On Optimal Replication of Data Object at Hierarchical and Transparent Web Proxies

Xiaohua Jia, Deying Li, Hongwei Du, and Jinli Cao

Abstract—This paper investigates the optimal replication of data objects at hierarchical and transparent web proxies. By transparent, we mean the proxies are capable of intercepting users' requests and forwarding the requests to a higher level proxy if the requested data are not present in their local cache. Two cases of data replication at proxies are studied: 1) proxies having unlimited storage capacities and 2) proxies having limited storage capacities. For the former case, an efficient algorithm for computing the optimal result is proposed. For the latter case, we prove the problem is NP-hard, and propose two heuristic algorithms. Extensive simulations have been conducted and the simulation results have demonstrated significant performance gain by using the proposed data replication algorithms and also shown the proposed algorithms out-perform the standard web caching algorithm (LRU threshold method).

Index Terms—Web server, Web proxy, Web caching, data replication, World Wide Web.

1 Introduction

Web-server proxy (Web proxy for short) is an important technology to improve response time of Web servers and alleviate Web data access traffic on the Internet. A Web proxy stores replicated data of an origin Web server and acts as a "front end" of this server to its clients. Basically, there are three major kinds of Web proxies, namely client-side proxies, surrogates and CDN (Content Distribution Network) proxies. Client-side proxies are usually operated by large corporations or ISPs (Internet Service Providers), and located close to end-users to reduce latency for data retrievals. Surrogates (also called Web accelerators) are installed in front of an origin Web server to enhance the performance of the Web server through resource sharing and load balancing. CDN proxies, operating in a CDN setting, are usually installed close to end-users like the first kind, but are operated by a CDN provider. The CDN provider has a contract with origin servers (called content owners) at one side, and at the other side has a contract with a large number of ISPs to host the contents for the content owners. The proxies discussed in this paper are client-side proxies. We assume the proxies are organized hierarchically [5], [25]. The levels of this hierarchy can be client proxies, institutional level proxies, regional level proxies and national level proxies. We also assume the proxies are transparent to clients, which are capable of intercepting users' requests and forwarding the requests to a higher level proxy if the requested data are not present in their local cache. A typical example of transparent proxies are *transparent en-route proxies* [6], [19], [21], [26], where proxies are colocated with routers and maintained by network providers. Most of the existing caching products include such a transparent operation mode [8].

In this paper, we address the problem of data replication at proxies. Data replication is a different technology from the typical Web caching techniques (extensive research has been done on Web caching and consistency maintenance, such as in [3], [9], [27]). Data replication *proactively* places a copy of data and anticipates many clients to make use of the copy, while a caching technique on-demand caches accessed data at a proxy. A cached data object will be purged out if there is not enough hit and not enough space to cache newly accessed data, but a replicated object will stay in the proxy for a relatively longer period of time. Replication and caching can be used to complement each other in improving the performance of a Web server.

A primary task of data replication is to decide what data objects should be replicated at proxies, given the users access patterns. Since a transparent proxy usually serves many origin servers, the storage space that can be allocated to a particular server is very much limited compared with the large number of data objects that are hosted at the server. It is very important to study data replication at Web proxies with limited space. In this paper, we first introduce the data replication problem at proxies and then formulate the optimal solution for the case where proxies have infinite storage space. Based on this optimal solution, two efficient heuristic algorithms are proposed for the case where proxies have limited storage space. Extensive simulations have been conducted to evaluate the performance of the proposed algorithms.

Notice that the data replication at proxies is different from traditional data replication in distributed database systems, due to its hierarchical structure. In a Web-proxy system discussed in this paper, a cache-missing request will be forwarded to a higher level proxy (or the origin server),

- *X. Jia and H. Du are with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. E-mail: {jia, hongwei}@cs.cityu.edu.hk.*
- *D. Li is with the School of Information, Renmin University, Beijing, China. E-mail: dyli@cs.cityu.edu.hk.*
- *J. Cao is with the Department of Computer Science, La Trobe University, Bundoora, Melbourne, VIC 3086, Australia. E-mail: jinli@cs.latrobe.edu.au.*

Manuscript received 2 June 2003; revised 17 Jan. 2004; accepted 22 Oct. 2004; published online 22 June 2005.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0086-0603.

while in distributed database systems data replicas are usually in flat structure.

2 Related Work

The performance of Web proxies heavily depends on two important factors: where in the network to place the proxies and what data objects to be replicated at a proxy.

The placement of Web proxies or mirrored Web servers has been an active research topic in recent years. In [21], Li et al. presented the optimal placement of Web proxies in a tree by using dynamic programming method. Jia et al. extended this work to an environment where Web servers are mirrored in [14] and both read and write operations are considered in [15]. In [19], a k-cache location problem and a k-TERC (transparent en-route cache) problem were formulated. The optimal solutions for some regular topologies and some greedy heuristic algorithms for general networks were proposed. The problem of placing mirrored Web servers to minimize the overall access distance was formulated as the p-median problem in [24], which is NP-complete [18], and several heuristic algorithms were proposed. Cronin et al. discussed the same problem in [7], but with the objective of minimizing the maximum distance between any client to a mirrored server. The problem was formalized as the k-center problem (which is again NP-complete), and a greedy algorithm was proposed. In the existing work on the Web proxy (or mirrored server) placement, it is assumed the whole Web server is replicated at a proxy, and the data replication at individual object level is not considered.

As Web proxies are widely used to improve the quality of Web data retrievals, some research has been done on the replication of data objects at proxies. The problem of data replication in database systems was extensively studied in [4], [10], [13], and it has been proven that optimal replication of data objects in general networks is NP-complete. The replication problem in the context of Web proxies is different from that in general database systems because of the special characteristics of the client accesses to Web proxies. A model targeting for Web data replication was presented in [16], where read, write, and storage costs are considered. A dynamic programming algorithm was proposed to find the optimal replication of data objects in a tree network. Pierre et al. presented in [23] an integrated method that uses different data replication strategies, such as delayed verification, server verification, server updates, etc., for different data objects according to their access pattern. In [26], Tang and Chanson studied data replication at en-route caches (proxy nodes), and a novel algorithm was proposed by using dynamic programming for the optimal replication of data objects at a subset of proxy nodes along the path from clients to the server. A more general model for data replication in CDN networks was proposed in [17]. The problem was formulated as a combinatorial optimization problem and was proved to be NP-hard. Four heuristics, namely, Random, Popularity, Greedy-Single, and Greedy-Global, were discussed and simulations have been conducted to show the performance of them.

3 Problem Formulation

The network is modeled by a connected graph $G(V,E)$, where V represents a set of network nodes and E represents a set of links. For a link $(u,v) \in E, d(u,v)$ is the distance of the link and can be the number of hops in the Internet.

Let s be the origin server. Suppose there are k proxies that can be used by s to host its contents, denoted by $P = \{p_1, p_2, \dots, p_k\}$. We assume the locations of the k proxies are given in prior (the proxy placement problem in the Internet can be found in [7], [14], [19], [21]). Each proxy, p_i, has a limited storage allocation to this server s, denoted by c_i.

The origin server has a set of m data objects, denoted by $O = \{o_1, o_2, \dots, o_m\}$. Each data object, o_i, has a size z_i. Every node, $u \in V$, has a read frequency to data object o_i, denoted by $r(u, o_i), 1 \le i \le m$. Each data object, o_i, has an update frequency $w(o_i)$.

We define the cost of data access as the distance of data travelling times the size of the data during a certain period of time. The cost is in the unit of (hops × bytes)/time. To formulate the cost function of data access, we extend the distance function to path $\pi(u,v)$ between nodes u and v as:

$$d(u,v) = \sum_{(x,y)\in\pi(u,v)} d(x,y).$$

Let $d(v,o_i)$ denote the distance from node v to object o_i. $d(v,o_i)$ is $d(v,p_j)$ if a request for retrieving o_i is served by proxy p_j and it becomes $d(v,s)$ if the request is missed by all the proxies and is finally served by the origin server s. The cost of the user at node v to retrieve o_i can be represented as:

$$r(v,o_i) \times d(v,o_i) \times z_i.$$

The total retrieval cost of o_i by all clients in the network is:

$$r\_cost(o_i) = \sum_{v\in V} r(v,o_i) \times d(v,o_i) \times z_i. \tag{1}$$

The replicas of a data object at proxies need to be updated when the original copy is modified. In Web systems, update operations usually originate from the origin server, where the update of Web information is performed by authorized persons. When data object o_i is updated, the new version of o_i needs to be transmitted to the proxies that hold o_i. We assume the multicast model is used for the server to transmit updated data to proxies. That is, the route for multicasting from server s to proxies is a multicast tree (*MT*). Let $P(o_i)$ denote a set of proxies where o_i is replicated (including the server s) and $MT(s, P(o_i))$ the multicast tree rooted from s to reach all proxies in $P(o_i)$. The updating cost (write cost) of o_i is:

$$w_{cost}(o_i) = w(o_i) \times z_i \times \sum_{(x,y)\in MT(s,P(o_i))} d(x,y). \tag{2}$$

