PAPER Special Section on Parallel and Distributed Computing and Networking

Tree-Based Consistency Maintenance Scheme for Peer-to-Peer File Sharing of Editable Contents*

Taishi NAKASHIMA[†], *Nonmember and* **Satoshi FUJITA**[†a)], *Member*

SUMMARY This paper proposes a consistency maintenance scheme for P2P file sharing systems. The basic idea of the proposed scheme is to construct a static tree for each shared file to efficiently propagate the update information to all replica peers. The link to the root of the trees is acquired by referring to a Chord ring which stores the mapping from the set of shared files to the set of tree roots. The performance of the scheme is evaluated by simulation. The simulation result indicates that: 1) it reduces the number of messages in the Li's scheme by 54%, 2) it reduces the propagation delay of the scheme by more than 10%, and 3) the increase of the delay due to peer churns is effectively bounded provided that the percentage of leaving peers is less than 40%.

key words: *Peer-to-Peer, file sharing, consistency maintenance, distributed hash table*

1. Introduction

Recently, Peer-to-Peer (P2P) file sharing systems have attracted considerable attention as a method to share digital contents through the Internet. In classical P2P file sharing systems such as Gnutella [10] and WinMX [11], given shared files cannot be updated by the end-users while they can be read out by any users. However, in these years, there have been proposed several file sharing applications which allow end-users to update shared files, which includes online storage systems such as OceanStore [8], mutable content sharing such as P2P WiKi [3], P2P collaborative workspace [2], and others.

Many P2P file sharing systems dynamically generate replicas of shared files and distribute them over several peers called **replica peers** to enjoy the following advantages of P2P systems [4]:

- To reduce the load of the owner of popular files accessed by many users.
- To enable participants to access (replica of) files even after the file owner leaves.
- To reduce the time before finding a replica under flooding-based query propagation schemes.

A scheme which updates all replicas of a given file to be consistent with the update of the original file is called the **consistency maintenance scheme**. A typical scenario which needs such a consistency maintenance is the realization of a forum of users in P2P environments, in which each user subscribes to articles in a thread published by other end-users (editors) which is locally cached by each subscriber to reduce the response time. The simplest way to realize such a consistency maintenance in a P2P environment is that the file owner keeps the information on all replica peers and when the file is updated, the owner directly notifies the update information to all replica peers. Although it is simple and natural, it has a drawback such that the load of the file owner becomes heavy as the number of replicas increases (it is reported that such a naive method works well if the number of replica peers is less than ten [14]). To overcome such a drawback of the simple scheme, several consistency maintenance schemes such as push/pull [9] and IRM (Integrated file Replication and consistency Maintenance mechanism) [5] have been proposed in the literature. Among those proposals, the scheme proposed by Li *et al.* realizes an efficient consistency maintenance using the following sophisticated techniques: 1) a Chord ring [7] is prepared for each shared file to store the set of replica peers, 2) after conducting an update, the updater dynamically extracts a tree from the Chord ring to realize an efficient tree-based notification to the replica peers, and 3) the churn of peers is effectively tolerated by the underlying Chord protocol.

In this paper, we propose a new consistency maintenance scheme for P2P file sharing systems based on the *construction of a static tree for each shared file*. More concretely, in the proposed scheme, each file is associated with a static tree, and each updater initializes the notification of update information by sending a message to the root of the tree which is not fixed unlike conventional owner-based schemes [5], [9]. To enhance the tolerance to the peer churn, we prepare links to few ancestors for each vertex in the tree in addition to links to the children and prepare several shadow peers for each root peer. The link to the root in a tree is acquired by referring to a Chord ring which registers the mapping from the set of shared files to the set of roots (and their shadow peers) of the trees. Since such an acquisition takes $O(\log X)$ sequential steps where X is the size of the Chord ring, in this paper, we take an approach such that a newly joined peer selects the root as the contact point with probability p and another known peer with probability $1 - p$ for an appropriate parameter p. The performance of the scheme is evaluated by simulation. The

Manuscript received December 24, 2013.
Manuscript revised April 7, 2014.

[†]The authors are with the Department of Information Engineering, Graduate School of Engineering, Hiroshima University, Higashihiroshima-shi, 739–8527 Japan.

*Earlier version of this paper was presented at "Tree-Based Consistency Maintenance Scheme for Peer-to-Peer File Sharing Systems," by Taishi Nakashima and Satoshi Fujita, In Proc. CANDAR 2013.

a) E-mail: fujita@se.hiroshima-u.ac.jp

DOI: 10.1587/transinf.2014PAP0007

result of simulations indicates that: 1) the proposed scheme reduces the number of messages in the Li's scheme by 54%, 2) the propagation delay of update messages is shorter than the Li's scheme and gradually decreases as the value of p increases, and 3) the increase of the delay due to peer churns is effectively bounded provided that the percentage of leaving peers is less than 40%.

