Optimal Replication Transition Strategy in Distributed Hierarchical Systems

Chun-Chen Hsu[1,2], Chien-Min Wang[2], and Pangfeng Liu[1,3]

[1] Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan
{d95006,pangfeng}@csie.ntu.edu.tw
[2]Institute of Information Science, Academia Sinica, Nankang, Taiwan
{tk,cmwang}@iis.sinica.edu.tw
[3]Graduated Institute of Networking and Multimedia
National Taiwan University, Taipei, Taiwan

Abstract

We study the replication transition problem in distributed hierarchical systems. Most distributed systems replicate data to increase data access efficiency. A replication strategy dictates where the replicas are stored in respond to the data access pattern, therefore a good strategy can effectively improve data access efficiency. However, the access pattern in a distributed system is constantly changing. As a result, a good replication strategy must evolve accordingly. The replication transition problem is to seek an efficient transition from one replication strategy to another, in order to cope with the dynamic access pattern. This paper focuses on solving the replication transition problem on tree topology, which is one of the most important models in Data Grid systems and web proxy systems from the literature. To the best of our knowledge, our work is the first that proposes an optimal algorithm for the replication transition problem on tree topology. The algorithm has a time complexity of $O(n \log \Delta \log(n\Lambda))$, where n is the number of sites, Δ is the maximum degree in the tree and Λ is the largest communication delay in the network.

1 Introduction

Grid Computing has enabled applications to utilize distributed data and computing resources with the help of the rapid increase in computer and network performance during the last decade. One example of Grid Computing is Data Grids, which consist of distributed storage infrastructures that integrate geographically distributed and independently managed data resources.

In recent years, a number of Data Grid projects have been developed, such as the EU DataGrid [1], GriPhyN [2], LCG [3], and TeraGrid [4]. These project aim to provide integrated service platforms that allow users to access those data resources transparently and efficiently. Therefore, these Data Grid projects focus on the issues of storage and data management, data transfers, data replication, data access optimization, and at the same time maintaining high data reliability and availability.

A common approach to increase data access efficiency in a distributed system is to replicate data at different sites. Data replication improves not only data access efficiency but also data availability and fault tolerance. A replication strategy dictates where the replicas are stored in respond to the data access pattern. Many researchers [8, 12, 13, 14, 15, 19, 20, 21, 22, 23, 24, 25, 26, 27] have proposed algorithms to find optimal or near-optimal replication strategies when data access patterns are given.

The problem we are interested in is to design an algorithm that finds an efficient transition from one replication strategy to another within the *minimum* amount of time. Since the data access pattern usually changes over time, the system needs to create new replication strategies in respond to the changing patterns. Therefore, we need to transit from the old replication strategy to the new one. We denote this problem as the *replication transition problem*, which has been studied on complete graph [7, 9, 11, 16, 18].

This paper focuses on solving the replication transition problem on tree topology, which is one of the most im-

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

portant model in Data Grid systems and web proxy systems from the literature [5, 12, 15, 23, 24, 25, 27]. To the best of our knowledge, replication transition on tree topology has never been studied. On the other hand, this paper proposes the first algorithm that gives optimal replication transitions on tree topology. This algorithm runs in time of $O(n \log \Delta \log(n\Lambda))$, where n is the number of sites, Δ is the maximum degree in the tree, and Λ is the largest communication delay in the network.

1.1 Related Works

The replication transition problem on complete graph has many variations in the literature. Hall et al. [11] studied the replication transition problem with two assumptions. First, the network bandwidths is the same for all links, so that it takes the same amount of time to transfer a data item from one site to any other site. Second, each data has only one copy i.e., we are not allowed to replicate data. The goal is to minimize the makespan. They developed a polynomial time algorithm that finds a near-optimal migration plan in the presence of space constraints when a certain number of additional nodes are available as temporary storage, and a 1.5-approximation algorithm for the case where data must migrate directly to their destinations [11].

Khuller et al. [16] studied the same problem as in [11], but assumed that data have replicas, and gave a 9.5-approximation algorithm. Kim [18] studied the replication transition problem with the same assumptions as in [16], except that the network bandwidth may vary and the goal is to minimize the average completion time. A 9-approximation algorithm was given in [18]. Later Gandhi et al [7] improved the approximation ratio to 5.06 [7]. Interested readers may refer to [7] for a good survey about the history of the problem.

The multi-casting problem is strongly related to the replication transition problem. Replication transition is essentially a multi-casting problem with *multiple* sources. Khuller et al. [17] studied the *single-source* multi-message multi-casting problem on complete graph. Gonzalez [9, 10] studied the same problem and further assumed that a sender can multicast a message to a set of sites in a single send. Both works assume that the size of messages and the network bandwidth are the same, and gave approximation algorithms for these two models respectively [17, 9].

The above mentioned works cannot be applied on tree topology systems, such as Data Grids and web proxy systems, due to the lack of a direct communication channel between any two sites in tree topology. On the other hand, this paper proposes the first algorithm that gives optimal replication transitions on tree topology.

The rest of this paper is organized as follows. Section 2 describes our system model and gives a formal definition of the replication transition on tree topology. Section 3 presents our algorithm and provides theoretical analysis. Section 4 gives experimental results, and Section 5 gives concluding remarks and open problems.

2 System Model and Problem Definition

We consider a hierarchical distributed system, which can be represented by a tree $T = (V, E, w)$, where V is the set of sites, $E \subseteq V \times V$ is the set of links between sites, and the function w: $E \rightarrow \mathbb{N}$ maps each edge to a positive integer which represents the communication delay of the corresponding edge. For the sake of simplicity, we use w_{uv} instead of $w(u, v)$ to denote the communication delay of link $(u, v) \in E$. Nodes and sites are interchangeable terms in this paper.

