An Update Propagation Algorithm for P2P File Sharing over Wireless Mobile Networks*

Haengrae Cho

Department of Computer Engineering, Yeungnam University
Gyungsan, Gyungbuk 712-749, Republic of Korea

Abstract. Peer-to-peer (P2P) file sharing systems often replicate files to multiple nodes. File replication is beneficial in the sense that it can achieve good query latency, load balance, and reliability. However, it introduces another problem of maintaining mutual consistency among replica when a file is updated. The new file has to be propagated to all of its replica. In this paper, we propose an update propagation algorithm for P2P file sharing over wireless mobile networks (MONET). Compared to the previous algorithms proposed in wired P2P file sharing systems, our algorithm has low communication overhead. It also guarantees safe delivery of updates even when the underlying network is unreliable. This means that our algorithm is well matched to the characteristics of MONET such as limited battery power of mobile nodes, lower network reliability, and frequent disconnection of mobile nodes.

Keywords: Mobile network, P2P, file sharing, update propagation.

1 Introduction

Peer-to-peer (P2P) computing is becoming a very popular computing paradigm due to the wide diffusion of file sharing applications [1]. Representative P2P systems such as Gnutella [6] and Kazaa [10] have millions of nodes sharing petabytes of files over Internet. The complexity of a file discovery in P2P systems would be very high since it may require scanning the entire network to find a required file. Efficient file replication can reduce the complexity [3,7]. However, it introduces another problem of maintaining mutual consistency among replica when a file is updated. The new file has to be propagated to all of its replica.

The growing number of mobile devices, together with the proliferation of wireless and pervasive communication technologies, also demand for the adoption of P2P paradigms in wireless mobile networks (MONET) [4,5,9]. An update propagation algorithm for MONET based P2P file sharing systems has to deal with the following issues arising from constraints specific to MONET.

- **Limited battery power:** A node in MONET runs using its battery power. Then the node might lose its power rapidly if it transmits information heavily. This means that the update propagation algorithm should be able to reduce message traffic between nodes.

* This research was supported by University IT Research Center Project.

- **Frequent disconnection of mobile nodes:** Mobile nodes often get disconnected from the network due to power failure or due to their mobility. In addition, some mobile users switch their units on and off regularly to save power, causing more network disconnections. So the update propagation algorithm should be able to deliver missing updates to mobile nodes when they are reconnected to the system.
- **Unreliable network:** An wireless connection is more unreliable than an wired one. This means that the update propagation algorithm has to be resilient to the message lost.

The P2P file sharing over MONET is still in its early stage [4,5,9]. Most of previous work in this area investigate only the file discovery algorithm. They do not consider file replication or do not describe how replicated files get same values if any of them is updated. In this paper, we propose a new update propagation algorithm for P2P file sharing over MONET. The novel features of our algorithm are as follows.

- The algorithm is *purely decentralized*. All updates must be eventually propagated to their replica, but update propagation between nodes is performed in asynchronous manner. This is a great advantage in MONET based P2P systems that may experience transient failures and network partitioning.
- The algorithm has *lower communication overhead*. It tries to reduce duplicate delivery of an update message between nodes. The lower communication overhead can contribute power savings of mobile nodes and efficient usage of limited network bandwidth.
- The algorithm is *adaptive* to the dynamic behavior of MONET. When most neighbor nodes leave the network, a node reconstructs new neighborhood autonomously with other nodes. The movement of mobile nodes also causes the neighborhood reconstruction. This contributes to fault-tolerance and fast update propagation.

The rest of this paper is organized as follows. Section 2 introduces previous work in wired P2P systems. Section 3 describes the proposed algorithm in detail. Section 4 presents the experiment results. Finally, Section 5 concludes the paper.

2 Update Propagation in Wired P2P Systems

There are two types of update propagation algorithms in wired P2P systems: *push* and *pull*. In the push-based algorithm [6,8], a new update is pushed by an initiator to its neighbor nodes, which in turn propagate it to their neighbor nodes. In the pull-based algorithm [2], a node polls periodically to one of neighbors if there are any new updates. The push-based algorithm can provide good consistency guarantees for nodes that are online and reachable from the initiator. However, it suffers from high communication overhead due to duplicate message delivery. Figure 1.(a) illustrates a case of duplicate message delivery. Both node C and node D propagate a same update message to each other. The pull-based

Fig. 1. Problems of previous algorithms

algorithm can reduce the communication overhead but causes longer propagation delay and weaker consistency guarantees. The consistency guarantees in pull are critically dependent on the effectiveness of polling.

