Adaptive Scheduling of Message Forwarding of DHT-Based P2P Network Broadcast

JENG-WEI LIN1, TUN WEI WANG2 AND RAY-I CHANG2,+
1Department of Information Management
Tunghai University
Taichung, 407 Taiwan
2Department of Engineering Science and Ocean Engineering
National Taiwan University
Taipei, 106 Taiwan

Aggregated information, such as the total and free capacity of storages, computing power, network bandwidth, and so on, is very important for many peer-to-peer (P2P) applications and P2P-based grid or cloud computing. Previous studies have proposed to build a broadcast tree for a structured P2P network. Along this tree, the root can broadcast a command message to all the other peers, and in the reversed redirection, information can be aggregated from leaves to the root. Although command broadcast and information aggregation can be processed within $O(\log N)$ hops expectedly, where $N$ is network size, the required time may vary dramatically. In this paper, the authors furthermore consider the order of message forwarding in a broadcast tree. A low-cost fully-distributed algorithm running on each peer is presented to adaptively schedule the message forwarding. The simulation result shows that the proposed algorithm can significantly reduce the time required to complete command broadcast, as well as information aggregation. In addition, the load of peers is furthermore balanced.

Keywords: peer-to-peer, cloud computing, broadcast, aggregation, spanning tree, scheduling

1. INTRODUCTION

Peer-to-peer (P2P) networking and cloud computing are the most attractive Internet techniques and have interested many researchers in the last decade. They aim at offering virtually infinite computing power, storage space, and networking bandwidth. However, they are different essentially. Cloud computing [1] uses the client/server architecture to offer three types of services: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). At this moment, existent cloud computing services are typically implemented at big data centers. This enables a better system utilization by virtual machine consolidation. P2P networking shifts service load from powerful central servers to peers. Every peer in a P2P network is not only a client that uses services from others but also a server that provides services to others. P2P networking has been widely used in areas such as content sharing [2, 3], and multimedia streaming [4, 5]. In recent years, various P2P-based cloud computing services [6-9] also emerged. They adopt P2P networking techniques to conceptually form their powerful central servers. In P2P-based cloud computing, it is usually required to maintain or monitor certain valuable global
information of the service or the P2P network. For example, it is very important to know the total and free storage capacities in a distributed storage cloud, e.g., in Cassandra [6]. Broadcast supports these operations. An originator first broadcasts a command message to all the other peers in the network and then waits for response messages sent back in the reverse direction for information aggregation.

A structured P2P network typically maintains a DHT (Distributed Hash Table) to support fast lookups of peers and objects by their hashed identifiers [10-13]. Unfortunately, it is not trivial for broadcast. Instead of blindly flooding broadcast messages to its all neighbors, an activated peer – that has received a broadcast message – has to cleverly find some non-activated peers – that have not yet received the message. Many broadcast algorithms had been presented, such as mesh-based approaches for mass data dissemination [14, 15], and divide-and-conquer approaches for small messages [16, 17]. In certain cases, an originator can use high-speed interconnection networks to broadcast messages to all peers. However, these algorithms do not support information aggregation well since the originator may suffer from a large number of responses from all other peers. In [18, 19], tree-based approaches proposed to explicitly maintain a spanning tree of the network to support operations such as command broadcast, status notification, network diagnoses, information aggregation, and semi-structured or arbitrary queries. When an originator wants to broadcast a command message, it first sends the message to the root of the tree. After receiving the message, every non-leaf peer forwards the message to each of its children. In the reverse direction, peers can aggregate the requested information from leaves to the root. When peers are randomly distributed in the network, the heights of the broadcast trees are expectedly $O(\log N)$, where $N$ is the network size. Thus, it is very likely that a command message can reach any peer within $O(\log N)$ hops. As well, information can be aggregated within $O(\log N)$ hops. However, the time required to complete command broadcast may vary dramatically, and thus to aggregate information.

In this study, a P2P-based cloud computing environment is considered, which consists of a dynamic number of servers located in a same data center. The order of message forwarding in a broadcast tree is considered. A discrete time model is used, since command and response messages are typically small and the propagation delay is short. An activated peer can send a message to a non-activated peer in a round (or time slot). This simply definition does not imply the clocks of peers are synchronous. Given a spanning tree, the schedules of message forwarding of all peers are investigated. For the tree shown in Fig. 1 (a) whose height is 5, a broadcast schedule shown in Fig. 1 (b) indicates that peer A initially sends the message to peer B. Hence, B is activated and can forward the message to others at the next round. At the 2nd round, A and B send the message to E and C, respectively. At the 3rd round, B, C and E send the message to F, I, and K, re-

Fig. 1. A broadcast spanning tree and its possible schedules.
spectively. Finally, at the 5th round, all peers have received the message. The schedule takes 5 rounds to disseminate the message to all peers. For the same tree, another schedule shown in Fig. 1 (c) takes 8 rounds. In the two schedules, all peers will receive the broadcast message within at most 4 hops. However, the number of total rounds (or the round number for short in this paper) required for the first schedule is smaller than the second one. The round number can be furthermore reduced. As shown in Fig. 1 (d), after D, I, and J are redirected to attach onto A, A, and B respectively, the broadcast schedule takes 4 rounds. The schedule is optimal since in every round every activated peer sends the message to a non-activated peer. The authors must note that these redirections result in another broadcast tree for command broadcast and information aggregation.