This paper adopts the full-duplex and one-port communication model [6]. In the full-duplex one-port model, each site can simultaneously send/receive one data item to/from one of its neighbors. We further assume that the communication is non-preemptive, i.e., a site cannot send another data item before the current one finishes. Similarly, a site cannot start receiving a new data item before completely receiving the current one. In addition, two sites u and v can communicate only if there exists a link $(u, v) \in E$.

The sites require data because they are planned to store replicas. If a site needs the data which is not available locally, it must request the data from other sites that have the data or the replica. For simplicity we will consider one data at a time, and assume that there is only one data in the system. Initially, there are two set of sites – the *owner set* $O \subset V$ is the set of sites that own the data, and the *requester set* $R \subset V$ is the set of sites that request the data. Other sites are *routers*. A router can temporarily request and store the data, if it is required by its parent or at least one of its children. We assume that $O \cap R = \varnothing$ since a node cannot own and request the same data at the same time.

We formally define the timing of all the message passing by *communication*. We define a communication as a quadruple (s, r, t_s, t_e), where s and r are the sender and the receiver in V, and t_s and t_e are the starting and ending time of the communication. For ease of notation we use $\tau.t_s$ to indicate the starting time t_s of a communication τ. The receiver owns the data when the communication is completed and r is removed from R if it belongs to R initially. A communication is *valid* if the following conditions hold.

1. The node s owns the data before or at time t_s.
2. There is a link in E between s and r.
3. The duration of the communication $t_e - t_s$ is the link delay w_{sr}.

A *valid* communication schedule S is a set of valid communications that obey the following two conditions.

Non-overlapping For any two communications τ, and $\bar{\tau}$, if they have the same sender or the same receiver, then they cannot overlap. That is, the t_s of τ must be equal to or later than the t_e of $\bar{\tau}$, or the t_s of $\bar{\tau}$ must be equal to or later than the t_e of τ.

Completeness All request sites own the data after the schedule.

We now define the completion time of a requester, which can be used to define the length of a schedule. Given a valid schedule S, we define the *completion time* of a requester r as $\min\{\tau.t_e | \tau \in S, \tau.r = r\}$, which is the earliest ending time of those communication having r as the receiver. The *length* of a schedule is the maximum completion time among all requesters.

The following lemma suggests that, in the search of an optimal schedule, we could consider only those valid schedules in which all nodes receive the data at most once. This gives us a "canonical" form of a communication schedule, at which we can focus our search of optimal schedule.

Lemma 1. *Given a valid schedule S, there exists another valid schedule S' such that S' has the same length as S and all nodes receive the data at most once in S'.*

Proof.

Suppose there are $k \geq 2$ communications $c_1, \ldots, c_k \in S$ with the same receiver r. Those communications do not overlap in time since S is valid. Without lose of generality, we assume that $c_1.t_e < \cdots < c_k.t_e$. We modify S by removing $c_2, \ldots, c_k$, and denote the resulting schedule by S'.

The schedule S' satisfies the **Non-overlapping** condition since we only remove communications from S. S' also satisfies the **Completeness** condition since r will have the data after the communication c_1 in S'. As a result the schedule S' is valid.

Furthermore, the schedule S' has the same length as S because the completion times of r and the other nodes remain the same. Therefore, by repeating this process, we obtain a valid schedule S' in which all nodes receive the data at most once, and S' has the same schedule length as S. ■

A replication transition problem instance I is a tuple of (T, O, R), where T represents the tree topology, O is the owner set, and R is the requester set. The goal is to find a valid communication schedule with the *minimum* length for a given problem instance I. In the following section, we describe an algorithm that finds the optimal schedule for the replication transition problem on tree topology.

3 The Optimal Replication Transition Algorithm

This section describes our optimal replication transition algorithm. A schedule is *D-feasible* if it is valid and its length is no more than D. The time D servers as a *deadline* of all requesters. We start by describing a $FindFeasible$ algorithm that given a problem instance I and a deadline D, determines whether a D-feasible schedule exists. Then we describe our optimal replication transition algorithm that calls $FindFeasible$ repeatedly within a binary search in order to find an optimal schedule.

3.1 The $FindFeasible$ Algorithm

Given a problem instance $I = (T, O, R)$ and a deadline D, the $FindFeasible$ algorithm determines whether a D-feasible schedule exists. Let I_i be a sub-problem instance consisting of (T_i, O_i, R_i), where T_i is the sub-tree of T rooted at node i, O_i is the set of data owners in T_i, and R_i is the set of data requesters in T_i. Formally we have $O_i = O \cap V(T_i)$ and $R_i = R \cap V(T_i)$.

A sub-tree *D-self-fulfilled* if all the requesters in the subtree can obtain data from those owners located in the same subtree within time D. Formally we have the following definition.

Definition 1. *Given a common deadline D, a sub-tree T_i is D-self-fulfilled if there exists a D-feasible schedule for I_i.*

Two observations can be derived from Definition 1. First, T_i is definitely self-fulfilled if there is no requester in T_i. Second, if T_i is not D-self-fulfilled, the root of T_i (node i) must receive the data from its parent in a D-feasible schedule for I. For simplicity, we use *feasible* schedule to represent D-feasible schedule.