Equation (1) + (2) is the total access cost to o_i by all clients in the network:

$$Cost(o_i) = \sum_{v\in V} r(v,o_i) \times d(v,o_i) \times z_i + w(o_i) \times z_i \times \sum_{(x,y)\in MT(s,P(o_i))} d(x,y). \tag{3}$$

The overall cost for all clients to access all m objects in the network is

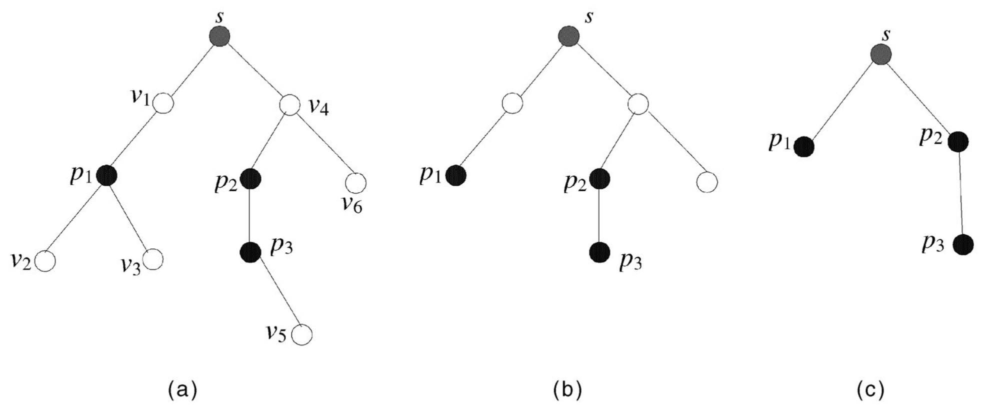


Fig. 1. An example of producing Induced Subtree (nodes in black are proxies). (a) T_s. (b) Intermediate Graph from T_s to $T_s(P)$. (c) $T_s(P)$.

$$Cost = \sum_{i=1}^{m} Cost(o_i). \tag{4}$$

From the above cost function, we can see the cost for accessing data objects depends on the locations where they are replicated, on the Web can be significantly reduced. Since each proxy has a limited storage capacity for s, the total size of s's data objects replicated at this proxy should not exceed this capacity. This constraint can be represented as:

$$\forall i: \sum_{j=1}^{m} \delta_{ij} \times z_j \leq c_i, \\ \text{where}|\ \delta_{ij} = \begin{cases} 1 & \text{if } p_i \in P(o_j) \\ 0 & \text{otherwise} \end{cases}, 1 \leq i \leq k. \tag{5}$$

We will focus on the two subproblems of replicating data objects at proxies:

1. Assuming each proxy has storage large enough to hold all data objects for s, find the optimal replication of data objects at proxies, such that the overall cost defined in (4) is minimized.
2. Assuming each proxy has a limited storage capacity for s, find the optimal replication of data objects at proxies, such that the overall cost defined in (4) is minimized under the constraint defined in (5).

Data access cost considered in our model is the cost defined in (4). We do not consider other overheads, such as CPU computational overhead and the overhead for visiting proxies (even objects are not cached).

4 Our Solutions

4.1 Computing the Induced Tree of Proxy Nodes

Since the proxies discussed in this paper are transparent to clients, clients use the URL address of the origin server for data retrieval. When a request is intercepted by a proxy on its way to the origin server and the requested data is available at the proxy, it will be served by the proxy; otherwise, it needs to go all the way to the origin server. If the routing in the network is stable (studies in [19], [20], [22] show that the routing is stable for most of the time in the Internet), requests would always take the same route to the origin server. In this case, the routes from all clients to access the origin server s form a tree, where the root of the tree is s and all the leaf nodes are clients. Some (not all) nonleaf nodes are proxy nodes. Let T_s denote such a routing tree from all clients to the server s. Since we only concern about data replication at proxies and the proxy nodes are given in prior, we can focus on the induced tree that contains only proxy nodes.

Let $T_s(P)$ denote the induced tree of T_s. $T_s(P)$ is obtained by contracting all nonproxy nodes (i.e., clients) to their corresponding proxy nodes in T_s. A nonproxy node is contracted to the first proxy node it meets on its way to the root in T_s. Fig. 1 illustrates the steps of producing the induced tree $T_s(P)$ from T_s. Fig. 1a is T_s. Fig. 1b is an intermediate induced tree after contracting the lowest levels of nonproxy nodes to their corresponding proxy nodes. Fig. 1c is the final induced tree. From this example, we can see that the nonproxy nodes contracted to a proxy p are those client nodes that will retrieve data from p.

Now, we define link distance and read frequency in $T_s(P)$. For each link $(p_i, p_j) \in T_s(P)$, $d(p_i, p_j)$ is the distance of path $\pi(p_i, p_j)$ in T_s. For each node $p \in T_s(P)$, the new read frequency of p to object o_j, denoted by $r^+(p, o_j)$, is the aggregate read frequency of all the non-proxy nodes that contract to it (notice that p is a proxy node). $r^+(p, o_j)$ can be formally expressed as below. Let T_p denote a subtree of T_s and the root of T_p is p.

Definition 1. *p' is said to be a* direct child proxy *of p if p' is a proxy node in the subtree of T_p and there is no other proxy nodes in the path between p and p' along the tree.*

For example, in Fig. 1a, p_1 and p_2 are the direct child proxies of s (s is a special proxy node).

For any $p \in T_s(P)$, let $C(p)$ denote the set of *direct child proxies* of p in T_s. The union of subtrees of *direct child proxies* of p is represented as:

$$\cup_{p' \in C(p)} T_{p'}.$$

Since p' is a direct child proxy of p, the nonproxy nodes in $T_{p'}$ will be contracted to p' or other proxies inside $T_{p'}$ (i.e., they will not be contracted to p) when producing the induced tree $T_s(P)$. The set of nonproxy nodes that will be contracted to proxy node p can be expressed as:

$$T_p - \cup_{p' \in C(p)} T_{p'}.$$

Taking Fig. 1a as an example again, $T_s - (T_{p_1} \cup T_{p_2}) = \{v_1, v_4, v_6\}$. This set of nodes go to s for data retrieval and will thus be contracted to s in $T_s(P)$. Their read frequencies to data objects will be aggregated to s.

Therefore, for any $p \in T_s(P)$, the aggregate read frequency of o_j is:

$$r^+(p, o_j) = \sum_{v \in T_p - \cup_{p' \in C(p)} T_{p'}} r(v, o_j). \tag{6}$$

We have obtained the induced tree $T_s(P)$. Each node $p \in T_s(P)$ is a proxy node in T_s and the aggregate frequency of p to read o_j is given in formula (6). Then, next, we will express the cost function (3) in the context of $T_s(P)$.

Let $p(v), v \in V$, be the first proxy node from v to root s along tree T_s. The distance $d(v, o_i)$ consists of two parts: $d(v, p(v))$ and $d(p(v), o_i)$. Notice that $d(p(v), o_i) = 0$ if $p(v)$ holds o_i. The cost function (3) can be rewritten as:

$$\begin{aligned} Cost(o_i) &= \sum_{v \in V} r(v, o_i) \times (d(v, p(v)) + d(p(v), o_i)) \times z_i \\ &\quad + w\_cost(o_i) \\ &= \sum_{v \in V} r(v, o_i) \times d(v, p(v)) \times z_i + \sum_{v \in V} r(v, o_i) \\ &\quad \times d(p(v), o_i) \times z_i + w\_cost(o_i). \end{aligned} \tag{7}$$

The first term in (7) is the cost by all clients to reach the proxies they are contracted to. It is a constant once the locations of P are given, irrespective to whether or not o_i is replicated at a proxy. Considering the second term in (7), the nonproxy nodes in T_s can be grouped under the proxies they are contracted to. Thus, (7) can be expressed as:

$$\begin{aligned} Cost(o_i) = &\sum_{v \in V} r(v, o_i) \times d(v, p(v)) \times z_i + \\ &\sum_{p \in T_s(P)} \left(\sum_{v \in T_p - \cup_{p' \in C(p)} T_{p'}} r(v, o_i) \times d(p(v), o_i) \times z_i \right) + \\ &w\_cost(o_i). \end{aligned}$$

