shim, Peter Scheuermann, and Radek Vingralek. Proxy cache algorithms: Design, implementation, and performance. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):549–562, July/August 1999.

[9] Jean-Marc Menaud, Valerie Issarny, and Michel Banatre. Improving the effectiveness of web caching. In *Advances in Distributed Systems, Advanced Distributed Computing: From Algorithms to Systems, Springer-Verlag LNCS-1752*, pages 375–401, 2000.

[10] Shudong Jin and Azer Bestavros. Greedydual* web caching algorithm: Exploiting the two sources of temporal locality in web request streams. *Computer Communications*, 24(2):174–183, February 2001.

[11] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical Internet object cache. In *Proceedings of the 1996 USENIX Annual Technical Conference*, pages 153–163, January 1996.

[12] Renu Tewari, Michael Dahlin, Harrick M. Vin, and Jonathan S. Kay. Design considerations for distributed caching on the internet. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 273–284, June 1999.

[13] Pablo Rodriguez, Christian Spanner, and Ernst W. Biersack. Analysis of web caching architectures: Hierarchical and distributed caching. *IEEE/ACM Transactions on Networking*, 9(4):404–418, August 2001.

[14] Proxy Cache Comparison. http://www.web-caching.com/proxy-comparison.html.

[15] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura. Self-organizing wide-area network caches. In *Proceedings of IEEE INFOCOM'98*, pages 600–608, March 1998.

[16] Pablo Rodriguez and Sandeep Sibal. SPREAD: scalable platform for reliable and efficient automated distribution. *Computer Networks*, 33(1–6):33–49, June 2000.

[17] P. Krishnan, Danny Raz, and Yuval Shavitt. The cache location problem. *IEEE/ACM Transactions on Networking*, 8(5):568–582, October 2000.

[18] Pablo Rodriguez, Sandeep Sibal, and Oliver Spatscheck. TPOT: Translucent proxying of TCP. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop (WCW)*, May 2000.

[19] Michael Rabinovich and Hua Wang. DHTTP: An efficient and cache-friendly transfer protocol for web traffic. In *Proceedings of IEEE INFOCOM'01*, pages 1597–1606, April 2001.

[20] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[21] Duane Wessels and Kim Claffy. ICP and the Squid web cache. *IEEE Journal on Selected Areas in Communications*, 16(3):345–357, April 1998.

[22] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, June 2000.

[23] Michael Rabinovich, Jeff Chase, and Syam Gadde. Not all hits are created equal: Cooperative proxy caching over a wide area network. *Computer Networks and ISDN Systems*, 30(22–23):2253–2259, November 1998.

[24] Bo Li, Mordecai J. Golin, Giuseppe F. Italiano, Xin Deng, and Kazem Sohraby. On the optimal placement of web proxies in the Internet. In *Proceedings of IEEE INFOCOM'99*, pages 1282–1290, March 1999.

[25] Keith W. Ross. Hash routing for collections of shared web caches. *IEEE Network*, 11(6):37–44, November/December 1997.

[26] Kun-Lung Wu and Philip S. Yu. Load balancing and hot spot relief for hash routing among a collection of proxy caches. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 536–543, June 1999.

[27] Philip S. Yu and Edward A. MacNair. Performance study of a collaborative method for hierarchical caching in proxy servers. *Computer Networks and ISDN Systems*, 30(1–7):215–224, April 1998.

[28] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox. Removal policies in network caches for world wide web documents. In *Proceedings of ACM SIGCOMM'96*, pages 293–305, August 1996.

[29] Peter Scheuermann, Junho Shim, and Radek Vingralek. A case for delay-conscious caching of web documents. *Computer Networks and ISDN Systems*, 29(8–13):997–1005, September 1997.

[30] V. N. Padmanabhan and Lili Qiu. The content and access dynamics of a busy web site: Findings and implications. In *Proceedings of ACM SIGCOMM'00*, pages 111–123, August 2000.

[31] Balachander Krishnamurthy and Craig E. Wills. Piggyback server invalidation for proxy cache coherency. *Computer Networks and ISDN Systems*, 30(1–7):185–193, April 1998.

[32] Vern Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, October 1997.

[33] Bo Li, Xin Deng, Mordecai J. Golin, and Kazem Sohraby. On the optimal placement of web proxies in the Internet: The linear topology. In *Proceedings of the 8th IFIP TC-6 International Conference on High Performance Networking (HPN)*, pages 485–495, September 1998.

[34] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[35] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS'98*, pages 151–160, July 1998.

[36] Shudong Jin and Azer Bestavros. Popularity-aware GreedyDual-Size web proxy caching algorithms. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 254–261, April 2000.

## Biography

**Xueyan Tang** received his BEng degree in computer science and engineering from Shanghai Jiao Tong University, Shanghai, China in 1998. He is currently a PhD candidate in the Department of Computer Science at the Hong Kong University of Science and Technology. His research interests include web caching and performance, Internet technologies, mobile computing, and distributed systems.

**Samuel T. Chanson** received his Ph.D. degree in Electrical Engineering and Computer Sciences from the University of California, Berkeley in 1975. He was a faculty member at the School of Electrical Engineering, Purdue University for two years before joining the Department of Computer Science at the University of British Columbia where he became a full professor and director of its Distributed Systems Research Group. He joined the Hong Kong University of Science and Technology in 1993. Professor Chanson has been widely consulted by industry and government institutes in North America and Asia in Internet and communication technologies, and has served on the program committees of numerous international conferences on distributed systems and computer communications. He was the Conference Co-Chair of IEEE ICDCS in 1998 and IFIP FORTE/PSTV in 1999. He currently serves on the editorial boards of three international journals, including IEEE/ACM Transactions on Networking. Professor Chanson's research interests include web technologies, multimedia communication, Internet security and distributed systems. He has published more than 150 papers in the above areas.