3.1.1 Earliest Supplying Time and Deadline

Definition 2. *When T_i is D-self-fulfilled, the* earliest supplying time *of node i, denoted by c_i, is the earliest time at which node i can start sending the data to its parent.*

Note that when T_i is not D-self-fulfilled, the earliest supplying time c_i is infinity. The reason is that when T_i is not D-self-fulfilled, instead of sending the data to its parent, node i must *receive* the data from its parent in a feasible schedule for I according to the second observation above.

Definition 3. *When T_i is not D-self-fulfilled, the* deadline *of node i, d_i, is the latest time at which node i must complete receiving the data from its parent so that requesters in T_i can complete receiving the data at time no later than D.*

Note that when T_i is self-fulfilled we set d_i to infinity. The reason is that when T_i is self-fulfilled it is not necessary for node i to receive the data from its parent; instead node i will receive the data from a node within T_i. In addition, we define a communication to be *feasible* when its ending time is no later than the deadline of its receiver.

3.1.2 Earliest Possible Time Calculation

We now compute the *earliest possible* time of a node i, which is the earliest possible time node i can receive the data from nodes other than its parent. We denote the earliest possible time time of node i by s_i. Let K_i be the set of children of node i whose earliest supplying time c is not infinity, and L_i be the set of children of i whose deadline d is not infinity.

Now we compute the earliest possible time of node i. $FindFeasible$ processes tree nodes in post-order. As a result we may assume that when we compute the earliest possible time for node i, we know the earliest supplying times and deadlines of all the children of node i. Therefore we can classify the children of node i into K_i and L_i and calculate earliest possible time s_i as follows.

$$s_i = \begin{cases} 0 & \text{, if } i \in O \\ \infty & \text{, if } i \notin O \text{ and } K_i = \varnothing \\ \min_{k \in K_i}\{c_k + w_{ik}\} & \text{, if } i \notin O \text{ and } K_i \neq \varnothing \end{cases} \quad (1)$$

The first equation states that if node i owns the data, the earliest possible time is 0. If node i does not own the data initially and none of its children nodes can supply the data, the earliest possible time s_i is set to infinity. On the other hand, if any child of i can supply the data, then node i chooses the child that can send the data at the earliest time.

We now describe $FindFeasible$ in details. $FindFeasible$ processes tree nodes in post-order. For each node i, $FindFeasible$ first decides whether T_i is self-fulfilled. If T_i is self-fulfilled, $FindFeasible$ computes c_i and set d_i to infinity; otherwise it computes d_i and sets c_i to infinity.

3.1.3 L_i is empty

We consider two cases in the computation of earliest supplying time and the deadline. First we consider the case where L_i is empty and we have three sub-cases.

i is not a requester T_i is self-fulfilled because neither i nor the children of i will request the data. $FindFeasible$ sets c_i to s_i because it is the earliest time i can send data to its parent. $FindFeasible$ sets d_i to infinity because T_i is self-fulfilled.

i is a requester and $s_i \leq D$ T_i is also self-fulfilled because node i can get the data from its children no later than time D, so $FindFeasible$ sets c_i to s_i and d_i to infinity.

i is a requester and $s_i > D$ In this case T_i is not self-fulfilled because i cannot receive the data from its children before time D, so it sets c_i to infinity. $FindFeasible$ sets d_i to D because i, as a requester, must receive the data no later than D.

In the first two sub-cases, if node i receives the data from one of its child k', then $s_i = c_{k'} + w_{ik'}$, and $FindFeasible$ creates a communication $\tau_s = (k', i, c_{k'}, s_i)$. We will show that the communication τ_s will not overlap with other existing communications of node k'.

3.1.4 L_i is not empty

We now consider the case when L_i is not empty, which implies that node i must send the data to those nodes in L_i before their corresponding deadlines. Consider a node l in L_i. It is necessary for node l to start receiving the data from node i at the time no later than $d_l - w_{il}$, otherwise requesters in T_l could not possibly complete receiving the data at time D. Therefore we must create communications such that each node l in L_i could start receiving the data from node i at time no later than $d_l - w_{il}$.

$FindFeasible$ creates communications for nodes in L_i as follows. Let the number of nodes in L_i be m, and $l_1, \cdots, l_m$ be the nodes in L_i sorted by their deadlines in a non-decreasing order. $FindFeasible$ creates m communications, and the j-th communication is $\tau_j = (i, l_j, t'_{l_j}, t_{l_j})$, where the starting time t'_{l_j} is $t_{l_j} - w_{il_j}$, and the ending time t_{l_j} is the minimum of $\tau_{j+1}.t_s$ and d_{l_j}. We also create a dummy communication τ_{m+1} where $\tau_{m+1}.t_s$ is infinity. This communication τ_{m+1} is only for boundary condition.

The equation $t_{l_j} = \min\{\tau_{j+1}.t_s, d_{l_j}\}$ ensures two properties. First, any two sending communications of node i cannot overlap because τ_j must end before the starting time of τ_{j+1}. Second, any communication will not miss its deadline because the equation guarantees that τ_j ends before d_{l_j}.

