IPSJ Transactions on Advanced Computing Systems Vol. 3 No. 3 138–152 (Sep. 2010)

Regular Paper

A Strategy for Efficient Update Propagation on Peer-to-Peer Based Content Distribution Networks

Ai Hayakawa,[†1] Masato Asahara,[†1] Kenji Kono[†1] and Toshinori Kojima[†1]

As demand for high fidelity multimedia content has soared, content distribution has emerged as a critical application. Large multimedia files require effective content distribution services such as content distribution networks (CDNs). A recent trend in CDN development is the use of peer-to-peer (P2P) techniques. P2P-based CDNs have several advantages over conventional non-P2P-based CDNs in scalability, fault resilience, and cost-effectiveness. Unfortunately, P2P-based content distribution poses a crucial problem in that update propagation is quite difficult to accomplish. This is because peers cannot obtain a global view of replica locations on the network. There are still several issues in conventional approaches to update propagation. They degrade the scalability, the fault resilience, and the cost-effectiveness of P2P-based content distribution, they also consume the network bandwidth, or take a long time. In this paper, we propose the speculative update, which quickly propagates an update to replicas with less bandwidth consumption in a pure P2P fashion. The speculative update enables a fast update propagation on structured P2P-based CDNs. Each server attempts to determine the directions in which there will be replicas with a high probability and speculatively relays update messages in those directions. Simulation results demonstrate that our mechanism quickly propagates an update to replicas with less bandwidth consumption. The The speculative update completes update propagation as fast as the simple gossip-based update propagation even with up to 69% fewer messages per second. Compared to the convergence-guaranteed random walk, the speculative update completes an update propagation faster by up to 92%.

1. Introduction

As the demand for high fidelity multimedia content has soared, content distribution has emerged as a critical application. Large multimedia files require effective content distribution services such as content distribution networks (CDNs). Typical content distribution solutions enable clients to obtain a replica of content from a dedicated server at the edge of the Internet. The best example of such solutions is Akamai [1], which runs several tens of thousands of servers all over the Internet. This solution provides several benefits, such as a short response time for obtaining content and the load balancing of servers over the network.

A recent trend in CDN development is the use of peer-to-peer (P2P) techniques. P2P-based CDNs [2)–8)] have several advantages over conventional (non-P2P-based) CDNs. First, P2P-based content distribution systems can easily scale out to increase the number of servers, clients, or the variety of content. Second, the decentralized architecture is resilient against a server failure. The entire system keeps working regardless of several servers crashes because every server has a homogeneous functionality and operates independently. Third, P2P-based CDNs are cost-effective to construct. Conventional CDNs require a powerful dedicated server responsible for edge server management, replica distribution, and client query routing. On the other hand, P2P-based CDNs aggregate commodity PCs to provide a scalable service.

While the P2P-based architecture provides the attractive features described above, it is quite difficult to propagate updates to all replicas distributed over the Internet, because no peer maintains every location of replica content. Conventional approaches to the update propagation can be classified into three categories. First, in the centralized server approaches, used in Globule [9], a centralized server holds a map of the current replica locations, and propagates updates to all replicas according to the map. This approach diminishes the scalability and the fault-resilience of P2P-based systems because the centralized server may become a bottleneck and brings a single point of failure. Second, the structured overlay approaches [10)–12)] construct structures, such as trees or distributed hash tables (DHTs), to locate replicas. Although these approaches enable a server to figure out replica locations without global information, it is expensive to build and maintain complex structures solely for update propagation. Finally, gossip protocols [13)–16)] and random walks [17)–22)] are suitable for large-scale, dynamic P2P-based CDNs. They propagate update messages without any structures nor global information. However, there are some issues with them. For example, the gossip protocols inherently induce many redundant messages which consume

†1 Department of Information and Computer Science, Keio University

the network bandwidth. Another example is that, to use the random walks, the underlying overlay must be carefully designed because the overlay topology significantly affects the duration and the coverage of the update propagation.

In this paper, we propose the *speculative update*; each server determines the directions in which there will be replicas with a high probability and speculatively relays update messages in those directions. The speculative update enables a fast update propagation on structured P2P-based CDNs. This mechanism is speculative in the sense that it is uncertain whether there are replicas in those directions, but we hope there will be. The speculative update uses a short record of the query directions (i.e., the directions from which queries come) to determine the directions. By propagating the update messages towards the probable replica locations, the speculative update accomplishes a quick propagation of updates; replicas are updated in a timely fashion.

