

Optimal Replica Placement on Transparent Replication Proxies for Read/Write Data*

Jianliang Xu Bo Li Dik L. Lee
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{xujl,bli,dlee}@cs.ust.hk

Abstract

Data server replication is a promising technique for improving system performance in a large distributed network. Recently, transparent data replication has been gaining increasing attention due to its low management overheads incurred. However, research on replica placement strategies for transparent data replication thus far focused on read operations only. In this paper, we consider examine for read/write applications two data replica placement problems (i.e., without and with constraint on the number of replicas respectively) for a single object on the deployed transparent replication proxies. The performance objective is to minimize the total data transfer cost for a target server under a given traffic pattern. Optimal solutions for these two problems are proposed, with complexities of $O(N)$ and $O(N + N_o P_o M^2)$ respectively. Numerical results show that the proposed solutions are very efficient.

1 Introduction

Most information systems today suffer from high communication cost and/or notoriously long access latency. To alleviate this problem, one solution is to use *data replication*. The idea of data replication is not new. Previous work has shown that a carefully designed placement scheme (i.e., the number and placement of replicas) can improve system performance significantly [4, 9, 10, 13]. However, early studies on data replication generally assumed that a client is aware of the replicas' locations so that each request can be serviced optimally (e.g., a data retrieval request is routed to its nearest replica) [4, 9, 13]. Obviously, this approach incurs considerable management overheads in identifying the optimal serving replica for each request beforehand, whether such knowledge is maintained at clients [4, 13] or is obtained on-the-fly [3, 11].

*The work was supported in part by the Research Grant Council, Hong Kong SAR, China under grant numbers HKUST-6154/98E, HKUST-6163/00E, and HKUST-6241/00E.

As a consequence, *transparent data replication* has been recently advocated by both academic and industrial communities because of its low management overheads incurred [2, 6, 7]. In this paper, we consider transparent data replication and call a computer/program that holds partial or full data replicas of the data server a *transparent replication proxy* (or *proxy* for short). One typical solution to transparent data replication is *en-route replication* [6, 7], where a proxy is co-located with a router. When a request arrives at a router, the router intercepts the request and either services it if a replica of the requested data is found in the local proxy or forwards it towards the server along the regular routing path. In other words, a request is transparently serviced by the nearest replica on the path upwards the server. Different from traditional data replication, here clients need not be aware of the serving replica for each request, and hence the management overheads are reduced greatly.

One of the major issues for data replication is the replica placement problem. In the context of transparent data replication, research on replica placement strategies is rather limited. Because accesses in transparent replication is *directed*, classic graph theoretic approaches to the problems such as the p -median problem and the facility location problem [5], where *undirected* accesses were assumed, are not applicable. In [8], Li et al. proposed an $O(N^3 M^2)$ solution to optimally allocate M Web replication servers in a tree network. Vigneron et al. improved the complexity of the algorithm to $O(PM^2)$ [12]. A similar solution was reported for the optimal placement of Web caches by Krishnan et al. [7]. In [1], Cidon et al. presented an $O(NH)$ algorithm for electronic content allocation on hierarchical servers when storage cost was considered. However, all these previous studies considered read operations only, thereby limiting their applications in the real world.

This paper examines the replica placement strategies for applications with both read and write operations. The performance objective is to minimize the total data transfer cost for a target data server under a given traffic pattern. Specifically, given N potential replication sites for a data object, we are interested in finding out how many replicas should be created and on which proxies to place them in order to obtain the optimal performance. Further, if a constraint on the maximum number of replicas is forced (i.e., only a maximum of M replicas are allowed),¹ we are interested in finding out which proxies to place these replicas on such that the performance is optimized.

The rest of this paper is organized as follows. The optimization problem of minimizing the total data transfer cost is formulated in Section 2. Sections 3 and 4 describe the optimal solutions to the placement problems without and with replica number constraint, respectively. The practical complexity of the proposed solutions is investigated in Section 5. Finally, the paper is concluded in Section 6.

2 Problem Formulation

The idea of using data replication is to improve system performance such as data transfer cost. Needless to say, the placement decision for the data replicas is crucial to the success of this idea. This is particularly true for read/write applications, in which the “wrong” placement of a proxy/replica may increase the update cost and hence the overall cost significantly. Therefore, we need some methods to obtain the optimal placement schemes. To do so, we formulate the cost model for the placement problems formally in this section. In the next two sections, we will present efficient solutions to the replica placement problems.

