An Efficient Update Propagation Algorithm for P2P Systems

Zhijun Wang[a], Sajal K. Das[b], Mohan Kumar[b] and Huaping Shen[c]

[a]*Department of Computing, The Hong Kong Polytechnic University, Hong Kong*

[b]*Center for Research in Wireless Mobility and Networking (CReWMaN)*
Department of Computer Science and Engineering
The University of Texas at Arlington, Arlington, TX 76019, USA

[c]*Ask Jeeves Inc., Piscataway, NJ, 808854, USA*

Abstract

As more and more applications with dynamic files are introduced in peer-to-peer (P2P) systems, file consistency maintenance becomes important. In this paper, we propose a novel file consistency maintenance algorithm, called *U*pdate *P*ropagation *T*hrough *Re*plica *C*hain (UPTReC), for decentralized and unstructured peer-to-peer (P2P) systems. UPTReC provides a probabilistically guaranteed file consistency. In UPTReC, each file has a logical replica chain composed of all replica peers (RPs) which is defined as a peer that has a replica of the file. Each RP acquires partial knowledge of the bi-directional chain by keeping a list of information about k nearest RPs in each direction. When an RP initiates an update, it pushes the update to all possible online (active) RPs through the replica chain. A reconnected RP pulls an online RP to synchronize the replica status and the chain information. An analytical model is derived to evaluate the performance of the UPTReC algorithm. The analytical results provide insights of the system design in choosing the parameters. Simulation experiments are conducted to compare the performance with an existing update propagation algorithm based on the rumor spreading scheme. The experimental results show that UPTReC can significantly reduce (up to 70%) overhead messages and also achieve smaller stale query ratio for files prone to frequent updates.

Key words: Peer-to-Peer, Consistency, file replica

1 Introduction

The peer-to-peer (P2P) systems are self-organizing distributed systems. There are two types of decentralized P2P systems: structured and unstructured. In a structured P2P system, the topology is tightly controlled and the files are well deployed [16] [19]. On the other hand, an unstructured P2P system has no central control of its topology and file placement [1][3] [5]-[8] [10] [14]. Chord [19] is an example of a structured P2P system whereas Gnutella [1] is an example of an unstructured P2P system. Compared to structured P2P systems which provide efficient file searches through distributed hash table (DHT), unstructured P2P system can support key word search and are efficient for extremely transient peers. In this paper, we focus on the decentralized and unstructured P2P systems, in which some files may be heavily replicated to improve the file availability, reduce file search cost and enhance system fault-tolerance. In order to make such systems scalable and efficient, significant efforts have been made on the development of search and replication algorithms [5] [8][10][14]. In these algorithms, the locations of replicas for a file are well deployed based on partial knowledge of the system to minimize the search cost and balance the network load. However, these algorithms assume that the files are rather static and updates occur very infrequently. Indeed, the impact of the file update has not received much attention.

As more and more applications, such as trust management [3], bulletin-board systems and distributed web cache [11], become available in P2P systems, the file consistency issues become important. Consider an update on a dynamic file, without update propagation, even if the replicas are invalidated properly, the well placed replicas may no longer exist, thus resulting in a large search cost or even unavailable for incoming file queries. Therefore, effective propagation of update to all replica peers (RPs), which are defined as peers that have the replicated files, is critical to maintain balanced network load, enhance file availability, and reduce access latency. Hence there is a need to develop efficient update propagation algorithms for P2P systems.

In this paper, we propose an efficient update propagation algorithm, called **U**pdate **P**ropagation **T**hrough **Re**plica **C**hain (UPTReC), for decentralized and unstructured P2P systems. UPTReC provides a probabilistically guaranteed file consistency. In the algorithm, each file has a logical replica chain composed of all RPs. Each RP has partial knowledge of the bi-directional chain by keeping information (i.e., identity (ID) and IP address) of k nearest RPs (called *probe* peers) in each direction. The replica chain can be naturally built and easily maintained during the file replica process. When an RP updates a file, it pushes the update to its online (i.e., active) probe peers in both direction. In each direction, the farthest online probe peer is in charge to forward the update to its online probe peers along the direction. This process is recursively executed to propagate the update to all possible online RPs. When an offline (i.e., inactive) RP gets reconnected, it pulls an online probe peer to synchronize the file status and the probe peers' information. If the RP's IP address is changed, the new IP address is pushed to all possible online probe peers which in turn update the recorded information of the reconnected RP. A preliminary version of this paper appeared in [20].

An analytical model is derived for the UPTReC algorithm. The optimal number of probe peers is calculated to achieve the minimum file consistency maintenance overhead messages. The numerical results show that UPTReC is efficient and scalable. The simulations are also conducted, the results show that UPTReC significantly reduces (up to 70%) overhead messages for update propagation compared to that of the rumor spreading based algorithm [9].

The rest of the paper is organized as follows. Section 2 gives an overview of the related work. A detailed description of UPTReC is given in Section 3. An analytical model is derived in Section 4, and numerical results are given in Section 5. Section 6 presents performance comparisons of UPTReC with an existing propagation algorithm. The conclusions are drawn in Section 7.

2 Related Work

The problem of searching and replicating files in P2P systems has received much attention. However, most existing P2P systems consider files to be static and do not address the file updating issues. In this section, we present an overview of file consistency maintenance algorithms in both structured and unstructured P2P systems.