Since any nonproxy node $v \in T_p - \cup_{p' \in C(p)} T_{p'}$ is contracted to proxy node p, $p(v)$ is p in $T_s(P)$. Thus, we have:

$$\begin{aligned} Cost(o_i) &= \sum_{v \in V} r(v, o_i) \times d(v, p(v)) \times z_i + \\ &\quad \sum_{p \in T_s(P)} \left(\sum_{v \in T_p - \cup_{p' \in C(P)} T_{p'}} r(v, o_i) \times d(p, o_i) \times z_i \right) + \\ &\quad w\_cost(o_i) \\ &= \sum_{v \in V} r(v, o_i) \times d(v, p(v)) \times z_i + \\ &\quad \sum_{p \in T_s(P)} r^+(p, o_i) \times d(p, o_i) \times z_i + w\_cost(o_i). \end{aligned} \tag{8}$$

In (8), the third term, $w\_cost(o_i)$ is

$$w(o_i) \times z_i \sum_{(x,y) \in MT(s, P(o_i))} d(x, y).$$

Multicast tree $MT(s, P(o_i))$ is a subtree of T_s, where all its leaf nodes are in $P(o_i)$. $\sum_{(x,y) \in MT(s,P(o_i))} d(x, y)$ is the overall distance of $MT(s, P(o_i))$, which is the same in both $T_s(P)$ and T_s (notice that the distance of a link between two nodes in $T_s(P)$ is the sum of distance of a path between the two nodes in T_s). Therefore, the cost function (8) can be computed in the induced tree $T_s(P)$. For the rest of the paper, our algorithms are developed on $T_s(P)$.

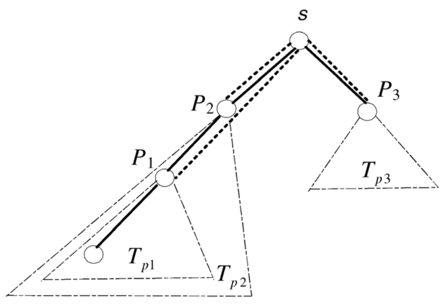


Fig. 2. An example of cost-saving by replicating data at proxies.

We further rewrite the cost expression in the case of replicating objects at proxies in the supplementary fashion to the case of not replicating any objects.

Assuming there is no replica of o_i placed at any proxies, i.e., only the server holds o_i, the total cost for accessing o_i is:

$$Cost^0(o_i) = \sum_{p \in T_s(P)} r^+(p, o_i) \times d(p, s) \times z_i. \tag{9}$$

This is because all the clients need go to the root s to retrieve the data and no updated cost is incurred.

Now, we consider the cost-saving by replicating o_i at proxies. Before formulating the general case of cost-saving, let's take the example in Fig. 2 and consider the cost-saving of replicating o_i at three proxies, p_1, p_2, and p_3. When o_i is replicated at p_1, all nodes in T_{p1} come to p_1 to read o_i, instead of going to s. Thus, $d(p_1, s)$ is the distance saved for them to read o_i. The saving gained by reading o_i at p_1 is:

$$\left(\sum_{u \in T_{p_1}} r^+(u, o_i) \right) \times d(p_1, s) \times z_i. \tag{10}$$

The cost incurred by updating o_i at p_1 is:

$$w(o_i) \times d(p_1, s) \times z_i. \tag{11}$$

Equation (10) - (11) is the net gain by replicating o_i at p_1:

$$\left(\sum_{u \in T_{p_1}} r^+(u, o_i) \right) \times d(p_1, s) \times z_i - w(o_i) \times d(p_1, s) \times z_i. \tag{12}$$

Since $d(p_1, s) = d(p_1, p_2) + d(p_2, s)$, as shown in Fig. 2, (12) can be rewritten as:

$$\begin{aligned} &\left(\sum_{u \in T_{p_1}} r^+(u, o_i) \right) \times d(p_1, p_2) \times z_i + \left(\sum_{u \in T_{p_1}} r^+(u, o_i) \right) \\ &\times d(p_2, s) \times z_i - w(o_i) \times d(p_1, p_2) \times z_i - w(o_i) \times d(p_2, s) \times z_i. \end{aligned} \tag{13}$$

When o_i is also replicated at p_2, all nodes in $(T_{p_2} - T_{p_1})$ will come to p_2 to read o_i. The distance saved for reading o_i at p_2 is $d(p_2, s)$. The saving gained by reading o_i at p_2 is:

$$\left(\sum_{u\in(T_{p_2}-T_{p_1})} r^+(u,o_i)\right) \times d(p_2,s)\times z_i \tag{14}$$

$$= \left(\sum_{u\in T_{p2}} r^+(u,o_i)\right) \times d(p_2,s)\times z_i - \left(\sum_{u\in T_{p_1}} r^+(u,o_i)\right) \times d(p_2,s)\times z_i. \tag{15}$$

Since the multicast model is used to send updated data to all replicas, by replicating another copy of o_i at p_2 it does not incur extra cost for updating o_i, because the new version of o_i will be transmitted to p_1 via p_2 (from s) anyway. Thus, (15) is also the net gain by replicating o_i at p_2.

Considering replicating o_i at p_3, which is similar to the case at p_1, the net gain by replicating o_i at p_3 is:

$$\left(\sum_{u\in T_{p_3}} r^+(u,o_i)\right) \times d(p_3,s)\times z_i - w(o_i)\times d(p_3,s)\times z_i. \tag{16}$$

Equation (13) + (15) + (16) is the net gain by replicating o_i at p_1, p_2, and p_3:

$$\left(\sum_{u\in T_{p_1}} r^+(u,o_i) - w(o_i)\right)\times d(p_1,p_2)\times z_i + \left(\sum_{u\in T_{p_2}} r^+(u,o_i) - w(o_i)\right)\times d(p_2,s)\times z_i + \left(\sum_{u\in T_{p_3}} r^+(u,o_i) - w(o_i)\right)\times d(p_3,s)\times z_i. \tag{17}$$

Notice that the distances $d(p_1,p_2)$, $d(p_2,s)$, and $d(p_3,s)$ in (17) are the distances from a proxy that has o_i to its first ancestor that also holds o_i in $T_s(P)$. Let $d(p,o_i), p\in T_s(P)$, denote the distance from a proxy p to its first ancestor holding o_i in $T_s(P)$. For example, in Fig. 2, $d(p_1,o_i)$ is $d(p_1,p_2)$.

Now, we formulate the general expression of cost-saving of replicating objects at proxies. The net gain of replicating o_i at p against the case of no replication of o_i is (i.e., the case in (9)):

$$\left(\sum_{u\in T_p} r^+(u,o_i) - w(o_i)\right)\times d(p,o_i)\times z_i. \tag{18}$$

The total net gain of replicating o_i at all proxies in $P(o_i)$ is:

$$\sum_{p\in P(o_i)}\left(\sum_{u\in T_p} r^+(u,o_i) - w(o_i)\right)\times d(p,o_i)\times z_i. \tag{19}$$

The proof of the correctness of expression is straightforward following the example of formula (17).

Equation (9) - (19) is the cost of accessing o_i when o_i is replicated at all proxies in $P(o_i)$:

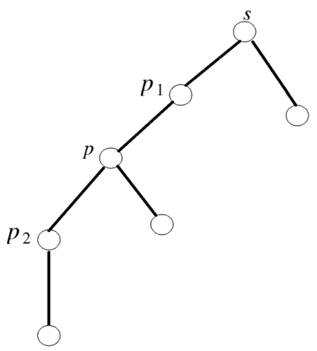


Fig. 3. Making another replica of o_i at p.

$$Cost(o_i) = Cost^0(o_i)\sum_{p\in P(o_i)}\left(\sum_{u\in T_p} r^+(u,o_i) - w(o_i)\right)\times d(p,o_i)\times z_i. \tag{20}$$

The overall cost for the accesses to all m objects is:

$$Cost = \sum_{i=1}^{m} Cost(o_i) = \sum_{i=1}^{m} Cost^0(o_i)\sum_{i=1}^{m}\sum_{p\in P(o_i)}\left(\sum_{u\in T_p} r^+(u,o_i) - w(o_i)\right)\times d(p,o_i)\times z_i. \tag{21}$$

In cost function (21), the first term is a constant, which is the overall access cost to the Web server without replicating any data at proxies, and the second term is the total net gain. Therefore, minimizing the overall cost is equivalent to maximizing the total gain. Our objective becomes to maximize the total gain when developing our algorithms.

In the next two sections, we will discuss the solutions to the two problems defined in Section 3. For case 1), we prove there is a polynomial time optimal solution and propose an optimal algorithm. As for case 2), it is NP-hard and two heuristic algorithms are proposed.

4.2 Proxies with Unlimited Storage Capacities

We first consider the case that all proxies are assumed to have unlimited storage capacities (i.e., large enough to hold all objects of s). The solutions to this case will lead to two effective heuristics for the limited storage case and also provide a benchmark for the maximum storage requirement in the simulations.