The remainder of this paper is organized as follows. Section 2 overviews related works. Section 3 describes the proposed scheme. Section 4 shows the result of simulations. Finally, Sect. 5 concludes the paper with future work.

2. Related Work

Push/pull proposed by Lan *et al.* is a typical consistency maintenance scheme for P2P file sharing systems [9]. In this scheme, a peer which generated a shared file serves as the "owner" of the file, and the consistency maintenance procedure is always initiated by the owner peer triggered by the update information received from an updater. Under this mechanism, the file held by the owner is always the newest among all replicas. After updating the original file, the owner "pushes" the update information to the nearby peers using a flooding with an appropriate TTL (time-to-live), while peers outside of the TTL can acquire the update information by "pulling" the latest replica by conducting a periodical polling. As such, push/pull puts high responsibility on the owner peer rather than each updater. The load of the owner can be reduced by increasing TTL, but since a large TTL increases the load of non-replica peers, it is difficult to reduce the overall load by merely tuning the TTL.

In IRM [5], the update information is notified to replica peers merely by conducting pulling. In this scheme, the rate of pulling is determined so as not to exceed the update rate nor the query rate, where the update rate of file f is the frequency of update of f by a peer and the query rate of peer u on f is the rate of acquiring f by u. The reader should note that if the pulling rate is higher than the update rate, the probability of acquiring new update information becomes low, and if the query rate is higher than the update rate, the most of acquired information could not be used to update acquired replica. Although IRM could reduce the redundant pulling by tuning the pulling rate, it cannot adapt itself to the change of the update rate. In addition, although it can certainly bound the pulling frequency, since it must not be lower than a certain value, the load of the owner linearly increases as the number of replicas increases.

The drawback of the above two schemes could be overcome by using a "structure of replica peers" as in SCOPE (Scalable COnsistency maintenance in structured PEer-to-peer systems) [12] and the Li's scheme [14]. In the following, we will merely focus on the Li's scheme, since the superiority of the Li's scheme to SCOPE was experimentally confirmed in [14]. The outline of the Li's scheme is as follows. At first, it organizes a Chord ring [7] of replica peers *for each* shared file. The insertion and the deletion of peers to/from the ring are conducted according to the Chord protocol. When a replica peer x updates a shared file f, x "extracts" a tree structure rooted at x from the Chord ring corresponding to f, and propagates the update information along with the extracted tree from the root to all leaves.

The extraction of such tree is conducted as follows (in the following, we use parameter ℓ to denote the number of children of each peer in the extracted tree).

1. Assign all replica peers to x as the territory of x. Let $A(x)$ denote the set of peers assigned to x.
2. Peer x partitions $A(x)-\{x\}$ into ℓ subsets $A_1, A_2, \ldots, A_\ell$, and after letting the first peer u_i in A_i be the i^{th} child of x for each i, it hands over A_i to u_i as the territory of u_i.
3. Repeat the same procedure until the territory of all peers becomes a singleton.

The territory of each peer is given as a "chord" in the Chord ring, and the partition of a territory is realized by splitting the corresponding chord into ℓ sub-chords. In addition, the first peer in a territory means the first peer which is found by conducting the clock-wise traversal of the ring from the current peer. In the following, we will explain the behavior of the procedure using a simple example shown in Fig. 1. Let $\ell = 2$ and suppose that peer 0 updates a shared file corresponding to the Chord ring. Then a tree rooted at peer 0 is extracted from the ring as follows:

1. Let $A(0) := [0, 15]$ be the territory of peer 0.
2. Peer 0 partitions $A(0) - \{0\}$ into two ($= \ell$) subsets $[1, 8]$ and $[9, 15]$. The children of 0 in the tree are the first peer 1 in the first subset and the first peer 10 in the second subset. Peer 0 assigns a territory to each child such that $A(1) := [1, 8]$ and $A(10) := [9, 15]$.
3. Peer 1 partitions $A(1) - \{1\}$ into two subsets $[2, 5]$ and $[6, 8]$, and assigns a territory to children 2 and 6 as $A(2) := [2, 5]$ and $A(6) := [6, 8]$, respectively. Peer 10 partitions $A(10) - \{10\}$ into two subsets $[11, 13]$ and $[14, 15]$. Since $[14, 15]$ contains no peer, the first peer 12 in $[11, 13]$ becomes the unique child of peer 10.