2.1 Structured P2P systems

File consistency maintenance in structured P2P systems is relatively simpler compared to that in unstructured P2P systems due to the well-defined file locations. In [17], a controlled update propagation (CUP) algorithm was proposed to maintain consistency of cache index entries for structured P2P systems. CUP asynchronously builds caches of index entries while the files are searched and queried. However, the scheme only caches metadata and hence only provides limited consistency. More recently, an efficient scheme called Scalable COnsistency maintenance in structured PEer-to-peer system (SCOPE) was developed in [6]. SCOPE builds a replica-partition tree and keeps track of the locations of replicas through DHTs. However, this scheme cannot be applied to unstructured P2P systems.

2.2 Unstructured P2P systems

In [9], a hybrid push/pull update propagation algorithm based on the rumor spreading scheme was proposed for unstructured P2P systems, such as Gnutella [1] and P-Grid [2]. The algorithm provides probabilistically guarantees rather than strict consistency. In the algorithm, each RP maintains a subset of all RPs as its responsible peers. When an RP initiates an update, the update is pushed to its responsible peers, which in turn propagate the update to their responsible peers with some probabilities. This process continues until all possible online peers get the update. When a peer gets reconnected, it queries multiple responsible peers to synchronize itself with the peer having the most recent update. The algorithm in [9] is the first attempt to focus on the effective propagation of *updates* to RPs in decentralized and unstructured P2P systems. However, the overhead messages due to push updates are significant. Moreover, the maintenance of the subset of responsible peers is not easy, especially for RPs with dynamic IP addresses. The discussion on how to maintain the responsible subset is not presented in [9].

An invalidation report based on push and pull (PAP) algorithm is developed in [12] [13]. In PAP, each file has a master peer, only the master peer can update the file. An estimated Time-To-Expire (TTE) and the master peer information are associated with each replica. When a file is updated, its invalidation report is broadcast to the network. Any online peers that have replicas of the file invalidate the replicas. Once the TTE of a file expires, the file must be pulled from the master peer if it is accessed. Only the master peer updating the file is a strong constraint in P2P systems. Moreover, the master peer may change its IP address and go offline, thus resulting in a small probability of an RP successfully pulling a master peer.

3 Update Propagation Through R*e*plica Chain (UPTReC)

In this section, we present the details of the proposed Update Propagation Through R*e*plica Chain (UPTReC) algorithm. The main motivation behind UPTReC is to minimize the overhead messages for propagating updates to RPs in decentralized and unstructured P2P systems.

3.1 System Model and Assumptions

We consider decentralized and unstructured P2P systems, such as Gnutella where all peers are equal and no peer has a global view of the system. A peer frequently joins (online) and leaves (offline) the system and has some probability to change its IP address for each reconnection. The physical connectivity and topology are ignored. We assume that an online peer can communicate with the other if it knows the IP address of that peer. This is not a strong assumption because if two peers cannot communicate with each other, they can perceive each other as offline. In summary, the assumptions are as follows:

(1) No strong file consistency is required, but a probabilistically guaranteed file consistency is required.
(2) All peers frequently join and leave the system.
(3) An online peer that gets an update has the ability to finish its push process.
(4) An online peer can communicate with any other online peer if it knows the IP address of that peer.
(5) The physical connectivity and system topology are ignored.
(6) Each RP has an ID and an IP address, the ID is fixed but the IP address may be changed for each reconnection.
(7) Each file is associated with a version and generation time used for synchronization.

The probability of an online peer to successfully finish its push process is usually over 0.95 [9]. If the probability is low for a system, the assumption (3) can be remedied by using a reliable push process. In a reliable push process, the push process of RP_a does not stop after it propagates the update to RP_b, which in turn forwards the update to other RPs. RP_a must wait for the confirmation from RP_b indicating the update has been successfully propagated. If the confirmation is not received within a certain period, RP_a probes RP_b again; and if RP_b is offline, RP_a contacts another RP to continue the push process. The reliable push process incurs some additional overhead messages for confirmation. Due to low fail rates, we make assumption (3) to simplify our algorithm analysis.

3.2 Push Update Through Replica Chain

Figure 1 (a) shows a logical replica chain for a file with N replicas. Each RP is a node on the chain and has a unique ID. For simplification, node and RP are used alternatively in the following of the paper. Each node maintains information (i.e., ID and IP address) about k (typically k is tens) nearest nodes in each (left and right) direction of the chain. These $2k$ nodes are called *probe* nodes (peers). Two nodes are said to have h-hop distance if there are h-1 nodes between them. For example, node i and node i+k-1 have k-hop distance.

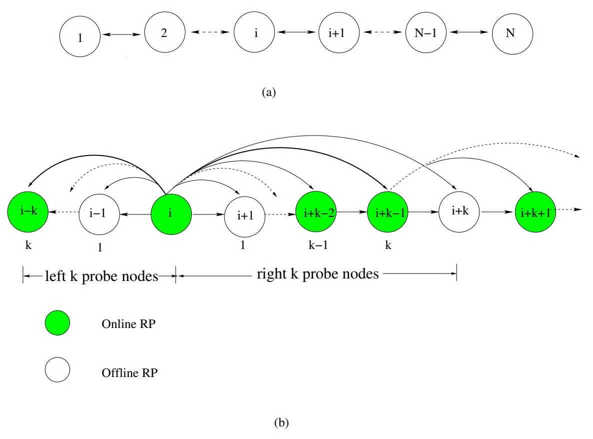

Fig. 1. (a) Logical Replica Chain; (b) Update propagation of node i. Node $i \pm m$ ($0 < m \leq k$) is the m^{th} probe node in right (left) direction.