If proxies have unlimited storage, the optimal replication of all objects at proxies consists of the optimal replication of each object at proxies. The optimal replication of each object is independent from the others. The following lemmas will lead to an efficient algorithm for this case.

Lemma 1. *If $P(o_i)$ is the optimal replication of o_i in $T_s(P)$, for any path $\pi(p_1,p_2)$ in $T_s(P)$, where $p_1,p_2\in P(o_i)$, then $P(o_i)\cup\{p|p\in\pi(p_1,p_2)\}$ is still an optimal replication.*

Proof. Suppose p_1 and p_2 are in $P(o_i)$, and a node p is in the path between p_1 and p_2, but not in $P(o_i)$, as shown in Fig. 3 (where p_1 is an ancestor of p_2 and p_1 could be s). From (19), the gain by replicating o_i at every proxy node in $P(o_i)$ can be expressed as:

$$\left(\sum_{u\in T_{p_1}} r^+(u,o_i) - w(o_i)\right) \times d(p_1,o_i)) \times z_i + \left(\sum_{u\in T_{p_2}} r^+(u,o_i) - w(o_i)\right) \times d(p_2,p_1) \times z_i + \sum_{x\in P(o_i)-\{p_1,p_2\}} \left(\sum_{u\in T_x} r^+(u,o_i) - w(o_i)\right) \times d(x,o_i)) \times z_i. \tag{22}$$

By replicating o_i at node p, i.e., $P(o_i)' = P(o_i) \cup \{p\}$, the gain becomes:

$$\left(\sum_{u\in T_{p_1}} r^+(u,o_i) - w(o_i)\right) \times d(p_1,o_i)) \times z_i + \left(\sum_{u\in T_p} r^+(u,o_i) - w(o_i)\right) \times d(p,p_1) \times z_i + \left(\sum_{u\in T_{p_2}} r^+(u,o_i) - w(o_i)\right) \times d(p_2,p) \times z_i + \sum_{x\in P(o_i)'-\{p_1,p,p_2\}} \left(\sum_{u\in T_x} r^+(u,o_i) - w(o_i)\right) \times d(x,o_i)) \times z_i. \tag{23}$$

Equation (23) - (22) is the extra gain by making an extra replica at node p, which is:

$$\left(\sum_{u\in T_p} r^+(u,o_i) - w(o_i)\right) \times d(p,p_1) \times z_i + \left(\sum_{u\in T_{p_2}} r^+(u,o_i) - w(o_i)\right) \times d(p_2,p) \times z_i - \left(\sum_{u\in T_{p_2}} r^+(u,o_i) - w(o_i)\right) \times d(p_2,p_1) \times z_i. \tag{24}$$

This is because $P(o_i)' - \{p_1,p,p_2\} = P(o_i) - \{p_1,p_2\}$.

Furthermore, since $d(p_2,p_1) = d(p_2,p) + d(p,p_1)$ (see Fig. 3), the gain in (24) becomes:

$$\left(\sum_{u\in T_p} r^+(u,o_i) - w(o_i)\right) \times d(p,p_1) \times z_i - \left(\sum_{u\in T_{p_2}} r^+(u,o_i) - w(o_i)\right) \times d(p,p_1) \times z_i = \left(\sum_{u\in T_p} r^+(u,o_i) - \sum_{u\in T_{p_2}} r^+(u,o_i)\right) \times d(p,p_1) \times z_i. \tag{25}$$

The extra gain in (25) is greater than or equal to zero, because $\sum_{u\in T_p} r^+(u,o_i) \geq \sum_{u\in T_{p_2}} r^+(u,o_i)$ (because $T_{p_2} \subset T_p$). Thus, by adding p to $P(o_i)$, the access cost will decrease or at least remain unchanged. Therefore, $P(o_i) \cup \{p\}$ is still an optimal replication of o_i in $T_s(P)$. Lemma 1 is proven. □

Notice that Lemma 1 states that if o_i is replicated at p_1 and p_2 and they are in a path in $T_s(P)$, then o_i can be replicated at any proxy node in between p_1 and p_2, because it does not incur any extra cost for updating the object but save the cost for reading the object. However, if there is no read access to o_i at a proxy node, there is no need to replicate it due to the overhead of maintaining this unused replica at the proxy node. The next lemma states that an object will be replicated at a proxy only if there is a positive gain. That is, an optimal set of replicas also contains only the minimal number of replicas.

Lemma 2. *Let* $P(o_i) = \{p | \sum_{u\in T_p} r^+(u,o_i) > w(o_i)$, *for any* $p \in T_s(P)\}$. *Then,* $P(o_i)$ *is the optimal replication of* o_i *in* $T_s(P)$.

Proof. We prove it by contradiction. We assume $P(o_i)$ is not optimal and P_i^{opt} is the optimal replication of o_i. Then, there must exist a proxy $p' \in P_i^{opt}$, but $p' \notin P(o_i)$. Since $p' \notin P(o_i)$, we have $\sum_{u\in T_{p'}} r^+(u,o_i) \leq w(o_i)$. We consider the two cases:

Case $\sum_{u\in T_{p'}} r^+(u,o_i) < w(o_i)$: In this case, p' can be removed from P_i^{opt}, which would result in less cost in the access to o_i, because the cost of updating o_i at p' overweighs the saving of reading o_i at p' by the clients in the subtree of $T_{p'}$. Thus, P_i^{opt} is not optimal, which contradicts to the assumption.

Case $\sum_{u\in T_{p'}} r^+(u,o_i) = w(o_i)$: In this case, p' can be removed from P_i^{opt} but the cost remains unchanged. Thus, P_i^{opt} is not optimal in the sense that it contains unnecessary objects, which contradicts to the assumption.

Therefore, $P(o_i)$ is an optimal replication. □

Lemma 1 and Lemma 2 lead to an efficient algorithm of searching for the optimal replication of o_i at proxies. According to Lemma 1, the algorithm employs a top-down search on the tree nodes in $T_s(P)$, starting from root s. By using Lemma 2, it tests condition $\sum_{u\in T_p} r^+(u,o_i) > w(o_i)$ for each node p. If the condition is true, p is included in $P(o_i)$, and p's children are added into the candidate set. This search operation stops when all the nodes in $T_s(P)$ are searched. The algorithm is called *Opt-replic*.

The detailed algorithm is given below.

```
Opt-replic{
    $P(o_i) \leftarrow s$;
    $C \leftarrow \{s\text{'s children}\}$;
    while $C \neq \emptyset$ do
        pick any node $p \in C$ and remove $p$ from $C$;
        if $\sum_{u\in T_p} r^+(u,o_i) > w(o_i)$, then
            $P(o_i) \leftarrow P(o_i) \cup p$;
            $C \leftarrow C \cup \{p\text{'s children}\}$;
    end-while
}
```

Theorem 1. $P(o_i)$ *produced by* Opt-replic *algorithm is the set of optimal replication of* o_i.

Proof. *Opt-replic* algorithm searches all nodes in $T_s(P)$ and includes in $P(o_i)$ any node p that satisfies $\sum_{u\in T_p} r^+(u,o_i) > w(o_i)$. According to Lemma 2, $P(o_i)$ is the optimal replication of o_i. □

Theorem 2. Opt-replic *algorithm can produce the optimal replication of o_i in time $O(|P|^2)$, and the optimal replication of m objects in time $O(m|P|^2)$, where $|P|$ is the number of nodes in $T_s(P)$.*

Proof. The algorithm searches nodes in $\mathrm{T_s(P)}$ at most once. At each node p, it computes $\sum_{u\in T_p} r^+(u,o_i) > w(o_i)$, which takes time $\mathrm{O(|T_p|)}$. Thus, Opt-replic has complexity of $\sum_{p\in T_s(P)} |T_p| = \mathrm{O(|P|^2)}$. Because the optimal replication of objects is independent from each other, by using Opt-replic algorithm to compute the optimal replication of m objects, it takes time $\mathrm{O(m|P|^2)}$. □

4.3 Proxies with Limited Storage Capacities

We refer the problem of optimal replication of data objects at proxies that have limited storage capacities as ORLS. We first prove that the ORLS problem is NP-hard and then propose two heuristic algorithms.

Theorem 3. *The ORLS problem is NP-hard.*

Proof. We first introduce the following definition. □

Definition 2: Partition problem. *Given a list of integers $\{a_1, a_2, \ldots, a_n\}$, determine whether there is a set $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i\in I} a_i = \sum_{i\notin I} a_i$.*

The partition problem has been proved to be NP-hard [12]. In the following, we will prove that the partition problem is a special case of the ORLS problem.

