Placement Problems for Transparent Data Replication Proxy Services

Jianliang Xu, *Student Member, IEEE*, Bo Li, *Senior Member, IEEE*, and Dik Lun Lee

***Abstract*—Transparent data replication has been considered a promising technique for improving system performance for a large distributed network. In this paper, a hybrid transparent replication model is presented. We address the problems of replication proxy placement in the network and data replica placement on the installed proxies given that a maximum of M proxies are allowed. Both reads and writes are considered in these problems. The performance objective is to minimize the total data transfer cost. To address the placement problems, we first present the optimal solutions for a single object in a tree network without/with constraint on the number of replicas. Based on that, two schemes, namely, AGGregate Access *(AGGA)* and Weighted POPularity *(WPOP)*, are proposed for the replication proxy placement problem. An optimal solution is described for the replica placement problem. The performance of the proposed placement schemes is evaluated with a set of carefully designed simulation experiments over a wide range of system parameters. The results give us several helpful intuitions in deploying transparent replication proxies in a practical system.**

***Index Terms*—Caching, data placement, en route, hierarchical, performance evaluation, replication, transparent data access.**

I. Introduction

A. Transparent Data Replication Model

THE EXPLOSIVE growth of network-based computing is moving us toward an interconnected and distributed information environment. In such an environment, a data object can be retrieved (read) and updated (written) by various geographically distributed clients. Several applications are described below. Other examples, including stock quote services, distributed ticketing systems, cooperative authoring systems, etc., can be found elsewhere [1]–[3].

Distributed Sensor System: The sensor system takes the average temperature (or pollution, traffic, etc.) of an area. Each sensor periodically takes a sample at a fixed point in the area, and updates the average value according to the new sample value taken. Users may retrieve the average value at any location of the network.

Wireless Location Lookup: In a wireless cellular system, users are located in system-defined *zones*. A user's location is updated due to his movement from one zone to another, and is retrieved by other people who want to look up his current location from different areas.

Online Auction: In an online auction system, the server maintains the highest current bid for a number of items. Each participant acquires the up-to-date information for the item of his interest, and may update the highest bid by offering a higher price.

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]–[6]. In this paper, we call a computer/program that holds partial or full data replicas of the server a *replication proxy* (or simply *proxy*).

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]–[6]. Obviously, this approach incurs considerable management overheads in identifying the optimal replica before each request is served, whether such knowledge is maintained at the clients [4], [6] or obtained on-the-fly [7], [8]. Thus, it is not scalable to a large network. As a consequence, *transparent data access* has been advocated by both academic and industrial communities because of its low management overheads incurred [9]–[12].

In general, there are two basic approaches to transparent data access for replication proxy services: *en-route replication* [11], [12] and *hierarchical replication* [9], [13]. Suppose that the replication proxies are colocated with the routers in the network. We denote requests that require interception by the routers as *router-aware* requests in order to distinguish them from *normal* requests.[1] In the en-route model, each client request is represented by a router-aware request. When it arrives at a router, the router intercepts the request. If a replica of the requested data is found on the local proxy, the router will service the request. Otherwise, it forwards the request toward the server along the regular routing path. In the hierarchical model, proxies are organized in a hierarchical manner such that the server resides at the top of the hierarchy and every proxy knows the location of its parent proxy. When a request, represented by a normal request, is issued by the client, it is routed to a predetermined proxy. A proxy either services the request if a replica of the requested data is stored locally or otherwise forwards it to the parent.

Manuscript received May 1, 2001; revised January 3, 2002. This work was supported in part by the Research Grant Council, Hong Kong SAR, China, under Grant HKUST-6154/98E, Grant HKUST-6163/00E, and Grant HKUST-6241/00E. This paper was presented in part at the 21st IEEE International Performance, Computing, and Communications Conference (IPCCC'02), Phoenix, AZ, April 3–5, 2002.

The authors are with the Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong (e-mail: xujl@cs.ust.hk; bli@cs.ust.hk; dlee@cs.ust.hk).

Publisher Item Identifier 10.1109/JSAC.2002.802068.

[1]In a typical implementation, router-aware requests can be distinguished from normal requests by the destination addresses and/or ports using $L4$ switches.

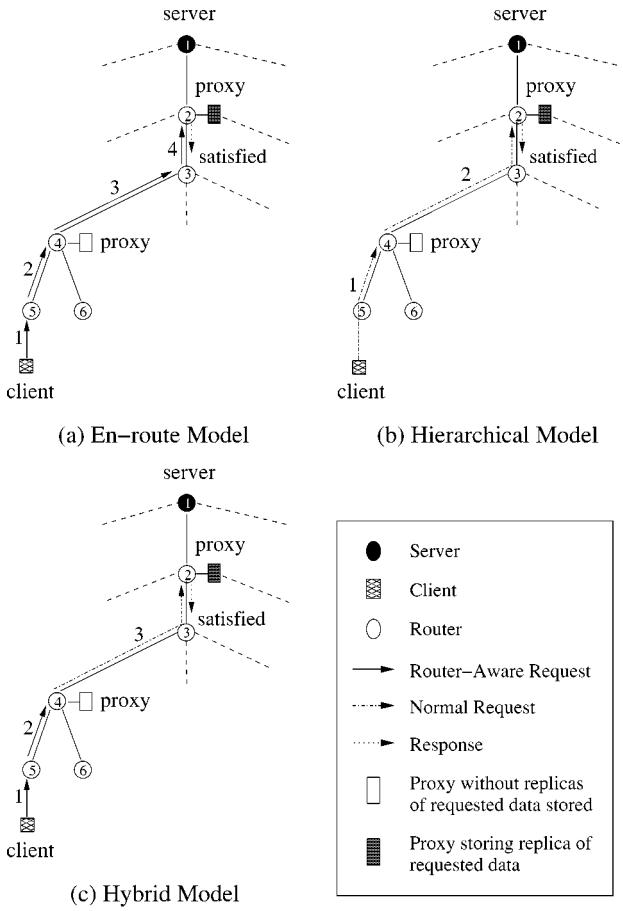


Fig. 1. Request flows under three transparent replication models.

In this paper, we present a hybrid model that combines the en-route model and the hierarchical model. In the hybrid model, proxies are organized in a hierarchical manner as in the hierarchical model but each client request is represented by a router-aware request as in the en-route model. A router-aware request is forwarded toward the server transparently until a replication proxy is reached. Then, the request is handled as in the hierarchical model. Fig. 1 shows an example for different request flows under these three models. As can be observed, compared with the en-route model, the hybrid model improves the loads for routers 2 and 3 since they do not need to intercept the request. Moreover, the hybrid model eliminates the need to configure the serving proxy for each client, which is required in the hierarchical model. We will explain the hybrid model in detail in Section III.

B. Placement Problems in Transparent Data Replication

The proxy and replica placement problems are two major issues of data replication. They have not been investigated extensively thus far for transparent data access. As will be discussed in Section II, there are three major limitations in the existing work. First, only read operations were considered, whereas many applications such as those presented earlier require write operations. Second, the problem of proxy placement in the network and the problem of data replica placement on the installed proxies were not well distinguished. This is partly because the previous work only considered read-only applications, in which putting more replicas on the installed proxies does not hurt the performance as long as storage cost is not a concern. However, for read/write applications, as we will show in Section VII-C, introducing replicas blindly could significantly deteriorate the write cost and, hence, the overall performance. Third, while previous studies were devoted to the development of optimal placement algorithms, performance evaluation of transparent replication strategies reported in the literature is rather limited.

This paper aims at filling the void mentioned above. The performance objective is to minimize the total data transfer cost in the network, which can be interpreted as hop counts, link cost, delay, etc. We consider the following placement problems for read/write applications. Given that a maximum of M proxies are allowed,[2] we want to find out how many replication proxies are needed, where to install them, and the placement of the replicas on the installed proxies. These two placement problems are related in that data replicas can only be placed on the installed proxies and that the placement of the proxies can be affected by the data replicas residing on them.

To study these placement problems, we model the network as a weighted physical tree with the server at the root. By considering tree-based topologies, we can focus on a target server and represent its access traffic by *a tree structure*, which is best for capturing the long-term traffic behavior [12], [14]. This is also consistent with the typical assumption used in the literature [6], [12], [13], [15].

Observing the complexity of solving the placement problems in one single step, we split them into two subproblems: the problem of proxy placement (hereinafter called *PP*) in the network and the problem of replica placement (hereinafter called *RP*) on the installed proxies. For the PP problem, two schemes, *AGGregate Access* (*AGGA*) and *Weighted POPularity* (*WPOP*), are proposed. The AGGA scheme returns the optimal solution when different nodes have the same access distribution over the data objects. For the RP problem, a linear optimal algorithm is presented.

We conduct a series of simulation experiments to evaluate the performance of the proposed placement schemes. We run the experiments over a wide range of system parameters so that the results can apply to a wide variety of applications. Through careful and thorough examinations, we hope to provide some helpful intuitions in deploying transparent replication proxies in a practical system.

C. Contributions and Organization of the Paper

The main contributions of this paper are summarized as follows.

- The placement problems for read/write applications under the transparent data replication model are formulated for the first time.
- A linear algorithm is presented to determine the optimal replica placement for a single object in a tree network.

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

- A low-complexity polynomial algorithm is developed to determine the optimal replica placement for a single object with a maximum of M replicas in a tree network.
- Two schemes, AGGA and WPOP, are proposed to solve the proxy placement problem in a tree network. AGGA obtains the optimal solution for a homogeneous case.
- It is 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 access can be achieved without any penalty on the data transfer cost.
- Extensive performance evaluation over a wide range of system parameters is carried out, from which a number of insightful observations are obtained for deploying transparent replication proxies in a practical system.

The rest of this paper is organized as follows. Section II reviews the related work. The hybrid transparent replication model is presented in Section III. The optimization problem of minimizing the total data transfer cost is formulated in Section IV. Section V describes the optimal solutions to two fundamental placement problems concerning a single object, based on which Section VI proposes two schemes, AGGA and WPOP, for the PP problem and the optimal algorithm for the RP problem. Performance evaluation is shown in Section VII. Finally, the paper is concluded in Section VIII.

II. Related Work

Data replication and caching are two common techniques to improve system performance [6], [8], [12]. Data replication can be viewed as a special kind of "push-based" caching. Compared with general caching, replication can take advantage of multicast for update dissemination [16]. Further, replication allows elegant placement schemes to be developed based on the knowledge of midterm/long-term data access patterns, whereas general caching relies on short-term temporal locality of client requests. In this paper, we only consider data replication.

The problem of replica placement in a communication network has been extensively studied in the area of file allocation problem (FAP) [4] and distributed database allocation problem (DAP) [5]. With different optimization objectives, such as communication cost alone or with other parameters such as storage constraint and load balancing requirement, the optimal placement problems were translated to a set of *nonlinear integer 0–1 programming* problems [4], [5]. However, solutions for these problems have been proven to be *NP-complete* [17]. Thus, various heuristic techniques have been adopted to solve the FAP and DAP problems. Examples are the knapsack solution [18], branch-and-bound [19], and network flow algorithms [20].