Figure 1 (b) shows the update propagation process of node i. When node i initiates an update (node i is called the update initiating node), the update is pushed symmetrically along both left and right directions of the chain.

Now let us show the process of node i pushing the update to node N (right side of the chain). Node i has information of k probe nodes in right direction (from node i+1, called the 1^{st} probe node, to node i+k, called the k^{th} probe node). To propagate the update, node i sends a probe message to each of the probe nodes in this direction (i.e., node i+k, ..., i+1). The farthest online probe node (here node i+k-1) is chosen to be the update *relay* node, which will further propagate the update through the chain along the direction. All other online probe nodes of i, such as node i+k-2, will receive but do not propagate the update. After node i determines its update relay node i+k-1, it first sends the update to that node with the relay flag bit set as 1 and then sends the update to all other online probe nodes with the relay flag bit set as 0. When an online probe node receives the update, it first checks the update relay flag bit. If the bit is 0, it only needs to receive the update. Otherwise, it needs to propagate the update through the chain along the direction. The process of the update propagation is similar to node i except not to send the probe messages to its probe nodes which are also the probe nodes of i. Because all these nodes are probed by i and they should be offline. As shown in Fig. 1(b), when node i+k-1 gets the update, it finds that the update relay flag bit is 1, and hence it immediately sends the probe messages to its probe nodes in the right hand side which are not the probe nodes of i, i.e., nodes i+2k-1, ..., i+k+1. The process is repeatedly executed to propagate the update through the chain. If all k probe nodes of an update relay node are offline, the propagation process is stopped and the update cannot be propagated in this direction. The same process is executed for node i to propagate the update to node 1. Figure 2 gives the update propagation pseudo codes of a node in the chain.

```
***Procedure of Update Propagation of an RP:***
**IF** (Initiate an update)
    For ( all probe peers in both right and left hand side)
        Send a probe message
    Wait for probe peers' acknowledgment back
    For the farthest online probe peers in both right and left sides
        set the relay fag bit as 1, and send the update
    For other online probe peers
        set the relay flag bit as 0, and send the update
**IF** (Receive an update from $RP_j$ in right (left) side)
    IF (the relay flag bit is 1)
        For ( all probe peers which are not the probe peers of
            $RP_j$ in the left (right) side)
            Send a probe message
        Wait for probe peers' acknowledgment back
        For the farthest online probe peers
            set the relay fag bit as 1, and send the update
        For other online probe peers
            set the relay flag bit as 0, and send the update
```

Fig. 2. Procedure of Update Propagation of an RP

3.3 Pull After Online

An RP may go offline. During an RP's offline period, it may miss some file updates and/or some updated chain information. Therefore, when an offline RP gets reconnected, it needs to pull some online RPs to synchronize the status of the file and its probe nodes. An RP can probe an online probe node from its nearest probe node to farthest one in each direction. Whenever an online RP is probed in one direction, the file and the information of its probe nodes are synchronized. If its IP address is not changed, the pull process in this direction is finished. The same process is executed in the other direction. If the IP address of the reconnected RP is changed, it needs to send its ID and new IP address to all its probe nodes. Then the pull process is finished. If no online probe node can be pulled (due to probe nodes going offline or changing IP addresses), the reconnected RP needs to connect its probe nodes through flooding search to synchronize the file status and the chain information if its IP address is changed.

3.4 Write-write protection

In UPTReC, if two RPs update a file and push the updates through the chain at the same time, there is a write-write confliction. In this case, one RP in the chain can detect a write-write confliction when it receives two updated files generated by two different RPs at the same time. Whenever the write-write confliction is detected, the RP can send the confliction information back to the two update initiating RPs, which in turn solve the confliction through communication with each other, then the latest updated file is pushed through the chain again. Due to low write-write confliction rate [15] in P2P systems, we ignore the write-write confliction costs in our analysis below.

3.5 Chain Construction and Maintenance

Now let us discuss how to construct and maintain a replica chain. After a peer initiates a file in a system, the file can be searched, fetched and replicated by other peers. Each replica is copied from one of other replicas. If each RP maintains the information of all RPs which fetched the file from it, then a replica tree is naturally constructed. Figure 3 (a) shows a replica tree composed of 5 RPs as the root node at RP_1. If all RPs are always online, an update from any RP can be successfully propagated to any other RPs. For example, when RP_3 initiates an update, it sends the update to RP_1, RP_4 and RP_5. Each RP in turn updates its replica and then relays the update to all its children and parent except the one which sent the update. The update is successfully propagated through all RPs. A new replica tree with RP_3 as the root is shown in Figure 3 (b). However, frequently disconnected peers make such a replica tree ineffective in terms of update delivery. In order to increase the probability of successfully propagating the update, each RP must maintain the information of multiple RPs along each path. Due to the properties of the general tree, some RPs may maintain the information of a large number of RPs, while some other RPs maintain information of very few RPs. To balance the overhead associated with the file maintained by each RP, a replica chain can be constructed from the replica tree as explained below.

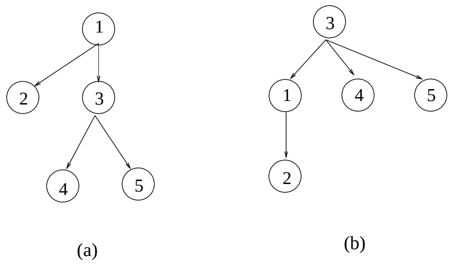

Fig. 3. RPs naturally constructs a replica tree. (a) Root at RP_1; (b) New tree with root at RP_3.