Assuming the height and average degree of a broadcast tree are \( h \) and \( w \) respectively, without redirection, a random broadcast on this tree expectedly takes \( O(w^h) \) rounds to disseminate the broadcast message to all peers. With redirection, the broadcast could be accelerated. Assuming the root has \( w_{\text{root}} \) children, after the first \( w_{\text{root}} \) rounds, the root has forwarded the broadcast message to all its children. If the root continues forwarding the message to some other non-activated peers, who had been redirected to attach onto the root previously, these peers will receive the message earlier from the root than from their original parents, as shown in Fig. 1 (d). Similar redirections could be applied to all activated peers in the network. If one is able to find enough redirections and arrange every activated peer to forward the message to a non-activated peer in every round, the number of activated peers doubles every round. Thus, any broadcast schedule takes at least \( \log_2 N \) rounds. However, it is not trivial for an activate peer to find a non-activated peer in a P2P network. Furthermore, a P2P network is usually highly dynamic since peers may join or leave the network at any time.

In [20], a centralized algorithm, referred to as OPBS (Optimal P2P Broadcast Scheduling), had been presented to compute an optimal broadcast schedule which completes a broadcast on a spanning tree within minimal rounds for a stable network. Instead of searching a non-activated peer in a top-down fashion, OPBS uses a bottom-up fashion: each non-activated peer finds an activated peer along the path from its parent to the root. The simulation result showed that the bottom-up approach is very effective. The resultant broadcast schedule is expected to complete a broadcast nearly within the theoretical minimal rounds. However, OPBS recomputes a whole new broadcast schedule whenever the network topology changes due to the join, departure, or failure of peers. In other words, OPBS is preferred in a stable P2P network where peers do not shutdown and/or restart frequently. For example, Cassandra [6] adopts P2P techniques to assemble a dynamic number of private-owned servers. These servers join and leave mostly because of scheduled maintenance or failure of some components. However, in some P2P-based cloud computing services, the network topology may change much more frequently. The resources provided by the peers, such as storage space and CPU power, are more unreliable due to churn situation [7-9]. For example, in a video-on-demand system [7], a P2P cloud is designed to pool local cache space and network bandwidth on all participating peers via an overlay network. To compensate for the dynamic nature of distributed peers, a set of media servers are deployed to cooperate with the P2P cloud to optimize video availability and streaming quality. A broadcast scheduling algorithm that can quickly respond to the frequent change of network topology is required.

In this paper, a fully-distributed adaptive algorithm, named as ABS (Adaptive P2P
Broadcast Scheduling), is presented for highly dynamic P2P networks. Each peer in the P2P network maintains a local partial broadcast schedule of message forwarding. All peers together explicitly construct a broadcast tree for information aggregation. Whenever a peer joins or leaves the P2P network, only a small number of peers are triggered to find redirections in a bottom-up fashion. If suitable redirections are found, these peers modify their local broadcast schedules and thus form a new broadcast tree. The simulation result shows that the round number of the resultant broadcast schedule required to broadcast a command message is significantly reduced, and thus to complete an information aggregation. In addition, the load of peers is more balanced.

The remainder of this paper is organized as follows. Section 2 presents the ABS algorithm. Section 3 discusses the simulation result. Finally in section 4, the authors conclude the paper.

2. ADAPTIVE SCHEDULING

In this section, the authors describe how to build a broadcast tree in a DHT-based P2P network. Then, the authors present the ABS algorithm and related data structures for broadcast scheduling.

2.1 Construction of a Broadcast Tree

Similarly to [18-20], a broadcast tree is constructed in a bottom-up fashion in a DHT-based P2P network. Each peer in the network locates and attaches onto its parent computed by a parent function. As a result, all peers together build the tree.

An efficient Chord-like DHT with a circular and continuous identifier space $I$ is assumed. A peer $x$ in the DHT is identified by a unique identifier $x$ in $I$ and owns a partial identifier space (or an interval) $[x, w]$ of $I$, where $w$ is the immediate successor of $x$ in the DHT. A peer $x$ owns an identifier $t$ means that $t$ is in the interval owned by $x$. This definition only requires a total order relation on identifiers, which is true or derivable in most DHTs.

The parent function $P_s(x)$ and its power functions $P_s^i(x)$ are defined as Eqs. (1) and (2).

![Fig. 2. Peer $x$, who owns an identifier space $[x, w]$, locates its parent by a parent function $P_s(x)$. In (a), $P_s(x)$, $P_s^2(x)$, ..., $P_s^∞(x)=α$ are in $[x, w]$. In this case, $x$ becomes the root of the spanning tree. In (b), $P_s(x)$, $P_s^2(x)$, ..., $P_s^i(x)$ are in $[x, w]$. However, $P_s^i(x)$ is in the identifier space owned by $y$. In this case, $y$ is the parent of $x$ in the spanning tree. $x$ locates $y$ by a generic lookup of $P_s(x)$ provided by the DHT.](image-url)
In Eq. (1), $\alpha$ is a constant identifier, $\beta$ is a constant larger than 1, and $2^s$ is the size of the identifier space. The function is modular arithmetic modulo $2^s$. It is easy to see that $P_s(x)$ is closer to $\alpha$ than $P_{s-1}(x)$, and $P_\infty(x) = \alpha$. $\alpha$, $\beta$, and $2^s$ are well known by all peers. These parameters of a particular application can be stored in the DHT as a regular object. Peers who want to join this broadcast can look up these parameters via generic DHT procedures.