Wolfson *et al.* discussed the replica placement problem when the multicast write policy is employed [6]. Optimal placement solutions were proposed for *unweighted* clique, *logical* tree, and ring networks in [6]. In [1], Shivakumar *et al.* proposed to replicate user profiles based on a network flow solution in mobile environments.

A common problem of these early studies is that they are based on models which require read/write operations to be aware of the replicas' locations. Recently, transparent data replication has been proposed as a promising solution to reduce the management overheads of the previous models [9], [10], [12]. *Hierarchical replication* [9], [13] and *en-route replication* [11], [12] are two typical solutions, where a request is serviced by the nearest replica on the path toward the server. The disadvantage with transparent replication is that it may lengthen the serving paths. Fortunately, from the results presented in [21], only a slight penalty on the data transfer cost was observed. In this paper, we further show that when optimal placement was adopted, without any constraint on the number of replicas the cost will not deteriorate at all in a tree network. For en-route access, in a typical implementation [10] *router-aware* requests are intercepted and diverted to an appropriate proxy if $L4$ switches are used. With the support of various commercial products such as Cisco's CacheEngine, interception operations are expected to impose insignificant loading on the routers [8], [10], [12]. The hybrid model proposed in this paper can offload the tasks to be performed by the routers and, therefore, further reducing the loading on the routers.

The placement problems for transparent data replication have received little attention so far. Since accesses in transparent replication are *directed*, classic graph theoretic approaches such as the p-median problem and the facility location problem [22], where *undirected* accesses were assumed, are not applicable. In [15], Li *et al.* proposed an $O(N^3M^2)$ solution to optimally allocate M Web replication servers in a tree network. Vigneron *et al.* improved the complexity of the algorithm to $O(PM^2)$ [24]. A similar solution was reported for the optimal placement of Web caches by Krishnan *et al.* [12]. However, the problem of replica placement on Web servers/caches was not discussed in [12] and [15]. In [13], Cidon *et al.* presented an $O(NH)$ algorithm for electronic content allocation on hierarchical servers when storage cost was considered. The server placement problem in a tree network was also briefly discussed. However, there was no constraint forced on the number of replication servers. With some budget scheme, we believe that a model limiting the number of replication proxies would be more reasonable. In addition, all these previous studies considered read operations only, thus, limiting their applications in the real world.

A related issue to data replication is the *replicas control protocol* [5]. Considerable studies have been done on this subject, such as *primary copy*, *quorum consensus*, and *lazy propagation* [5], [23]. Interested reader is referred to [2], [5], and [23] for details.

III. Hybrid Transparent Replication Model

This section describes in detail the hybrid transparent 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. The core of the network consists of a set of inter-connected *router*s, each of which is responsible for a local domain or a site. As mentioned in Section I, replication proxies are placed alongside with some selected routers and are further organized in a hierarchical manner with the server at the top of the hierarchy. The *replicated node*s of a data object are nodes that have proxies installed and replicas of the object are stored on the proxies. In contrast, the *nonreplicated node*s of a data object are nodes that have no replica stored locally. We assume that each object has a globally unique identifier (e.g., a

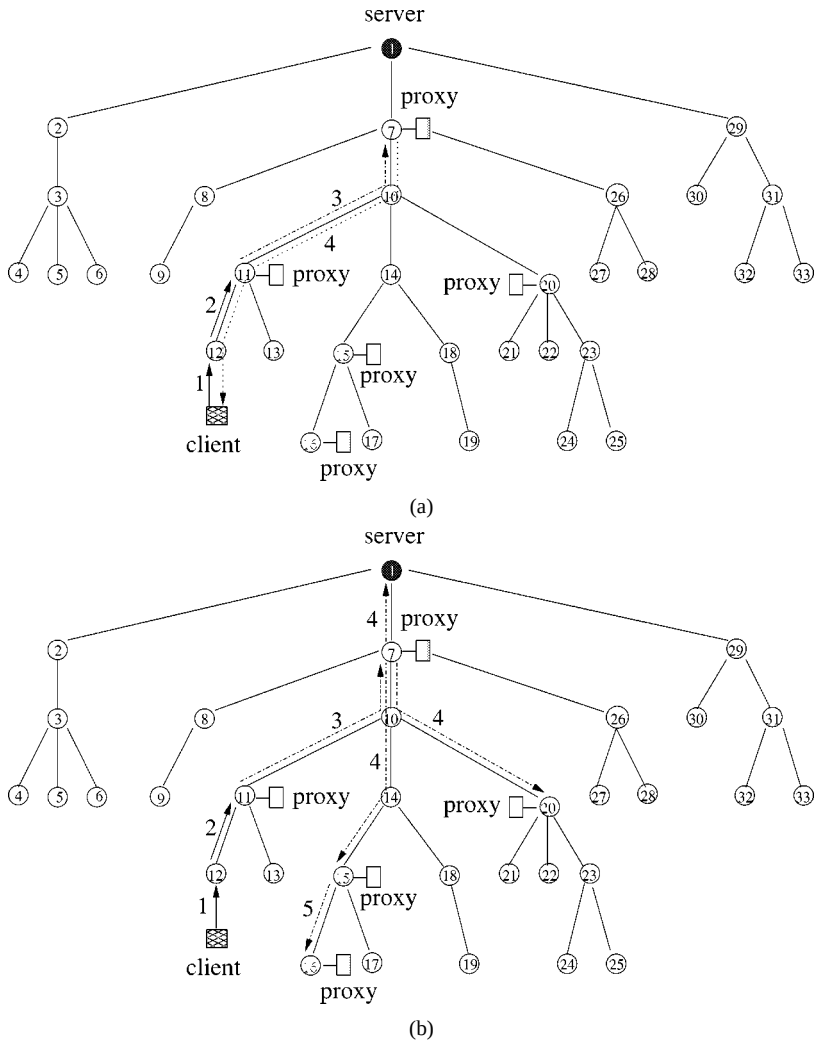


Fig. 2. Message flows of hybrid transparent replication model.

table name plus a key value or a *URL*) and that a replica maintains the replicated data as well as the locations of the other replicas. In the following paragraphs, we specify the read and write methods. For simplicity, we assume that a read (write) operation consists of a single retrieval (update) request.

A client initiates a *read* operation by sending to the server a *router-aware retrieval request* containing the requested object's id. When a router-aware retrieval request reaches a router, it is intercepted by the router. If a replication proxy of the server is found alongside with the router, the request is directed to the proxy. Otherwise, the router forwards the request to its parent router toward the server. When a proxy receives a retrieval request, if a replica of the requested data is found locally, the proxy satisfies the request by returning the replica. Otherwise, it sends a *normal retrieval request* to its parent proxy to request the data.

Since the multicast write policy can save the data transfer cost significantly [6], we adopt this policy for write operations in the model. Each *write* operation is initiated by a *router-aware update request* sent from a client node toward the server. An update request contains the id of the object to be updated and the new value for the update operation. As in read operations, the routers forwards the router-aware request toward the server until a replication proxy is found. If the object to be updated is found on the proxy, it retrieves the locations of the other replicas and starts a multicast write. Otherwise, the proxy sends a *normal update request* to its parent proxy and the parent performs the same operation. In this paper, we assume that multicast writes are implemented at the application level, as this can be achieved even if the underlying network infrastructure does not support multicast. As we will see in Section V-A, under optimal placement the replicas of an object are *connected* among the proxies. Thus, an application-level multicast write can be implemented easily in the following manner. When a replicated node receives an update request, it updates the object and further forward the update with a *normal update request* to its parent and children proxies (except the sender) that are also replicated nodes of the object. The following example illustrates the read/write operations more clearly.

Example 1: As illustrated in Fig. 2, node 1 is the data server and five replication proxies are placed at nodes 7, 11, 15, 16, and 20. Suppose that a data object is replicated on the proxies at nodes 7, 15, 16, and 20. For a read operation accessing this object from router 12's domain, the retrieval request is accepted by the proxy at node 11, and finally satisfied by the proxy at node 7 [see Fig. 2(a)]. For an update operation on this object from router 12's domain, as in the read case, the update request

is serviced by the proxy at node 7, i.e., a multicast write starts at node 7 [see Fig. 2(b)]. Since its parent proxy (i.e., node 1) and two of its children proxies (i.e., nodes 15 and 20) are also replicated nodes of the object to be updated, three individual update request are sent to them. After the proxy at node 15 receives the update request, it further forwards it to its child proxy at node 16.

IV. 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 proxies and 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 methods to obtain the optimal placement schemes. To do so, we formulate the placement problems formally in this section. In the next two sections, we will present several solutions to the formulated problems.

As stated in Section I, the network topology in this paper is modeled as a physical tree with the server at the root. Consider 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 router. Each edge corresponds to a physical link. A node $v \in T_r$ is ordered in preorder traversal (see Fig. 2). For simplicity, we hereinafter identify each node of T_r with its preorder numbering. Assume that the data server contains K objects. For each object i ($1 \le i \le K$), 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 $v \in T_r$, T_v denotes the subtree of T_r rooted at v. Further, let $\lambda^t_{v,i}$ be the total read rate and $\mu^t_{v,i}$ the total write rate generated from subtree T_v for object i, i.e., $\lambda^t_{v,i} = \sum_{u \in T_v} \lambda_{u,i}$, $\mu^t_{v,i} = \sum_{u \in T_v} \mu_{u,i}$. 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 one. 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 set of proxies $\mathcal{P} \subseteq T_r$ (including the server) and a residence set of $\mathcal{R}_i \subseteq \mathcal{P}$ of object i, we derive in the following paragraphs the total data transfer cost for object i.

Let us first consider the case where no replication proxies are installed 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:

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

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

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 a replica of object i and no other replicas have been installed in T_v. The incremental data transfer cost due to the installation of v can be expressed in the following formula:

$$cost^i(T_r, \mathcal{R}_i, v) = -\left(\lambda^t_{v,i} - \alpha\left(\mu^t_{r,i} - \mu^t_{v,i}\right)\right) \cdot d(v, c(v, \mathcal{R}_i)). \quad (2)$$

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 is given by

$$cost(T_r, \mathcal{R}_i) = cost(T_r, \{r\}) - \sum_{v \in \mathcal{R}_i - \{r\}} cost^i(T_r, \mathcal{R}_i, v)$$

where the first term corresponds to the total cost when no replica is installed 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 $cost(T_r, \mathcal{R}_i)$ is fixed. As such, we only need to consider the problem of optimizing the following cost instead of $cost(T_r, \mathcal{R}_i)$:

$$cost'(T_r, \mathcal{R}_i) = \sum_{v \in \mathcal{R}_i - \{r\}} \left(\lambda^t_{v,i} - \alpha\left(\mu^t_{r,i} - \mu^t_{v,i}\right)\right) \cdot d(v, c(v, \mathcal{R}_i)). \quad (3)$$

This cost physically means the improved data transfer cost for object i due to the installation of a set of replicas $\mathcal{R}_i - \{r\}$. Consequently, the overall data transfer cost for all K objects that we are concerned with is $\sum_{i=1}^{K} cost'(T_r, \mathcal{R}_i)$. Thus, the optimization problem can be formally defined as follows.