Figure 4 shows the process of constructing a replica chain during the replica process. Figure 4 (a) presents the first four RPs which naturally form a chain. In this case, a new node locates at the head or tail of the chain. When a new peer replicates the file fetched from another RP, the corresponding chain information is also fetched. The information of a new RP is forwarded to all possible RPs which should have the information of the new RP. Figure 4 (a) illustrates the process for an RP, such as RP_4 joining the chain. When RP_4 fetches the file from RP_3, the replica chain including information about RP_1 and RP_2 is also fetched. The RP_3 adds RP_4 into the chain, and pushes the information about RP_4 to RP_1 and RP_2. However, if RP_1 for example is offline at that time, it needs to probe either RP_2 or RP_3 to get the updated chain information.

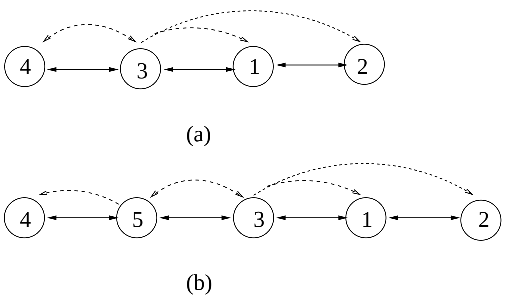

Fig. 4. The process for construction a replica chain. (a) The new replica locates on the head or tail of the chain; (b) The new replica locates at the middle of the chain.

If a new peer joins in the middle of a chain, it needs to push its information to at most k RPs in the chain along the direction opposite to the RP which provides the file. For example, when RP_5 joins the chain by obtaining the chain information from RP_3, RP_5 pushes the information to RP_4.

When RP_i removes a replicated file, it sends a message to each of its probe peers to get removed from the replica chain. All online probe peers get the message and in turn remove RP_i from the chain. All offline probe peers get this message when they reconnect. If all probe peers are offline, the RP is not removed from the chain but will inform the reconnecting peers when they probe. The process of adding or removing a replica requires up to $2k$ messages.

4 Performance Analysis

The performance of UPTReC is analyzed in this section. One critical issue concerning the UPTReC algorithm is to determine the value of k. If k is too small, an update may not be successfully propagated through the chain. If k is too large, the overhead cost of chain maintenance is high.

4.1 Performance Analysis

Our analytical modeling is based on the assumptions made in Section 3.1. Some parameters and measurement metrics are defined below.

- N: number of nodes in the chain, i.e., the total number of replicas of a file
- k: number of probe nodes in one direction
- P_{on}: probability of an RP to be online
- P_{off}: probability of an RP to be offline (P_{off}=1-P_{on})
- P_{cIP}: probability of an RP to change its IP address after each reconnection
- h: number of hops an online RP from the update initiating peer
- T: average period of a peer online and offline cycle
- λ: access rate of a file in the whole system
- T_{up}: average file update period
- P_h^s:probability of successfully propagating an update to an online RP with h-hop distance
- $P_h^s(m)$: probability of successfully propagating an update to an online RP with h-hop distance while the online RP only counts the contributions of its m farthest ($1 \leq m \leq k$) probe peers (i.e., the k^{th}, ..., $(k$-m+$1)^{th}$ probe peers)
- $P_{pull}^s(k)$: probability of a reconnected RP to successfully pull an online RP
- C_{flood}: average number of messages to find an online probe peer through flooding search
- $C_{push}(N)$: maximum number of messages to push an update through an N-node replica chain
- $C_{pull}(k)$: average number of messages in each pull procedure of a reconnected peer
- OHQ: number of overhead messages per query of file consistency maintenance (including overhead of push and pull)
- $P_{stale}(N)$: stale query probability for a file with N replicas
- D: update propagation delay (in hops)

In UPTReC, for a chain with N RPs, the maximum number of probe messages to push an update through the chain is N, because each RP at most receives one probe message. Thus we have

$$C_{push}(N) \leq N \tag{1}$$

When an offline RP rejoins the system, it pulls an online RP from its probe peers in each direction to synchronize the file status and probe peers' information, the pull process in one direction stops whenever an online RP is pulled. If a probe peer is offline or online but with different IP address from that recorded by the reconnected RP, it cannot be pulled. We use P_{fail}=P_{off}+$P_{on}P_{cIP}$ to represent the probability that a probe peer cannot be pulled by a reconnected peer. Then the probability of a reconnected RP to successfully pull an online RP is

$$P^s_{pull}(k) = 1 - (P_{fail})^{2k} \tag{2}$$

If the IP address of the reconnected RP is changed, it needs to contact all probe peers once, hence $2k$ probe messages are needed. If no probe peer is probed, it needs to search a probe peer through flooding. So the average number of probe messages for each pull process is:

$$\begin{aligned} C_{pull}(k) &= P_{cIP}[2k + (1 - P^s_{pull}(k))C_{flood}] + \\ 2(1 - P_{cIP})&[(1 - P_{fail})\sum_{i=1}^{k-1} i(P_{fail})^{i-1} + k(P_{fail})^{k-1})] \\ &= P_{cIP}[2k + (1 - P^s_{pull}(k))C_{flood}] + \\ &\qquad 2(1 - P_{cIP})(\frac{1 - P^k_{fail}}{1 - P_{fail}}) \end{aligned} \tag{3}$$

In Eqn. (3), the first term is the pull cost for a reconnected RP with changed IP address, and the second term is the pull cost when its IP address is not changed. In the second term, if an online probe peer is pulled in a direction, the pull process is stopped in that direction, and if no online probe peer is pulled, all k probe peers are needed to be pulled once. The pull process is symmetrical in both directions.