For a peer $x$ owning an interval $[x, w)$, where $w$ is the immediate successor of $x$ in the DHT, there are two cases: (a) If $P_s(x)$, $P_{s-1}(x)$, ..., $P_\infty(x) = \alpha$ are all in $[x, w)$, $x$ is the root, as shown in Fig. 2 (a); (b) Otherwise, there is some $i$ such that $P_s(x)$, $P_{s-1}(x)$, ..., $P_{i-1}(x)$ are in $[x, w)$, but not $P_i(x)$, ..., $P_\infty(x) = \alpha$. In the latter case, the parent of $x$ is defined as the peer who owns $P_i(x)$, as shown in Fig. 2(b). In practice, if $x$ owns $\alpha$, $x$ is the root. Otherwise, $x$ computes $P_s(x)$, $P_{s-1}(x)$, ..., $P_i(x)$ iteratively. When some $i$ is found such that $x$ does not own $P_s(x)$, $x$ looks up for the peer who owns $P_i(x)$ via generic DHT peer lookup procedures. Let peer $y$ owns $P_i(x)$. $x$ attaches onto $y$ and becomes a child of $y$.

This procedure is fully distributed. Clearly, there is exactly one peer served as the root and every other peer has a parent. Since the parent function is directional, there are no cycles. Hence, all peers together build a spanning tree for information aggregation.

Apparently, it is likely that the parent of a peer switches among peers due to join or departure of other peers, so does the root of the broadcast tree. Thus, every peer periodically examines the status of its parent. A new-coming peer probably becomes a child of some peer and the new parent of other peers. As well, when a peer leaves, each of its children relocates and attaches onto a new parent. More scenarios about parent switch of a peer had been discussed in [18].

### 2.2 Data Structures

In this subsection, the data structures for the ABS algorithm maintained in all peers are described. Their values together describe a distributed broadcast schedule along a broadcast tree.

In addition to the parent $P_s$, a peer $x$ maintains a forwarding array $F_x$ of fingers, its upstream peer $U_x$, and the inverse round order $A_x$. If $x$ is the root, $P_x$ and $U_x$ point to nothing (null). Otherwise, $U_x$ points to the peer who will send broadcast messages to $x$. It is not unusual that $U_x$ is other than $P_x$ because of redirections. Each finger in $F_x$ points to a downstream peer. On receipt of a broadcast message from its upstream peer $U_x$, $x$ starts forwarding. It sends the message to each peer listed in $F_x$ in the decreasing order. Referring to Fig. 3, which is corresponding to the schedule shown in Fig. 1 (b), peer A sends the message first to B (pointed by $F_A[5]$) and next to C (pointed by $F_A[4]$); B sends the message first to E, next to F, and then to D (pointed by $F_B[4]$, $F_B[3]$, and $F_B[2]$, respectively).
$K_x$, defined as Eq. (3), denotes the maximum index of non-null fingers in $F_x$.

$A_x$, defined as Eq. (4), denotes the inverse round order of $x$.

\[
K_x = \text{MAX}\{ \text{idx} \mid F_x[\text{idx}] \text{ points to a peer }\}
\]

(3)

\[
A_x = \begin{cases} 
\text{undefined} & , \text{$U_x$ is null} \\
\text{k st. $F_u[k] = x$} & , \text{otherwise.}
\end{cases}
\]

(4)

If $x$’s upstream peer $U_x$ is determined, the $A_x$th finger in the forwarding array of $U_x$ points to $x$. Since $x$ forwards broadcast messages to its children in $F_x$ in the decreasing order, it should keep $A_x > K_x$, as shown in Fig. 3. It is clear that all forwarding arrays together describe a broadcast schedule along the broadcast tree of the network. Because of redirections, it is possible that $x$’s upstream peer $U_x$ is different from its original parent $P_x$ defined by the parent function.

By definition, $K_{\text{root}}$ is the round number of the resultant broadcast schedule, and $K_{\text{root}} - A_x + 1$ is expectedly the round at which a non-root peer $x$ will receive the broadcast message from its upstream peer $U_x$. As shown in Fig. 3, it is expected that B and C receive broadcast messages at the first and second round respectively. However, a peer $x$ does not have to know the value of $K_{\text{root}}$ and thus $K_{\text{root}} - A_x + 1$.

![Fig. 3. Forwarding array $F_x$, upstream peer $U_x$, and the inverse round order $A_x$ of peer $x$. This figure is corresponding to the schedule shown in Fig. 1 (b).](image-url)

Before a peer $x$ joins the network, $P_x$ and $U_x$ point to null, $A_x = 0$, and all fingers in $F_x$ also point to null. After $x$ joins the network, it learns these values adaptively by performing the procedures described in next subsection. When a peer wants to perform an information aggregation, it first sends the command message to the root. The root can perform some access control if necessary. Then, it starts forwarding the message to each peer listed in its forwarding array in decreasing order. On receipt of a broadcast message, a peer starts forwarding the message in the same manner as its upstream peer does. Recursively, the message arrives to all peers.
2.3 The ABS Algorithm

All peers in the network run the ABS algorithm to maintain the broadcast schedule by performing two procedures: FindUpstream() and OnReceiveRedirectionRequest(). The pseudo-codes of the two procedures are shown in Figs. 4 and 5 respectively. The authors note that the pseudo-codes are written in a hard-state style. However, soft-state procedures with timeout and retry mechanism are preferred in real implementations.