Suppose there are m objects $\{o_1, o_2, \ldots, o_m\}$ stored at server s and the sum of the sizes of the m objects is $\sum_{i=1}^{m} z_i = 2b$. We consider a special case of the ORLS problem that the replication of data at a proxy is independent from the others. Consider proxy p whose storage allocation to s is b.

We first assume all nodes in T_s have the same access frequency to any object, i.e.,

$$\forall u, v \in T_s : r(u,o_i) = r(v,o_i), 1 \le i \le m.$$

We can always design read frequencies, write frequencies and the sizes of objects in such a way to meet the following constraints. For $u \in T_s$ and $1 \le i, j \le m$:

$$z_i r(u,o_i) = z_j r(u,o_j), \tag{26}$$

$$w(o_i) z_i = w(o_j) z_j, \tag{27}$$

$$w(o_i) < \sum_{u\in T_p} r(u,o_i). \tag{28}$$

The purpose of (26) and (27) is to make all the data objects contributing the same to the cost function. That is, the selection of objects to be replicated at proxies can be arbitrary. Constraint (28) ensures that all the objects are qualified to be replicated at proxy p if p has unlimited storage capacity (according to Lemma 2).

Since all objects have the same weight to be chosen for replication at p, the optimal replication of them at p with capacity b becomes choosing a subset of objects $I = \{o_{i_1}, o_{i_2}, \ldots, o_{i_t}\}$, $I \subseteq \{o_1, o_2, \ldots, o_m\}$, such that $\sum_{o_{i_j}\in I} z_{i_j} = \sum_{o_{i_j}\notin I} z_{i_j} = b$. This is exactly the partition problem, which is NP-hard. Therefore, the ORLS problem is NP-hard.

4.3.1 A Greedy Heuristic

The greedy algorithm traverses the nodes in $T_s(P)$ in a breadth-first fashion, starting from root s. At each node p it traverses, it evaluates the gain of replicating o_i at p defined as:

$$gain(p,o_i) = \left(\sum_{u\in T_p} r^+(u,o_i) - w(o_i)\right) \times d(p,o_i) \times z_i. \tag{29}$$

The objects are sorted in descending order according to their gains (the objects with 0 or negative gains are dropped out). Then, the algorithm simply replicates at p the first k objects that have the largest gains and can be accommodated at p. For the rest of the storage at p, the objects from the rest of the list are chosen to fill it up.

Since the algorithm considers the replication of objects in a top-down fashion, distance $d(p,o_i)$ in (29) is always known when considering the replicating of o_i at p.

```
Greedy{
    append $s$'s children to a list $L$ in the order from left to right;
    $p =$ get-head$(L)$;
    greedy-call$(p)$;
}
greedy-call$(p)$ {
    if $p =$ nil then return;
    sort all objects o_i in descending order of $gain(p,o_i) > 0$;
    // denote the sorted object list as $o_{i_1}, o_{i_2}, \ldots, o_{i_l}$

    if $z_{i_1}+z_{i_2}+\ldots+z_{i_k} \le c_p$ but $z_{i1}+z_{i_2}+\ldots+z_{i_k}+z_{i_{k+1}} > c_p$ then
        replicate $o_{i_1}, o_{i_2}, \ldots, o_{i_k}$ at $p$;
        search the rest of objects in the list to fill up the rest of space in $p$;
    append $p$'s children to $L$ in the order from left to right;
    $p' =$ get-head$(L)$;
    greedy-call$(p')$;
}
```

Theorem 4. *The time complexity of* Greedy *algorithm is $O(|P|^2 m + |P| m \lg m)$.*

Proof. *Greedy* algorithm computes the replication at each node in $T_s(P)$ by calling subroutine greedy-call(p). At each node p, it computes $gain(p,o_i)$ for all objects, which take time $O(|T_p|m)$. The sorting of $gain(p,o_i)$ for $1 \le i \le m$ takes time $O(m\lg m)$. Therefore, computing the replication at all nodes in $T_s(P)$ takes time:

$$O\left(\sum_{p\in T_s(P)} (|T_p|m + m\lg m)\right) = O(|P|2m + |P|m\lg m).$$

□

4.3.2 A Knapsack-Based Heuristic

If the object sizes vary drastically, which is the usual case for Web objects, the greedy algorithm may not perform well, because several large size objects may quickly fill up the storage space of a proxy.

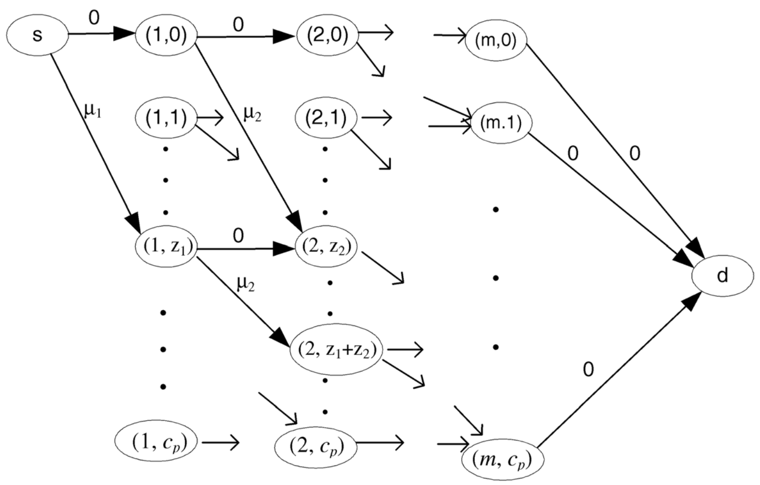


Fig. 4. The auxiliary graph.

In the knapsack-based heuristic, when considering data replication at proxy node p, we try to maximize the overall gain contributed by all objects under the limited storage space of p. The overall gain of replicating all possible objects is:

$$\sum_{i=1}^{m}\left(\left(\sum_{u\in T_p} r^{+}(u,o_i)-w(o_i)\right)\times d(p,o_i)\times z_i\right)x_i, \tag{30}$$
$$\text{where } x_i=\begin{cases}1 & \text{if } o_i \text{ is replicated;}\\ 0 & \text{otherwise.}\end{cases}$$

Let $\mu_i=\left(\sum_{u\in T_p} r^{+}(u,o_i)-w(o_i)\right)\times d(p,o_i)\times z_i$ in (30), which is a constant for a given o_i at p. The problem of finding the optimal replication of objects at p can be formulated as the following optimization problem:

$$\max\sum_{i=1}^{m}\mu_i x_i$$
$$s.t.\begin{cases} z_1x_1+z_2x_2+\ldots+z_mx_m\le c_p\\ x_i=0 \text{ or } 1,\ \text{for } i=1,2,\ldots,m.\end{cases} \tag{31}$$

This is a typical knapsack problem. We use an auxiliary direct graph $G_A=(V_A,E_A)$ to solve this problem.

The vertices in G_A are arranged as an array, as shown in Fig. 4. Each column corresponds to an object $o_i, 1\le i\le m$. We assume the storage size is in multiple of pages (a page could be in size of 1K, 2K, or even larger). Thus, each column has (c_p+1) vertices, representing the sequence of storage size from $0,1,\ldots,$ to c_p, respectively. Therefore, G_A has $m\times(c_p+1)$ vertices in total.

Each vertex in G_A is indexed by $(i,j), 1\le i\le m$ and $0\le j\le c_p$, indicating a combinatorial case of replicating a subset of objects from set $\{o_1,o_2,\ldots,o_i\}$ at proxy p, and the total storage size of this subset of objects being j. In G_A, each vertex (i,j) has two outgoing arcs pointing to two vertices in the $(i+1)$th column, $(i+1,j)$ and $(i+1,jz_{i+1})$, indicating the two cases of not replicating or replicating o_{i+1}, respectively. The weight on the former arc is 0, and the weight on the latter is μ_{i+1} (i.e., the gain of replicating o_{i+1} at p).

We add two pseudonodes s and d to G_A, representing the source and the destination of the graph. s has an arc to $(1,0)$ with weight 0 and another arc to $(1,z_1)$ with weight μ_1. To vertex d, there is an arc with weight 0 from each vertex $(m,k), 0\le k\le c_p$. See Fig. 4.

From the construction of G_A, we can see the total weight of any path from s to d is $\sum_{i=1}^{m}\mu_i x_i$, where $x_i=1$ if the weight of the arc from the $(i-1)$th column vertex to the ith column vertex is μ_i; otherwise, 0. Therefore, the optimization problem formulated in (31) is equivalent to find the path from s to d that has the largest weight, i.e., $\max\{\sum_{i=1}^{m}\mu_i x_i\}$. The value of x_i along this path, $0\le i\le m$, determines if o_i is replicated at this proxy or not. The path with the largest weight can be computed by using any shortest path algorithm, such as the *Bellman-Ford* algorithm. It takes time $O(|V_A||E_A|)=O(m^2c_p^2)$, because $|V_A|$ is $m\times(c_p+1)+2$, and $|E_A|$ is at most $2|V_A|$.