To combine the best features of push and pull, a *hybrid push-pull* algorithm [3] is proposed. An updating node initiates a push phase by propagates the new value to its neighbors. A push message includes a receiver list. The receiver list includes identifiers of nodes to which the same message has been sent. The algorithm can avoid duplicate message delivery by restricting an update message to be sent to any node in the receiver list. However, a node could not receive an update at the push phase if messages are lost. Figure 1.(b) illustrates the case of lost update. A receiver list sent to C is $[A, B, C, D]$. Then node C does not propagate an update message to node D because the receiver list includes D. When a node joins the network, it enters a pull phase to synchronize and reconcile. The pull phase also introduces another problem. At the pull phase, a node inquires for missed updates based on version vector to one of its neighbors. However, a version vector of entire replicated data items is a huge unit for messaging and consumes lots of CPU cycles of recipient node for version comparison. A selective pull on part of replicated files is strongly required.

3 Update Propagation over MONET

In this section, we propose a new hybrid push-pull algorithm, named UPM (*Update Propagation over MONET*) for MONET based P2P file sharing systems. UPM consists of four sub-algorithms for push, selective pull, entire pull, and reconstruction of neighborhood.

3.1 Data Structures

A node creates new neighborhood with other nodes when it joins a P2P system. Gnutella's ping-pong protocol [1] is an example to make such neighborhood. A node N_i maintains the identifiers of its neighbor nodes at $neighbor(i)$.

We assume that each file has an owner. The owner is a node that has a role to synchronize updates on the file. Initial owner is set to a node that creates the file. When a non-owner node tries to update the file, it first contacts the owner.

If the owner grants the update operation, the ownership is changed to the new node. The old owner keeps the information of new owner so that it can forward following update requests to the new owner.

Each node N_i maintains an *update counter* $UC(i)$. N_i increments $UC(i)$ whenever it updates any file. N_i also maintains an *update counter vector* UV_i for other nodes. $UV_i[k]$ stores a value of $UC(k)$ that N_i knows. Finally, N_i maintains an *update history table* H_i . If $H_i[k, UC(k)] = f$, then N_i knows that another node N_k updates a file f at $UC(k)$. The initial settings of the data structures are $UC(i) = 0$, $UV_i[*] = 0$, and $H_i[*, *] = \emptyset$. The contents of UV and H are updated at push phase and pull phase.

3.2 Push Phase

When an owner updates a file, it initiates update propagation to its neighbor nodes. The update message consists of six attributes: <sender, owner, file id, new file, update counter of initiator, sender list>. The sender list is a set of nodes that have sent the same message. Let us suppose that node N_i updates a file f . Then the following steps are performed.

- (1) $UC(i) = UC(i) + 1$
- (2) $H_i[i, UC(i)] = f$, $UV_i[i] = UC(i)$
- (3) To every node in $neighbor(i)$, send an update message < i , i , f , value of f , $UC(i)$, $\{i\}$ >.

Since N_i is both an owner and a sender, the update message includes i twice. The sender list includes only i with the same reason. Now let us suppose that node N_k receives an update message < s , i , f , value of f , $UC(i)$, $\{i, s\}$ > from node N_s . Then N_k performs the following steps to process the message.

- (1) If $UC(i) == UV_k[i] + 1$, N_k has received every update message of N_i that was sent before $UC(i)$. So N_k needs to process this message only as follows.
 - $H_k[i, UC(i)] = f$, $UV_k[i] = UC(i)$
 - Replace N_k 's local copy of f by the value of f at the message.
 - To every node in $neighbor(k) - \{i, s\}$, send an update message < k , i , f , value of f , $UC(i)$, $\{i, s, k\}$ >.
- (2) Else if $UC(i) == UV_k[i]$, this is a duplicate message. N_k just ignores it.
- (3) Else if $UC(i) > UV_k[i] + 1$, there are some updates that N_k did not receive. Then N_k inquires for the missed updates to N_s at the pull phase.

If a node is in the sender list, it actually received the message. So other nodes do not need to send the message to any nodes in the sender list. As a result, compared with sending the receiver list [3], UPM can increase the probability of receiving the update message. Figure 2.(a) illustrates a scenario that UPM can propagate an update when there is a message lost. Node D can receive the update from other link of node C. Furthermore, unlike the pure push-based algorithm, D need not propagate the update to node B again since the sender list includes B. This contributes to reduce the duplicate message delivery.

Fig. 2. Scenarios of push phase and selective pull of UPM

3.3 Pull Phase

A node performs a pull phase at two cases: when it detects any missed updates at the push phase (*selective pull*), or just after it joins the system (*entire pull*).