By repeating the same operation, we have a tree structure depicted in Fig. 2. The propagation of the update information is conducted from the root to the leaves, which can be concurrently done with the extraction of the tree structure.

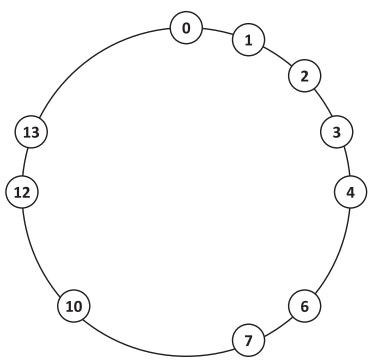


Fig. 1 Chord ring (to clarify the exposition, we omit shortcut links called "fingers" in this figure).

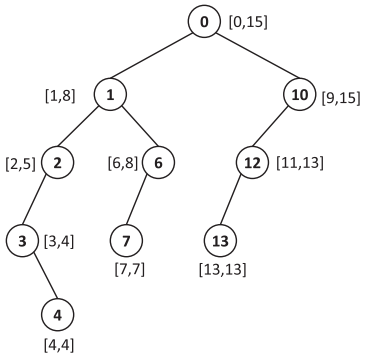


Fig. 2 Tree structure extracted from the Chord ring shown in Fig. 1.

The message propagation in the Li's scheme is much more efficient than push/pull, since it sends the update information merely to the replica peers. In addition, since such a propagation is conducted with the responsibility of the updater, it could avoid the concentration of the responsibility to a specific peer (i.e., single point of failure) as in IRM and push/pull. However, since it constructs a Chord ring for *every* shared file, the cost of each peer increases as the number of replicas held by the peer increases. In particular, since the number of adjacent peers in a Chord ring is $\Theta(\log X)$ even in the best case [7], where X is the number of peers in the ring, if it accesses many popular files, the cost of the peer increases super-linearly.

The idea of constructing a tree-structured overlay is used in many consistency maintenance schemes with various objectives. For example, Shen *et al.* recently proposed a poll-based consistency maintenance scheme called GeWave (Geographically aware Wave) [6] for the files geographically distributed over the network, and Hu *et al.* proposed a framework called BCoM (Balanced Consistency Maintenance) to tune the balance between the consistency strictness, the availability of files for updates, and the update dissemination latency [13]. The reader should note that the proposed scheme described in the next section can be applied to those schemes, since the key points of our scheme is that: 1) an efficient access to the tree roots through Chord ring, 2) several techniques to increase the churn tolerance, and 3) the probabilistic technique to balance the cost and the height of the resulting trees.

3. Proposed Method

3.1 Overview

In the proposed scheme, we statically construct a rooted tree for each shared file unlike the Li's scheme which dynamically constructs a rooted tree for each update request. As will be described later, the proposed scheme requests each peer to maintain constant number of neighbors for each shared file, which is much smaller than the Li's scheme which requests to maintain $\Theta(\log X)$ neighbors for each shared file.

We consider the following four issues during the design of the proposed scheme:

- To reduce the time required for the message transmission, we bound the number of children of each peer by constant d (appropriate value of d will be determined experimentally in Sect. 4).
- To reduce the time required for the message propagation, we balance the height of subtrees so that almost all leaves have the same depth. To this end, the proposed scheme includes a method to balance the depth of leaves subject to the constraint on the number of children.
- The leave of an internal peer easily disconnects the tree. The proposed scheme includes a procedure to tolerate the churn of peers so that the resulting graph after the leave easily becomes connected in many cases.
- To start the propagation of the update information through the tree, we need to find the root of the tree as quickly as possible. The proposed scheme includes a mechanism for such a quick identification of the root.

The reader should note that the proposed scheme can avoid the concentration of the responsibility to specific peers as in the Li's scheme and SCOPE. In fact, although an update message will always be delivered from the root of the tree, it does not mean that the responsibility is concentrated at the root, since every internal peer in the tree is responsible for the delivery of the message to the down streams. It should also be noted that in the proposed scheme, even if several peers try to update the same file at the same time, each update message is delivered to replica peers through the root of static tree associated with the file, i.e., those updates are naturally serialized at the tree root.

3.2 Identification of the Root

Let T_f denote the tree corresponding to shared file f, where the way of organizing T_f will be described later. When peer u updates f, after identifying the root of T_f, u sends the update information to the root, which will be propagated to all replica peers by repeating message forwarding along tree links. The identification of the root is realized by using a Chord ring. This Chord ring, which will be referred to as R hereafter, consists of all peers participating in the file sharing system and the join and the leave of peers is realized by using the normal Chord protocol (we could modify the scheme so that R consists of several selected super-peers with a high computational power). In R, the mapping from the set of shared files to the set of roots is stored in the form of key-value pairs such that the ID of f is the "key" and the ID of the root of T_f is the "value."