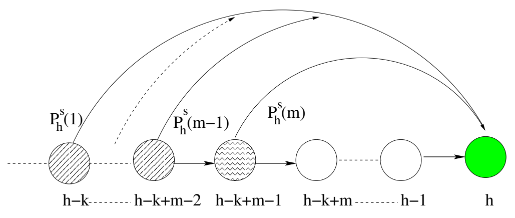

Fig. 5. Calculation diagram of $P^s_h(m)$.

Based on the definitions, we have $P^s_h = P^s_h(k)$, and $P^s_h(m)$ can be recursively calculated. Figure 5 shows the calculation diagram of $P^s_h(m)$. Here RP_h has h-hop distance from the update initiating peer. $P^s_h(m)$ represents the probability of RP_h to get the update if only its farthest m probe peers (i.e., its k^{th}, $(k-1)^{th}$, ..., $(k-m)^{th}$ probe peers) are considered, these probe peers are h-k, h-k+1, ..., h-k+m-1 hop distance from the update initiating peer, we call these RPs as RP_{h-k}, RP_{h-k+1}, ..., and $RP_{h-k+m-1}$ as shown in Figure 5. For example, $P^s_h(2)$ is the probability for RP_h to get an update if only probe peers RP_{h-k} and RP_{h-k+1} are considered to push the update to RP_h, and all probe peers RP_{h-k+2}, ..., RP_{h-1} are not considered. All these probabilities can be recursively calculated by the following three equations:

If $h \leq k$ and $1 \leq m \leq k$,

$$P_h^s(m) = 1 \tag{4}$$

If $h > k$ and $m = 1$,

$$P_h^s(m) = P_{on} P_{h-k}^s(k) \tag{5}$$

If $h > k$ and $1 < m \leq k$,

$$P_h^s(m) = P_h^s(m-1) + P_{on} P_{off}^{m-1} P_{h-k+m-1}^s(k-m+1) \tag{6}$$

Eqn. (4) means that an online RP is a probe peer of the update initiating peer, it can absolutely get the update. Eqn. (5) indicates that only considering its farthest probe peer RP_{h-k}, if it is online and successfully receives the update, then RP_h can successfully get the update. Eqn. (6) can be explained by considering the m^{th} farthest probe peer $RP_{h-k+m-1}$, the probability of successfully receiving the update by peer RP_h is the probability of successfully receiving the update through its farthest m-1 probe peers plus the contribution of the m^{th} farthest probe peer. The m^{th} probe peer has contributions only if all farthest m-1 probe peers are offline, because if any of these peer is online, the contribution has been counted through that peer. In this case, the probability of successfully getting the update for the m^{th} farthest probe peer is only through its k-m+1 probe peers (its first m-1 probe peers are offline), i.e., $P_{h-k+m-1}^s(k\text{-}m\text{+}1)$.

The number of overhead message per query of file consistency maintenance is:

$$OHQ = \frac{1}{\lambda}\left(\frac{C_{push}}{T_{up}} + N\frac{C_{pull}(k)}{T}\right) \tag{7}$$

The optimal k is determined by $\frac{\partial OHQ}{\partial k} = 0$, that is:

$$P_{cIP} + P_{fail}^{2k} C_{flood} P_{cIP} ln(P_{fail}) - (1 - P_{cIP})\frac{P_{fail}^{k} ln(P_{fail})}{1 - P_{fail}} = 0 \tag{8}$$

However, this optimal k may not satisfy the guaranteed consistency requirements. For a replica chain with N RPs, the maximum number of hops from an update initiating peer to an online RP is N-1. Hence any online RP has a probability larger than P_N^s to get the update. An offline RP has $P_{pull}^s(k)$

probability to synchronize with an online RP, hence each online RP has at least $P^s_N P^s_{pull}$ probability with a valid file. Then the stale query probability is upper bounded by:

$$P_{stale}(N) \leq 1 - P^s_N(k) P^s_{pull}(k) \tag{9}$$

In Eqn. (9), P_{stale} decreases as k increases. So the optimal k must satisfy Eqn. (9) while have the minimal OHQ. The optimal k can be obtained by Eqns. (7) - (9). The overall performance of UPTReC is formulated by Eqn.s (1) - (9).

5 Numerical Results

The numerical results are presented here to characterize the optimal value of k under some probabilistically guaranteed file consistency. The characteristics of Gnutella system are measured in [18] and [4]. The results show that about 60% peers have 0.2 or less online probability, and 10% peers have more than 0.8 online probability, and 50% peers stay 60 minutes or less while 10% peers stay 300 minutes or more in each online session. Moreover, about 40% peers change their IP addresses in one day and about 50% in seven days. Our parameters are chosen based on theses statistics.

5.1 Impact of peer online probability

We first study the impact of the peer online probability. In this case, we set N = 5,000, C_{flood} = 1,000,000, T = 7,200 *sec*, and T_{up}=1800 *sec*. The optimal values of k and the probability of successfully propagating update through the chain versus P_{on} with P_{cIP} = 0.5 and 0.3 are shown in Tables 1 and 2, respectively.