2.1 Transparent Replication Model

This subsection describes the transparent data replication model. A unit of data to be replicated is referred to as a *data object*. A data object could be a *tuple*, a *relation*, a *block* or an *XML/HTML page* etc. Replication proxies are located alongside some selected routers and form a tree structure (see Figure 1). *Replicated nodes* of a data object are those nodes that replicas of the object are stored. In contrast, *non-replicated nodes* are those nodes that have no replica stored locally. We assume that each object has a globally unique identifier (e.g., a table name plus a key value or a *URL*) and that a replica maintains the replicated data as well as the locations of its parent

¹The number of replicas allowed is based on various management factors, such as system resources and financial issues, which is out of the scope of this paper.

and children replicas. In the following paragraphs, we specify the read and write methods. For simplicity, we assume that a read operation consists of a single retrieval request and a write operation consists of a single update request.

A client initiates a read operation by sending to the server a *retrieval message* containing the requested object’s ID. When the retrieval request reaches a proxy, if a replica of the requested data is found locally, the proxy satisfies the request by returning the replica. Otherwise, it forwards the request to its parent proxy to request the data. In the worst case, if none of the proxies on the path upwards the server contains the requested data, the request is eventually serviced by the server.

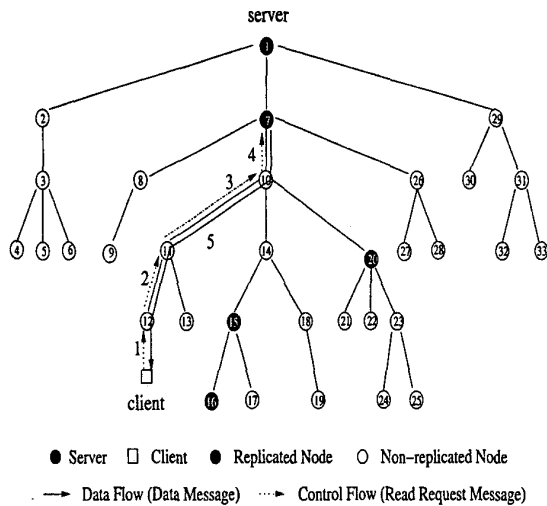
Since the multicast write policy can save the data transfer cost significantly [13], we adopt this policy for write operations in the model. Each *write operation* is initiated by an *update message* sent from a client node towards the server. An update message contains the ID of the object to be updated and the new value for the update operation. When the update message reaches a proxy, if the object to be updated is found locally, it starts the multicast write. Otherwise, the proxy forwards the update message to its parent proxy and the parent performs the same operation. In this paper, we assume multicast writes are implemented at the application level, as this can be achieved even if the underlying network infrastructure does not support multicast. The following example illustrates the read/write operations more clearly.

Example 1 As illustrated in Figure 1, node 1 is the data server and four replicas of an object are placed on the proxies at nodes 7, 15, 16, and 20. For a read operation accessing this object from node 12’s domain, the retrieval request is satisfied by the proxy at node 7 (see Figure 1(a)). For an update operation on this object from node 12’s domain, as in the read case, the update message is serviced by the proxy at node 7, i.e., a multicast write starts at node 7 (see Figure 1(b)). Since its parent proxy (i.e., node 1) and two of the descendant proxies (i.e., nodes 15 and 20) are parent and children replicas of the replica at node 7, three individual update messages are sent to them.² After the proxy at node 15 receives the update message, it further forwards it to its child proxy at node 16.

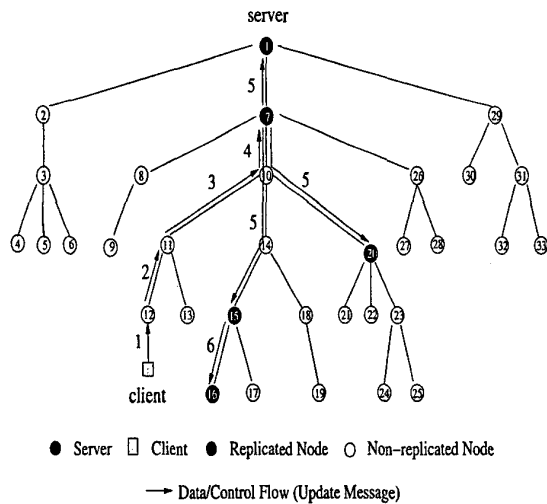
2.2 Cost Model

The topology of the proxy network is modeled as a physical tree with the server at the root. Consider

²There are slightly different ways to implement the multicast write. Our proposed algorithms are expected to handle them by slight modifications.