3.3 Addition of New Replica Peers

Each peer u in T_f locally stores two variables $Child_f(u)$ and $D_f(u)$, which represent the set of children and the number of descendants in tree T_f, respectively. Suppose that peer x newly accesses file f. After acquiring a replica of the

file from the system, peer x initializes its local variables as $Child_f(x) := \emptyset$ and $D_f(x) := 1$, and sends message $\mathsf{Add}(x)$ to peer y in the tree which is selected by using a procedure described later. After receiving $\mathsf{Add}(x)$, y increments $D_f(y)$ by one, and conducts the following operation:

- If $|Child_f(y)| < d$, then it adds x to $Child_f(y)$ and notifies the fact to x.
- Otherwise, it selects a peer in $Child_f(y)$ to have the smallest value of D_f and forwards $\mathsf{Add}(x)$ to the selected peer.

A child which received the forwarded message conducts the same operation, and such a forwarding is repeated until the parent of x is determined. Note that the above iteration always terminates since any path from the root to a leaf contains at least one peer y' such that $|Child_f(y')| < d$ (in fact, any leaf satisfies this condition).

The first receiver y of request $\mathsf{Add}(x)$ is determined as follows:

1. If x knows the IP address of the root of T_f, the root is selected as y with probability one.
2. Otherwise, the root of T_f is selected as y with probability p and the peer which sent a replica to x is selected as y with probability $1 - p$.

The reason of why we adopted such a probabilistic approach is as follows. If we fix the receiver of Add request to the root of the tree, we can guarantee the balance of the resulting tree even if the tree grows monotonically (as in the grow of a binary heap), although the identification of the root takes $\Theta(\log N)$ sequential steps unless it has known the IP address of it. On the other hand, if we select a peer which just sent a replica to the peer as y, although it eliminates the cost for the identification, we could not guarantee the balance of the resulting tree. In order to resolve such a trade-off, we take an approach such that: 1) it corrects the imbalance of the tree caused by the transmission to known peers by "inserting" the transmission to the root, and 2) to bound the cost of such a correction, we limit the probability of such an event to p.

3.4 Churn Tolerance

This subsection describes how to tolerate the churn of peers in the proposed scheme. As for the churn at the root of the tree, we tolerate the churn by preparing a backup of the root (the details will be described later). For the other cases, we tolerate the churn by preparing a variable Anc_f to remember a part of the ancestors for each peer. The leave of a peer could be detected by an adjacent peer by periodically exchanging message between adjacent peers, i.e., a peer can judge that an adjacent peer leaves when it misses consecutive α messages for some $\alpha > 0$. Upon detecting the leave of the parent, peer x sends message $\mathsf{ReAdd}(x, D_f(x))$ to peer y in the tree, where y is a peer selected by using a procedure described later. After receiving $\mathsf{ReAdd}(x, D_f(x))$, y increments $D_f(y)$ by $D_f(x)$, and conducts the following operation:

- If $|Child_f(y)| < d$, then it adds x to $Child_f(y)$ and notifies the fact to x.
- Otherwise, it selects a peer in $Child_f(y)$ to have the smallest value of D_f and forwards $\mathsf{ReAdd}(x, D_f(x))$ to the selected peer.

A child which received the forwarded message conducts the same operation, and such a forwarding is repeated until the parent of x is determined.

The first receiver y of request $\mathsf{ReAdd}(x, D_f(x))$ is determined as follows:

1. If x knows the IP address of the root of T_f, the root is selected as receiver y with probability one.
2. Otherwise, the root of T_f is selected as y with probability p and a peer *which is closest to the root among all peers in* Anc_f *currently participating in the system* is selected as y with probability $1 - p$.

In other words, peer y is probabilistically selected by considering the balance of the cost and the tree height similar to Add described in Sect. 3.3.

In order to tolerate the churn of the root peer, in the proposed scheme, we randomly nominate c (≥ 1) peers from the set of replica peers as the *shadow peers* and register the set of IDs of shadow and original peers as the value of key-value pairs to the Chord ring. Shadow peers of the root r of T_f share the replica of file f and variables $Child_f(r)$ and $D_f(r)$ with r. The original root r is responsible to notify any update of such information to all shadows. Under the procedure described in the last subsection, a peer u which updated the replica of file f detects the leave of r since it needs to contact the root of the tree associated with the replica (via the Chord ring if it is not aware of the IP address of the root). Upon detecting the leave of r, u randomly selects a peer from the set of shadow peers associated with r, and promotes it to the new root of the tree. On the other hand, if the root r detects that the number of shadows is smaller than c, it randomly selects a peer from $Child_f(r)$ and promotes it to a new shadow peer.