From the tables, we know that the optimal value k decreases as the peer online probability increases. The optimal k is determined by the chain maintenance and RP rejoining costs. For each rejoining RP, it needs to probe at lease one online probe peer if its IP address is changed. If no one can be pulled successfully, it needs to search one online RP through flooding search which costs a lot of messages. Hence, an RP needs to keep information of more probe peers with smaller P_{on}. Similarly, an RP needs to record information of more probe peers with larger P_{cIP} so that it has high probability to successfully probe one online probe peer to rejoin the chain. This can be seen in Tables 1 and 2.

For $P_{cIP} = 0.5$, P^s_N is over 0.99 for all P_{on}, and hence we can choose the optimal

Table 1
Optimal k and P_N^s vs P_{on} at $P_{cIP} = 0.5$

P_{on}	0.1	0.2	0.3	0.4	0.5
Optimal k	105	54	36	27	21
P_N^s	0.992	0.994	0.996	0.998	0.999

Table 2
Optimal k and P_N^s vs P_{on} at $P_{cIP} = 0.2$

P_{on}	0.1	0.2	0.3	0.4	0.5
Optimal k	77	39	26	19	15
P_N^s	0.862	0.848	0.886	0.927	0.999
K	103	52	34	24	18

Table 3
Optimal k and P_N^s vs N

N	2000	4000	8000	16000	32000
Optimal k	36	36	36	36	36
P_N^s	0.998	0.997	0.994	0.987	0.975

k number of probe peers. But for $P_{cIP} = 0.2$, the P_N^s is low for small P_{on}. To maintain P_N^s larger than 0.99, the number of probe peers must be increased, Table 2 shows the minimal k (denoted as K) to ensure $P_N^s > 0.99$. Then K is the optimal value of k which can provide P_N^s larger than 0.99.

5.1.1 Impact of number of replicas

We study the scalability of UPTReC. The parameters are set to the same as in the previous case, excluding $P_{on} = 0.3$ and $P_{cIP} = 0.5$. The maximum number of hops in a replica chain increases as the number of RPs increases. In P2P systems, the typical number of replicas for a file varies from tens to thousands. Table 3 shows the results of P_h^s as h increases from 2,000 to 32,000. The results indicate that by keeping the optimal number of $k = 36$ probe peers, P_N^s only decreases from 0.998 to 0.975 when N increases from 2,000 to 32,000. This indicates that the UPTReC algorithm has good scalability in terms of the number of RPs.

6 Performance Comparisons

The performance comparisons between UPTReC and the update propagation algorithm based on the rumor spreading scheme (in short, *Rumor*) proposed in [9] are presented in this section. The overhead messages of file consistency maintenance come from push and pull processes, the major messages of a fast (slow) updating file is from the push (pull) process. We use simulations to study the impact on the performance of update frequency that is not analyzed in Rumor algorithm.

The both algorithms , i.e., UPTReC and Rumor, focus on the efficient update propagation to all online RPs. Note that the update propagation is only through the RPs. Moreover, both algorithms are independent on the file search and replication. Therefore, we simulate only RPs instead of a whole P2P system to focus on the file consistency maintenance cost. The system topology and physical connectivity are ignored.

In the simulations, each RP alternatively leaves and joins the system as a Poisson process. The file update is also assumed to follow Poisson distribution. When an update comes, the update initiating peer is randomly chosen from an online RP. In a real P2P system, a file can be searched and replicated by other peers, and an RP may drop a replica. As stated in the previous section, adding a new RP or removing an RP costs $2k$ messages to maintain the chain, but the subset maintenance is not discussed in Rumor algorithm [9]. Hence, we ignore the comparison on the costs of the chain and subset maintenance in the simulation by assuming a static chain and subsets. Moreover, all RPs are considered to have static IP addresses, because no method is discussed to deal with dynamic IP address in Rumor algorithm. The chain is randomly built, i.e., each RP has equal probability to appear at any location on the chain. Each RP keeps information of k probe peers in each direction. In the Rumor algorithm, each peer randomly picks up R RPs as its responsible peers. In the 0 push round, the update as well as a replica list are forwarded to its all responsible peers. The replica list records all RPs in which the update has been sent. In the t (≥ 1) push round, a peer has a probability $P_F(t) = f^t$ to push the update to its any responsible peer that is not on the replica list, where f is a constant between 0 and 1. A RP that receives an update is assumed to have ability to finish its push process. The pull process in both algorithms is similar. In UPTReC, when an online probe peer is probed in a direction, the pull process in this direction is finished. In Rumor, two online probe peers are probed in each pull process.

Let the file have an access rate λ for the whole system, each access randomly fetches the file from an online RP. When an online RP answers a query, if the file is in its newest version, a valid query is counted, otherwise a stale query

Table 4
Parameter Setup I

N	λ	T	T_{up}
10000	1	10000	10000

is counted. Due to focus on the efficiency of file consistency maintenance, the parameters λ, T, and T_{up} are set to unit time.

In our simulation model, when RP_a pushes an update to RP_b, it first probes RP_b. If RP_b is online, the update is forwarded. Thus the total number of update sent out is equal to the number of online RPs which have received the update. This number is almost equal in both algorithms if the stale query ratio is close to each other. We compare the overhead messages for probing all RPs rather than the number of update themselves. Of course, the update can be sent out instead of the probe messages. However, if the update is large, this may cause large extra traffic for sending the update to offline RPs.