A bottom-up approach is used: each non-activated peer finds an activated peer along the path from its parent to the root. When a non-root peer loses contact with its upstream peer due to join or departure of other peers, it invokes FindUpstream() to find a new upstream peer. FindUpstream() iteratively probes an ancestor for a redirection at a specific round by sending a redirection request. When a peer is probed, it invokes OnReceiveRedirectionRequest(). It may accept the redirection request or pass the request to its parent. Detail of the two procedures follows.

When a non-root peer $x$ loses contact with its upstream peer, it first sets $U_x$ point

---

**Fig. 4. Procedure FindUpstream().**

1. Procedure FindUpstream($x$)
2. if ($x$ is not the root and $U_x$ points to nothing) {
3. \[ K_x = \text{MAX}( \{ \text{idx} \mid P_x[\text{idx}] \text{ points to a peer} \} ) \]
4. for (\text{idx} from $K_x + 1$ to MAX_FINGERS) {
5. \[ y = \text{SendRedirectionRequest(\text{idx}, P_x, x)} \]
6. if ($y$ points to a peer) {
7. \[ x \text{ and } y \text{ handshake and make the redirection} \]
8. \[ \text{return} \]
9. \}
10. \}
11. \}
12. End Procedure

---

**Fig. 5. Procedure OnReceiveRedirectionRequest().**

13. Procedure OnReceiveRedirectionRequest($\text{idx}, x, z$)
14. if ($U_x$ points to nothing) {
15. \[ \text{if (} P_x[\text{idx}] \text{ points to nothing) } \text{// case (1)} \]
16. \[ \text{Return and tell } z \text{ that } x \text{ could forward the message} \]
17. \}
18. else {
19. \[ \text{if (} \text{idx} < A_x \} \]
20. \[ \text{if (} P_x[\text{idx}] \text{ points to nothing) } \text{// case (2)} \]
21. \[ \text{Return and tell } z \text{ that } x \text{ could forward the message} \]
22. \}
23. else if (\text{idx} == A_x) { // case (3)
24. \[ \text{UnsetForwarding(\text{idx}, U_x, x)} \]
25. \[ \text{Return and tell } z \text{ that } x \text{ could forward the message} \]
26. \}
27. \}
28. // Pass the request to $x$’s parent
29. \[ \text{SendRedirectionRequest(\text{idx}, P_x, z)} \]
30. End Procedure
to null, asks its parent $P_x$ forward broadcast messages to itself, and then invokes FindUpstream() to find a new upstream peer. Since $A_x$ should be larger than $K_x$, FindUpstream() starts testing the value of $A_x$ from $idx = K_x + 1$. It sends a redirection request to $P_x$. $P_x$ may accept the request or pass the request to $P_x$’s parent, which absolutely is an ancestor of $x$. If there is an ancestor $y$ in the path upward to the root answering to $x$ with that $y$ can forward the message at the specified round, a redirection is found. $x$ and $y$ handshake and make the redirection. They update their internal data structures. $y$ sets $F_y[\text{idx}]$ point to $x$. At the same time, $x$ sets $U_x = y$ and $A_x = \text{idx}$. If no peers answer, FindUpstream() increases $\text{idx}$ by 1 and searches again until a peer answers.

When a peer $x$ receives a redirection request originated from another peer $z$, it checks its internal data structures. As shown in Fig. 6, there are three positive cases. (1) If both $U_x$ and $F_x[\text{idx}]$ point to null, i.e., $x$ is the root or has not yet determined its upstream peer and the specific round is free, $x$ accepts the request and answers to $z$ with that $x$ can forward the message to $z$ at the specified round. (2) If $U_x \neq \text{null}$ and $F_x[\text{idx}] = \text{null}$, and $\text{idx} < A_x$, i.e., the specific round is free and after $x$ receives the broadcast message, $x$ accepts the request and answers to $z$. (3) $U_x \neq \text{null}$ and $\text{idx} = A_x$, i.e., the specific round is the round at which $x$ receives the broadcast message, $x$ breaks the forwarding from its current upstream $U_x$, accepts the request, and answers to $z$. If no redirection is possible in the three cases, $x$ passes the redirection request to its parent $P_x$.

In Case (3), $x$ breaks contact with its upstream peer. Thus, it has to find a new upstream peer by invoking FindUpstream(). This will make a peer receive the broadcast message from a lower ancestor. Higher peers are kept available to redirection requests from descendant peers in a larger range. As a result, more redirections can be found. The ABS algorithm is capable of handling situations in which multiple peers are losing contacts with their upstream peers, as described in Case (1).

When a peer $x$ joins the network, it first locates and attaches onto its parent $P_x$. 