(a) Read Operations



(b) Multicast Write Operations

Figure 1: Message Flows for an Example of Transparent Replication Model

a tree $T_r = (V, E)$, where V is the set of nodes or vertices, E is the set of edges or links and r is the root.³ Each node in the tree representation corresponds to a proxy. Each edge corresponds to a physical link. A node $v \in T_r$ is ordered in pre-order traversal (see Figure 1). For simplicity, we hereinafter identify each node of T_r with its pre-order numbering. For each object i , every node $v \in T_r$ is associated with a nonnegative read rate $\lambda_{v,i}$ and a nonnegative write rate $\mu_{v,i}$, which represent the traffic generated within this node's local domain. For each node $v \in T_r$, denote by T_v the subtree of T_r rooted at v . Further, let $\lambda_{v,i}^t$ be the total read rate and $\mu_{v,i}^t$ the total write rate generated from subtree T_v for object i , i.e., $\lambda_{v,i}^t = \sum_{u \in T_v} \lambda_{u,i}$, $\mu_{v,i}^t = \sum_{u \in T_v} \mu_{u,i}$. Every edge $(u, v) \in T_r$ is associated with a unit transmission cost $d(u, v)$, which could be interpreted as bandwidth, link cost, hop counts, etc. We further extend the cost function $d(x, y)$ as follows: denote the unique path from node x to y by $\pi_{x,y}$; then $d(x, y) = \sum_{(u,v) \in \pi_{x,y}} d(u, v)$ is the sum of the edge costs along the path.

Suppose a data transfer cost of R is involved in a retrieval operation, and W in an update operation. Let $W/R = \alpha$. To simplify our cost model, we normalize the retrieval cost to 1. Consequently, the update cost is α . Since α could also be viewed as the ratio of the average write rate to the average read rate in the system, we refer to α as *write/read ratio* in the rest of this paper to facilitate the presentation. Given a residence set of $\mathcal{R}_i \subseteq \mathcal{P}$ (including the server) of object i , we derive in the following paragraphs the total data transfer cost for object i .

Let's first consider the case where no replicas are created in the network, i.e., only the server holds the data. It is easy to obtain the total data transfer cost for object i as follows:

$$\text{cost}(T_r, \{r\}) = \sum_{v \in T_r} (\lambda_{v,i} + \alpha \mu_{v,i}) d(v, r). \quad (1)$$

Now we are going to calculate the total data transfer cost for object i with a residence set of \mathcal{R}_i using an incremental method in a way from top to bottom and left to right.⁴ Define $c(v, \mathcal{R}_i)$ to be the lowest ancestor of $v \in T_r$ which is contained in $\mathcal{R}_i - \{v\}$, i.e., the first node in $\mathcal{R}_i - \{v\}$ that is seen while going up from v to the root r . Note that this node must not be v itself. If v is a replicated node, $c(v, \mathcal{R}_i)$ is v 's parent replica for object i . Suppose that $v \in \mathcal{R}_i - \{r\}$ is going to be

³In this and subsequent sections, for convenience, T_r could be interpreted as V or E , depending on the context.

⁴Note that there are other ways to formulate the write cost. We use this one because it is consistent with our proposed dynamic programming algorithm in the next two sections.

a replica of object i and no other replicas have been placed in T_v . The incremental data transfer cost for placing a replica at v can be expressed in the following formula:

$$\begin{aligned} \text{cost}^i(T_r, \mathcal{R}_i, v) &= -\lambda_{v,i}^t d(v, c(v, \mathcal{R}_i)) + \\ &\quad \alpha(\mu_{r,i}^t - \mu_{v,i}^t) d(v, c(v, \mathcal{R}_i)) \\ &= -(\lambda_{v,i}^t - \alpha(\mu_{r,i}^t - \mu_{v,i}^t)) \cdot \\ &\quad d(v, c(v, \mathcal{R}_i)). \end{aligned} \quad (2)$$

This is because adding v to \mathcal{R}_i decreases the read cost of each node in T_v by $d(v, c(v, \mathcal{R}_i))$ and increases the write cost of each node in $T_r - T_v$ by $d(v, c(v, \mathcal{R}_i))$, but it does not change other costs. Thus, the total data transfer cost for object i with a residence set of \mathcal{R}_i is given by:

$$\begin{aligned} \text{cost}(T_r, \mathcal{R}_i) &= \text{cost}(T_r, \{r\}) + \\ &\quad \sum_{v \in \mathcal{R}_i - \{r\}} \text{cost}^i(T_r, \mathcal{R}_i, v) \\ &= \text{cost}(T_r, \{r\}) - \sum_{v \in \mathcal{R}_i - \{r\}} (\lambda_{v,i}^t - \\ &\quad \alpha(\mu_{r,i}^t - \mu_{v,i}^t)) d(v, c(v, \mathcal{R}_i)), \end{aligned} \quad (3)$$

where the first term corresponds to the total cost when no replicas are created in the network and the second term calculates the improved cost for a set of replicas $\mathcal{R}_i - \{r\}$ incrementally.