Figure 1 illustrates the idea of the communication creation. Node i has 4 children – l_1, l_2, l_3 and l_4. The communication delays between node i and l_1, l_2, l_3 and l_4 are 4, 3, 2 and 3 respectively. The deadlines of node l_1, l_2, l_3 and l_4 are 9, 9, 14 and 16 respectively. Each block represents a communication, and the length of a block is the cost of the corresponding communication. The communication creation starts from node l_4, which has the latest deadline 16. We first create the communication $\tau_4 = (i, l_4, 13, 16)$ in Figure 1(a), since l_4 must get the data before its deadline 16. Then, we create the communication $\tau_3 = (i, l_3, t_{l_3} - w_{il_3}, t_{l_3})$, where $t_{l_3} = \min\{\tau_4.t_s, d_{l_3}\} =$

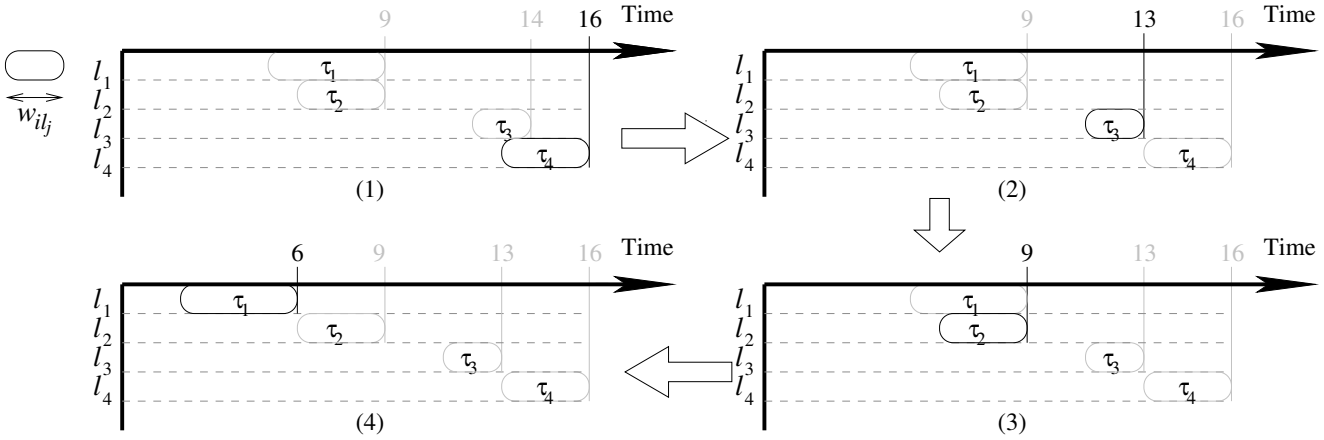

Figure 1. An illustration of the creation of communications.

$\min\{13, 14\} = 13$, as illustrated in Figure 1(b). This ensures that the two sending communications of node i (τ_3 and τ_4) do not overlap. Then we create communication $\tau_2 = (i, l_2, t_{l_2} - w_{il_2}, t_{l_2})$. The ending time t_{l_2} is set to $\min\{\tau_3.t_s, d_{l_2}\} = \min\{11, 9\} = 9$, which ensures that node l_2 meets its deadline. Finally, we create $\tau_1 = (i, l_1, 2, 6)$ as in Figure 1(d).

The communications created above can form groups and communications within the same group are contiguous in time. For example, there are two groups in Figure 1(d) – τ_1 and τ_2 are in one group, and τ_3 and τ_4 are in the other group. The starting time of a group is the starting time of its first communication and the ending time of a group is the ending time of its last communication. Furthermore, the ending time of a group is the deadline the receiver of its last communication.

The following lemma states an important property of the created communications.

Lemma 2. *Let $\tau_1, \cdots, \tau_m$ be the communications created by $FindFeasible$ between node i and its child nodes such that $\tau_i.t_s < \tau_j.t_s$ for $i < j$. $\tau_1.t_s$ is the latest time for node i to start sending the data to its child nodes in all feasible schedules.*

Proof.

We prove this lemma by contradiction. Let S be the schedule created by $FindFeasible$ and g_1 be the earliest communication group , $\tau_1, \ldots, \tau_u$, of node i. Suppose there is a feasible schedule S' such that the earlies time at which node i starts sending the data to its child nods in S' is later than $\tau_1.t_s$. Let g'_1 be the earliest communication group of node i in S' and t'_s be the starting time of g'_1, which is later than $\tau_1.t_s$, i.e. $t'_s > \tau_1.t_s$.

By Definition 3, all subtrees rooted at the receivers in g_1 are all not self-fulfilled in any feasible schedule. As a result, those receivers must receive the data from node i in S'. Furthermore, none of them can complete after $\tau_u.t_e$ in S' because those receivers must complete receiving the data before their deadlines and $\tau_u.t_e$ is the latest deadline among those receivers.

Then, we claim that there is at least one communication in g_1 that is not in the duration between t'_s and $\tau_u.t_e$ in S'. First, the duration between $\tau_1.t_s$ and $\tau_u.t_e$ is the minimum duration to contain all the communications of g_1 since $\tau_u.t_e$ is the latest deadline among those receivers and g_1 consists of contiguous communications. And, second, $t'_s > \tau_1.t_s$. Along with our assumption that t'_s is the starting time of g'_1, we can conclude that, in S', there is at least one communication in g_1 that completes later than $\tau_u.t_e$, a contradiction. Thus, $\tau_1.t_s$ is the latest time for node i to start sending the data to its child nodes in all feasible schedules. ■

Lemma 2 implies that $\tau_1.t_s$ is the latest time at which node i must start sending the data to its child nodes so that all children in L_i can meet their deadlines in all feasible schedules.

3.1.5 $s_i > \tau_1.t_s$

We can determine whether T_i is self-fulfilled by comparing $\tau_1.t_s$ with s_i. Recall that s_i is the *earliest possible* time at which node i can obtain the data without receiving it from the parent, and by Lemma 2, $\tau_1.t_s$ is the *latest* time for node i to start sending the data to its children. Thus, if $s_i > \tau_1.t_s$, T_i could not be self-fulfilled. In this case we set the deadline d_i to $\tau_1.t_s$, which means the parent must supply the data to i no later than $\tau_1.t_s$ so that all requesters in the subtrees in L_i can complete within time D. And we set the earliest supplying time c_i to infinity to indicate that node i will not send the data to its parent.