OP: *Given a tree network $T_r = (V, E)$, M, and α, find subsets $\mathcal{P}$ and $\mathcal{R}_i$ for all $1 \le i \le K$, $\mathcal{P} \subseteq V$, $r \in \mathcal{P}$, $|\mathcal{P}| \le M$, and $\mathcal{R}_i \subseteq \mathcal{P}$, which maximize $\sum_{i=1}^{K} cost'(T_r, \mathcal{R}_i)$, where $cost'(T_r, \mathcal{R}_i)$ is given in (3).*

V. Two Fundamental Placement Problems for Transparent Replication

Before describing the proposed schemes for the OP problem, we discuss two fundamental placement problems for transparent replication, i.e., the optimal replica placement problems for a single object without/with constraint on the number of replicas in a tree network. The proposed schemes for the OP problem presented in the next section are based on the optimal solutions to these two fundamental problems.

A. Optimal Placement Without Replica Number Constraint

This section considers the replica placement problem for object i in a tree network where the number of replicas could

[4]Note that there are other ways to formulate the write cost. We use this one because it is consistent with our dynamic programming algorithm in Section V-B.

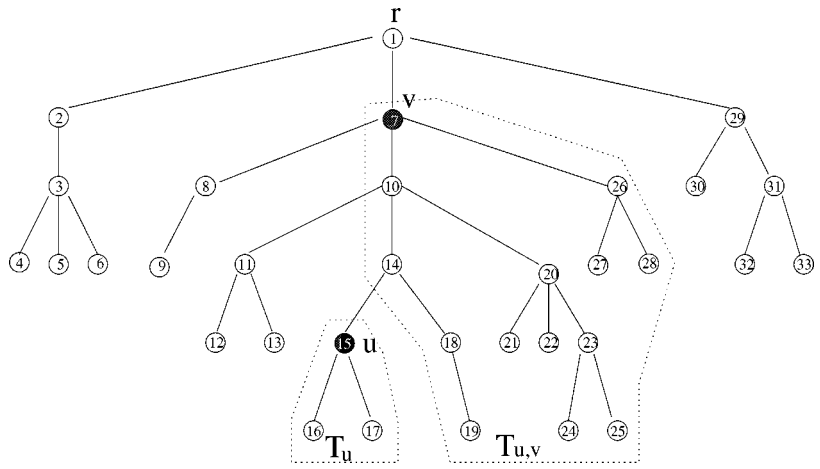


Fig. 3. Definition of $T_{u,v}$.

be arbitrary. As can be observed, this is an abstraction of the replica placement problem on a set of installed hierarchical proxies, which naturally form a tree structure. The proposed WPOP scheme for the PP problem (Section VI-A) is also based on the solution to this problem. In addition, as we will see later in Section V-B2, an efficient solution to this problem helps to reduce the complexity of the optimal solution to 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 cost$'(T_r, \mathcal{R}_i)$ given in (3).*

In the following, we present an $O(N)$ algorithm for the above problem, followed by the proof of its correctness. The algorithm is developed in the spirit of [6], where an *unweighted logical* tree was considered for a similar problem.

Algorithm 1 Algorithm for finding optimal residence set without replica number constraint.

```
1: add the root r to the optimal resi-
   dence set R_i
2: add the children of the root to a can-
   didate set C_i
3: while C_i is not empty do
4:   remove the first node v from C_i
5:   if λ^t_{v,i} > α(μ^t_{r,i} − μ^t_{v,i}) then
6:     add v to R_i
7:     add v's children to C_i
8:   end if
9: end while
10: output R_i
```

The proposed 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 [6, Lemma 4.2.0] for the full proof. Although [6, Lemma 4.2.0] was presented in the context of an *unweighted logical* tree, the proof still holds for our *weighted physical* tree. ■

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

Proof: See Appendix I. ■

It is also easy to obtain the following property.

Property 1: Suppose that 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 $\lambda^t_{v,i} > \alpha(\mu^t_{r,i} - \mu^t_{v,i})$ if $v \in \mathcal{R}_i$; and $\lambda^t_{v,i} \leq \alpha(\mu^t_{r,i} - \mu^t_{v,i})$ otherwise.

B. Optimal Data Replication 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. The proposed AGGA scheme for the PP problem (Section VI-A) is based on the solution to this problem. Formally, the problem we consider here is defined as follows.

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 (3).*

In the following, Section V-B1 develops a dynamic programming algorithm to solve this problem. The algorithm is extended from a previous paper [15] and has a complexity of $O(NPM^2)$. In Section V-B2, we present several techniques to reduce its complexity.

1) Dynamic Programming Algorithm: We first define the notations used in the algorithm. Recall that each node of T_r is identified with its preorder 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 Fig. 3, $m_7 = 28$. 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

$$T_{u,v} = \{x \in T_v : m_u < x \le m_v \quad \text{or} \quad x \in s_{u,v} - \{u\}\}. \quad (4)$$

As shown in Fig. 3, 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.

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

$$C^i(T_r, v, u) = \left(\lambda_{u,i}^t - \alpha\left(\mu_{r,i}^t - \mu_{u,i}^t\right)\right) d(u, v). \quad (5)$$

If $\mathcal{R}_i^{t\prime}$ is any residence set of object i such that $v, u \in \mathcal{R}_i^{t\prime}$, and no replicas of $\mathcal{R}_i^{t\prime}$ are in $s_{u,v} - \{u, v\}$, then we have $c(u, \mathcal{R}_i^{t\prime}) = v$. This means that $C^i(T_r, v, u)$ is the contribution to $cost'(T_r, \mathcal{R}_i^{t\prime})$ [see Equation (3)] by installation of u.

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

$$C(v, t) = \begin{cases} 0, & \text{if } t = 1 \\ \max\limits_{v < u \le m_v, 0 < t' < t} (C(u, t') \\ \quad + C(u, v, t - t') \\ \quad + C^i(T_r, v, u)), & \text{if } t > 1 \end{cases} \quad (6)$$

and

$$C(u, v, t) = \begin{cases} 0, & \text{if } t = 1; \\ \max\limits_{m_u < x \le m_v, 0 < t' < t} (C(x, t') \\ \quad + C(x, v, t - t') \\ \quad + C^i(T_r, v, x)), & \text{if } t > 1. \end{cases} \quad (7)$$

In (6), set $P(v, t) = (u, t')$, where $v < u \le m_v$, $0 < t' < t$ are the values that maximize the expression, and in (7) set $P(u, v, t) = (x, t')$, where $m_u < x \le m_v$, $0 < t' < t$ 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 $1 \le t \le M$ 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 $\mathcal{R}_i'$ of t^* replicas for T_{u^*} [i.e., from $P(u^*, t^*)$] and the optimal set $\mathcal{R}_i''$ of $m^* - t^*$ replicas for $T_{u^*,r}$ [i.e., from $P(u^*, r, m^* - u^*)$] and return $\mathcal{R}_i^M = \mathcal{R}_i' \cup \mathcal{R}_i''$. This procedure does not stop until all m^* locations are found out.

Algorithm 2 Algorithm for finding optimal residence set with replica number constraint.

```
1:  order the nodes of the tree; ∀v compute m_v
2:  for v = 1 to n do
3:    ∀v < u ≤ 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 equation (6), record P(v, t)
8:      for u = v + 1 to m_v do
9:        compute C(u, v, t) using equation (7), record P(u, v, t)
10:     end for
11:   end for
12: end for
13: m* = arg_{1≤t≤M} max C(1, t)
14: compute 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: See Appendix II. ∎

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

Theorem 3: Suppose that the optimal residence set output by Algorithm 1 is $\mathcal{R}_i$ when the number of replicas could be arbitrary and that 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 $\mathcal{R}_i^t \subset \mathcal{R}_i$ from Lemma 1, which is a contradiction.

Since $u \notin \mathcal{R}_i$, according to Property 1, $\lambda_{u,i}^t \le \alpha(\mu_{r,i}^t - \mu_{u,i}^t)$. 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 $\mathcal{R}_i^t$, which is a contradiction. Now consider

the case of $\lambda^t_{u,i} = \alpha(\mu^t_{r,i} - \mu^t_{u,i})$. Similarly, we can obtain $cost'(T_r, \mathcal{R}^t_i) = cost'(T_r, \mathcal{R}^t_i - \{u\})$. Since Algorithm 2 breaks ties by favoring the residence set with a smaller size, it will output $\mathcal{R}^t_i - \{u\}$ rather than $\mathcal{R}^t_i$, which is again a contradiction. ■

From the above theorem, an alternative way to obtain $\mathcal{R}^M_i$ is as follows. We first find out $\mathcal{R}_i$. If $M \geq |\mathcal{R}_i|$ then $\mathcal{R}^M_i = \mathcal{R}_i$. 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 $O(N)$]. As we will see in Section VII-B, this algorithm can solve the placement problem with $\alpha \geq 0.001$ for a network of one million nodes in a few minutes on an Ultra Sparc 2 machine with 256-M memory.

Algorithm 3 A low complexity version of Algorithm 2.