4. Evaluation

4.1 Setup

We evaluate the performance of the proposed scheme by simulation using PeerSim [1]. Parameters used in the simulations are as follows. We consider a P2P file sharing system consisting of 5000 peers and 5000 shared files. For each file, the initial distribution of the number of replica peers follows the Zipf's distribution. Each simulation is of length 1000 unit times, and during a simulation, each file is updated by a replica peer according to the Poisson distribution with $\lambda = 0.05$. The arrival of peers follows the Poisson distribution with $\lambda = 0.1$ and to keep the number of participants to be 5000, we delete a random peer when a new peer joins. Default values of p and d are set as $p = 0.5$ and $d = 16$,

respectively.

We evaluate the performance of the proposed scheme in terms of the following three metrics and compare it with the Li's scheme described in Sect. 2.

- The number of messages, which includes messages to propagate the update information and messages for the maintenance of data structure due to join and leave of the participant peers.
- Delay of update propagation, which is defined as the elapsed time from the point of update of a replica to the point at which all replica peers receive the update information. We are particularly interested in the impact of two parameters d and p to the delay, as well as the influence of the number of replica peers, i.e., scalability of the scheme.
- Churn tolerance of the schemes, which is evaluated by the impact of the churn rate to the delay of update propagation, where the definition of the churn rate is described later.

The reason of why we focus on the Li's scheme as a competitor is as follows: 1) it is a representative structured overlay designed for the consistency maintenance of P2P file sharing systems, 2) it is an extension of SCOPE [12] which is widely used as a reference scheme in many papers [6], [13], and 3) the superiority of the Li's scheme to SCOPE is confirmed in [14].

4.2 Number of Messages

At first, we evaluate the number of messages. The result for the Li's scheme and the result for the proposed scheme are summarized in Tables 1 and 2, respectively. In this table, for example, items starting with `CHORD_`, `LI_`, and `TCM_` are messages used for the management of the underlying Chord rings, messages used in the Li's scheme, and messages used in the proposed scheme, respectively; and `_UPDATE` means messages used to notify the update information.

We can make the following observations from the tables. Li's scheme issues more `CHORD_` messages than the proposed scheme, which is apparently because of the large number of Chord rings maintained by the scheme (recall that the proposed scheme maintains exactly one Chord ring in contrast to that). As for the number of update messages, Li's scheme issues more messages than the proposed scheme (i.e., 2.48×10^6 v.s. 1.68×10^6), which is because of a high overhead before conducting an actual message forwarding. In the proposed scheme, every update message is simply propagated from the root to the leaves through tree links connected beforehand. In contrast, in the Li's scheme, before conducting a message forwarding to the children, each peer should partition a given chord into several sub-chords and find the first peer in each chord to identify the set of children. Such operations requires additional messages compared with the proposed scheme, which results in a significant reduction of the total number of messages by the proposed scheme, i.e., it reduces the total number of messages of the Li's scheme by 54%.

Table 1 The number of messages in the Li's scheme.

CHORD_PING_PREDECESSOR	519028
CHORD_PONG_PREDECESSOR	518595
CHORD_PING_SUCCESSOR	519062
CHORD_PONG_SUCCESSOR	518548
CHORD_NOTIFY	522484
CHORD_FIND_JOIN	469
CHORD_RE_JOIN	86
CHORD_FIND_FIXFINGERS	401601
CHORD_RE_FIXFINGERS	126882
LI_UPDATE	2478815
LI_GET_REPLICA	86
LI_RE_GET_REPLICA	86
Total	5605742

Table 2 The number of messages in the proposed scheme.