3.1.6 $s_i \leq \tau_1.t_s$

On the other hand, if s_i is less than or equal to $\tau_1.t_s$, node i can send the data to nodes in L_i so that they can meet their deadlines. In this case T_i is self-fulfilled since all requesters in T_i can get the data no later than D.

Now if T_i is self-fulfilled then we need to compute the earliest supplying time c_i, which is the earliest time node i can start sending the data to its parent. Therefore, in order to compute c_i, we need to find the earliest idle period that is large enough to insert a communication between node i and its parent. The idea is to find an idle period with length at least w_{ip} among communications $\tau_j, 1 \leq j \leq m$, where p is the parent of node i.

Before describing the algorithm, we use an example to demonstrate how to find such an idle period. Please refer to Figure 2 for an illustration.

In this example, s_i is 0 and w_{ip} is 4. $\tau_1, \tau_2, \tau_3, \tau_4$ are shown in Figure 2(a). We start searching the idle period from $s_i = 0$. As shown in Figure 2(a), the length of the first idle period is 2, which is not enough for τ_p. Therefore, we move τ_1 and τ_2 forward to time 0 and time 4 respectively, as shown in Figure 2(b). Please note that, in order to meet the deadlines of those child nodes, we can only move the existent communications forward rather than dylaying them. Now we combine the first and the second idle periods and a larger idle period is formed. The new idle period is now large enough for τ_p to fit in, therefore we set c_i to 7.

Now we describe the process that finds the idle period for τ_p. Let a_j and b_j represent the starting and ending time of τ_j, where $1 \leq j \leq m$. We set b_0 to s_i and a_{m+1} to infinity as the boundary conditions. We consider the gaps between communications one at a time, and start from the first idle period. If the j-th idle period $a_{j+1} - b_j$ is at least w_{ip}, then we use this gap as the idle period for τ_p, and set c_i to b_j. Otherwise, we move the communication τ_{j+1} forward by setting its starting time to b_j and its ending time to $b_j + w_{il_{j+1}}$. Then we consider the next gap. This process will eventually terminate because a_{m+1} is set to infinity.

Lemma 3. *The time c_i found by $FindFeasible$ is the earliest time at which node i can start sending the data to its parent node when T_i is self-fulfilled.*

Proof. Since the procedure of the idle period starts from the earliest possible time s_i, the time c_i found by $FindFeasible$ is the starting time of the first idle period, which conforms to Definition 2. ■

Note that the communication τ_p has not yet been created at this point since we still do not known whether it is necessary for node p to have τ_p. However, if node p requires the communication $\tau_p = (i, p, c_i, c_i + w_{ip})$, it will not overlap with existing communications of node i.

The following theorem states that $FindFeasible$ can find a feasible schedule if such a schedule does exist.

Theorem 1. *Given a problem instance $I = (T, O, R)$ and a deadline D, $FindFeasible$ determines whether these exists a D-feasible schedule.*

Proof. We would like to show that for each node i, $FindFeasible$ determines whether T_i is self-fulfilled, and all communications created by $FindFeasible$ do not overlap and are feasible.

For each node i, we can determine whether T_i is self-fulfilled by s_i and $\tau_1.t_s$. The earliest possible time s_i is the earliest time node i can obtain the data from its children or itself. By Lemma 3, all the earliest supplying times of its child nodes are computed correctly by $FindFeasible$, which implies that s_i can be correctly determined by Equation 1. In addition, Lemma 2 implies that $\tau_1.t_s$ is the latest time for node i to start sending the data to its child nodes so that they will not miss deadlines. Therefore, if $s_i > \tau_1.t_s$, there is no D-feasible schedule for T_i. If that is the case, the deadline d_i is set to $\tau_1.t_s$, which conforms to Definition 3. On the other hand, if T_i is self-fulfilled, by Lemma 3, $FindFeasible$ can also calculate the earliest possible time of node i correctly.

Now we argue that the communications created by $FindFeasible$ for node i do not overlap and are feasible. During the creation, communications $\tau_1, \ldots, \tau_m$ do not overlap and, when the communication τ_s is created, it will not overlap with the existing communications of its children by construction. In addition, the created communications are feasible since the deadlines of receivers are all satisfied by construction. Therefore, $FindFeasible$ can determine whether T is self-fulfilled and, if it is, the created communications form a valid schedule. Hence, $FindFeasible$ can find a feasible schedule if there exists one. ■

The time complexity of $FindFeasible$ is as follows. Let δ_i be the number of child nodes of $i \in V$ and $\Delta = \max_{i \in V}\{\delta_i\}$. For each node i, the processing time is dominated by sorting L_i since the rest of the algorithm runs in time proportional to either the size of the set K_i or the size of the set L_i. Therefore, for each node i, the processing time is bounded by $O(\delta_i \log \delta_i)$, and the time complexity of $FindFeasible$ is $\sum_{i=1}^{n} O(\delta_i \log \delta_i) = O(\sum_{i=1}^{n} \delta_i \log \delta_i) \leq O(\sum_{i=1}^{n} \delta_i \log \Delta) = O(\log \Delta \sum_{i=1}^{n} \delta_i) = O(n \log \Delta)$.

The pseudo code of $FindFeasible$ is given in Figure 3.