6.1 Overhead messages for each push process

The number of overhead messages in the push process and the stale query ratio are studied by varying the peer online probabilities. The probability of successfully propagating an update is determined by the probability of a peer being online and the number of probe (responsible) peers. Based on the analytical results in the previous section, we set $2kP_{on} = 20$ (or $RP_{on} = 20$) to ensure a low stale hit probability. Thus, $P_{on} = 10\%$ corresponds to $k = 100$ ($R = 200$), and $P_{on} = 50\%$ corresponds to $k = 20$ ($R = 40$). The other parameters are set as in Table 4. Based on these setups, there are 1 query per RP and 1 update in each RP online and offline cycle (T period) on the average. Two different f values (0.8 and 0.9) are used in the Rumor algorithm to show the relationship between the stale query ratio and the number of overhead messages. The number of overhead messages in Rumor algorithm is determined by the stale query ratio, a larger f or R makes a lower stale query ratio. We set the R value as $2k$ which is the total number of probe peers kept by an RP in UPTReC. For such R value, high f values are needed to ensure a similar stale query ratio between UPTReC and *Rumor* algorithm, and hence f is set to 0.8 and 0.9.

Figures 6 and 7 show the number of overhead messages in the push process and the stale query ratio of both algorithms. As shown in these figures, a smaller f reduces the number of overhead messages in the Rumor algorithm, but the stale query ratio is increased. When f drops from 0.9 to 0.8, the number of overhead messages drops about 20%, but the stale query ratio is almost doubled.

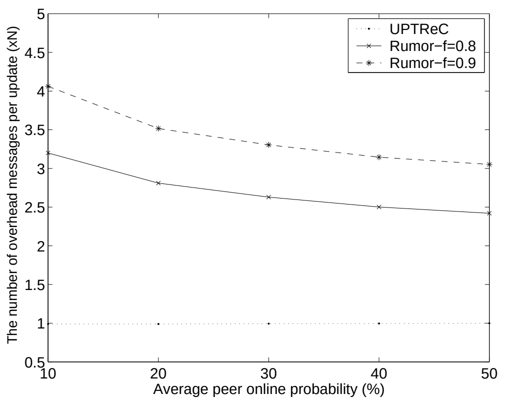

Fig. 6. The number of overhead messages for push process versus peer online probability

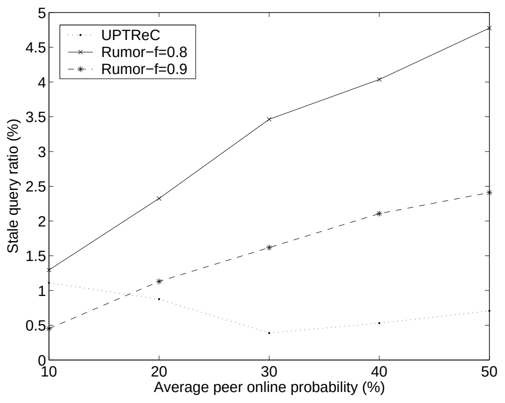

Fig. 7. Stale query ratio (%) versus peer online probability

The results show that the number of push overhead messages divided by the number of RPs (N) in UPTReC is almost 1, and this value is more than 2.4 in Rumor. The stale query ratio for the UPTReC is less than 1.2% for all ranges of P_{on} from 10% to 50%. But the stale query ratio for Rumor increases from 1.2% to 4.8% when P_{on} increases from 10% to 50% with $f = 0.8$. The stale query ratio can be reduced to less than 2.5% but incurs more than 20% overhead messages if f is set to 0.9. The results indicate that compared with Rumor, UPTReC reduces more than 60% overhead messages to put an update while achieving a smaller stale query ratio.

6.2 Overhead messages per query

The number of overhead messages per query in various update frequency is investigated in this case. We measure two performance metrics: the number of overhead messages per query and stale query ratio. The number of overhead messages per query is defined as the total number of consistency maintenance messages which include overhead messages of the push and pull processes divided by the total number of queries in the system. We set two R values (80 and 100) for Rumor algorithm in the simulation to study the effects of R. The other system parameters are set as in Table 5.

Table 5
Parameter Setup II

N	λ	P_{on}	T	k	f
10000	1	30%	10000	40	0.9

Figures 8 and 9 show the results of the number of overhead messages per query and stale query ratio versus different update frequencies. When the average update period ($T_{up} = 10^5$) is much larger than the peer online and offline cycle, the overhead messages of the pull process are the major source. Due to the similar pull process, the number of overhead messages per query for two algorithms is close in this case. As the update period decreases, the number of overhead messages from the push process increases and dominates the number of overhead messages from the pull process. This leads to a better performance of UPTReC than that of Rumor. When the update period is much shorter than the peer online and offline cycle, the number of overhead messages per query in UPTReC is more than 70% lower than that of Rumor. The stale query ratio in UPTReC is less than 0.1% in all range of update periods. In Rumor, when the update frequency is high, the stale query ratio is about 2% for $R = 80$, and it is reduced to less than 1% when $R = 100$. The effect of R is similar to f. A larger R or f gives a lower stale query ratio but costs more overhead messages. The results show that the UPTReC can save up to 70% overhead messages while providing better probabilistically consistency guarantee for highly update files compared to the Rumor algorithm.

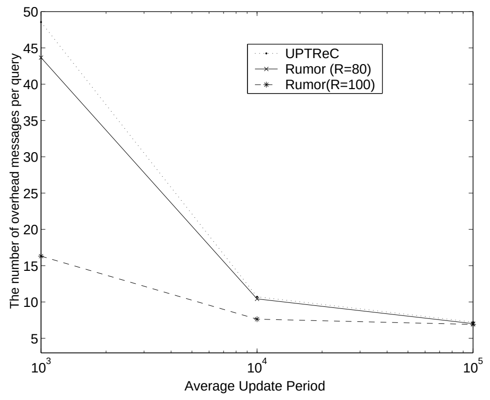

Fig. 8. The number of overhead messages per query versus average update period

Through these comparisons, we know that UPTReC can significantly reduce overhead messages to propagate an update with a smaller stale hit ratio compared to the Rumor algorithm.