CHORD_PING_PREDECESSOR	94362
CHORD_PONG_PREDECESSOR	93728
CHORD_PING_SUCCESSOR	94362
CHORD_PONG_SUCCESSOR	93728
CHORD_NOTIFY	94362
CHORD_FIND_JOIN	561
CHORD_RE_JOIN	79
CHORD_FIND_FIXFINGERS	284356
CHORD_RE_FIXFINGERS	62069
TCM_UPDATE	1647151
TCM_PING	453771
TCM_JOIN	13882
TCM_PARENT	24339
TCM_ROOT_WITHJOIN	69713
TCM_ROOTJOIN	13716
Total	3040179

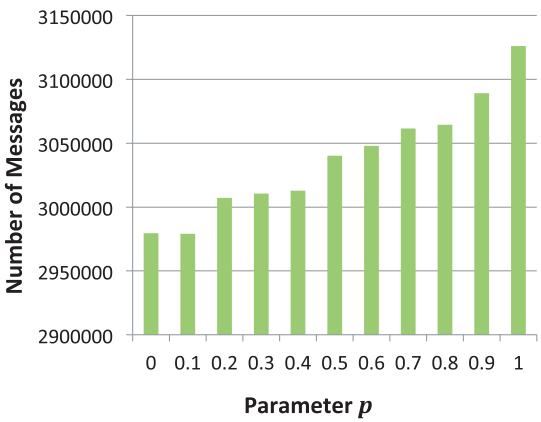
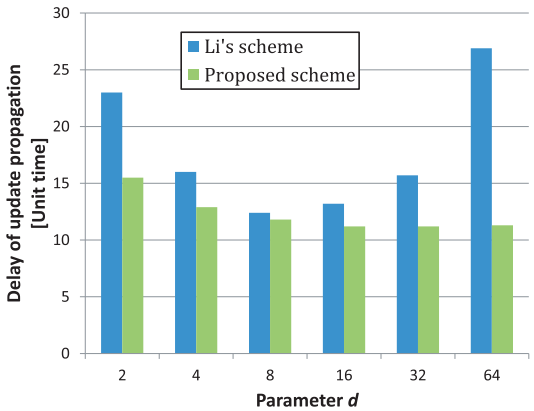


Fig. 3 The number of messages for each p.

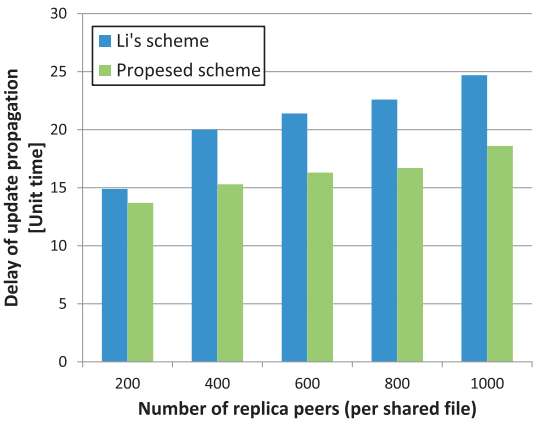
The number of messages in the proposed scheme depends on parameter p. Recall that p is the probability of selecting the root peer as the first receiver of the Add message and that the access to the root needs additional messages compared with the access to a known peer (see Sect. 3 for the details). Figure 3 illustrates the relationship between parameter p and the number of messages. The number of messages slightly increases as the value of p increases. In particular, the number of messages at $p = 1.0$ is 1.05 times of that at $p = 0.0$.

4.3 Delay of Update Propagation

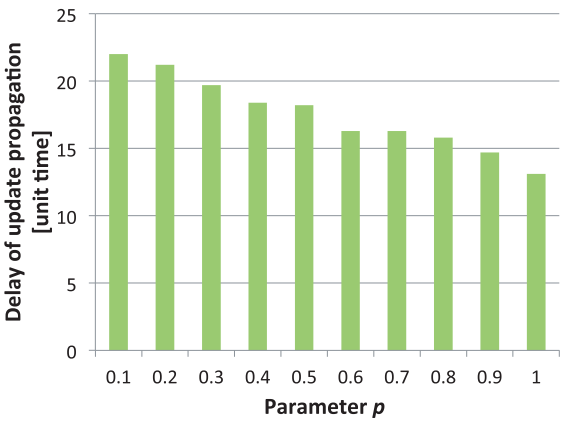
Next, we evaluate the impact of parameter d to the propagation delay (recall that d is the upper bound on the number of children of each peer and a large d generally reduces the height of the resulting tree). The result for 100 replica peers with $p = 0.5$ is summarized in Fig. 4 (a). Although the delay in the proposed scheme gradually reduces as d increases, in the Li's scheme, it rapidly "increases" after exceeding certain value, i.e., $d = 8$. The badness of the Li's scheme for large d's is due to the heavy cost to reconstruct the tree structure which is necessary before conducting an actual message forwarding [14]. Such a superiority of the proposed scheme does hold even when the number of replica peers becomes large. See Fig. 4 (b) for illustration. This figure summarizes the result for $d = 16$ and $p = 0.5$, where the horizontal axis is the number of replica peers. In general, the propagation delay increases as the height of the tree becomes large and the height and the imbalance of the resulting tree are (implicitly) controlled by parameter p. Figure 4 (c) summarizes the result, where the horizontal axis is the value of p ranged from 0.1 to 0.9 (in this figure, the number of replicas is fixed to 1000). As shown in the figure, the delay at $p = 1.0$ is 60% of the delay at $p = 0.1$, i.e., the shape of the tree certainly becomes balanced as p increases. Under the setting used in the simulation, values larger than 0.5 will give a better trade-off point for parameter p (probably 0.8 would be better than 0.5). However, the value of p giving the best trade-off point between the delay and the number of messages decreases as the number of participants increases, since it increases the cost of identifying the location of the root of the tree through the Chord ring, while for files to have many replicas should have larger p, since as the size of the tree increases, the merit of decreasing the delay dominates the demerit of increasing the number of messages. In this way, an optimal value of p would depend on: 1) the total number of peers in the Chord ring and 2) the number of peers participating in each tree.