3.2 The Scheduling Algorithm

Given a problem instance I, our algorithm finds an optimal solution by a binary search. First, the algorithm determines the initial values of the upper bound U and the lower bound L of the optimal schedule length. Then, wee set the deadline D to the average of the upper and the lower bound, and call $FindFeasible$ to determine whether a D-feasible schedule exists. If we do find a D-feasible sched-

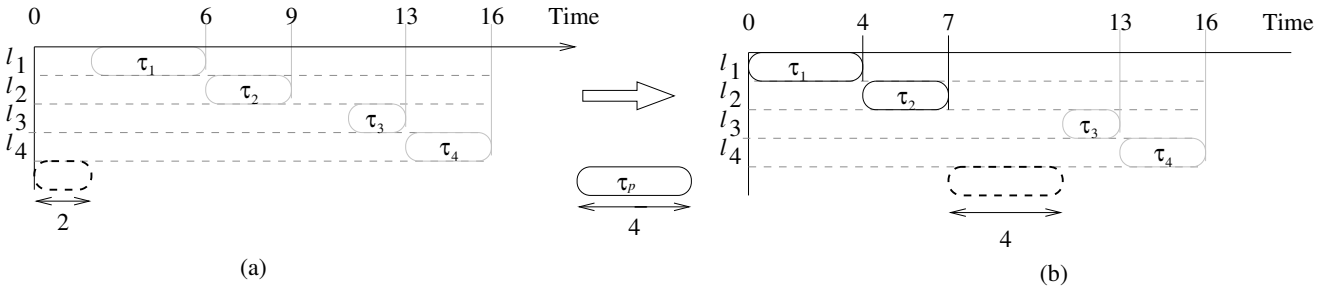

Figure 2. An illustration of how to create an idle period for communication τ_p.

ule, we lower the upper bound to D. Otherwise, no D-feasible schedule exists according to Theorem 1, and we raise the lower bound to D. By this binary search we can find the minimum D such that a D-feasible schedule does exist. This is the optimal schedule length we are looking for.

We now analyze the number of iterations of the binary search by bounding the upper bound. Let Λ be the largest network delay among all links in the tree network. The longest distance between any two nodes in T is bounded by $n\Lambda$ since there are only n nodes. Since there are at most $|R|$ requesters, it is sufficient to set the schedule upper bound to $n\Lambda \times |R|$. For simplicity we set the upper bound to $n^2\Lambda$ since $|R|$ is at most n. Our algorithm calls $FindFeasible$ for at most $O(\log(n\Lambda))$ times and the time complexity is $O(n \log \Delta \log(n\Lambda))$.

4 Experiment

We conduct experiments to evaluate the performance of our scheduling algorithm. We first describe the settings of our experiments and then discuss and analyze the experimental results.

4.1 Experiment Settings

We use the Shortest Path First (SPF) algorithm, a greedy heuristic algorithm, as a performance comparison with our algorithm. SPF algorithm works as follows. For each requester, SPF chooses the nearest owner. Once an owner is chosen by a requester, the data will be sent along the path from the owner to the requester. There is a upper bound k that limits the number of requesters a owner could serve.

When all paths are found, the communications are created as follows. The owners will first create communications to their neighbours. If a receiver is scheduled to receive the data multiple times, it chooses the communication with the earliest ending time and drops other communications. The receiver becomes an owner after it receives the data. The process repeats until all requesters receives the data.

The settings of experiments are as follows. All test cases are generated based on the proposed system model. The height of each tree is at most 8. For each node the number of children are generated from a uniform distribution between 0 and 6. The number of nodes in each generated tree is between 8000 and 14500. There are 20 trees generated in the experiments.

Requester and owners are selected from tree nodes randomly. For each tree, the sum of the numbers of requesters and owners is one tenth of the total number of nodes. We test 19 ratios between the numbers of requesters and owners, i.e., $|R|/|O|$, from $\frac{10}{1}$ to $\frac{1}{10}$. The communication cost of each link is from a uniform distribution between 1000 and 20000. In our experiments, k is set to 5, 10, 15, 20, 25 and infinity.

4.2 Experimental Results

For each R/O ratio, our algorithm and SPF generate schedules for 20 trees and take the average makespan as the performance metric. The results are shown in Figure 4. The horizontal axis represents the different R/O ratios and the vertical axis is the average makespan. For each algorithm, Figure 4 illustrates the average, minimum, and the maximum makespan. Note that the vertical axis is in log-scaled due to the large range among different results. There are no results of SPF for $10/1 \sim 6/1$in Figure 4(a) when k is 5 since there is not enough owners to send data to requesters sue to the limit of k.

Figure 4 clearly shows the large performance gap between SPF and our algorithm when the R/O ratio is large. When the R/O ratio is from $10/1$ to $1/1$ in Figure 4(a), the average makespans for SPF are about 675000$\sim$150000 and the average makespans for our algorithm are about 147000$\sim$99000. This is because when the number of owners is relatively small compared to the number of requesters, the requesters tend to find the same owner as their senders in SPF. On the other hand, our algorithm does not assign