Fig. 6. Adaption of the ABS algorithm.
Since $U_x$ points to null, $x$ executes FindUpstream() to find a upstream peer. At the same time, it probably becomes the new parent of other peers. When probed, it executes OnReceiveRedirectionRequest(). During its lifetime, $P_x$ or $U_x$ are likely to change due to join or departure of other peers. $x$ can detect these events by active periodic checks or notifications from other peers. As a result, $x$ is triggered to recompute $P_x$ or $U_x$. When $x$ leaves the network, its children and downstream peers are triggered to recompute their parents and upstream peers similarly.

Here is a simple scenario in which peer $I$ in Fig. 3 leaves the network and peer $C$ becomes the new parent of peer $O$, as shown in Fig. 7. The authors note that when $O$ recomputes its new parent by the parent function, there is no guarantee that the parent of the leaving peer $I$, i.e. $C$ in this case, will become the new parent of $O$. After $O$ attaches onto $C$, $O$ then tries to find a new upstream peer. Since $O$ does not forward to other peers, $K_{O}=0$. $O$ tests $idx$ from 1 at first and makes a redirection request to $C$. $C$ is thus probed. Since $F_C[1]$ is not null, $C$ passes the request to its parent $A$. When $A$ is probed, $F_A[1]$ points to null, $A$ accepts the request. After $A$ and $O$ handshake, $A$ sets $F_A[1]$ point to $O$, and $O$ sets $U_O$ point to $A$ and $A_O=1$. A new broadcast schedule is therefore generated.

Subsequent redirections are possible. Peers may act more aggressively. For example, when $K_C$ decreases from 3 to 2 due to the departure of $I$, peer $C$ may also want to decrease $A_C$ by finding another upstream peer. $C$ tests $idx$ from 3 and makes a redirection request to peer $A$. $A$ is probed. Since $F_A[3]$ points to null, $A$ accepts the request. After $A$ and $C$ handshake, $A$ sets $F_A[4]$ point to null and sets $F_A[3]$ to $C$; at the same time, $C$ decreases $A_C$ to 3. A new broadcast schedule is generated, as shown in Fig. 8.

When a peer joins or leaves the network, some other peers may be triggered to invoke the ABS algorithm. In general, the higher the position of the peer in the original broadcast tree, the more peers triggered. This may prolong the time required to stabilize the resultant broadcast schedule. However, the coverage of the new broadcast schedule is the same as the original one. During the handshaking for redirection, the old and new upstream peer may both send broadcast messages to the downstream peer. Thus, the downstream peer will not lose any broadcast message during the handshaking for redirection.

Fig. 7. Lazy adaption after Peer $I$ leaves the network shown in Fig. 3.
3. THE SIMULATION

This section discusses the simulation result. All peers run the ABS algorithm. When a peer joins or leaves the network, some peers are triggered to recompute their parent and upstream peers due to active periodic check or notification from other peers. The following values are examined: (1) the round number of the resultant broadcast schedule; (2) the number of downstream peers a peer might have; and (3) the number of peers triggered to recompute their schedules whenever the network topology changes. The simulation result is compared with the original broadcast tree and the result by the centralized OPBS algorithm.

3.1 Simulation Setup

The experiment is divided into discrete time periods. Peer identifiers are randomly distributed over an identifier space of the size $2^{24}$. Similar to [19], an original spanning tree is constructed. For the parent function $P_\beta(x)$, $\alpha$ is randomly chosen when the experiment begins. $\beta$ is set between 2 and 8 to evaluate the ABS algorithm for broadcast trees with different heights and degrees. Peers join and leave the network according to Poisson distributions with parameters $\lambda_j$ and $\lambda_l$ respectively. There are two phases. Initially, $\lambda_j$ is set twice of $\lambda_l$ from time 0 to 16,000. In this phase, the number of peers increases with time. After there are more than 10,000 peers in the network, $\lambda_j$ and $\lambda_l$ are set alternately larger than each other and the number of peers oscillates between 8,000 and 12,000. In practical P2P implementation, peers, if possible, inform their neighbors before they leave the network. Thus, the DHT can reorganize its structure immediately and perform more stable and better. To understand worse-case performance, a fail-stop model is used where all nodes do not inform their neighbors when departing the network. Each peer has to periodically check whether its parent and upstream peer still exist in the network.

As shown Fig. 9, the height of the original broadcast tree defined by the parent function $P_\beta(x)$ is $c \log N$ approximately, where $N$ is the number of peers in the network and $c$ is a constant dependent on $\beta$. As $\beta$ increases, so do $c$ and thus the height of the spanning tree. On the other hand, as $\beta$ increases, the number of children to which a
peer might have to forward broadcast messages decreases, as shown in Table 2. Although the average number of children is small, the maximal number is several larger. The network load of peers is unbalanced and the required round number varies much for any peer to forward a broadcast message to its all children. When the spanning tree is constructed as described in [19] without redirection, whose average degree is $d$, a peer in the $h$th level of the tree will expect to receive broadcast messages at the $d^{*}(h-1)$th round. In the simulation, when there are 8,000–12,000 peers in the network, this value of a leaf at the lowest level is between $3.5*13–2.5*60$ (or 46–150) in average dependent on $\beta$. This value is much larger than the theoretical lower bound $\log_2N$. In fact, the total round number to complete a broadcast in the network is larger than this value, as shown in Fig. 10, since the maximal number of children a peer might have is several larger than the average. Fig. 10 also depicts the round numbers to complete a broadcast by using the divide-and-conquer approach presented in [16]. The numbers are comparable with those of the case where $\beta = 2$ without redirection; however, this approach does not support information aggregation well.