(a) Impact of parameter d (100 replica peers, $p = 0.5$).



(b) Impact of the number of replica peers ($d = 16$, $p = 0.5$).



(c) Impact of parameter p in the proposed scheme (1000 replica peers, $d = 16$).

Fig. 4 Delay of update propagation.

The reader should note that although the reduction of the delay of update propagation does not directly reduce the access time of each peer since each replica peer holds its own copy in its local storage, such a reduction of the propagation delay certainly improves the users' experience since it realize the sharing of the latest copy of the shared file existing in the network.

4.4 Churn Tolerance

Let us consider a situation in which D [%] of the participating peers "simultaneously" leave the system (since we fix the number of participants to 5000, for example, $D = 10\%$ means that 500 peers simultaneously leave). In the following, we call D the churn rate. In this subsection, we evaluate the impact of D to the delay of the schemes by varying D from 0% to 50%. Figure 5 summarizes the result for a file with 500 replica peers. From the figure, we can observe that in both schemes, the delay increases as D increases, but the delay of the proposed scheme increases more rapidly than the Li's scheme for large D.

The badness of the proposed scheme for large D's can be explained as follows. In the proposed scheme, the leave of a peer disconnects the given tree into subtrees and peers in a subtree which does not contain the root peer can receive the update information issued by other peers only after connecting with a subtree containing the root peer. The transition of the average delay after the simultaneous leaves is illustrated in Fig. 6 (the horizontal axis is the time units and

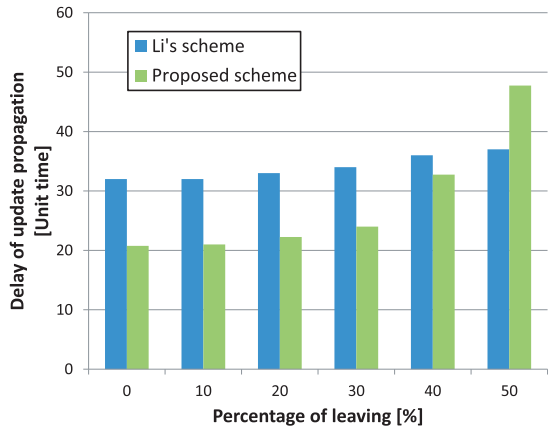


Fig. 5 Churn tolerance of the schemes.

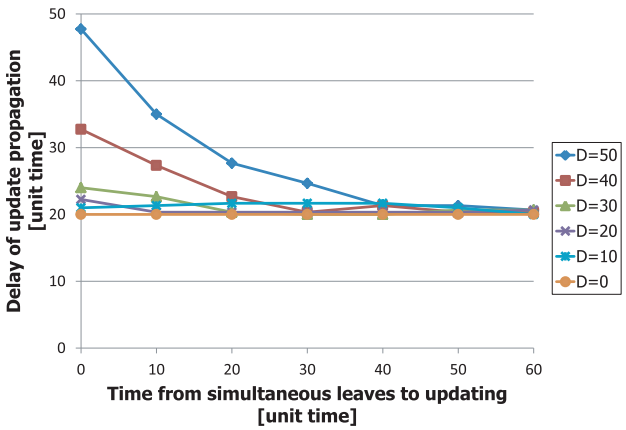


Fig. 6 Convergence of the delay after the simultaneous leaves.

each curve represents the convergence of the delay for each D). As shown in the figure, the time before the convergence increases as the value of D increases. When the churn rate is not high, such a re-connection completes in a relatively short time, but as the churn rate increases, the time before completing the re-connection increases because of the conflict of requests at the root peer. Such a long waiting time does not occur in the Li's scheme because in this scheme, each Chord ring has a sufficient redundancy under a random selection of leaving peers and the extraction of a tree structure is always possible if the underlying Chord ring is connected, while the leave of peers would (slightly) degrade the routing performance of the Chord ring. In fact, the delay in the Li's scheme increases from 32 to 37 (i.e., the amount of increase is less than 20%) by increasing the churn rate from 0% to 50%, whereas the delay in the proposed scheme increases from 20 to 48 (i.e., it increases to more than twice) in the same range of the churn rate.