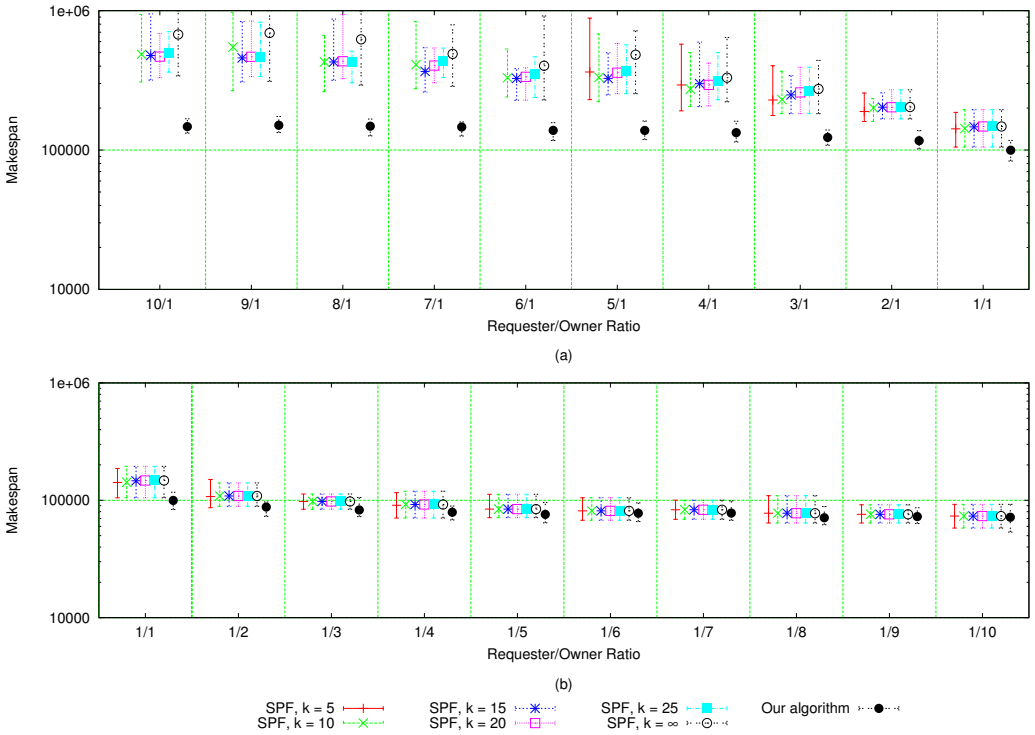

Figure 4. Illustrations of the experimental results.

owners to requesters in advance. Instead, our algorithm dynamically assign owners to requesters, while considering all the communication that have been determined.

When the R/O ratio is smaller the average makespans in SPF decreases as shown in Figure 4(b). When the R/O ratio is $1/10$, the makespan of SPF is closed to the optimal makespan. This is reasonable since requesters may now choose different senders and hence distribute the communications into different sections of the tree.

Figure 4 also shows the influence of different k on SPF. The performances difference is small from $k = 5$ to $k = 25$. However, it is interesting that the average makespan is the worst when $k = \infty$. The reason is that, when $k = \infty$, requesters tend to choose the same owner since there is no limitation on the number of requester that can go to the same owner. When R/O ratios is smaller than 1 the performance of SPFs are almost the same, independent of k.

From Figure 4(a) and (b) we conclude that although SPF is straightforward and can be easily implemented, it has relatively poor performance compared with our algorithm.

5 Concluding Remarks

This paper considers the data replication transition problem on tree topology. A replication strategy dictates where the replicas are stored according the data access pattern. A good replication strategy must evolve in respond to the constantly changing data access pattern. Therefore, the replication transition problem is to seek an efficient transition from one replication strategy to another on tree topology networks. The goal is to minimize the makespan of the replication transition. To the best of our knowledge, our work is the first that proposes an optimal algorithm for the replication transition problem in tree topology.

Given a hierarchical distributed system $T = (V, E, w)$ and an initial owner set $O \subset V$ and requester set $R \subset V$, we give an efficient algorithm that finds an optimal transition schedule. The time complexity of our algorithm is $O(n \log \Delta \log(n\Lambda))$, where $n = |V|$, Δ is the maximum degree among all nodes in T, and Λ is the largest communication delay in the network. The experimental results indicate that the algorithm produces very efficient schedules, when compared those produced with Shortest Path First heuristic.

There are still many variations of the replication transition problem on tree. This paper address a single message multi-source multi-casting problem on tree topology. When more than one data are involved, the replication transition problem become a multi-message multi-source multicasting problem. These problems are very exciting and

```
Input: A problem instance I = (V, E, w, O, R) and a common deadline D
Output: A feasible schedule S or NULL if no such schedule exits
begin
    S ← ∅
    Order nodes in V into postorder
    foreach i ∈ V in postorder do
        K_i ← ∅, L_i ← ∅
        foreach child k of i do
            if c_k ≠ ∞ then K_i ← K_i ∪ {k}
            if d_k ≠ ∞ then L_i ← L_i ∪ {k}
        end
        if i ∈ O then s_i ← 0
        else if K_i = ∅ then s_i ← ∞
        else s_i ← c_{k'} + w_{ik'} = min_{k∈K_i}{c_k + w_ik}
        if L_i = ∅ then
            if i ∉ R OR s_i ≤ D then
                c_i ← s_i, d_i ← ∞
                if s_i ≠ ∞ AND s_i ≠ 0 then
                    // add communication τ into S
                    S ← S + τ where τ = (k', i, c_{k'}, c_{k'} + w_{ik'})
                end
            else
                c_i ← ∞, d_i ← D
            end
        else
            m ← |L_i|
            sort L_i by deadlines in a non-decreasing order
            L_i = {l_1, ..., l_m} // sorted in a non-decreasing order
            τ_m ← (i, l_m, d_m − w_{il_m}, d_m)
            for j ← m − 1 downto 1 do
                t_{l_j} ← min{t_{l_{j+1}} − w_{il_{j+1}}, d_{l_j}}
                τ_j ← (i, l_j, t_{l_j} − w_{il_j}, t_{l_j})
            end
            if τ_1.t_s < 0 then // τ_1.t_s must not be smaller than time 0
                return NULL
            end
            if s_i > τ_1.t_s then // T_i is not self-fulfilled
                c_i ← ∞, d_i ← τ_1.t_s
            else
                d_i ← ∞
                if i is not root then
                    // compute c_i
                    p ← parent(i)
                    Let a_j represents τ_j.t_s and b_j represents τ_j.t_e,
                    1 ≤ j ≤ m
                    Let b_0 ← s_i and a_{m+1} ← ∞
                    for j ← 0 to m + 1 do
                        idle_period ← a_{j+1} − b_j
                        if idle_period ≥ w_ip then
                            c_i ← b_j
                            break
                        else
                            τ_{j+1} ← (i, l_{j+1}, b_j, b_j + w_{il_{j+1}})
                        end
                    end
                end
            end
            // Add communications τ_j into S, 1 ≤ j ≤ m
            for j ← 1 to m do
                S ← τ_j
            end
        end
    end
    if d_root = ∞ then // T is self-fulfilled.
        return S
    else T is not self-fulfilled
        return NULL
    end
end
```