Given a tree structure and the associated read/write patterns for object i , the first term in (3) is fixed. As such, we only need to consider the problem of maximizing the following cost instead of $\text{cost}(T_r, \mathcal{R}_i)$:

$$\begin{aligned} \text{cost}'(T_r, \mathcal{R}_i) &= \sum_{v \in \mathcal{R}_i - \{r\}} (\lambda_{v,i}^t - \alpha(\mu_{r,i}^t - \mu_{v,i}^t)) \cdot \\ &\quad d(v, c(v, \mathcal{R}_i)). \end{aligned} \quad (4)$$

This cost physically means the improved data transfer cost for object i due to the placement of a set of replicas $\mathcal{R}_i - \{r\}$. In the next two sections, we examine the optimal replica placement problems for a single object without and with constraint on the number of replicas, respectively.

3 Optimal Placement Without Replica Number Constraint

This section considers the replica placement problem for object i on the deployed proxies where the number of replicas could be arbitrary. In addition, as we will see later in Section 4.2, an efficient solution to this problem helps to reduce the complexity of the optimal algorithm for the second problem, where a

constraint exists on the number of replicas. Formally, the problem we consider here can be defined as follows:

OP1 Given a tree network $T_r = (V, E)$ and α , find a subset of \mathcal{R}_i , $\mathcal{R}_i \subseteq V, r \in \mathcal{R}_i$, which maximizes $\text{cost}'(T_r, \mathcal{R}_i)$ given in Equation (4).

Algorithm 1 Algorithm for Finding Optimal Residence Set Without Constraint

- 1: add the root r to the optimal residence set \mathcal{R}_i
 - 2: add the children of the root to a candidate set C_i
 - 3: **while** C_i is not empty **do**
 - 4: remove the first node v from C_i
 - 5: **if** $\lambda_{v,i}^t > \alpha(\mu_{r,i}^t - \mu_{v,i}^t)$ **then**
 - 6: add v to \mathcal{R}_i
 - 7: add v 's children to C_i
 - 8: **end if**
 - 9: **end while**
 - 10: output \mathcal{R}_i
-

We propose an $O(N)$ algorithm for the above problem. The algorithm is described in Algorithm 1. We are going to show the correctness of this algorithm by starting with Lemma 1. In Lemma 1, the *maximal* optimal residence set means the set has the minimal data transfer cost while the size of \mathcal{R}_i is maximized.

Lemma 1 *The maximal optimal residence set of an object must induce a connected subtree of T_r .*

Proof: The sketch of the proof is as follows. If u and v are two replicated nodes of some object i , with multicast writes adding any node on the path between u and v to \mathcal{R}_i will not increase the write cost and will probably reduce the read cost for object i . Interested reader is referred to Lemma 4.2.0 in [13] for the full proof. \square

Theorem 1 *Algorithm 1 is correct and has a complexity of $O(N)$.*

Proof: The proof is omitted, see [14] for details. \square

It is also easy to obtain the following property.

Property 1 *Suppose the optimal residence set output by Algorithm 1 is \mathcal{R}_i , then for any node $v \in T_r$ and $v \neq r$, we have $\lambda_{v,i}^t > \alpha(\mu_{r,i}^t - \mu_{v,i}^t)$ if $v \in \mathcal{R}_i$; and $\lambda_{v,i}^t \leq \alpha(\mu_{r,i}^t - \mu_{v,i}^t)$ otherwise.*

In addition, an important observation is obtained. Since Lemma 1 states that the maximum optimal residence set of an object induces a connected subtree on a

tree network, it means that under the optimal replica placement the nearest replica of an object from a node is along the path from the node towards the server. Applying this result to the problem at hand, we can conclude that under the optimal replica placement on the proxies the nearest replica to a client is always residing at a proxy upwards the hierarchy. This indicates that, in terms of data transfer cost, transparent data replication can be achieved without any penalty in performance.

4 Optimal Placement With Replica Number Constraint

In this section, we discuss the replica placement problem for object i in a tree network where a maximum of M replicas are allowed. Formally, the problem we consider here is defined as:

OP2 Given a tree network $T_r = (V, E)$, M , and α , find a subset of \mathcal{R}_i^M , $\mathcal{R}_i^M \subseteq V, |\mathcal{R}_i^M| \leq M, r \in \mathcal{R}_i^M$, which maximizes $cost'(T_r, \mathcal{R}_i^M)$ given in Equation (4).

In the following, Section 4.1 develops a dynamic programming algorithm to solve this problem. The algorithm is extended from a previous paper [8] and has a complexity of $O(NPM^2)$. In Section 4.2, we present several techniques to reduce its complexity.