The above result indicates that the proposed scheme is more robust against peer churns than the Li's scheme provided that the churn rate is relatively small (e.g., less than 40%) and if the churn rate is expected to be very high, we should use other schemes such as the Li's scheme.

5. Concluding Remarks

This paper proposes a consistency maintenance scheme for P2P file sharing systems based on the construction of static trees for each shared file. Unlike the Li's scheme which is regarded as the most efficient scheme in the literature, the proposed scheme uses exactly one Chord ring for the lookup of the root of the trees and the churn of peers is effectively tolerated by preparing links to few ancestors and by adopting the notion of shadow roots. The result of simulations indicates that the proposed scheme outperforms the Li's scheme in terms of the number of messages and the propagation delay of update messages. In addition, it improves the churn tolerance of the Li's scheme particularly when the churn rate is less than 40%.

A future work is an extensive comparison with other schemes such as push/pull and IRM. An improvement of the proposed scheme by adaptively selecting the number of children in each tree is another challenging issue.

References

[1] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," Proc. IEEE 9th International Conference on Peer-to-Peer Computing, pp.99–100, 2009.
[2] G. Oster, P. Urso, P. Molli, and A. Imine, "Data consistency for P2P collaborative editing," Proc. 20th Anniversary Conference on Computer Supported Cooperative Work, pp.259–268, 2006.
[3] G. Urdaneta, G. Pierre, and M.V. Steen, "A decentralized Wiki engine for collaborative Wikipedia hosting," Proc. 3rd IEEE Workshop on Internet Applications, pp.156–163, 2007.
[4] H.-Y. Shen, "An efficient and adaptive decentralized file replication algorithm in P2P file sharing systems," IEEE Trans. Parallel Distrib. Syst., vol.21, no.6, pp.827–840, 2010.
[5] H.-Y. Shen, "IRM: Integrated file replication and consistency maintenance in P2P systems," IEEE Trans. Parallel Distrib. Syst., vol.21, no.1, pp.100–113, 2010.
[6] H.-Y. Shen and G.-X. Liu, "A geographically aware poll-based distributed file consistency maintenance method for P2P systems," IEEE Trans. Parallel Distrib. Syst., vol.24, no.11, pp.2148–2159, 2013.
[7] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications, IEEE/ACM Trans. Netw., vol.11, no.1, pp.17–32, 2003.
[8] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An architecture for global-scale persistent storage," ACM SIGPLAN Notices, vol.35, no.11, pp.190–201, 2000.
[9] J. Lan, X.-T. Liu, P. Shenoy, and K. Ramamritham, "Consistency maintenance in peer-to-peer file sharing networks," Proc. 3rd IEEE Workshop on Internet Applications, pp.90–94, 2003.
[10] Gnutella Protocol Specification. http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html
[11] WinMX. http://www.winmxworld.com/
[12] X. Chen, S. Ren, H.N. Wang, and X.-D. Zhang, "SCOPE: Scalable consistency maintenance in structured P2P systems," Proc. IEEE INFOCOM, vol.3, pp.1502–1513, 2005.
[13] Y. Hu, L.N. Bhuyan, and M. Feng, "Maintaining data consistency in structured P2P systems," IEEE Trans. Parallel Distrib. Syst., vol.23, no.11, pp.2125–2137, 2012.

[14] Z.-Y. Li, G.-G. Xie, and Z.-C. Li, "Efficient and scalable consistency maintenance for heterogeneous peer-to-peer systems," IEEE Trans. Parallel Distrib. Syst., vol.19, no.12, pp.1695–1708, 2008.

Taishi Nakashima received his B.E. degree in information engineering from Hiroshima University in 2013. His current research interests include peer-to-peer networks and distributed systems.

Satoshi Fujita received the B.E. degree in electrical engineering, M.E. degree in systems engineering, and Dr.E. degree in information engineering from Hiroshima University in 1985, 1987, and 1990, respectively. He is a Professor at Graduate School of Engineering, Hiroshima University. His research interests include communication algorithms, parallel algorithms, graph algorithms, and parallel computer systems. He is a member of the Information Processing Society of Japan, SIAM Japan, IEEE Computer Society, and SIAM.