The overall structure of the algorithm is similar to the greedy algorithm. It traverses the nodes in $T_s(P)$ in a breadth-first fashion, and at each node, it uses the knapsack algorithm to compute the optimal replication of objects at this node.

```
Knapsack-h{
    append $s$'s children to a list $L$ in the order from left to right;
    $p$ = get-head($L$);
    knapsack-call($p$);
}
knapsack-call($p$){
    if $p$ = nil then return;
    construct $G_A=(V_A,E_A)$ for $p$ with capacity $c_p$;
    compute the largest path from s to d and obtain values of $x_i, 1\le i\le m$;
    append $p$'s children to $L$ in the order from left to right;
    $p'$ = get-head($L$);
    knapsack-call($p'$);
}
```

Theorem 5. *The time complexity of* Knapsack-h *algorithm is* $O(m^2\sum_{p\in T_s(P)}c_p^2)$.

Proof. At node p, the construction of $G_A=(V_A,E_A)$ and the computing of the longest path takes time $O(m^2c_p^2)$. The algorithm traverses every node p in $T_s(P)$. It takes time $O(\sum_{p\in T_s(P)}m^2c_p^2)=O(m^2\sum_{p\in T_s(P)}c_p^2)$ in total. □

5 SIMULATIONS

5.1 Simulation Setup and Configurations

Extensive simulations have been conducted to evaluate the performance of the algorithms. Network topologies used in the simulations are generated using the *Inet* topology generator, downloaded from http://topology.eecs.umich.edu/inet/. It generates random network topologies following the characteristics of the Internet topology reported in [11]. The size of the networks used in the simulations is of 5,000 nodes (i.e., $n = 5,000$).

When a network graph is generated with the parameters of links and nodes, all the simulations are conducted on this graph. In each run of the simulation, a server node, s, is randomly picked up from the graph. Since we are concerned about the data replication at the given set of proxy nodes, a set of proxy nodes are also randomly picked from the graph nodes. The default number of proxies in the simulations is 50 (except the simulations in Fig. 8). A tree T_s, which is rooted from s and linking all the nodes (with shortest path) in the graph, is computed. As the preprocessing of our method, an induced tree $T_s(P)$ is computed based on the method described in Section 4.1. All the simulations of data replication algorithms are then performed on the induced tree $T_s(P)$.

The total number of data objects stored at the Web server is 10,000. The distribution of object sizes follows a heavy-tailed characterization [1], which consists of a body and a tail. The body is a lognormal distribution:

$$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x - \mu)^2/2\sigma^2}, \text{ where } \mu = 8.5 \text{ and } \sigma = 1.318.$$

The tail of the object size distribution is a Pareto distribution, which has a large variation on sizes (i.e., heavy-tailed):

$$p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}, \text{ where } \alpha = 1.1 \text{ and } k = 133K.$$

The cut-off point of the body and the tail is approximately at 133K. By using this setting, more than 93 percent of objects fall into the body distribution. The mean size of objects in is about 11K.

The read frequency to data object o_i follows the Zipf-like distribution [2]. Let $p(i)$ be the probability of accessing the ith most popular object. The Zipf-like distribution is:

$$p(i) = \frac{1}{i^{\alpha}}, \text{ where } \alpha \text{ is typically between 0.6 and 0.8.}$$

During the simulations, the parameter α in the Zipf distribution is set to 0.75, which is the average number obtained from the popular Web sites studied in [2]. Studies in [2] also show that the correlation between access frequency and object sizes is weak and can be ignored. We simply make the object o_i as the ith most popular object in the simulations. The mean value of read frequency is 0.00998 from each client.

The cost presented in simulation results below is a relative cost, which is a ratio:

$$\textit{relative-cost} = \frac{\text{overall cost with data replication at proxies}}{\text{overall cost with no data replication}}.$$

All the simulations were conducted on a PC with 2.4GHz CPU and1GB RAM. It takes a few minutes to run the *Greedy* algorithm and about 10 minutes to run the *Opt-replic* and *Knapsack* algorithms for an input size of 500 proxies and 10,000 objects.

5.2 Simulations on Optimal Data Replication with Unlimited Storage

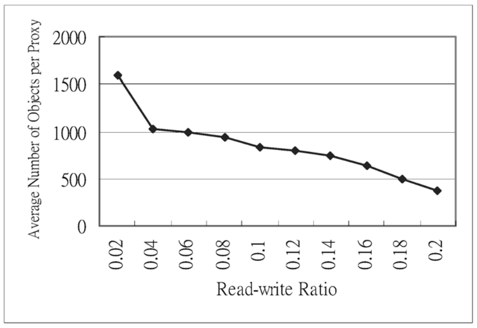
The first session of simulations was on the optimal data replication at proxies (i.e., *Opt-replic* algorithm), where the storage capacities of proxies are unlimited.

When the storage capacity at each proxy is unlimited, whether to replicate an object at a proxy purely depends on the read-write ratio of the object. Since there is little correlation between read frequency and update frequency of objects revealed in the studies of [2], we simply assume all objects have the same level of read-write ratio. Let α be the read-write ratio. The update frequency to o_i is $w(o_i) = \alpha \sum_{v \in T_s} r(v, o_i)$. In Fig. 5, the X-axis is the read-write ratio α. For a given α value, the *Opt-replic* algorithm is used to compute the optimal replication of objects at each proxy. Fig. 5a shows the average number of objects replicated at proxies and Fig. 5b the average total-size of all the objects at the proxies. For a proxy, we call the total size of the objects that are be replicated the *maximal cache size*, which will be used as the baseline of the proxy storage capacity in the simulation of Fig. 6. Fig. 5c shows the relative cost by replicating objects optimally at proxies. The following observations can be made from Fig. 5:

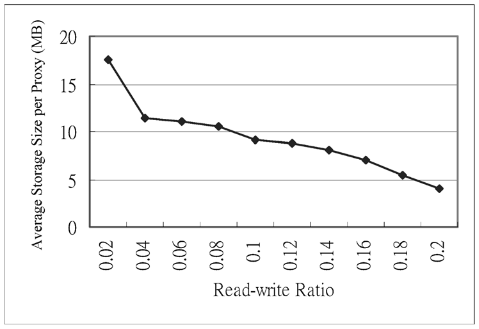
1. The number of objects that should be replicated at proxies is very sensitive to α and it drops quickly as the increase of α (about 15 percent of the total objects that are replicated even when α is 0.02). See Fig. 5a. When α value reaches some extent ($\alpha = 0.04$), the drop of the number of objects replicated at proxies becomes steady as the further increase of α. Fig. 5b shows the storage capacity required for holding the objects at proxies, in the unit of MB. It is consistent with Fig. 5a. This result tells us that increasing cache sizes may not help much in reduing the access cost when objects have relatively high update rates (say above 0.02).
2. The cost reduction by using data replication is pretty significant when the read-write ratio is low (see Fig. 5c). In fact, we did simulations by setting α to smaller values ($\alpha = 0.0005$), the relative cost can reach as low as 40 percent (Fig. 6 and Fig. 8 are the results with $\alpha = 0.001$). This result tells us that data replication is only effective when α is small and it can hardly save any cost when α reaches the level of 0.2 (i.e., update rate is 20 percent of the total read rate).