```
1:  run Algorithm 1 to obtain R_i
2:  if M ≥ |R_i| then
3:    output R_i
4:  else
5:    construct a subtree T'_r consisting of
    R_i
6:    for each node v in T'_r do
7:      λ_{v,i} = λ^t_{v,i} − Σ_{u∈CH(v), u∈R_i} λ^t_{u,i}
8:      μ_{v,i} = μ^t_{v,i} − Σ_{u∈CH(v), u∈R_i} μ^t_{u,i}
9:    end for
10:  end if
11:  run Algorithm 2 over T'_r to obtain R^M_i
```

VI. Proposed Schemes for the Optimization Problem

We are now back to the original OP optimization problem. As we can see, it is difficult to solve the OP problem in just one step. One possible way is to extend Algorithm 2 to take into account the combination of all K objects in every recurrence of the dynamic programming algorithm. Unfortunately, the complexity of this method is as high as $O(PM(NM)^K)$. Thus, we propose to split the OP problem into two sub-problems. First, we consider the placement problem for M replication proxies in Section VI-A. Second, we consider the problem of replica placement on M installed proxies in Section VI-B. We show that the proposed AGGA scheme can obtain the optimal solution for a homogeneous case.

A. Placement of Replication Proxies in the Network

This subsection proposes two schemes, namely AGGA and WPOP, for selecting placement locations for M proxies in the network. As their names suggest, the AGGA scheme makes placement decision based on the aggregate access patterns of each node, while the WPOP scheme selects placement locations based on the weighted popularity of each node.

AGGA: In this scheme, we collect the aggregate read and write rates over all objects for each node. Then, Algorithm 3 is executed with the aggregate access patterns as the input parameters to determine the optimal residence set $\mathcal{R}^M$. If $|\mathcal{R}^M| = M$, M replication proxies are placed according to $\mathcal{R}^M$. Otherwise if $|\mathcal{R}^M| < M$, the locations of $|\mathcal{R}^M|$ proxies are selected according to $\mathcal{R}^M$. The remaining $M - |\mathcal{R}^M|$ placement locations are chosen using a greedy method. For each additional proxy, the node which would introduce the least incremental cost penalty is selected. Obviously, this scheme favors the scenario where access distributions over the objects on each node are similar. In a homogeneous case (i.e., access distributions over the objects on each node are the same), the optimization problem can be modeled as in the single object case. Thus, this scheme obtains the optimal solution by choosing $\mathcal{R}^M$. This scheme adds a complexity of $O(KN)$ to Algorithm 3

WPOP : This scheme executes Algorithm 1 for each object i, from which we can obtain $\mathcal{R}_i$ for all $1 \leq i \leq K$. Let $\lambda^t_{r,i}$ and $\mu^t_{r,i}$ be the total read and write requests, generated in the network for object i, respectively. The weight for object i is then defined as $\lambda^t_{r,i} + \alpha\mu^t_{r,i}$. For each node v, we maintain a $pop(v)$ factor as follows:

$$pop(v) = \sum_{i=1}^{K} \left(\lambda^t_{r,i} + \alpha\mu^t_{r,i}\right) \cdot uf(v, \mathcal{R}_i) \tag{8}$$

where

$$uf(v, \mathcal{R}_i) = \begin{cases} 1, & \text{if } v \in \mathcal{R}_i; \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Then the N potential nodes are sorted according to their $pop(v)$ values, and the replication proxies are placed at the top M nodes that have the highest $pop(v)$ values. Compared with the AGGA scheme, the WPOP scheme is expected to take into consideration the diversity of access patterns for different nodes. The complexity of this scheme is $O(KN + N\log N)$.

B. Optimal Placement of Replicas on the Proxies

As stated in Section I, we argue that data replicas should be placed on proxies selectively according to their access patterns. This section discusses the optimal placement of data replicas on M installed proxies. Since the proxies are organized in a hierarchical manner, they naturally form a tree structure. Therefore, the placement problem for each object i can be exactly modeled by the OP1 problem. Thus, Algorithm 1 can be used to determine the optimal residence set for each object. The whole procedure takes $O(KM)$ time.

We can make another important observation. Since Lemma 1 states that the optimal residence set of an object induces a connected subtree in 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 toward the server. It follows that under the optimal replica placement on hierarchical proxies the nearest replica to a client is always resides at a proxy upwards

the hierarchy. Thus, transparent data access can be achieved without any penalty in terms of data transfer cost.

VII. Performance Evaluation

In this section, the performance of the proposed placement schemes is evaluated with simulation experiments. The simulation model is described in Section VII-A. Section VII-B investigates the practical complexity of Algorithm 3. In Section VII-C, the effect of partial data replication on proxies is examined. In Section VII-D, we compare the proposed schemes with the baseline *NREP* scheme and the *RAND* scheme under various system parameters. Section VII-E evaluates the sensitivity of the proposed schemes to the inaccuracy of input data.

A. Experiment Setup

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 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. That is, starting from the root node, we recursively create a random number of 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 of λ_v and a write rate of μ_v. Two types of access models, *uniform* and *nonuniform*, are simulated. In *uniform access*, a read (write) rate is uniformly distributed between (*MinRdRate*, *MaxRdRate*) ((*MinWtRate*, *MaxWtRate*)). In *nonuniform access*, we consider two patterns, namely *Hot/Cold* and *Partial Update*. For these two nonuniform models, read/write rates are first generated as in the uniform access model. With a *Hot/Cold* model, for both read and write we randomly select *HnodeRatio* of the nodes each time and adjust their read (write) rates with values uniformly distributed between (*MinRdHotRate*, *MaxRdHotRate*) ((*MinWtHotRate*, *MaxWtHotRate*)). With a *Partial Update* model, we randomly select (1–*PnodeRatio*) of the nodes and set their write rates to zero. The default model is uniform access.

The data server is a collection of K data objects and is partitioned into disjointed data regions, each with *RegionSize* items. Data retrieval (update) pattern over the objects follows a *Zipf* distribution with a skewness parameter of θ_r (θ_w) [25]. When $\theta = 0$, the access pattern over the objects is uniform. The larger the θ value, the more skewed the access pattern. We assume that the probability of accessing any object within a data region is uniform, while access to the data regions follows the *Zipf* distribution. Thus, the *pdf*s over the objects are $d_{r,i} = (i/RegionSize)^{-\theta_r} / \sum_{j=1}^{K/RegionSize} j^{-\theta_r}$ and $d_{w,i} = (i/RegionSize)^{-\theta_w} / \sum_{j=1}^{K/RegionSize} j^{-\theta_w}$ $(1 \le i \le K)$ for retrieval and update access, respectively. To simulate access distributions for different nodes, two types of models are used: *homogeneous* and *heterogeneous*. For *homogeneous distribution*, we set $\lambda_{v,i} = d_{r,i}\lambda_v$ and $\mu_{v,i} = d_{w,i}\mu_v$. For *heterogeneous distribution*, we introduce an *offset* parameter. When generating a tree structure, the access distributions on two neighboring nodes are shifted by *offset* regions. Specifically, l_i is initialized to i, for each node v and object i, we set $\lambda_{v,i} = d_{r,l_i}\lambda_v$ and update $l_i = (l_i + OffsetRd * RegionSize) \bmod K$. A similar method is used to generate $\mu_{v,i}$ using an offset parameter of *OffsetWt*. The default pattern is heterogeneous distribution. The default parameter settings are described in Table I.