Figure 3. The pseudo code of FindFeasible algorithm

challenge, and we give two such problems, which we will further investigate.

- When multiple messages are present in the distributed systems, is there a polynomial time algorithm for the replication transition problem on tree topology or other topologies such as paths and cycles?
- Is there a polynomial time algorithm for replication transition problem when multiple messages are present and each site has limited storage capacity?

6 Acknowledge

The authors would like to acknowledge the anonymous reviewers for their valuable advises. This research is supported in part by the National Science Council, Republic of China, under Grant NSC 96-2221-E-002-025 , and by the Ministry of Education, Republic of China, under the national project 95R0062-AE00-07.

References

[1] EU DataGrid. http://www.edg.org.
[2] Grid Physics Network (GriphyN). http://www.griphyn.org.
[3] LCG: LHC Computing Grid. http://lcg.web.cern.ch/LCG/.
[4] TeraGrid. http://www.teragrid.org.
[5] J. H. Abawajy. Placement of File Replicas in Data Grid Environments. In *International Conference on Computational Science*, pages 66–73, 2004.
[6] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Pipelining broadcasts on heterogeneous platforms. *IEEE Transactions on Parallel and Distributed Systems*, 16(4):300–313, 2005.
[7] R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Improved results for data migration and open shop scheduling. *ACM Transactions on Algorithms*, 2(1):116–129, 2006.
[8] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *SODA*, pages 223–232, 2000.
[9] T. F. Gonzalez. Simple algorithms for multimessage multicasting with forwarding. *Algorithmica*, 29(4):511–533, 2001.
[10] T. F. Gonzalez. An efficient algorithm for gossiping in the multicasting communication environment. *IEEE Transactions on Parallel and Distributed Systems*, 14(7):701–708, 2003.
[11] J. Hall, J. D. Hartline, A. R. Karlin, J. Saia, and J. Wilkes. On algorithms for efficient data migration. In *SODA*, pages 620–629, 2001.
[12] W. Hoschek, F. J. Jaén-Martínez, A. Samar, H. Stockinger, and K. Stockinger. Data Management in an International Data Grid Project. In *GRID 2000*, pages 77–90, 2000.
[13] X. Jia, D. Li, X.-D. Hu, and D.-Z. Du. Placement of read-write web proxies in the internet. In *ICDCS*, pages 687–690, 2001.

[14] K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):628–637, 2001.

[15] S. R. Kashyap and S. Khuller. Algorithms for non-uniform size data placement on parallel disks. In *FSTTCS*, pages 265–276, 2003.

[16] S. Khuller, Y. A. Kim, and Y.-C. J. Wan. Algorithms for data migration with cloning. *SIAM Journal of Computing*, 33(2):448–461, 2004.

[17] S. Khuller, Y. A. Kim, and Y.-C. J. Wan. On generalized gossiping and broadcasting. *Journal of Algorithms*, 59(2):81–106, 2006.

[18] Y. A. Kim. Data migration to minimize the total completion time. *Journal of Algorithms*, 55(1):42–57, 2005.

[19] C. Krick, H. Räcke, and M. Westermann. Approximation algorithms for data management in networks. In *SPAA '01*, pages 237–246, New York, NY, USA, 2001. ACM Press.

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

[21] H. Lamehamedi, Z. Shentu, B. K. Szymanski, and E. Deelman. Simulation of Dynamic Data Replication Strategies in Data Grids. In *IPDPS 2003*, page 100, 2003.

[22] Y.-F. Lin, P. Liu, and J.-J. Wu. Optimal placement of replicas in data grid environments with locality assurance. In *ICPADS (1)*, pages 465–474, 2006.

[23] K. Ranganathan and I. T. Foster. Identifying Dynamic Replication Strategies for a High-Performance Data Grid. In *GRID 2001*, pages 75–86, 2001.

[24] M.-X. Tang and M.-J. Xu. QoS-aware replica placement for content distribution. *IEEE Transactions on Parallel and Distributed Systems*, 16(10):921–932, 2005.

[25] C.-M. Wang, C.-C. Hsu, H.-M. Chen, and J.-J. Wu. Efficient multi-source data transfer in data grids. In *CCGRID '06*, pages 421–424, 2006.

[26] C.-M. Wang, C.-C. Hsu, P. Liu, H.-M. Chen, and J.-J. Wu. Optimizing server placement for qos requirements in hierarchical grid environments. In *GPC*, pages 181–192, 2007.

[27] H. Wang, P. Liu, and J.-J. Wu. A QoS-aware heuristic algorithm for replica placement. In *International Conference on Grid Computing*, pages 96–103, 2006.