4.1 Dynamic Programming Algorithm

We first define the notations used in the algorithm. Recall that each node of T_r is identified with its pre-order numbering. Denote by m_v the highest numbered node of a subtree T_v , i.e., the rightmost leaf of T_v . For example, in Figure 2, $m_7 = 28$. It can be easily proven that for all vertices $v \in T_r, T_v = [v, m_v]$. Let $s_{u,v}$ be the set of nodes on the path between u and v . Now suppose $u \in T_v$, and we can define

$$T_{u,v} = \{x \in T_v : m_u < x \leq m_v \text{ or } x \in s_{u,v} - \{u\}\} \quad (5)$$

As shown in Figure 2, intuitively, $T_{u,v}$ contains the nodes in $s_{u,v} - \{u\}$ plus all the nodes of T_v which are to the right of u . We define the following two cost functions associated with T_v and $T_{u,v}$:

- $C(v, t) = \max_{\mathcal{R}_i^t \subseteq T_v, |\mathcal{R}_i^t|=t, v \in \mathcal{R}_i^t} cost'(T_v, \mathcal{R}_i^t)$ is the optimal improved cost of placing t replicas of object i in T_v , assuming v is a replica.
- $C(u, v, t) = \max_{\mathcal{R}_i^t \subseteq T_{u,v}, |\mathcal{R}_i^t|=t, v \in \mathcal{R}_i^t} cost'(T_{u,v}, \mathcal{R}_i^t)$ is the optimal improved cost of placing t replicas of object i in $T_{u,v}$, assuming v is a replica.

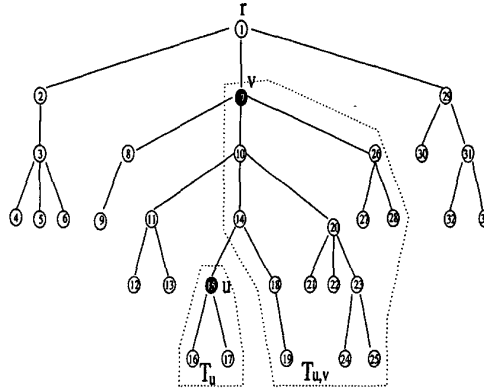


Figure 2: Definition of $T_{u,v}$

Further, we define $C^i(T_r, v, u)$ for $u \in T_v, v \in T_r$ as follow:

$$C^i(T_r, v, u) = (\lambda_{u,i}^t - \alpha(\mu_{r,i}^t - \mu_{u,i}^t))d(u, v) \quad (6)$$

If $\mathcal{R}_i^{t'}$ is any residence set of object i such that $v, u \in \mathcal{R}_i^{t'}$, and no replicas of $\mathcal{R}_i^{t'}$ are in $s_{u,v} - \{u, v\}$, then we have $c(u, \mathcal{R}_i^{t'}) = v$. This means that $C^i(T_r, v, u)$ is the contribution to $cost'(T_r, \mathcal{R}_i^{t'})$ (see Equation (4)) for placing a replica at u .

Now we are ready to derive two recurrence relations used in our algorithm as follows:

$$C(v, t) = \begin{cases} 0 & t = 1 \\ \max_{v < u \leq m_v, 0 < t' < t} (C(u, t') + C(u, v, t - t') + C^i(T_r, v, u)) & t > 1 \end{cases} \quad (7)$$

and

$$C(u, v, t) = \begin{cases} 0 & t = 1 \\ \max_{m_u < x \leq m_v, 0 < t' < t} (C(x, t') + C(x, v, t - t') + C^i(T_r, v, x)) & t > 1 \end{cases} \quad (8)$$

In Equation (7) set $P(v, t) = (u, t')$, where $v < u \leq m_v, 0 < t' < t$ are the values that maximize the expression, and in Equation (8) set $P(u, v, t) = (x, t')$, where $m_u < x \leq m_v, 0 < t' < t$ are the values that maximize the expression. For these two equations we break ties by favoring the residence set with a smaller size. For example, for $P(v, t) = (u, t')$ and $P(v, t) = (u', t'')$ which give the same cost $C(v, t)$, we choose $P(v, t) = (u, t')$ if $t' < t''$.

The algorithm for the OP2 problem is described in Algorithm 2. Basically, the algorithm can be divided into two phases. In the first phase (step 1-step 13),

we compute $C(u, t)$, $C(u, v, t)$ and the associated $P()$ entries for each node u, v and $1 \leq t \leq M$ via dynamic programming. In the second phase (step 14), we compute the optimal set of replicas \mathcal{R}_i^M recursively. This works as follows. Suppose that in the first phase the maximum cost of $cost'(T_r, \mathcal{R}_i^M)$ is achieved by $C(1, m^*)$ and $P(1, m^*) = (u^*, t^*)$. If $m^* = 1$, choose 1 (i.e., r) as the replica. Otherwise, find the optimal set \mathcal{R}_i^t of t^* replicas for T_{u^*} (i.e., from $P(u^*, t^*)$) and the optimal set $\mathcal{R}_i^{m^* - t^*}$ of $m^* - t^*$ replicas for $T_{u^*, r}$ (i.e., from $P(u^*, r, m^* - u^*)$) and return $\mathcal{R}_i^M = \mathcal{R}_i^t \cup \mathcal{R}_i^{m^* - t^*}$. Similar procedure is continued til all m^* locations are found out.