TABLE I
Default System Parameter Settings

Parameter	Setting	Parameter	Setting
TreeSize	100, 1000	*MaxDegree*	5, 10
α	0.001–0.1	M	20
MinDist	1	*MaxDist*	20
HnodeRatio	20%	*PnodeRatio*	10%
MinRdRate	1	*MinRdHotRate*	11
MinWtRate	1	*MinWtHotRate*	11
MaxRdRate	10	*MaxRdHotRate*	100
MaxWtRate	10	*MaxWtHotRate*	100
K	1000	*RegionSize*	50
θ_r	1.2	θ_w	0.4
OffsetRd	10	*OffsetWt*	2

TABLE II
Complexity of the Schemes Evaluated

	Full Replication	Partial Replication
NREP	$O(1)$	$O(1)$
RAND	$O(N)$	$O(N+KM)$
AGGA	$O(KN)$, $M \ge N_o$; $O(KN + N_oP_oM^2)$, $M < N_o$	$O(KN+KM)$, $M \ge N_o$; $O(KN + N_oP_oM^2 + KM)$, $M < N_o$
WPOP	$O(KN + NlogN)$	$O(KN + NlogN + KM)$

In the performance evaluation, in addition to the proposed AGGA and WPOP schemes, we also include the no replication (NREP) scheme as the baseline and a random (RAND) scheme for comparison.

NREP: No replication proxies are installed in the network; only the server contains the data objects.

RAND: This scheme randomly selects M nodes for placing the replication proxies. To improve performance, we execute the scheme over ten times and choose the random assignment that yields the lowest cost.

For AGGA, WPOP, and RAND, after choosing the placement locations of a maximum of M proxies, the optimal algorithm for replica placement on proxies can be used. We call this approach *partial replication*, compared with the *full replication* scheme in which the whole data are blindly replicated to every installed proxy. We summarize the complexity of the schemes in Table II.

To compare the performance of the schemes, we use *normalized cost* as the metric. The normalized cost of a scheme is defined as the ratio of the cost of the feasible solution found by the scheme to the cost of the NREP scheme. The smaller the normalized cost, the better the scheme. Each set of experiments are repeated hundreds of times, and we measure the *average*, *worst*, and *best* performance.

B. Practical Complexity of Algorithm 3

We have shown that the complexity of Algorithm 3 is $O(N + N_oP_oM^2)$ in the last section. Since $P_o \le N_o^2$, it is $O(N+N_o^3M^2)$. Therefore, the complexity depends heavily on the optimal number of replicas (*ONR*), N_o, when no constraint

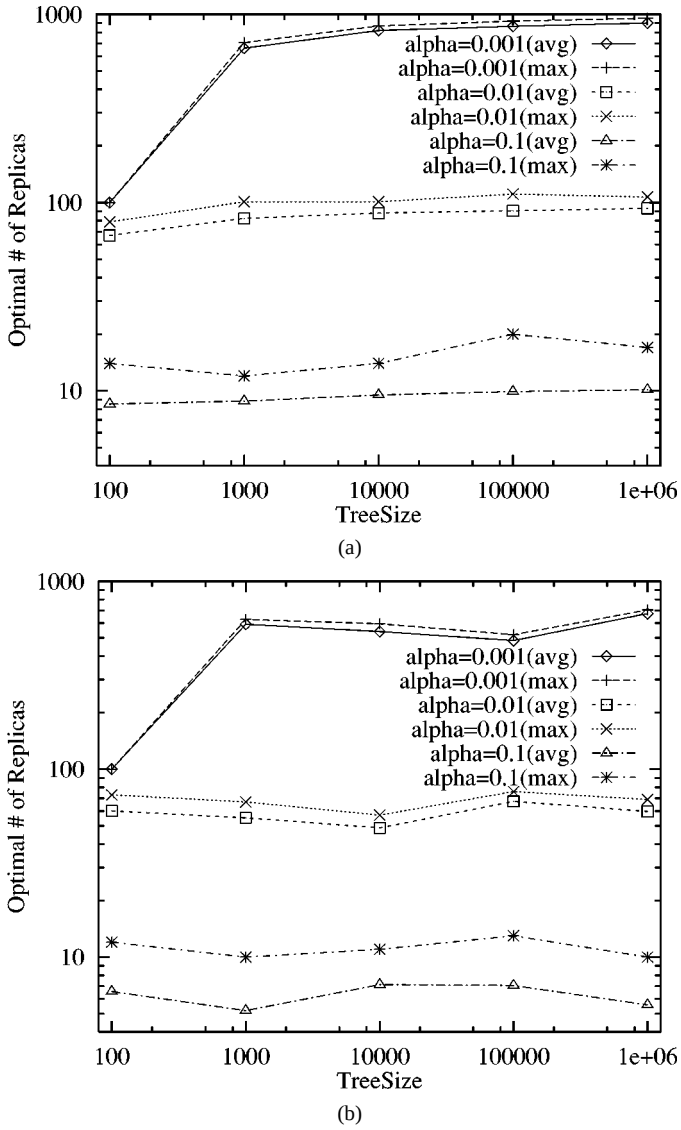


Fig. 4. The optimal number of replicas for various tree sizes. (a) *MaxDegree* = 5. (b) *MaxDegree* = 10.

is forced.[5] This subsection shows by simulation that N_o is normally small and the algorithm can solve the placement problems efficiently.

Fig. 4(a) and (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 optimal point is a balance between these two costs. From Fig. 4, 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, 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 $O(NPM^2)$ toward $O(N)$ since N_o is almost fixed at a small value.

[5]In the rest of this paper, *ONR* stands for the optimal number of replicas in the case without replica number constraint.

TABLE III
PERFORMANCE OF ALGORITHM 3 FOR *TreeSize* $= 10^6$, $\alpha = 0.001$, *MaxDegree* $= 5$

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

In Fig. 4, 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 256-M memory. Table III 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.

C. Effect of Partial Data Replication on Proxies

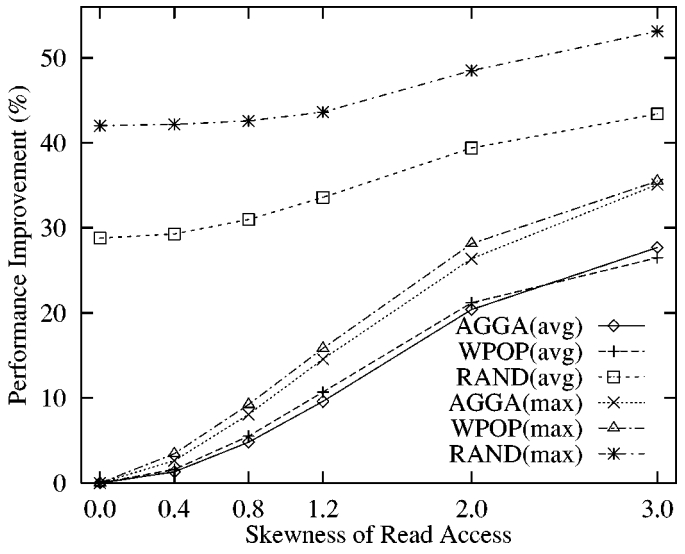
We have argued in the previous sections that data should be replicated on the installed proxies according to their data access patterns. In this subsection, we examine the effect of *partial replication* on the performance. To make a fair comparison, when the *full replication* approach is employed, for AGGA the set of nodes $\mathcal{R}_i^M$ are selected to install proxies; for WPOP we select the set of top $M' \leq M$ nodes that achieves the minimal data transfer cost.

In this set of experiments, we set θ_w to 0 and vary θ_r from 0 to 3. When θ_r is 0, different nodes have the same access distribution over the data objects. The larger the θ_r value, the more diverse the access distributions. The performance metric employed is the relative improvement of the partial replication approach over the full replication approach for each placement scheme. Fig. 5 shows the average improvement and the maximal improvement. We can see that the partial replication approach improves the performance significantly for $\theta_r > 0$. In particular, the performance improvement for the RAND scheme is the greatest, up to 52% for *TreeSize* $= 100$. This is because the RAND scheme determines the proxies' locations in an *ad-hoc* manner and, thus, there is more space to improve the performance. As θ_r increases, as expected more improvement is observed for all the schemes. When $\theta_r = 3$, the average improvement is about 15%–43%. This implies that partial replication is particularly important when access distributions over the objects for different nodes are observed very diverse.

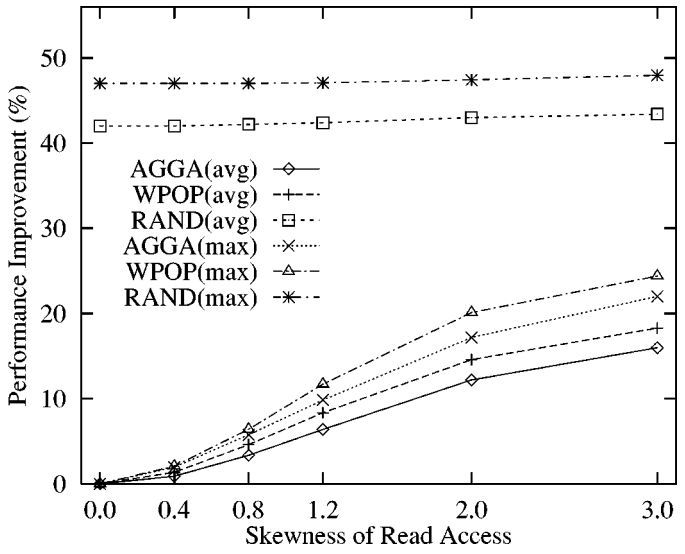
D. Performance Comparison of the Placement Schemes

In this section, we compare the performance of three placement schemes, namely AGGA, WPOP, and RAND, to that of NREP. The partial replication approach is employed for all these three schemes. As mentioned before, we use *normalized cost* as the metric in performance comparison. The results for homogeneous and heterogeneous access distributions are presented in Sections VII-D1 and VII-D2, respectively. The presented results are for uniform access, similar performance trends are obtained for nonuniform access (interested reader is referred to [27] for details).

1) Homogeneous Access Distribution: As discussed before, the AGGA scheme obtains the optimal solution for the

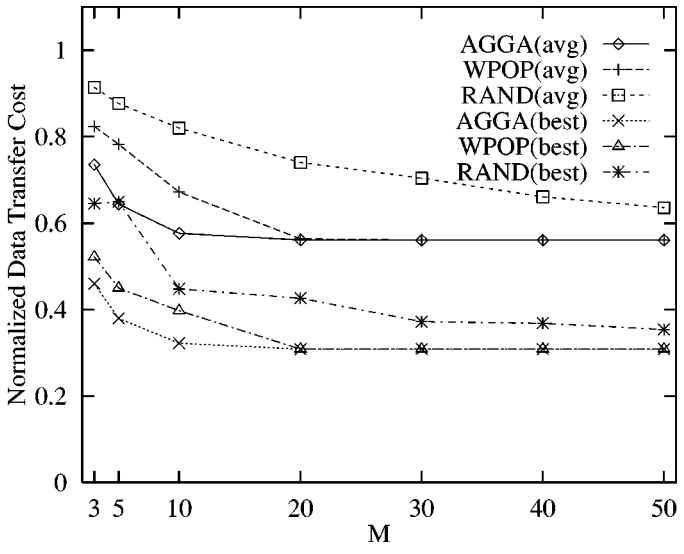


(a)

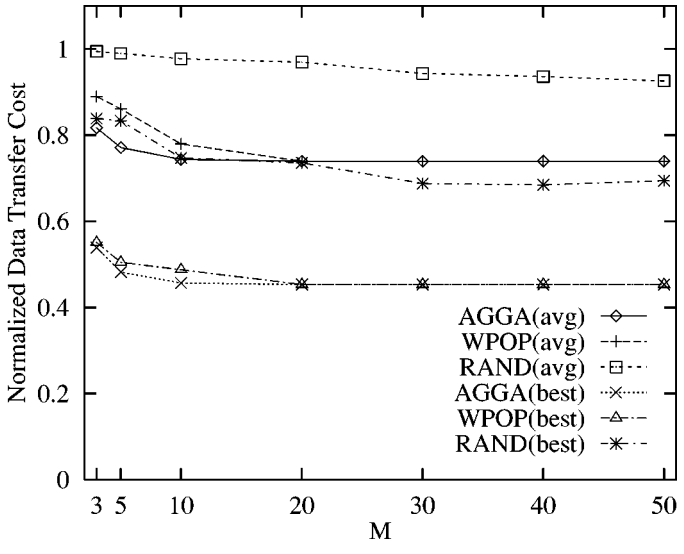


(b)

Fig. 5. Relative performance of partial replication versus full replication.



(a)



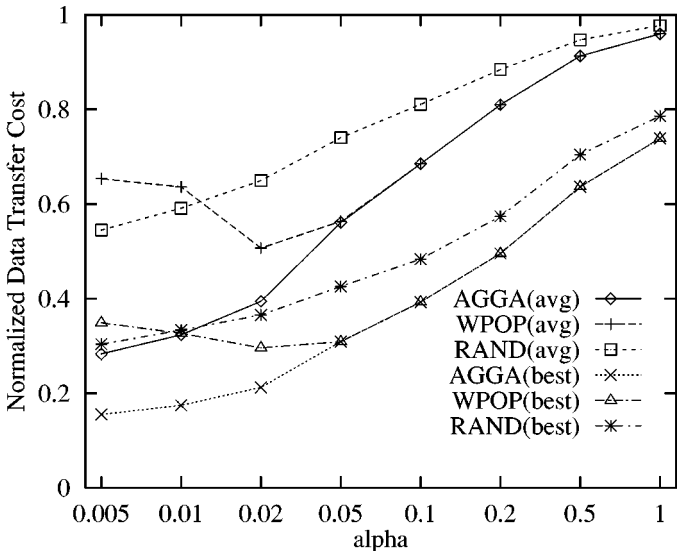
(b)

Fig. 6. Performance for various Ms (homo distribution).

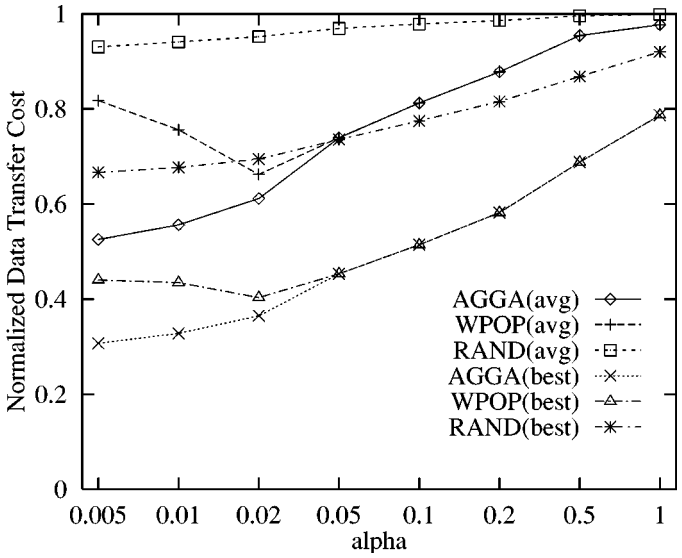
homogeneous distribution pattern. Fig. 6 shows the average and best performance when the maximum number of proxies, M, is varied from 3 to 50. The performance for various write/read ratios, αs, is shown in Fig. 7.