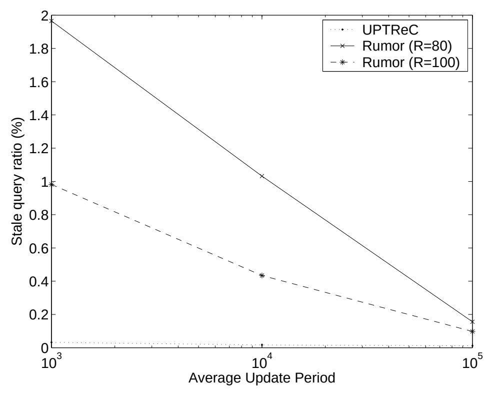

Fig. 9. Stale query ratio (%) versus average update period

7 Conclusions and Remarks

In this paper, we propose an efficient algorithm, UPTReC, to propagate update for decentralized and unstructured P2P systems. UPTReC provides probabilistically guaranteed file consistency. In UPTReC, each file has a logical replica chain composed of all RPs. Each RP has a partial knowledge of the chain. When an RP updates the file, it pushes the update to all possible online RPs through the replica chain. When an offline RP gets reconnected, the file status is synchronized by pulling an online RP.

An analytical model of the proposed algorithm is derived. The performance results of UPTReC compared to that of the Rumor algorithm shows that UPTReC can reduce up to 70% overhead messages while ensures a smaller query ratio for highly updating files.

As the growing of application in P2P systems, strong cache consistency is a further requirement, and this will be considered in our future works.

References

[1] Open Source Community, Gnutella. In *http://gnutella.wego.com*, 2001

[2] K. Aberer, P-Grid: A Self-organizing Access Structure for P2P information Systems. In *Proceedings of the Sixth International Conference on Cooperative Information Systems*, Trento, Italy, 2001.

[3] K. Aberer, Z. Despotovic, Managing Trust in a P2P Information System. In *Proceedings of the 10th International Conference on Information and Knowledge management*, pp310-317, ACM press 2001.

[4] R. Bhagwan, S. Savage and G. M. Voelker, Understanding Availability. In *Proceedings of the 2nd International Workshop on Peer-to-peer systems*, 2003.

[5] Y. Chawathe, S. Ratnasamy, L. Breslau, N. lanham and S. Shenker. Making Gnutella-like Peer-to-Peer Systems Scalable. In *Proceedings of ACM SIGCOMM'03*, 2003.

[6] X. Chen, S. Ren, H. Wang and X. Zhang. SCOPE:Scalable Consistency Maintenance in Structured P2P Systems. In *Proceedings of IEEE INFOCOM*, pp 1502-1513, 2005.

[7] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *Proceedings of the ACM SIGCOMM'02 Conference*, 2002.

[8] E. Cohen, A. Fiat and H. Kaplan, Associative Search in Peer-to-Peer Networks: Harnessing and Latent Semantics. In *Proceedings of IEEE INFOCOM'03*, 2003.

[9] A. Datta, M. Hauswirth and K. Aberer, Updates in Highly Unreliable, Replicated Peer-to-Peer Systems, In *Proceedings of IEEE ICDCS'03*, pp76-88, Rhode Island, May, 2003.

[10] B. Gedik and L. Liu, PeerCQ: A Decentralized and Self-Configuration P2P Information Monitoring System, In *Proceedings of IEEE ICDCS'03*, pp490-499, Rhode Island, May, 2003.

[11] S. Iyer, A. Rowstron and P. Druschel. Squirrel: A Decentralized Peer-to-peer Web Cache. In *Proceedings of the 21th ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[12] J. Lan, X. Liu, P. Shenoy and K. Ramaritham. Consistency Maintenance in Peer-to-Peer File Sharing Networks. In *Proceedings of Third IEEE Workshop on Internet Applications*, June, 2002

[13] X. Liu, J. Lan, P. Shenoy, and K. Ramaritham. Consistency Maintenance in Dynamic Peer-to-Peer Overlay Networks. In *Computer Networks*, v50, pp 859-876, 2006

[14] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer networks. In *Proceedings of the 16th Annual ACM International Conference on Supercomputing*, 2002

[15] T. W. Page, R. G. Guly, J. S. Heidemann, D. Reiher, A. Goel, G. H. Kuenning, and G. J. Popek. Perspectives on Optimistically Replicated Peer-to-Peer Filing. *Software-Practice and Experience*, pp155-180, 28(2), 1998.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM*, 2001

[17] M. Roussopoulos and M. Baker. CUP: Controlled Update Propagation in Peer-to-Peer Networks. In *Proceedings of the 2003 Annual USENIX Technical Conference*, June 2003.

[18] S. Saroiu, P.K. Gummadi and S. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of SPIE Conference on MUltimedia COmputing and Networking*, 2002.

[19] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM*, 2001.

[20] Z. Wang, S. K Das, M. Kumar and H. Shen, Update Propagation Through Replica Chain in Decentralized and Unstructured P2P Systems. In *Proceedings of IEEE International Conference on Peer-to-Peer Computing (P2P)*, pp 64-71, 2004