![Fig. 9. Height of the original broadcast tree without redirection.](image)

![Fig. 10. The total rounds for broadcast using the approaches presented in [16, 19].](image)
3.2 Simulation Result

Two scenarios are simulated. In the first, referred to as L-ABS (Lazy ABS), a peer runs the ABS algorithm only when it loses contact with its upstream peer. In another, referred to as A-ABS (Aggressive ABS), a peer more aggressively runs the ABS algorithm. A possibility is specified that even when a peer still has contact with its upstream peer, it may actively run the ABS algorithm to find a better upstream peer. In the simulation, 1\% is specified when in the A-ABS scenario.

![Fig. 11. Round number of the resultant broadcast schedule when OPBS is used.](image)

Fig. 11 shows the round number of the resultant broadcast schedule generated by the centralized OPBS algorithm, which is very close to the theoretical minimal round number \( \log_2 N \). When \( \beta \) is smaller than 4, the broadcast tree is too short and the reduction on the round number is not significant. As \( \beta \) increases, the tree gets higher. OPBS has a larger search space for a redirection and the round number significantly decreases. When there are 8,000–12,000 peers in the network and \( \beta \) is larger than and equals to 4, the round number is reduced from 46–150 to 15–20, which is close to the theoretical minimal round number 13. However, OPBS recomputes a whole new broadcast schedule for all peers even only a peer joins or leaves the network. As a result, OPBS is preferred in a relatively static network where peers do not shutdown and restart frequently. More discussion on OPBS can be found in [20].

In the L-ABS scenario, however, as \( \beta \) increases, the round number does not constantly decrease, as shown in Fig. 12. When there are 8,000–12,000 peers in the network and \( \beta \) is 3 or 4, the round number is reduced from 46–150 to 26–31, and not far from the one generated by OPBS. When \( \beta \) is larger than 4, the round number increases as the height of the spanning tree increases. In this scenario, a non-root peer \( x \) increases or decreases \( A_x \) only when it loses contact with its upstream peer. When one of \( x \)’s children leaves, \( x \) might unnecessarily keep many unused slots in its forwarding array between \( K_x \) and \( A_x \). However, \( x \) does not try to decrease \( A_x \), as the example shown in Fig. 7. Since \( x \) occupies a higher slot in the forwarding array of its upstream peer \( U_x \), \( U_x \) still keeps higher values of \( K_{UX} \) and \( A_{UX} \). This may repeat upward to the root. As a result, the round number might unnecessarily keep high.

In the A-ABS scenario, peers may act more aggressively, as described above and the example shown in Fig. 8. In the experiment, there is a 1\% possibility that a non-root peer \( x \) may try to decrease \( A_x \). The simulation result shows the round number furthermore reduced, as shown in Fig. 13. When there are 8,000–12,000 peers in the network and \( \beta \) is 4, the round number is reduced to 17–21, and very close to the one generated by
ADAPTIVE SCHEDULING OF MESSAGE FORWARDING OF DHT-BASED P2P BROADCAST

Fig. 12. Round number of the resultant broadcast schedule when in the L-ABS scenario.

OPBS. Since the round number is smaller than the height of the original broadcast tree, the time for broadcast is significantly reduced in terms of not only round number, but also hop count. Scenarios with different possibilities (5%, 10%, 50%, and 100%) are also simulated. The round number is furthermore reduced, but not significantly. Table 1 summarizes the required round numbers before and after redirection where there are 8,000~12,000 peers in the network and $\beta$ is 4.

Table 1. Required round number to complete a broadcast when there are 8,000~12,000 peers in the network ($\beta$ = 4).

<table>
<thead>
<tr>
<th>Divide &amp; Conquer [16]</th>
<th>[19] (w/o redirection)</th>
<th>OPBS</th>
<th>A-ABS</th>
<th>L-ABS</th>
</tr>
</thead>
<tbody>
<tr>
<td>40~120</td>
<td>42~280</td>
<td>15~20</td>
<td>17~21</td>
<td>26~31</td>
</tr>
</tbody>
</table>

Table 2 shows the maximal and average numbers of downstream peer to which a peer might have to forward broadcast messages after redirections have been made by the OPBS and ABS algorithms. The average numbers of downstream peers after redirection remain almost the same. These numbers are small because many peers are in the very bottom of the broadcast spanning tree. In fact, near a half of all peers are leaves. However, the maximal numbers are significantly reduced from 59~49 to 25~17 dependent on $\beta$. Thus, the load of peers is furthermore balanced after redirections have been made. In
Table 2. The maximal and average numbers of forwardings of a non-leaf peers when the network has 8,000~12,000 peers.