5.3 Simulations on Data Replication Algorithms with Limited Capacities

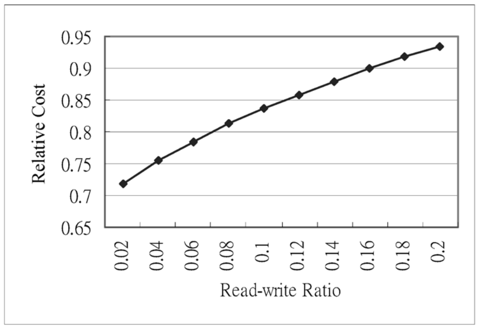
To compare with our replication algorithms, we introduce a widely used caching algorithm, LRU-threshold method (called LRU-Th). LRU-Th is based on the traditional LRU [5], but it does not cache objects whose sizes exceed a preconfigured threshold. By doing so, LRU-Th can significantly improve the hit-ratio in the situations where storage capacities are very much limited. The threshold value of the LRU-Th is set at 25K, where 88 percent of



(a)



(b)



(c)

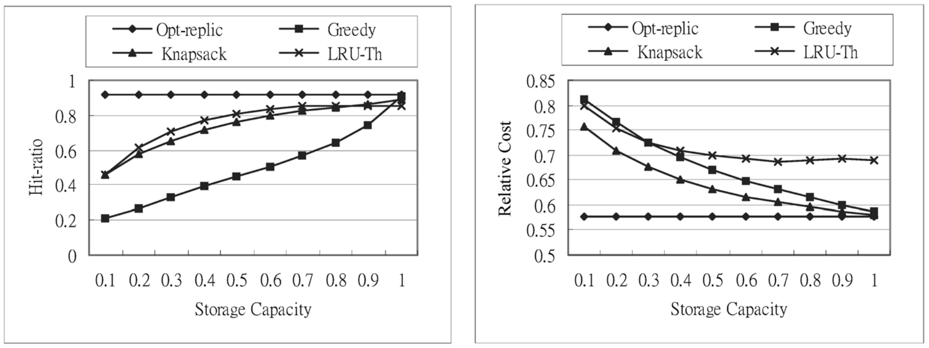
Fig. 5. (a) Average number of objects replicated at proxies versus read-write ratio. (b) Storage Size versus read-write ratio. (c) Relative cost versus read-write ratio.

objects are below this threshold (any threshold greater than 25K makes the cost of LRU-Th method higher even with a better hit-ratio). To be consistent with our data replication method, each time when an object is updated, a copy of the new data is transmitted to the proxies where this object is cached. A cache-hit is counted whenever a requested object is present in the cache. The request sequence generated is a random mixture of read and update operations. The distribution of requests to access objects still follows the Zipf-like distribution.

In this group of simulations, α is fixed at 0.001. In Fig. 6a and Fig. 6b, the storage capacities on the X-axis are the percentages of the maximal storage sizes of proxies obtained in Fig. 5a. For example, at an X-axis point 0.2, it means the storage capacity of each proxy is set as 20 percent of its maximal storage size computed by using *Opt-replic* algorithm. Fig. 6a shows the hit-ratio versus storage capacity. The hit-ratio is calculated as the number of requests served by proxies divided by the total number of requests in the whole system. Fig. 6b shows the corresponding relative cost. We include the curve of the *Opt-replic* method as the benchmark (the *Opt-replic* algorithm is not constrained by storage capacity). From Fig. 6a and 6b, the following can observed:

1. In terms of the hit-ratio (Fig. 6a), *Knapsack-h* and LRU-Th perform closely, both are significantly better than the *Greedy* method. When storage capacities reach 1 (i.e., the capacity of a proxy is large enough to hold all the objects that should be replicated based on the optimal replication algorithm), the hit-ratio of all the three methods reaches almost the same as the *Opt-replic* algorithm.
2. In terms of relative cost (Fig. 6b), the *Knapsack-h* and the *Greedy* methods behave very stable as the increase of capacities. The performance of the LRU-Th method becomes the worst of the three methods when storage capacities are greater than 30 percent. It should be noted that even though the LRU-Th method has a higher hit-ratio when storage capacities become large, the relative cost by the LRU-Th method becomes higher than the data replication algorithms. This is mainly because the LRU-Th method caches an object based on its past read accesses, which ignores the update cost (notice that each time when the object is updated, the LRU-Th method also bears the cost of refreshing the cache content). As the result, with more storage available, it caches more objects and pays a heavier price for updating them. While the data replication methods replicate objects based on the statistical prediction of read and write operations over an entire period, which takes the update cost into consideration when replicating an object. This difference between the LRU-Th method and our replication methods becomes more obvious with a slight increase of α, which is evidenced by the results in Fig. 7.

Fig. 7 shows the relative cost of the replication methods versus read-write ratio α. In this simulation, the storage capacities of proxies are set at 20 percent of their maximal sizes. From Fig. 7, we can find the cost of LRU-Th method is the highest among all the methods all the time (α starts from 0.02). This is because the LRU-Th method caches



(a) (b)

Fig. 6. (a) Hit-ratio versus storage capacity. (b) Relative cost versus storage capacity.

objects whenever they are accessed and this causes a heavy overhead to refresh them when they are updated. This is consistent with the result of the LRU-Th method shown in Fig. 6b. For both the *Knapsack-h* and the *Greedy* methods, the relative cost increases steadily as the increase of α. Their performances are pretty close to the *Opt-replic* algorithm that does not have storage constraint (notice that the storage capacity of the two heuristics is only 20 percent of the *Opt-replic* algorithm). This result again tells us that the increase of cache sizes to a large volume is not cost-effective in reducing the access cost.

Fig. 8 shows the relative cost versus the number of proxies. In this group of simulations, the α value is 0.001 and the proxy capacity is 20 percent. From Fig. 8, we can see that the performance of all three methods (except the Opt-replic method) becomes flat after the number of proxies reaches 50. This is mainly because the performance of the data replication methods is much more sensitive to the read-write ratio α than to the number of proxies. The methods of *Knapsack-h*, *Greedy*, and LRU-Th all consider the replication (or caching) of objects at a proxy relatively independent from the others. As more proxies are introduced into the system, more update cost is incurred to reach the proxies. In fact, even the *Opt-replic* method has limited cost reduction in Fig. 8. We actually simulated the number of proxies up to 500 and found the relative cost even has a slight increase. This result shows that there is no need to have a large number of proxies for the reduction of access cost. It is more cost-effective to install a small number of proxies at some heavy traffic nodes.

Finally, we simulate our data replication algorithms and LRU-Th method by using data of real Web traces. The traced data (obtained from http://ita.ee.lbl.gov/) were all HTTP requests to ClarkNet WWW server recorded on 28 August 1995 from 00:00:00 through 23:59:59 (ClarkNet is a full Internet access provider for the Metro Baltimore-Washington DC area). There were a total of 170,216 requests from 12,245 different clients that accessed 9,547 different objects. The mean size of objects is 13.7 K. The threshold for LRU-Th method is set at 100K, where about 98.8 percent of the objects are below this threshold. It is worthy pointing out that the distributions of read frequency and object size obtained from the traced data conform very well with the Zipf-like and the heavy-tailed distributions, respectively. Since there is no record of updating data objects, α value (read-write ratio) is set to 0. Clients are grouped into a tree according to the hierarchical structure of their IP addresses (or domain names). The ClarkNet Web server is the root of the tree. The number of hops between tree nodes is based on the distance between their domain names. Due to the large number of clients, 311 proxies are placed in the tree. Fig. 9a and Fig. 9b show the hit-ratio and relative cost, respectively as the increase of storage capacity of proxies (which is measured at a percentage to the total object size). We can find the results in Fig. 9 show remarkable consistency with Fig. 6, where the input data are generated from our simulation model. Again, although LRU-Th has better hit-

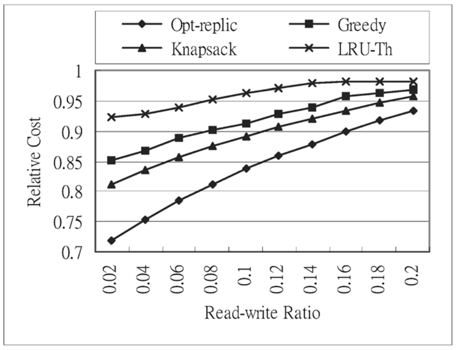


Fig. 7. Relative cost versus read-write ratio.

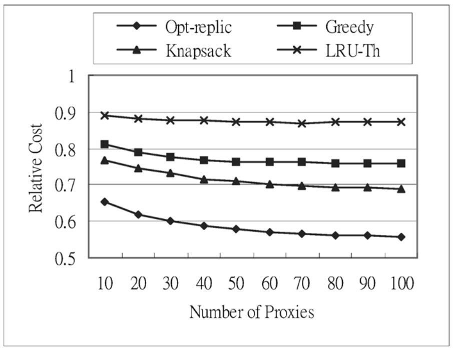
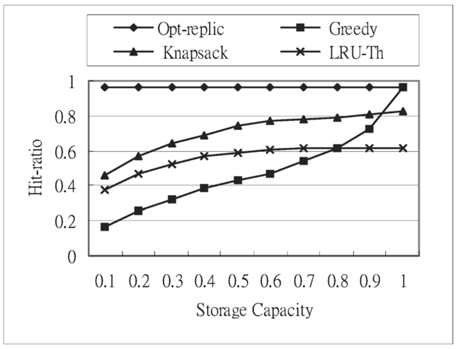
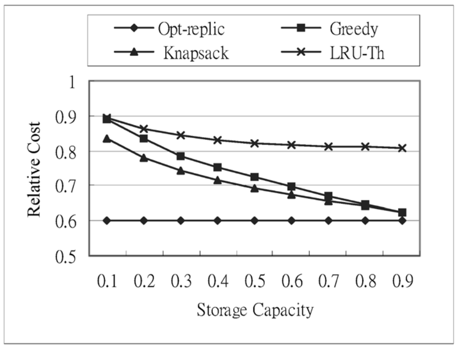


Fig. 8. Relative cost versus number of proxies.