Algorithm 2 Algorithm for Finding Optimal Residence Set With Number Constraint

- 1: order the nodes of the tree; $\forall v$ compute m_v
 - 2: **for** $v = 1$ to n **do**
 - 3: $\forall u < v \leq m_v$ compute $C^i(T_r, v, u)$
 - 4: **end for**
 - 5: **for** $t = 1$ to M **do**
 - 6: **for** $v = 1$ to n **do**
 - 7: compute $C(v, t)$ using Equations (7), record $P(v, t)$
 - 8: **for** $u = v + 1$ to m_v **do**
 - 9: compute $C(u, v, t)$ using Equations (8), record $P(u, v, t)$
 - 10: **end for**
 - 11: **end for**
 - 12: **end for**
 - 13: $m^* = \underset{1 \leq t \leq M}{\arg \max} C(1, t)$; breaking ties in favor of smaller t
 - 14: compute \mathcal{R}_i^M recursively
-

Let P be the *path length* of tree T_r , which is defined as the sum over T_r of the number of ancestors of each node. We have the following theorem:

Theorem 2 *Algorithm 2 is correct and has a complexity of $O(NPM^2)$.*

Proof: The proof is omitted, see [14] for details. \square

4.2 Reducing Algorithm Complexity

As shown in Theorem 2, Algorithm 2 has a time complexity of $O(NPM^2)$. This is significant for a large network. In this subsection, we present mechanisms to reduce its complexity, which is motivated by the following observation.

Lemma 2 *Suppose that the optimal residence set output by Algorithm 1 is \mathcal{R}_i when the number of replicas could be arbitrary, and the optimal residence set output*

by Algorithm 2 is \mathcal{R}_i^t when a maximum of t replicas are allowed. If $t < |\mathcal{R}_i|$, then we must have $\mathcal{R}_i^t \subset \mathcal{R}_i$.

Proof: Suppose some nodes in \mathcal{R}_i^t do not belong to \mathcal{R}_i . Then there exists at least one node u such that $u \notin \mathcal{R}_i$ and u is a leaf node in \mathcal{R}_i^t . This is because otherwise any leaf node in \mathcal{R}_i^t would belong to \mathcal{R}_i , which implies $\mathcal{R}_i^t \subset \mathcal{R}_i$, a contradiction.

Since $u \notin \mathcal{R}_i$, according to Property 1, $\lambda_{u,i}^t \leq \alpha(\mu_{r,i}^t - \mu_{u,i}^t)$. Let's first consider the case of $\lambda_{u,i}^t < \alpha(\mu_{r,i}^t - \mu_{u,i}^t)$. As u is a leaf node in \mathcal{R}_i^t , suppose that u 's parent replica is v , and we have $cost'(T_r, \mathcal{R}_i^t) = cost'(T_r, \mathcal{R}_i^t - \{u\}) + d(u, v)\lambda_{u,i}^t - d(u, v)\alpha(\mu_{r,i}^t - \mu_{u,i}^t) < cost'(T_r, \mathcal{R}_i^t - \{u\})$. Thus, if we remove u , the cost would be greater. This means that the optimal residence set for a maximum of t replicas would be $\mathcal{R}_i^t - \{u\}$ rather than \mathcal{R}_i^t , a contradiction. Now consider the case of $\lambda_{u,i}^t = \alpha(\mu_{r,i}^t - \mu_{u,i}^t)$. Similarly, we can obtain $cost'(T_r, \mathcal{R}_i^t) = cost'(T_r, \mathcal{R}_i^t - \{u\})$. Since Algorithm 2 breaks ties by favoring the residence set with a smaller size, it will output $\mathcal{R}_i^t - \{u\}$ rather than \mathcal{R}_i^t , again, a contradiction. \square