<table>
<thead>
<tr>
<th></th>
<th>β</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o redirection</td>
<td>Max</td>
<td>59</td>
<td>54</td>
<td>56</td>
<td>54</td>
<td>45</td>
<td>40</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>3.483</td>
<td>2.913</td>
<td>2.695</td>
<td>2.613</td>
<td>2.552</td>
<td>2.534</td>
<td>2.484</td>
</tr>
<tr>
<td>OPBS</td>
<td>Max</td>
<td>31</td>
<td>27</td>
<td>23</td>
<td>21</td>
<td>20</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>3.505</td>
<td>2.948</td>
<td>2.748</td>
<td>2.672</td>
<td>2.633</td>
<td>2.615</td>
<td>2.568</td>
</tr>
<tr>
<td>A-ABS</td>
<td>Max</td>
<td>24</td>
<td>20</td>
<td>19</td>
<td>17</td>
<td>17</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>3.557</td>
<td>3.004</td>
<td>2.803</td>
<td>2.728</td>
<td>2.671</td>
<td>2.648</td>
<td>2.607</td>
</tr>
<tr>
<td>L-ABS</td>
<td>Max</td>
<td>25</td>
<td>21</td>
<td>19</td>
<td>18</td>
<td>18</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>3.570</td>
<td>3.016</td>
<td>2.826</td>
<td>2.747</td>
<td>2.691</td>
<td>2.666</td>
<td>2.628</td>
</tr>
</tbody>
</table>

general, OPBS reduces the round number mostly, and peers have slighter more downstream peers. In the two ABS scenarios, the maximal and average numbers of downstream peers are almost the same. However, the round number generated in the A-ABS scenario is lower than in the L-ABS scenario.

Unlike OPBS which recomputes a whole new broadcast schedule for all peers whenever the network topology changes, in the ABS scenarios, only a small portion of peers are triggered to recompute their local broadcast schedule. As shown in Fig. 14, when there are 8,000–12,000 peers in the network, less than 30 peers (<1%) in average are triggered because some peer joins or leaves the network. When a peer is triggered, it invokes the procedure FindUpstream() to find a new upstream peer. Since some peers are probably forced to lose their upstream peers in this process, e.g. Case (3) of the ABS algorithm, a peer may invoke FindUpstream() more than once. However, it takes a short time for the new broadcast schedule to become stable. As shown in Fig. 15, each triggered peer may invoke ABS algorithm at most 5 times. The search space of each ABS

![Fig. 14. Average number of triggered peers when the network topology changes.](image1)

![Fig. 15. Maximal number of ABS invocations of a peer when the network topology changes.](image2)
invocation is the height of the original spanning tree, \( i.e., O(\log N) \), which is 22–26 or so when \( \beta \) is 4. As a result, the overhead of the ABS algorithm is low for a single peer and the whole network. The overhead is slightly increased in the A-ABS scenario; however, which is much outperforms L-ABS.

Peers may dynamically join and leave a P2P network. When the network topology changes, the underlying DHT has to reorganize its structure and then ABS will maintain the broadcast schedule adaptively. At this time, some peers will not receive broadcast messages temporarily. The higher a peer is in the broadcast spanning tree, the more damage its departure will cause. Fortunately, as stated above, most peers are in the bottom of the tree and only a very small portion of peers are near the root. Table 3 shows the percentages of peers that a broadcast message can reach during these periods. Most of the time, more than 99% of peers will still receive broadcast messages. Among the 10,000 or so events of peer departure in the simulations, there are less than 50 events causing that more than 10% of peers cannot receive broadcast messages.

Table 3. Percentages of reachable peers during the period when some peers have left but ABS does not yet repair the broadcast spanning tree.*

<table>
<thead>
<tr>
<th>( \beta )</th>
<th>min</th>
<th>1st quartile</th>
<th>median</th>
<th>max</th>
<th>count (&gt;90%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>51.01%</td>
<td>99.95%</td>
<td>100%</td>
<td>100%</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>49.82%</td>
<td>99.94%</td>
<td>99.99%</td>
<td>100%</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>50.19%</td>
<td>99.94%</td>
<td>99.99%</td>
<td>100%</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>49.62%</td>
<td>99.93%</td>
<td>99.98%</td>
<td>100%</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>49.64%</td>
<td>99.93%</td>
<td>99.98%</td>
<td>100%</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>49.57%</td>
<td>99.92%</td>
<td>99.98%</td>
<td>100%</td>
<td>49</td>
</tr>
<tr>
<td>8</td>
<td>49.58%</td>
<td>99.92%</td>
<td>99.98%</td>
<td>100%</td>
<td>36</td>
</tr>
</tbody>
</table>

* In the experiments, there are about 10,000 events of peer departure.

The authors must also note that broadcast messages may get lost because of other reasons, \( e.g., \) congestion or errors in physical links. As well, it is possible that broadcast messages will out-of-order arrive at peers. Broadcast in a dynamic P2P network studied in this work does best-effort message delivery to all peers in the network. Although ABS needs only a short time to maintain a new schedule for fast broadcast, there is still no guarantee that all peers in the network will receive the same messages in the same sequence order before a deadline without any loss. When an application uses broadcast to carry out information aggregation or arbitrary queries, the result is an estimate with a possibility of loss at a particular instant.