Selective Pull. Let us suppose that a node N_k receives an update message $\langle s, i, f, \text{value of } f, UC(i), \{i, s\} \rangle$ from node N_s at a push phase. If $UC(i) > UV_k[i] + 1$, N_k performs a selective pull as follows.

- (1) N_k sends a selective pull message $\langle k, i, UV_k[i] \rangle$ to N_s .
- (2) N_s makes a response message PR as follows.
 - $PR = \{\}$
 - For $(t = UV_k[i] + 1; t \leq UV_s[i]; t = t + 1)$
 $PR = PR \cup \langle i, H_s[i, t], \text{value of } H_s[i, t], t \rangle$.
- (3) N_s returns PR to N_k . Then N_k updates its data structures and local copies as follows.
 - For each tuple $\langle i, f, v, t \rangle \in PR$
 Set $H_k[i, t] = f$ and replace N_k 's local copy of f by v .
 - $UV_k[i]$ is set to the maximum value of t for all tuples in PR .

Figure 2.(b) illustrates a case when a selective pull happens. Node A is an initiator of push phase and propagates an update message with $UC(A) = 3$. Node B missed an update of A with $UC(A) = 2$, and thus $UV_B[A]$ is set to 1. Since $UC(A) > UV_B[A] + 1$, node B can detect that it missed some updates. At the selective pull phase, node B first sends $UV_B[A]$ to node A. Then node A returns all of its updates after $UV_B[A]$ to node B.

Entire Pull. When a node joins a P2P system, it needs to validate the currency of its local copies by performing an entire pull. Let us suppose that node N_k joins the P2P system. The entire pull consists of the following steps.

- (1) N_k selects a node $N_s \in \text{neighbor}(k)$ in a random manner. Then N_k sends an entire pull message $\langle k, UV_k \rangle$ to N_s .
- (2) N_s makes a response message PR as follows.
 - $PR = \{\}$

- For each node n where $UV_k[n] < UV_s[n]$, perform the following steps.
 For ($t = UV_k[n] + 1; t \leq UV_s[n]; t = t + 1$)
 $PR = PR \cup \langle n, H_s[n, t], \text{value of } H_s[n, t], t \rangle$.
- (3) N_s returns PR to N_k . Then N_k updates its data structures and local copies as follows.
 - For each tuple $\langle i, f, v, t \rangle \in PR$
 Set $H_k[i, t] = f$ and replace N_k 's local copy of f by v .
 - $UV_k[i]$ is set to the maximum value of t for all tuples of i in PR .

Note that the random selection of target node would not be optimal. This is especially true when the target node has missed some updates also. However, both the push phase and the selective pull can guarantee to get the missed updates eventually. If a node contacts multiple nodes at the entire pull, it may reduce the amount of missed updates. This in turn increases the communication overhead and consumes CPU cycles of more neighbor nodes.

3.4 Reconstruction of Neighborhood

The correctness of UPM depends on the connectivity of a node with its neighbor nodes. If most neighbors leave the P2P system or the node moves to other network, it would not receive any push messages. To detect such condition, each node sends periodically an **are_you_alive** message to every neighbor node. The living nodes reply to the message. If the number of living nodes is under the threshold value, the node has to reconstruct new neighborhood with other nodes by Gnutella's ping-pong protocol [1]. Furthermore, if the node has been isolated completely (i.e. the number of living nodes is zero), it has to perform an entire pull to one of the new neighbors.

4 Experiments

We develop an experiment model of a MONET based P2P system using CSIM [11] simulation package. We compare UPM with two algorithms, PO (push only) and LRA (List of Receivers Algorithm). PO is a pure push algorithm [6]. LRA is a hybrid push-pull algorithm [3]. Table 1 summarizes simulation parameters. We model a P2P system with a limited number of data items and high update ratio. This setting helps us investigate the differences between algorithms. There are three types of nodes: online, join, and leaving. Join nodes are off-line initially and then join the network with a probability of *OnlineRate*. Online nodes leave the network with a probability of *LeaveRate*. *MsgLossRate* is a probability of a message being lost. The last two parameters are used to model the mobility and reliability of the P2P system over MONET.