From the above lemma, an alternative way to obtain \mathcal{R}_i^M is as follows. We first find out \mathcal{R}_i . If $M \geq |\mathcal{R}_i|$ then $\mathcal{R}_i^M = \mathcal{R}_i$. Otherwise, we run the algorithm over the subtree T_r' consisting of each node in \mathcal{R}_i . In T_r' , let $CH(v)$ be the children set of node v , the read rate for each node $v \in T_r'$ is adjusted as the rate for itself plus the aggregate rates from the subtrees rooted at the nodes that are v 's children but do not belong to \mathcal{R}_i . The write rate at each node $v \in T_r'$ can be adjusted in a similar manner. The detailed algorithm is shown in Algorithm 3. Let $N_o = |\mathcal{R}_i|$ and P_o be the path length in the subtree T_r' , the time complexity of this algorithm is $O(N)$ for $M \geq N_o$, and $O(N + N_o P_o M^2)$ for $M < N_o$. When $M \geq N_o$ or $N_o \ll N$, the complexity can be reduced significantly (i.e., from $O(NPM^2)$ to $O(N)$). Thus, we have the following theorem:

Theorem 3 *Algorithm 3 has a complexity of $O(N)$ for $M \geq N_o$ and $O(N + N_o P_o M^2)$ for $M < N_o$.*

5 Numerical Results

As shown in Section 3, the optimal solution for the problem without replica number constraint is $O(N)$ and thus is trivial. In this section, we investigate the practical complexity of the optimal solution with replica number constraint (i.e., Algorithm 3) by simulation experiments. In the simulation, a variety of random tree topologies are generated. We use three parameters to control the generation of a tree: the total number of nodes (*TreeSize*), the maximum degree

Algorithm 3 A Low Complexity Version of Algorithm 2

- 1: run Algorithm 1 to obtain \mathcal{R}_i
- 2: **if** $M \geq |\mathcal{R}_i|$ **then**
- 3: output \mathcal{R}_i
- 4: **else**
- 5: construct a subtree T'_r consisting of \mathcal{R}_i
- 6: **for each** node v in T'_r **do**
- 7: $\lambda_{v,i} = \lambda_{v,i}^t - \sum_{u \in CH(v), u \in \mathcal{R}_i} \lambda_{v,i}^t$
- 8: $\mu_{v,i} = \mu_{v,i}^t - \sum_{u \in CH(v), u \in \mathcal{R}_i} \mu_{v,i}^t$
- 9: **end for**
- 10: **end if**
- 11: run Algorithm 2 over T'_r to obtain \mathcal{R}_i^M

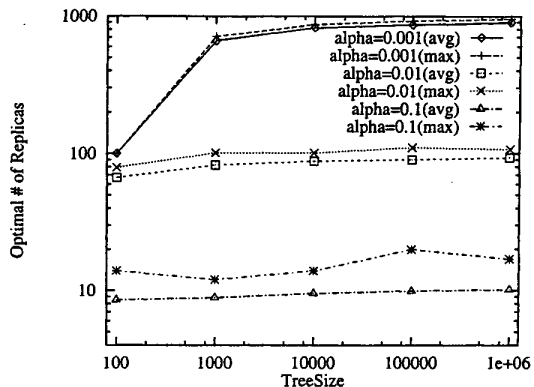
Parameter	Setting	Parameter	Setting
<i>TreeSize</i>	100 - 10^6	<i>MaxDegree</i>	5, 10
α	0.001 - 0.1	<i>M</i>	10 - 50
<i>MinDist</i>	1	<i>MaxDist</i>	20
<i>MinRdRate</i>	1	<i>MaxRdRate</i>	10
<i>MinWtRate</i>	1	<i>MaxWtRate</i>	10

Table 1: Default System Parameter Settings

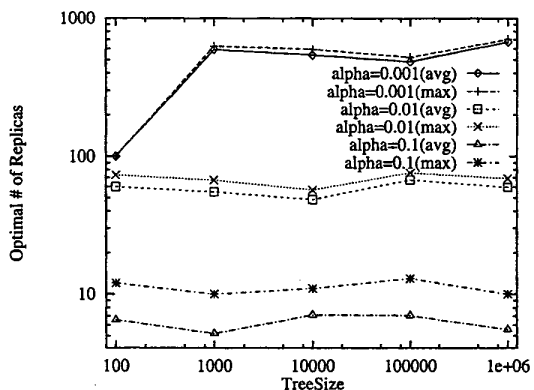
of a tree node (*MaxDegree*), and the distance range (*MinDist*, *MaxDist*) of a tree edge. A random tree is generated in a breadth-first manner, i.e., starting from the root node, we recursively create random children until the number of nodes specified is reached. An edge distance is randomly distributed between (*MinDist*, *MaxDist*). Every node v in a tree is associated with a read rate λ_v and a write rate μ_v . A read (write) rate is uniformly distributed between (*MinRdRate*, *MaxRdRate*) ((*MinWtRate*, *MaxWtRate*)). The default parameter settings are described in Table 1.