The following observations can be made from the results. First, in Fig. 6, introducing a few replication proxies in the network can substantially improve the data transfer cost. Furthermore, the greatest performance improvement is achieved by the installation of the first few proxies. For example, with the optimal placement scheme (i.e., AGGA), for $TreeSize = 100$, installing four proxies (i.e., $M = 5$) reduces the cost by 36% on average; however, installing five more proxies (i.e., $M = 10$) further improves the cost by 7% only. Second, in Fig. 7, as α increases, the optimal performance degrades. When α approaches one, the average optimal performance becomes close to the performance of the NREP scheme. However, even for $\alpha = 1$, with the optimal placement decision data replication can still improve the performance by 21% at best. This suggests that the decision to deploy replication proxies should not be made by looking at α only, but the network topology and the access patterns must be examined as well. Third, for a small M/α value, the WPOP scheme performs worse than the optimal solution. The reason is as follows. In these cases, the ONR values obtained are larger than the maximum number of proxies allowed. Since for homogeneous distribution each node in the optimal residence set has the same priority and WPOP breaks ties arbitrarily in selecting placement locations, it may perform very badly. On the other hand, for a large M/α value, as the ONR values become less than M, WPOP performs exactly the same as the optimal solution. Last, the RAND scheme has the worst performance. In particular, the difference of its performance to the optimal solution is larger in a larger network, because in this case the probability of finding a near-optimal solution by random selection is even lower.

2) Heterogeneous Access Distribution: We now study the performance of the placement schemes in a heterogeneous scenario where access distributions over the objects vary for different nodes. In this case, the AGGA scheme is not the optimal

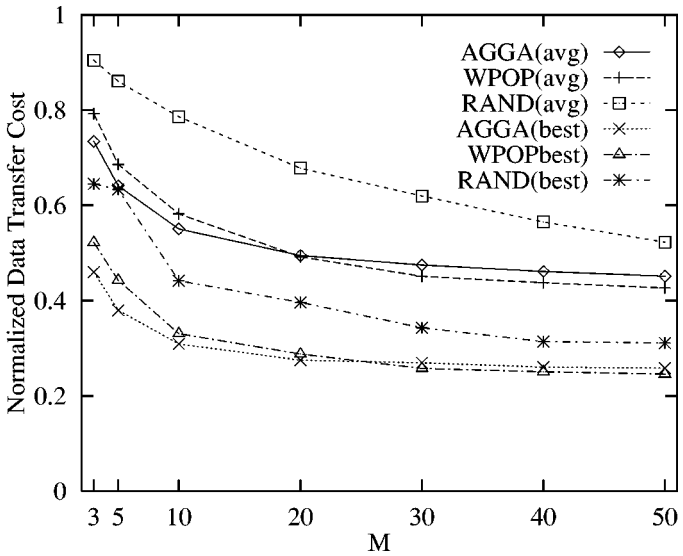


(a)

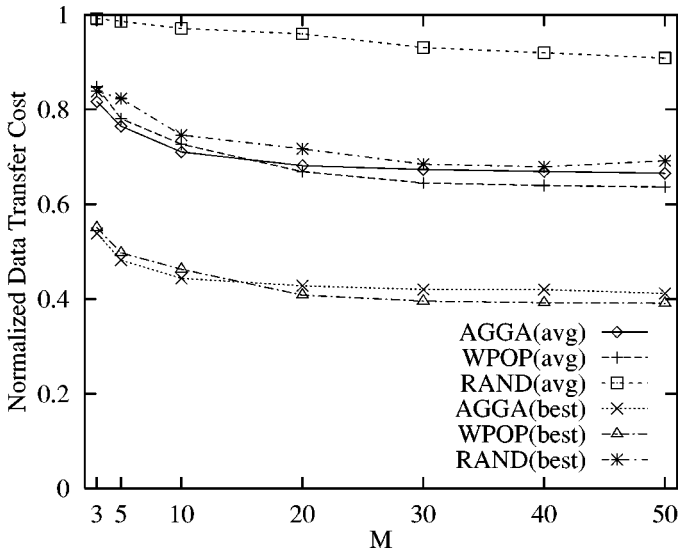


(b)

Fig. 7. Performance for write/read ratios (homo distribution).



(a)



(b)

Fig. 8. Performance for various Ms (heterogeneous distribution).

solution either. For heterogeneous access distribution, we set $OffsetRd = 10$, $OffsetWt = 2$, $\theta_r = 1.2$, and $\theta_w = 0.4$. The results are illustrated in Figs. 8 and 9.

Similar to the homogeneous case, for the proposed schemes of AGGA and WPOP, the installation of a few replication proxies improves the performance remarkably; both the AGGA and WPOP schemes reduce the cost greatly in the best case even for $\alpha = 1$; the RAND scheme has a much worse performance than AGGA and WPOP.

In contrast to the homogeneous case, two different phenomena for the heterogeneous case are observed as follows. First, as shown in Fig. 8, after $M > 20$, introducing more replication proxies can still improve the performance slightly for both AGGA and WPOP, while in the homogeneous case the performance cannot be further improved at all (see Fig. 6). This is due to the diversity of the optimal residence sets for the objects in the heterogeneous case. Second, compared with the homogeneous case, WPOP has a much better relative performance to AGGA. For a small M/α value, the performance difference between them becomes less. For example, when $M < 20$, the average performance of WPOP is 5% worse than that of AGGA, compared 12% in the homogeneous case. For a large M/α value, WPOP even has a slightly better average performance than AGGA. We discuss the reason in the next paragraph.

Fig. 10 shows the maximal relative performance between AGGA and WPOP observed in a series of experiments for $TreeSize = 100$. Similar performance trend is obtained for $TreeSize = 1000$. As can be seen, for a small M/α value, WPOP is always worse than AGGA since in these cases WPOP acts like the RAND scheme due to often tie-breaking operations. However, since the optimal residence set for each object varies in the heterogeneous case, for WPOP the number of tie-breaking operations is much less than that in the homogeneous case and thereby improving its relative performance to some degree. With increasing α, the ONR value for an object decreases as shown in Section VII-B. Thus, for a medium

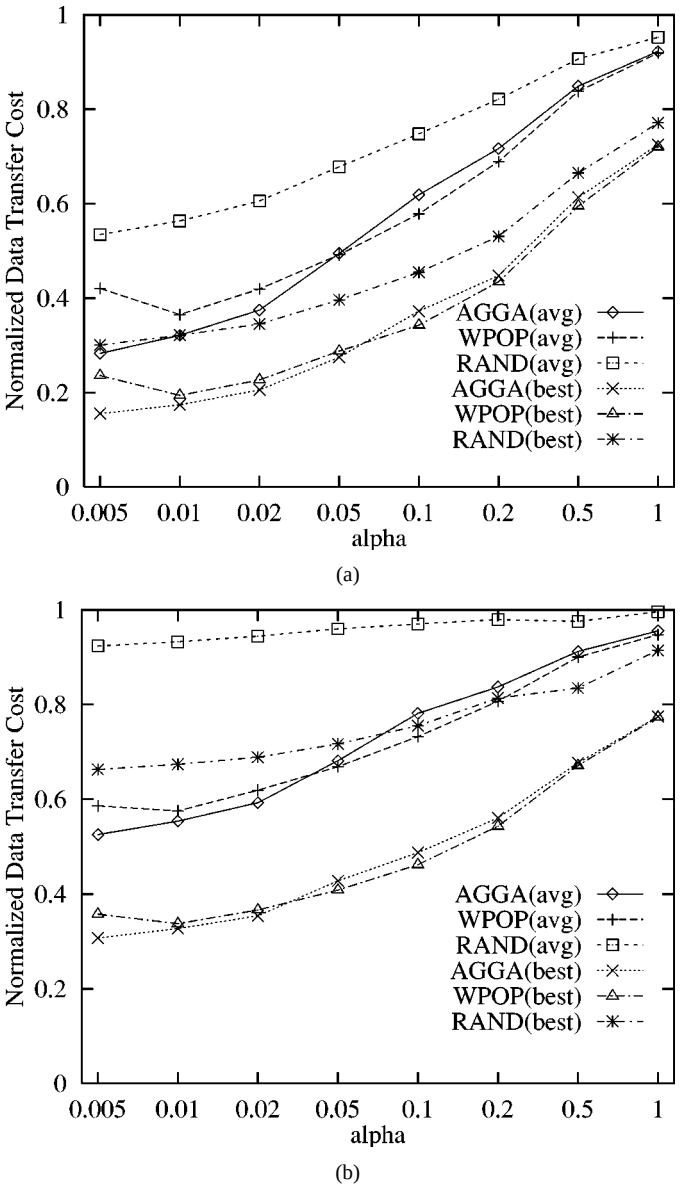


Fig. 9. Performance for write/read ratios (heterogeneous distribution).

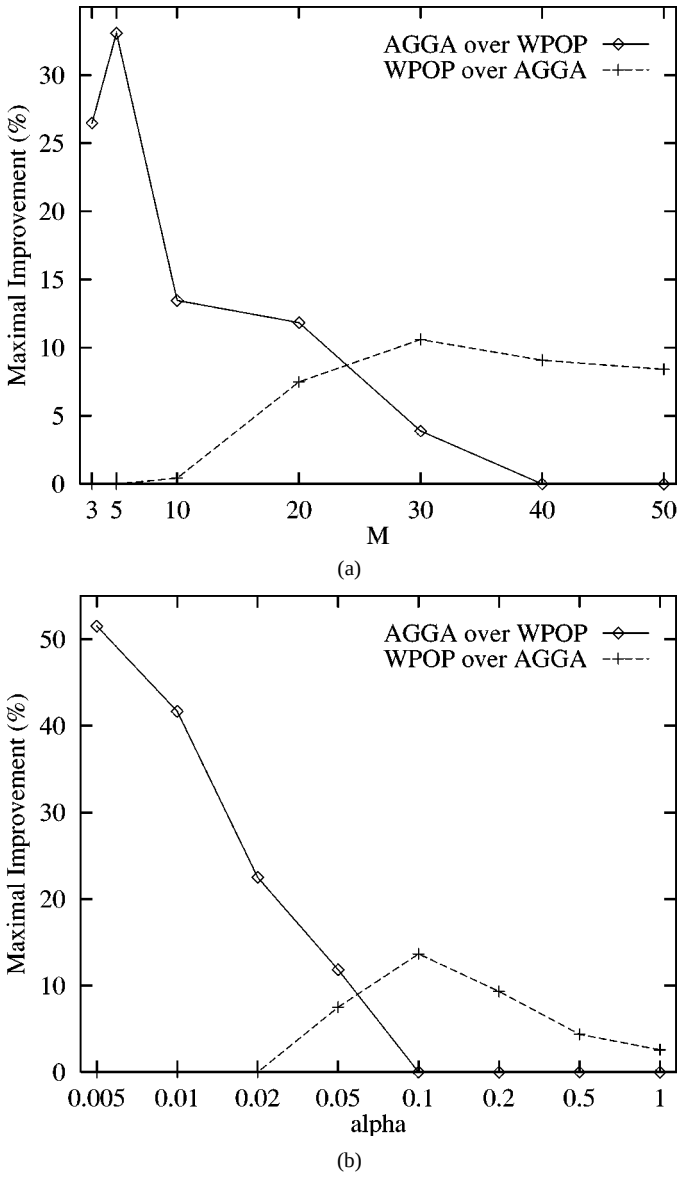


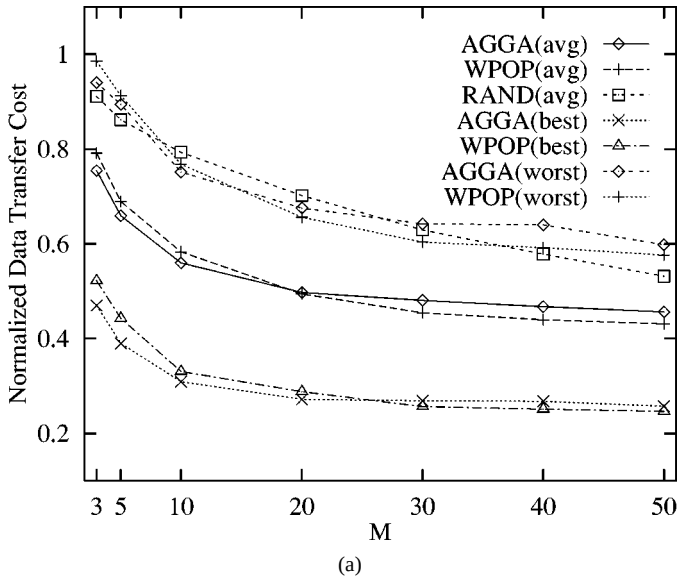
Fig. 10. Maximal relative performance between WPOP and AGGA ($TreeSize = 100$, heterogeneous distribution).

M/α value, tie-breaking operations become even less often for WPOP. Hence, WPOP covers most of the popular placement locations. On the other hand, AGGA cannot reflect the diversity of access distributions for various nodes and performs badly. As a result, for a medium M/α value, the improvement of WPOP over AGGA is the greatest, which is up to 13.6%. With further increasing M/α and, hence, further decreasing the ONR values, as AGGA can also cover most of the popular locations, the performance difference between them decreases. For a large M/α value, WPOP performs no worse than AGGA as shown in Fig. 10.

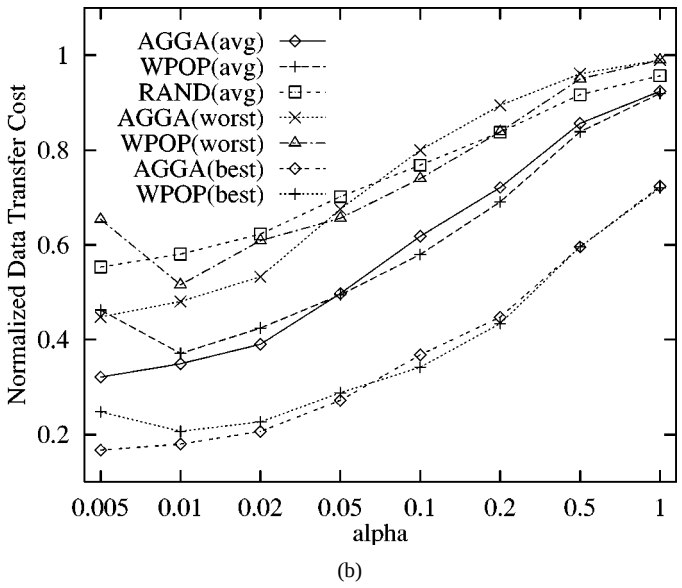
E. Impact of Imperfect Knowledge About Input Data

In the previous sections, we have assumed that perfect knowledge of the underlying network topologies and the access patterns is known to the placement schemes. However, in practice, only rough estimates are available. In this section, we investigate the sensitivity of the proposed placement schemes to imperfect knowledge about the input data.

As in [21], random *noise* is introduced to simulate the inaccuracy of the input data. *Noise* is introduced in two ways. First, the read/write rates for each node are perturbed by up to a factor of two, i.e., if the true rate is λ, the perturbed value is chosen randomly between $((\lambda/2), 2\lambda)$. This is used to model the inaccuracy caused by imperfect access estimate methods and possible changes in access patterns over time. Second, the distance for each edge is perturbed by up to a factor of five, i.e., if the true distance is $d(i, j)$, the perturbed value is chosen randomly between $((d(i, j)/5), 5d(i, j))$. This is used to model the inaccuracy due to imperfect distance measurement methods and local transient unstable routing. The noisy input data are fed to the schemes to determine the placement of proxies in the network and the placement of replicas on the proxies. We then compute the data transfer cost when the correct input data is applied to the imperfect placements.

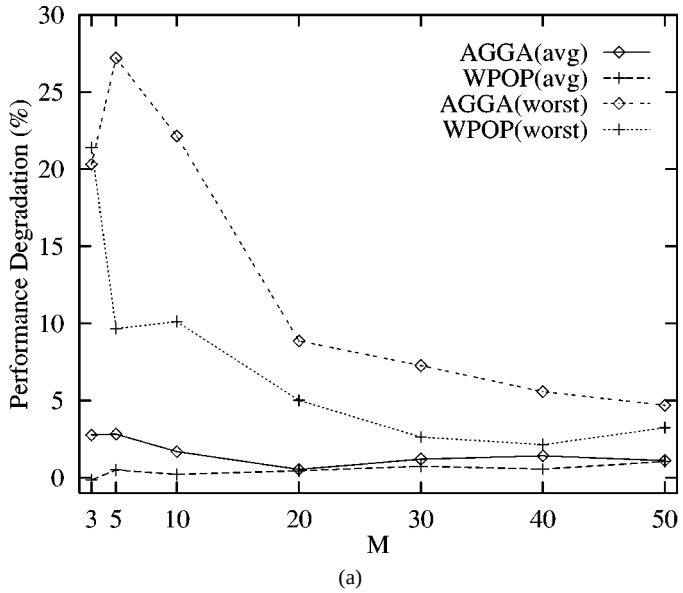


(a)

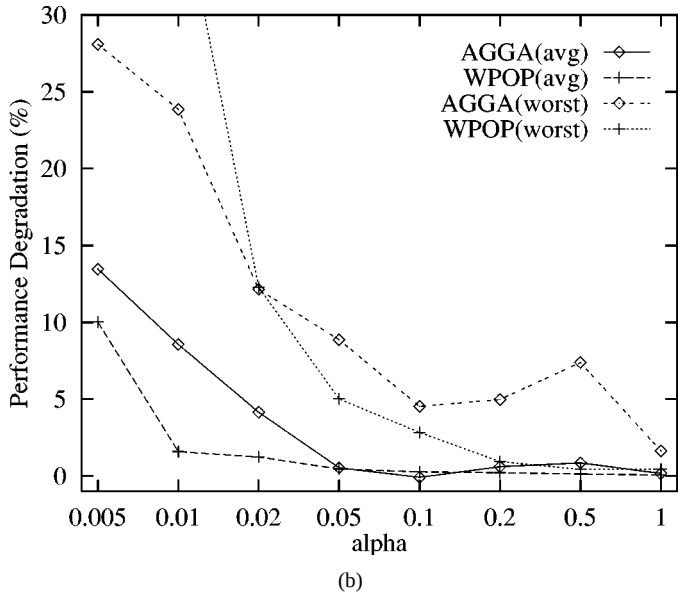


(b)

Fig. 11. Performance with noisy input ($TreeSize = 100$, heterogeneous distribution).



(a)



(b)

Fig. 12. Performance degradation with noisy input ($TreeSize = 100$, heterogeneous distribution).

Fig. 11 plots the average, worst, and best normalized costs for $TreeSize = 100$ when noise in collecting the input data is simulated. We can see that the proposed AGGA and WPOP schemes still achieve a great improvement over NREP and RAND even using noisy inputs.

To take a closer look at the impact of inaccurate inputs, we compare the relative performance of the placement schemes using noisy inputs and those having perfect knowledge. As illustrated in Fig. 12, only a small average performance deviation is observed. In all of the cases, the average performance degradation is within 13.4%; the worst degradation is within 76% [when $\alpha = 0.005$ for WPOP, which is not plotted in Fig. 12(b)], and within 28.1% for the better scheme at the same setting. From Fig. 12, as M or α increases, generally the degradation becomes less. This could be explained as follows. With increasing M or α, the ONR values will gradually become smaller than M. Thus, most of the objects can be placed optimally as in the case of no replica number constraint. Since such optimal placement decision is insensitive to path distances (see Algorithm 1), introducing noise on the distances will hardly affect the performance and, hence, the overall performance is improved.