4. CONCLUSION

In this paper, the authors study fast command broadcast and information aggregation along a broadcast tree in a P2P-based cloud computing network where peers are located in a same data center. A discrete time model is used. An activated peer can send a message to a non-activated peer in a round. If every activated peer can forward a command message to a non-activated peer in every round, a command broadcast can
complete within $\log_2 N$ rounds. However, it is not easy for an activated peer to find a 
non-activated peer. A fully-distributed algorithm, named as the ABS algorithm, is pre-
sented. Similarly to [18-20], all peers together build a spanning tree based on a parent 
function. A peer is triggered, due to join or departure of other peers, to invoke the ABS 
algorithm to adaptively find a new upstream peer along the path from its parent upward 
to the root.

OPBS [20] recomputes a whole new broadcast schedule for all peers whenever the 
network topology changes. Hence, OPBS is preferred in a computer environment where 
peers do not shutdown and/or restart frequently. Since ABS is fully-distributed, only a 
very small number of peers are triggered to adaptively modify their local partial broad-
cast schedule. In the simulation, less than 1% of peers in average are triggered. Since the 
search space of a peer to find a suitable redirection in ABS is $O(\log N)$, the load of each 
triggered peer is low. Therefore, the overall overhead of ABS is low. Hence, ABS is 
suitable in more dynamic P2P-based cloud computing, where peers join or leave the 
network frequently.

In the A(ggressive)-ABS scenario, the simulation result shows that the time re-
quired to complete a command broadcast is significantly reduced in terms of both round 
number and hop count, and thus to complete an information aggregation. The round 
numbers are very close to those generated by OPBS. The maximal number of down-
stream peers to which a peer has to forward messages is reduced. As a result, the load of 
peers is furthermore balanced.

The parameter $\beta$ in Eq. (1) has an impact on the height of the original spanning 
tree and thus the performance of the ABS algorithm. When the tree is too short or too high, 
the reduction of the round number of the resultant broadcast schedule is not significant. 
The simulation result shows that it is a good choice to set $\beta$ around 4.

Similarly to OPBS, ABS uses a simple bottom up approach to search for a suitable 
redirection. An algorithm can try to find more redirections by using more complex ap-
proaches in a larger search space; however, the overhead may also increase. The simula-
tion result shows that ABS has a very good performance compared to the centralized 
optimal OPBS.

It is worth noting that OPBS and ABS do not restructure the underlying DHT. In 
fact, they both rely on the DHT to provide routing service so that peers can communicate 
with each other and thus together maintain a spanning tree for fast broadcast. If the DHT 
is partitioned, OPBS and ABS cannot build a broadcast spanning tree. If a peer left and 
later rejoins the network, it may have knowledge of a partial broadcast schedule in its 
cache. It is under investigation how to use the cached information to further improve 
ABS. Generally speaking, broadcast along a spanning tree is not a reliable service. It is 
possible to construct multiple trees to prevent from a single point of failure [21]. Still, 
packets may get lost due to many reasons, e.g. drops or errors in physical links, or arrive 
at the destination out-of-order because of routing decision. The authors will further study 
these issues in the future.

REFERENCES

1. P. Mell and T. Grance, “The NIST definition of cloud computing,” Special Publica-
tion 800-145, Computer Security Division, Information Technology Laboratory, Na-
ADAPTIVE SCHEDULING OF MESSAGE FORWARDING OF DHT-BASED P2P BROADCAST


Jeng-Wei Lin (林正偉) received the B.S. degree in Computer Information Science from National Chiao Tung University, Hsinchu, Taiwan, in 1994, and the M.S. and Ph.D. degrees in Computer Science and Information Engineering from National Taiwan University, Taipei, Taiwan, in 1996 and 2005, respectively. Since 2005, he joined the Department of Information Management, Tunghai University, Taichung, Taiwan. From 1996 to 2005, he was a Research Assistant with the Institute of Information Science, Academia Sinica, Taipei, Taiwan. His current research interests include multimedia systems, P2P networking, cloud computing, and Chinese information processing.

Tun Wei Wang (王敦威) received the B.S. degree in Computer Science from Tunghai University, Taiwan in 2002 and the M.S. degree in Biomedical Engineering from National Taiwan University, Taiwan in 2005. Since 2009, he joined in the digital archive project of National Palace Museum, Taiwan. He is pursuing the Ph.D. degree in Engineering Science in National Taiwan University. His recent research interests include P2P networking, communication encryption, social networking, bioinformatics, and digital archive.

Ray-I Chang (張瑞益) received his Ph.D. degree in Electrical Engineering and Computer Science from National Chiao Tung University in 1996, where he was a member of the Operating Systems Laboratory. As his Ph.D. degree was pursued, he joined the Computer Systems and Communications Laboratory of the Institute of Information Science, Academia Sinica, as a Postdoctoral Researcher. In 2002, he joined the Department of Information Management, National Central University, Taiwan.
In the next year, he was invited to join the Department of Engineering Science, National Taiwan University. Dr. Chang has published over 150 original papers, including papers published in journals such as IEEE Transactions on Multimedia, IEEE Transactions on Broadcasting, and IEEE Transactions Neural Networks. He was in the Editorial Committee of some journals including International Journal of Information Processing and Management, Journal of the Chinese Institute of Industrial Engineers, Management and Information Science, and International Journal of Applied Metaheuristic Computing. His current research interests include multimedia networking and data mining. Dr. Chang is a member of IEEE, CERP and IICM.