(a)



(b)

Fig. 9. (a) Hit-ratio versus storage capacity. (b) Relative cost versus storage capacity.

ratios than the *Greedy* method when storage capacity is less than 80 percent, its relative-cost is the highest among all the methods at all time.

6 Conclusions and Discussions

In this paper, we discussed two cases of optimal data replication at transparent proxies: the proxies with unlimited storage capacities and those with limited capacities. For the former case, we gave detailed analysis of optimal data replication and proposed an efficient algorithm that computes the optimal result. For the latter, we proposed two heuristics. Extensive simulations have been conducted for performance analysis of the data replication algorithms under various network situations.

The effectiveness of transparent caching depends on the stability of the routing methods. If the routing is stable (i.e., packets from a source to a destination tend to take the same route over a long period of time), the routes used by clients to access the Web server could be modeled as a tree rooted from the server. In this case, the placement of proxies and the replication of data at proxies according to the clients' access patterns would certainly increase the hit-ratio of proxies. Studies in [20], [22] pointed out that, in reality, the Internet routing is stable. It was found that 80 percent of routes changes at a frequency lower than once a day. Krishnan et al. did another study in [19], which traced the routes from Bell Lab's Web server to 13,533 destinations. It was found that almost 93 percent of the routes are actually stable during their experiments. These studies justify our assumption on the stability of Internet routing, which reduces the problem in an arbitrary network to the problem in a tree structure.

The placement of transparent proxies in the routers requires static configuration work. Once a proxy is configured, it is expected the proxy would stay with the router for a relatively long period of time. Studies in [19] pointed out although the client population changes significantly from time to time, the traffic flow in the outgoing tree from the Web server to the clients remains pretty much stable in the sense that what are the objects accessed and how often they are accessed. This indicates that the client response time and the reduction of Web traffic can take great benefit from the proper locations of the proxies and the optimal replication of data objects.

It should also be pointed out that the performance of replication method heavily depends on the accuracy of past statistical data, such as read and update frequencies of objects. Enough statistical data on client access patterns must be collected before using the data replication method. Even during the time when the replication method is in use, collecting statistical data is also important to help adjust and improve data replication decisions from time to time.

Acknowledgments

The authors wish to thank the anonymous reviewers for their great patience and time spent in this manuscript. They gave many valuable comments and very constructive suggestions which helped improve the quality of the paper.

References

[1] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems*, pp. 151-160, July 1998.
[2] L. Breslau, P. Cao, L. Fan, G. Philips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM '99*, 1999.
[3] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World Wide Web," *IEEE Trans. Computers*, vol. 47, no. 4, pp. 445-457, Apr. 1998.
[4] S. Ceri, G. Martella, and G. Pelagatti, "Optimal File Allocation in a Computer Network: A Solution Method Based on the Knapsack Problem," *Computer Networks*, vol. 6, no. 5, pp. 345-357, 1982.
[5] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell, "A Hierarchical Internet Object Cache," *Proc. USENIX Technical Conf.*, Jan. 1996.
[6] M. Chatel, "Classical versus Transparent IP Proxies," RFC 1919, Mar. 1996.
[7] E. Cronin, S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt, "Constrained Mirror Placement on the Internet," *IEEE J. Selected Areas in Comm.*, vol. 20, no. 7, pp. 1369-1382, Sept. 2002.
[8] B.D. Davison, "Proxy Cache Comparison," http://www.web-caching.com/proxy-comparison.html, 2005.
[9] J. Dilley and M. Arlitt, "Improving Proxy Cache Performance: Analysis of Three Replacement Policies," *IEEE Internet Computing*, pp. 44-50, Nov.-Dec. 1999.
[10] L.W. Dowdy and D.V. Foster, "Comparative Models of the File Assignment Problem," *ACM Computing Survey*, vol. 14, no. 2, pp. 287-313, 1982.
[11] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," *Proc. ACM SIGCOMM*, Aug. 1999.
[12] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman, 1979.

[13] K.B. Irani and N.G. Khabbaz, "A Methodology for the Design of Communication Networks and the Distribution of Data in Distributed Supercomputer Systems," *IEEE Trans. Computers,* vol. 31, no. 5, pp. 419-434, May 1982.

[14] X. Jia, D. Li, X. Hu, and D. Du, "Optimal Placement of Web Proxies for Replicated Web Servers in the Internet," *The Computer J.,* vol. 44, no. 5, pp. 329-339, Oct. 2001.

[15] X. Jia, D. Li, X. Hu, W. Wu, and D. Du, "Placement of Web Server Proxies with Consideration of Read and Update Operations in the Internet," *The Computer J.,* vol. 46, no. 4, 2003.

[16] K. Kalpakis, K. Dasgupta, and O. Wolfson, "Optimal Placement of Replicas in Tree with Read, Write, and Storage Costs," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 6, pp. 628-636, June 2001.

[17] J. Kangasharju, J. Roberts, and K.W. Ross, "Object Replication Strategies in Content Distribution Networks," *Computer Comm.,* vol. 25, pp. 376-383, 2002.

[18] O. Kariv and S.L. Hakimi, "An Algorithmic Approach to Network Location Problems. II: The P-Medians," *SIAM J. Applied Math.,* vol. 37, no. 3, pp. 539-560, 1979.

[19] P. Krishnan, D. Raz, and Y. Shavitt, "The Cache Location Problem," *IEEE/ACM Trans. Networking,* vol. 8, no. 5, pp. 568-582, Oct. 2000.

[20] C. Labovitz, G.R. Malan, and F. Jahania, "Internet Routing Instability," *Proc. ACM SIGCOMM '97,* pp. 115-126, Aug. 1997.

[21] B. Li, M.J. Golin, G.F. Italiano, and X. Deng, "On the Optimal Placement of Web Proxies in the Internet," *Proc. IEEE INFOCOM '99,* pp. 1282-1290, Mar. 1999.

[22] V. Paxson, "End-to-End Routing Behavior in the Internet," *IEEE/ACM Trans. Networking,* vol. 5, pp. 601-615, Oct. 1997.

[23] G. Pierre, M. van Steen, and A.S. Tanenbaum, "Dynamically Selecting Optimal Distribution Strategies for Web Documents," *IEEE Trans. Computers,* vol. 51, no. 6, pp. 637-651, June 2002.

[24] L. Qiu, V. Padmanabhan, and G. Voelker, "On the Placement of Web Server Replicas," *Proc. IEEE INFOCOM '01,* Apr. 2001.

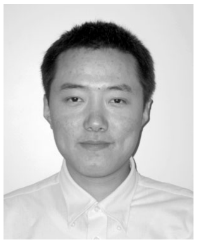
[25] P. Rodriguez, C. Spanner, and E.W. Biersack, "Analysis of Web Caching Architectures: Hierarchical and Distributed Caching," *IEEE/ACM Trans. Networking,* vol. 9, no. 4, pp. 404-418, Aug. 2001.

[26] X. Tang and S. Chanson, "Coordinated En-Route Web Caching," *IEEE Trans. Computers,* vol. 51, no. 6, pp. 595-607, June 2002.

[27] R. Tewari, T. Niranjan, S. Ramamurthy, "WCDP: A Protocol for Web Cache Consistency," *Proc. Web Caching and Content Distribution Workshop,* 2002.

Xiaohua Jia received the BSc (1984) and MEng (1987) degrees from the University of Science and Technolog of China, and the DSc degree (1991) in Information Science from the University of Tokyo, Japan. He is currently a professor in the Department of Computer Science at the City University of Hong Kong and School of Computing, Wuhan University, China. Professor Jia's research interests include distributed systems, computer networks, WDM optical networks, Internet technologies, and mobile computing.

Deying Li received the BSc (1985) and MSc (1988) degrees in mathematics from the Central China Normal University, China, and the PhD degree (2004) in computer science from the City University of Hong Kong. Dr. Li is currently an associate professor in the School of Information at Renmin University of China. Dr. Li's research interests include computer networks, WDM optical networks, Internet and mobile computing, and design of optimization algorithms.



Hongwei Du received the BSc degree in computer science from the Central China Normal University in 2003. He is currently am Mphil student in the Department of Computer Science, City University of Hong Kong. His research interests include computer networks and Internet and mobile computing.

Jinli Cao received the PhD degree in computer science from the University of Southern Queensland in 1997. She is a senior lecturer in the Department of Computer Science and Computer Engineering at La Trobe University, Australia. Her research interests include distributed systems, transaction management, distributed databases, mobile database management, Internet computing and Web information systems, electronic commerce, and Web services. She is a member of the Editorial Reviewer Board of the *International Journal of Distance Education Technologies*.