We also evaluate the performance for $TreeSize = 1000$. We observe that the deviation is even less for this case: the average degradation is within 3.9%, and the worst degradation is within 15.0%. This implies that in a large network the near-optimal placement decision is not very sensitive to the inaccurate inputs for an individual domain.

VIII. Conclusion

Transparent data replication is gaining increasing research interest since it enables high system scalability. In this paper, we presented a hybrid transparent replication model which combines the advantages of the en-route replication model and the hierarchical replication model. We examined the placement issues of replication proxies and data replicas with the objective

of minimizing the total data transfer cost, given that a maximum of M proxies are allowed. Two practical schemes, AGGA and WPOP, were proposed for the problem of replication proxy placement in the network. An optimal algorithm was presented for the problem of data replica placement on the installed proxies. The AGGA scheme is an optimal solution when access distributions over the objects for different nodes are the same.

We conducted a set of simulation experiments to study the performance of the proposed placement schemes. From the simulation results, we have made a number of insightful observations, which are summarized in the following. Note that the observations are valid for both uniform and nonuniform access patterns.

- Partial data replication on the installed proxies is important for read/write applications, especially for those with diverse access distributions over the objects for different nodes.
- Generally the installation of the first few (5–10) replication proxies improves the data transfer cost significantly; further installation of proxies improves the performance only slowly. This is important when the budget is limited.
- To determine if it would improve the performance by installing proxies in the network, the write/read ratio α is not a dominant factor. Instead, one should also examine the underlying network topology and the access patterns.
- For homogeneous access distribution, AGGA obtains the optimal solution and WPOP performs close to the optimal solution for large Ms/αs.
- For heterogeneous access distribution, AGGA outperforms WPOP for small Ms/αs, while WPOP is better for large Ms/αs. In particular, WPOP improves the performance over AGGA most greatly for medium Ms/αs.
- The proposed placement schemes are not sensitive to the inaccuracy of input data. For noise of up to a factor of five on distance estimates and up to a factor of two on access pattern estimates, the average performance degradation is within 15%.

In this paper, we discussed the proxy and replica placement problems under the hybrid transparent replication model. It is not difficult to see that the proposed schemes can also be applied to the en-route replication model and the hierarchical replication model if the serving proxy for each client is set to the first proxy on its path upwards the server, since the placement problems in these cases can be formulated in the same way as that in this paper.

As for future work, we are interested in exploring the effect of load balancing on proxy and replica placement, extending our work to other network topologies, and studying dynamic schemes [8], [28] for transparent data replication.

Appendix I
Proof of Theorem 1

Proof: Suppose that the maximal optimal residence set for the problem is $\mathcal{R}'_i$, $r \in \mathcal{R}'_i$. When $|\mathcal{R}'_i| = 1$, it contains the root only, and the algorithm is simply correct.

Consider the case of $|\mathcal{R}'_i| > 1$. Since the root (server) must be a replicated node according to the problem definition, from Lemma 1 $\mathcal{R}'_i$ is a connected subtree of T_r rooted at the server. For $\mathcal{R}'_i$, we recursively remove the leaf nodes xs with $\lambda^t_{x,i} = \alpha(\mu^t_{r,i} - \mu^t_{x,i})$. Denote the new set by $\mathcal{R}''_i$. It is easy to see that $\mathcal{R}''_i$ has the same cost as $\mathcal{R}'_i$, which means that $\mathcal{R}''_i$ is also an optimal residence set for object i. In the following, we first show that any node $x \in \mathcal{R}''_i$ will be added to $\mathcal{R}_i$ by Algorithm 1.

Without loss of generality, let u be a leaf node of $\mathcal{R}''_i$ and v be u's parent in T_r. As in Section IV, we have

$$cost'(T_r, \mathcal{R}''_i) = cost'(T_r, \mathcal{R}''_i - \{u\}) + d(u, v)\lambda^t_{u,i} - d(u, v)\alpha(\mu^t_{r,i} - \mu^t_{u,i}).$$

Since $\mathcal{R}''_i$ is an optimal residence set, $cost'(T_r, \mathcal{R}''_i) \geq cost'(T_r, \mathcal{R}''_i - \{u\})$. Thus, we can obtain

$$cost'(T_r, \mathcal{R}''_i - \{u\}) + d(u, v)\lambda^t_{u,i} - d(u, v)\alpha(\mu^t_{r,i} - \mu^t_{u,i}) \geq cost'(T_r, \mathcal{R}''_i - \{u\})$$

i.e., $\lambda^t_{u,i} \geq \alpha(\mu^t_{r,i} - \mu^t_{u,i})$. Since $\lambda^t_{u,i} \neq \alpha(\mu^t_{r,i} - \mu^t_{u,i})$ in $\mathcal{R}''_i$, we have $\lambda^t_{u,i} > \alpha(\mu^t_{r,i} - \mu^t_{u,i})$. If u is a child of the root r, then u will be added to $\mathcal{R}_i$ in the first few iterations of the algorithm. If u is not a direct child of r, denote the path between u and r by $u, b_1, \ldots, b_j, \ldots, b_k, r$ for $k \geq 1$. Obviously, $\lambda^t_{b_j,i} \geq \lambda^t_{u,i}$ and $\mu^t_{r,i} - \mu^t_{b_j,i} \leq \mu^t_{r,i} - \mu^t_{u,i}$, thus, we have $\lambda^t_{b_j,i} > \alpha(\mu^t_{r,i} - \mu^t_{b_j,i})$ for any $1 \leq j \leq k$. Consequently, according to the algorithm, these nodes will be added to $\mathcal{R}_i$ in the order of $b_k, \ldots, b_1, u$. Therefore, all the nodes in $\mathcal{R}''_i$ will be added to $\mathcal{R}_i$ eventually.

Next, we show any node $x \notin \mathcal{R}''_i$ will not be added to $\mathcal{R}_i$. Without loss of generality, let v be a leaf node of $\mathcal{R}''_i$ and u be a child of v, $u \notin \mathcal{R}''_i$. Thus, as above

$$cost'(T_r, \mathcal{R}''_i \cup \{u\}) = cost'(T_r, \mathcal{R}''_i) + d(u, v)\lambda^t_{u,i} - d(u, v)\alpha(\mu^t_{r,i} - \mu^t_{u,i}).$$

Since $\mathcal{R}''_i$ is an optimal residence set, $cost'(T_r, \mathcal{R}''_i \cup \{u\}) \leq cost'(T_r, \mathcal{R}''_i)$ and, hence, we can obtain $\lambda^t_{u,i} \leq \alpha(\mu^t_{r,i} - \mu^t_{u,i})$. Thus, according to the algorithm, u will not be added to $\mathcal{R}_i$, neither will u's descendant if any.

Obviously, the time complexity of the algorithm is $O(N)$. ■

Appendix II
Proof of Theorem 2

Proof: The correctness of Algorithm 2 follows by showing the two recurrence relations [i.e., (6) and (7)] are correct in Lemmas 2 and 3.

Lemma 2: Equation (6) is correct. Furthermore, an optimal way of placing $t > 1$ replicas of object i in T_v is to place t' replicas the optimal way in T_u and $t - t'$ replicas the optimal way in $T_{u,v}$, where $P(v, t) = (u, t')$.

Proof: Similar to the proof for Theorem 1 in [15, Theorem 1]. To save space, we do not present the proof here, and the details are left to the reader.

Lemma 3: Equation (7) is correct. Furthermore, an optimal way of placing $t > 1$ replicas of object i in $T_{u,v}$ is to place t' replicas the optimal way in T_x and $t - t'$ replicas the optimal way in $T_{x,v}$, where $P(u, v, t) = (x, t')$.

Proof: Similar to the proof of Lemma 2.