We have shown that the complexity of Algorithm 3 is $O(N + N_o P_o M^2)$ when $M < N_o$ in the last section. Since $P_o \leq N_o^2$, it is $O(N + N_o^3 M^2)$. Therefore, the complexity depends heavily on the optimal number of replicas (*ONR*), N_o , when no constraint is forced. We now show by simulation that N_o is normally small and the algorithm can solve the placement problems efficiently.

Figures 3(a) and 3(b) show the average and maximal *ONR* values we obtain when the tree size is varied from 100 to 10^6 . We can see that as *TreeSize* is enlarged rapidly, the *ONR* value increases very slowly or even decreases in some cases. Because putting more replicas in a network reduces read cost significantly as well as increasing update cost greatly, the



(a) *MaxDegree*=5



(b) *MaxDegree*=10

Figure 3: The Optimal Number of Replicas for Various Tree Sizes

optimal point is a balance between these two costs. From Figure 3, it turns out that the *ONR* value is relatively small even for a low write/read ratio in a very large network. For example, for *TreeSize*= 10^6 and $\alpha = 0.001$, the average *ONR* is 898 for *MaxDegree*=5 and 673 for *MaxDegree*=10. Thus, this implies that Algorithm 3 has a very nice property, i.e., for certain write/read ratio, as the network size grows, the algorithm complexity is reduced from $O(N + N_o P_o M^2)$ towards $O(N)$ since N_o is almost fixed at a small value.

In Figure 3, the worst *ONR* value is 955 for the setting of *TreeSize*= 10^6 , $\alpha = 0.001$, and *MaxDegree*=5. We ran Algorithm 3 with this setting on an Ultra Sparc 2 machine with 256M memory. Table 2 shows the setup times, overall running times, and occupied

memory sizes of the algorithm when M is set to 10 to 50. The setup time is the time used to construct the random tree. As can be seen, the algorithm is very efficient and can solve the problems in just a few minutes.

M (max # replicas)	10	20	30	40	50
Setup time (s)	19	30	37	44	54
Overall time (s)	60	128	242	389	590
Memory size (Mbyte)	144	185	216	248	279

Table 2: Performance of Algorithm 3 for $TreeSize=10^6$, $\alpha = 0.001$, $MaxDegree=5$

6 Conclusion

Transparent data replication is gaining increasing research interest. In this paper, we have examined two replica placement problems for applications with both read and write operations. We proposed two efficient optimal placement algorithms for replicating data on the proxies with and without replica number constraint respectively. We observed in theory that for a tree network if there is no constraint on the number of replicas and the optimal replica placement is used, transparent data replication can be achieved without any penalty in the data transfer cost. We have also conducted a set of simulation experiments to study the practical complexity of the proposed solutions. Numerical results showed that the proposed solutions are very efficient and can solve the optimal placement problems for a network of one million nodes in just a few minutes on an Ultra Sparc 2 machine with 256 M memory.

Acknowledgments

The authors would like to thank Mr. Xueyan Tang for his valuable discussions.

References

- [1] I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electrocnic content. In *Proceedings of IEEE INFOCOM'2001*, April 2001.
- [2] Proxy Cache Comparison. Website at <http://www.web-caching.com/proxy-comparison.html>.
- [3] CISCO DistributedDirector. Website at http://www.cisco.com/warp/public/cc/pd/cxsr/dd/tech/dd_wp.htm.
- [4] L. Dowdy and D. Foster. Comparative models fo the file assignment problem. *ACM Computing Surveys*, 14(2):287–313, June 1982.
- [5] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, November 1979.
- [6] A. Heddaya and A. Mirdad. Webwave: Globally load balanced fully distributed caching of hot published documents. In *Proceedings of IEEE ICDCS'97*, pages 160–168, May 1997.
- [7] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. *IEEE/ACM Transactions on Networking*, 8(5):568–582, October 2000.
- [8] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohrawy. On the optimal placement of web proxies in the internet. In *Proceedings of IEEE INFOCOM'99*, pages 1282–1290, March 1999.
- [9] M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, second edition, 1999.
- [10] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *Proceedings of IEEE INFOCOM'2001*, April 2001.
- [11] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal. A dynamic object replication and migration protocol for an internet hosting service. In *Proceedings of IEEE ICDCS'99*, pages 101–113, June 1999.
- [12] A. Vigneron, L. Gao, M. J. Golin, G. F. Italiano, and B. Li. An algorithm for finding a k-median in a directed tree. *Information Processing Letter*, 74:81–88, 2000.
- [13] O. Wolfson and A. Milo. The multicast policy and its relationship to replicated data placement. *ACM Transactions on Database Systems*, 16(1):181–205, March 1991.
- [14] J. Xu, B. Li, and D. L. Lee. Placement problems for transparent data replication proxy services. Technical Report HKUST-CS01-06, Department of Computer Science, Hong Kong Univ. of Science and Technology, March 2001.