We first compare the message overhead of three algorithms. Figure 3.(a) show the experiment results when *LeaveRate* varies from 0 to 0.5. *MsgLossRate* is set to 0.3. PO suffers from the heavy communication traffic due to large number of messages. Note that PO does not filter any duplicate messages. Every node just propagates an update message to all of its neighbors. UPM reduces the number of

Table 1. Simulation Parameters

Parameter	Description	Setting
<i>CPU Speed</i>	Speed of nodes' CPU	500 Mips
<i>NetBandwidth</i>	Wireless network bandwidth	1 Mbit/s
<i>NumNode</i>	Number of nodes	500
<i>NumNeighbor</i>	Number of neighbor nodes	8
<i>DiskTime</i>	Disk access time	10 ms ~ 30 ms
<i>MsgInst</i>	CPU instructions to process a message	22000
<i>PerIOInst</i>	CPU instructions for a disk I/O	5000
<i>NumDataItem</i>	Number of data items	1000
<i>UpdateRate</i>	Probability of a data item being updated	1.0
<i>OnlineRate</i>	Probability of a node being online	0.2
<i>LeaveRate</i>	Probability of an online node leaving network	0.0 ~ 0.5
<i>MsgLossRate</i>	Probability of a message being lost	0.0 ~ 0.3
<i>MsgDelay</i>	Delay to transfer a message	1 ms ~ 5 ms

messages about 35% compared with PO. This is because UPM attaches a sender list at the update message so that it can reduce the duplicate message delivery significantly. As expected, LRA reduces the number of messages the most. It was about half of PO. The receiver list can prevent the duplicate message delivery completely, even though lost updates would happen.

Then we evaluate the number of lost updates for each algorithm. Figure 3.(b) shows the experiment results by varying *LeaveRate* while *MsgLossRate* is set to either 0.15 or 0.3. PO performs worse as *LeaveRate* increases due to the lack of pull phase. If a node is off-line during a push phase of an update, it should lose the update. On the other hand, the performance of UPM and LRA are nearly constant under the change of *LeaveRate*. They can allow a new joining node to receive any missed updates at the pull phase.

The performance of LRA degrades when *MsgLossRate* is high. The receiver list approach of LRA restricts a node to receive an update message from only

Fig. 3. Experiment Results

one of its neighbor nodes. If the message is lost, the node cannot receive it again from any other nodes. UPM does not cause any lost updates. The duplicate message delivery is beneficial at this setting. As a result, UPM can guarantee safe update propagation at unstable network when messages could be lost and many nodes leave or join the network.

5 Concluding Remarks

In this paper, we describe UPM (*Update Propagation over MONET*), a new update propagation algorithm for P2P file sharing over MONET. UPM is novel in the sense that it is purely decentralized and has lower communication overhead. UPM also reconstructs the neighborhood autonomously and thus is resilient to node failures and can achieve fast update propagation.

We have demonstrated the efficacy of UPM using a number of different experiments. In the experiments, UPM propagates most update messages even when messages could be lost or nodes are disconnected frequently. This means that UPM can perform well at P2P systems on highly dynamic network such as MONET. Furthermore, UPM also reduces the amount of duplicate message delivery significantly compared with the push only algorithm. This contributes to the saving of network bandwidth and battery power of mobile nodes.

References

1. Androutsellis-Theotokis, S., Spinellis, D.: A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Surveys* **36** (2004) 335-371
2. Cetintemel, U., Keleher, P., Bhattacharjee, B., Franklin, M.: Deno: A Decentralized, Peer-to-Peer Object-Replication System for Weakly Connected Environments. *IEEE Trans. Computers* **52** (2003) 943-959
3. Datta, A., Hauswirth, M., Aberer, K.: Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. *Proc. 23rd ICDCS* (2003)
4. Ding, G., Bhargava, B.: Peer-to-peer File Sharing over Mobile Ad hoc Networks. *Proc. 2nd IEEE Conf. Pervasive Computing and Comm. Workshops* (2004)
5. Duran, A., Shen, C.: Mobile Ad hoc P2P File Sharing. *Proc. Wireless Comm. and Networking* (2004)
6. Gnutella, <http://www.gnutelliums.com/>
7. Gopalakrishnan, V., Silaghi, B., Bhattacharjee, B., Keleher, P.: Adaptive Replication in Peer-to-Peer Systems. *Proc. 24th ICDCS* (2004)
8. Holliday, J., Steinke, R., Agrawal, D., Abbadi, A.: Epidemic Algorithms for Replicated Databases. *IEEE Trans. Knowledge and Data Eng.* **15** (2003) 1218-1238
9. Huang, C-M., Hsu, T-H., Hsu, M-F.: A File Discovery Control Scheme for P2P File Sharing Applications in Wireless Mobile Environments. *Proc. 28th Australian Computer Science Conf.* (2005)
10. Kazaa, <http://www.kazaa.com/>.
11. Schwetmann, H.: User's Guide of CSIM18 Simulation Engine. Mesquite Software, Inc. (1996)