Now, we are going to show that Algorithm 2 requires $O(PNM^2)$ time. The first step can be implemented in linear time in a straight-forward manner. Step 3 can be done in $O(|T_v|)$ time. Thus, the total running time of the loop for step 2 is $O(\sum_{v \in T_r} |T_v|) = O(P)$. By Equations (6) and (7), the computation for $C(v, t)$ and $C(u, v, t)$ takes $O(NM)$. Hence, the loop for step 8 takes $O(|T_v|NM)$ time. Step 6 takes $O(\sum_{v \in T_r} 1)$ time and step 5 $O(M)$. Thus, the total running time of the loop for step 5 is $O(M \sum_{v \in T_r} |T_v|NM) = O(PNM^2)$. Steps 13 and 14 are $O(M)$. The total complexity of this algorithm is $O(PNM^2)$.■

ACKNOWLEDGMENT

The authors would like to thank X. Tang for his valuable discussions and the anonymous reviewers for their constructive comments and suggestions.

REFERENCES

[1] N. Shivakumar, J. Jannink, and J. Widom, "Per-user profile replication in mobile environments: Algorithms, analysis and simulation results," *ACM/Baltzer J. Mobile Networks Appl.*, vol. 2, pp. 129–140, 1997.

[2] H. Yu and A. Vahdat, "Efficient numerical error bounding for replicated network services," in *Proc. VLDB'2000*, Sept. 2000, pp. 123–133.

[3] M. Dilman and D. Raz, "Efficient reactive monitoring," in *Proc. IEEE INFOCOM'2001*, Apr. 2001, pp. 1012–1019.

[4] L. Dowdy and D. Foster, "Comparative models of the file assignment problem," *ACM Comput. Surv.*, vol. 14, no. 2, pp. 287–313, June 1982.

[5] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1999.

[6] O. Wolfson and A. Milo, "The multicast policy and its relationship to replicated data placement," *ACM Trans. Database Syst.*, vol. 16, no. 1, pp. 181–205, Mar. 1991.

[7] CISCO Distributed Director. [Online]. Available: http://www.cisco.com/warp/public/cc/pd/cxsr/dd/tech/dd_wp.htm

[8] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal, "A dynamic object replication and migration protocol for an internet hosting service," in *Proc. IEEE ICDCS'99*, May 31-June 4 1999, pp. 101–113.

[9] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "Hierarchical internet object cache," in *Proc. USENIX Annual Technical Conf.*, Jan. 1996, pp. 153–164.

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

[11] A. Heddaya and A. Mirdad, "Webwave: Globally load balanced fully distributed caching of hot published documents," in *Proc. IEEE ICDCS'97*, May 1997, pp. 160–168.

[12] P. Krishnan, D. Raz, and Y. Shavitt, "The cache location problem," *IEEE/ACM Trans. Networking*, vol. 8, pp. 568–582, Oct. 2000.

[13] I. Cidon, S. Kutten, and R. Soffer, "Optimal allocation of electronic content," in *Proc. IEEE INFOCOM'2001*, Apr. 2001, pp. 1773–1780.

[14] V. Paxson, "End-to-end routing behavior in the internet," *IEEE/ACM Trans. Networking*, vol. 5, pp. 601–615, Oct. 1997.

[15] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby, "On the optimal placement of web proxies in the internet," in *Proc. IEEE INFOCOM'99*, Mar. 1999, pp. 1282–1290.

[16] H. Yu, L. Breslau, and S. Shenker, "A scalable web cache consistency architecture," in *Proc. ACM SIGCOMM'99*, Aug. 1999, pp. 163–174.

[17] K. P. Eswaran, "Placement of records in a file and file allocation in a computer network," in *Proc. IFIP Congress*, Aug. 1974, pp. 304–307.

[18] S. Ceri, G. Martella, and G. Pelagatti, "Optimal file allocation in a computer network: A solution method based on the knapsack problem," *Comput. Networks*, vol. 6, pp. 345–357, Nov. 1982.

[19] M. K. Fisher and D. S. Hochbaum, "Database location in computer networks," *J. ACM*, vol. 27, pp. 718–735, Oct. 1980.

[20] S. K. Chang and A. C. Liu, "File allocation in a distributed database," *Int. J. Comput. Inform. Sci.*, vol. 11, pp. 325–340, 1982.

[21] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *Proc. IEEE INFOCOM'2001*, Apr. 2001, pp. 1587–1596.

[22] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

[23] A. Helal, A. Heddaya, and B. Bhargava, *Replication Techniques in Distributed Systems*. Norwell, MA: Kluwer, 1996.

[24] A. Vigneron, L. Gao, M. J. Golin, G. F. Italiano, and B. Li, "An algorithm for finding a k-median in a directed tree," *Inform. Process. Lett.*, vol. 74, pp. 81–88, Apr. 2000.

[25] G. K. Zipf, *Human Behavior and the Principle of Least Effort*. Reading, MA: Addison-Wesley, 1949.

[26] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On the placement of internet instrumentation," in *Proc. IEEE INFOCOM'2000*, Mar. 2000, pp. 295–304.

[27] J. Xu, B. Li, and D. L. Lee, "Placement problems for transparent data replication proxy services," Dept. Comput. Sci. Hong Kong Univ. Sci. Technol., Tech. Rep. HKUST-CS01-05, Feb. 2001.

[28] O. Wolfson, S. Jajodia, and Y. Huang, "An adaptive data replication algorithm," *ACM Trans. Database Syst.*, vol. 22, pp. 255–314, June 1997.

Jianliang Xu (S'02) received the B.Eng. (*summa cum laude*) degree in computer science and engineering from Zhejiang University, Hangzhou, China, in 1998. He is currently working toward the Ph.D. degree in the Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong.

His research interests include mobile computing, wireless networks, Internet technologies, and distributed systems.

Bo Li (S'88–M'92–SM'99) received the B.S. (*summa cum laude*) and M.S. degrees in computer science from Tsinghua University, Beijing, China, in 1987 and 1989, respectively, and the Ph.D. degree in computer engineering from University of Massachusetts at Amherst in 1993.

Between 1994 and 1996, he worked on high-performance routers and ATM switches in the IBM Networking System Division, Research Triangle Park, NC. Since then, he has been with the Computer Science Department, Hong Kong University of Science and Technology, Hong Kong. His current research interests include wireless mobile networking supporting multimedia, video multicast, and all optical networks using WDM. In addition, he has been involved in organizing over two dozen conferences. He has been on the Editorial Board for *ACM Mobile Computing and Communications Review*, *ACM/Kluwer Journal of Wireless Networks*, and *SPIE/Kluwer Optical Networking Magazine*. He has served as a Guest Editor for *SPIE/Kluwer Optical Networks Magazine*, *ACM Performance Evaluation Review*, and the *KICS/IEEE Journal of Communications and Networks*.

Dr. Li has been on the Editorial Board for IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS—Wireless Communication Series, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS. He has served as a Guest Editor for several special issues, including *IEEE Communications Magazine*, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS. He has been involved with IEEE INFOCOM since 1996 and will serve as the Co-TPC Chair for INFOCOM'2004.

Dik Lun Lee received the M.S. and Ph.D. degrees in computer science from the University of Toronto, Canada, in 1981 and 1985, respectively.

He is a Professor in the Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, and was an Associate Professor in the Department of Computer and Information Science, The Ohio State University, Columbus. He has served as a Guest Editor for several special issues on database-related topics, and as a Program Committee Member and Chair for numerous international conferences. He was the Founding Conference Chair for the International Conference on Mobile Data Management. His research interests include document retrieval and management, discovery, management and integration of information resources on the Internet, and mobile and pervasive computing. He was the Chairman of the ACM Hong Kong Chapter.