The speculative update has three advantages over the current approaches. First, it maintains the scalability, the fault resilience, and the cost-effectiveness of P2P-based content distribution; no dedicated server nor an additional structure are needed to manage replicas. Second, the speculative update reduces the bandwidth consumption compared to simple gossip protocols. Although the speculative update also employs gossip-based update messages to keep up the coverage of update propagation, it assigns these messages a small TTL unlike simple gossip protocols. By combining the update message which is relayed in the determined direction and gossip-based messages with a small TTL, the speculative update efficiently propagates updates to replicas with less bandwidth consumption. Finally, the speculative update accelerates the update propagation by determining the direction in which an update message should be relayed. The speculative update completes an update propagation faster than the random walks.

We applied the speculative update to ExaPeer [23], a P2P-based CDN, and evaluated it on three synthetic networks simulating the Internet. Simulation results demonstrate that the speculative update quickly propagates updates to replicas while keeping the bandwidth consumption low. Although the speculative update completes an update propagation as fast as a gossip protocol, it relays up to 69% fewer messages per second than a gossip protocol. Compared to the convergence-guaranteed random walk [24], the speculative update completes the update propagation faster by up to 92%.

The rest of the paper is organized as follows. Section 2 clarifies issues involved in the update propagation on P2P systems and discusses related work. Section 3 defines the system model to which the speculative update can be applied. Section 4 explains the basic architecture of ExaPeer. Section 5 describes the design of the speculative update and applies it to ExaPeer in a case study. In Section 6, we evaluate the speculative update through several simulations. Finally, Section 7 concludes the paper.

2. Problem Statement and Related Work

In this section, we discuss issues involved in the update propagation on P2P-based CDNs. In P2P-based CDNs, the lack of global information about locations of replicas makes it difficult to efficiently propagate updates. We introduce conventional approaches to update propagation in P2P-based CDNs and clarify the limitations of these approaches. **Table 1** summarizes the classification of the current approaches.

The centralized server approaches, which are used in popular CDNs, assign a dedicated server to maintain a map of current replica locations and propagate updates. Globule [9] takes the centralized server approach. Each content has one master server responsible for the update propagation. The master server maintains IP addresses of servers holding a replica. This method enables a server to easily figure out where replicas are positioned, but has difficulty in coping with a network growth. Though centralized server approaches allow P2P-based CDNs to directly advertise an update to all replicas, they suffer from a poor scalability and bring a single point of failure problem.

Table 1 Design alternatives for update propagation.

	Scalability	Fault resilience	Building cost	Bandwidth consumption	Dissemination time
Centralized server	×	×	×	√ √	√ √
Structured overlay	√ √	√	×	√ √	√ √
Gossip	√ √	√ √	√ √	×	√
Random walk	√ √	√ √	√ √	√√	×
Speculative update	√ √	√ √	√ √	√	√

√ √ good √ moderate × bad

The structured overlay approaches construct structures, such as distributed hash tables (DHTs) or trees, to locate replicas. These approaches construct a structure for update propagation completely or partially independent of the one for content distribution. The update propagation structure enables P2P-based CDNs to complete their update operations with a shorter convergence time and less redundant update messages. However, the structured overlay approaches make P2P-based CDNs more complex. They introduce additional development and maintenance costs for the update propagation structure. In order to avoid the degradation of P2P-based CDN's scalability in terms of client growth and resilience to server and network failures, the update propagation structure will become complex as well as the content distribution structure. In particular, it is difficult to make the update propagation structure deal with drastically rapid demand fluctuations such as flash crowds [25]. This is because in flash crowds, P2P-based CDNs dynamically change the number and the location of replicas in a short term, e.g., in several tens of seconds.

OceanStore [10] takes the structured overlay approach. OceanStore maintains two tier replicas. A small durable primitive tier keeps the most up-to-date data. Once the update is committed, the primitive tier multicasts the result of the update down the dissemination tree based on Tapestry [26]. However, the implementation of OceanStore is quite complex.

Li, et al. [11] proposed a locality-aware hierarchical structure. The DHT-based upper layer is Chord-based [27] and consists of more reliable and powerful servers. A replica server in the second layer attaches to a physically close server in the upper layer. When an update occurs, an update tree is built dynamically upon the upper layer to propagate the update. However, their approach introduces an additional building cost of the update propagation mechanism.

The gossip-based approaches are much simpler; they flood update messages on the network. Each server relays the update message to all the servers directly connected to itself. Despite the fact that they do not require a particular structure to maintain replica locations, they propagate an update fully and quickly. The gossip-based approaches are also scalable and resilient to failures because there is no dedicated server unlike the centralized server approaches. However, they inherently induce many redundant messages. Datta, et al. [28] proposed a gossip-based update propagation for highly unreliable P2P systems. Based on the number of duplicate messages received at a particular server, the server tunes the probability to relay the update message in order to reduce the bandwidth consumption. But the effectiveness of update propagation still depends on the number of messages.

Random walks are particularly attractive in P2P-based CDNs. Although random walks do not require the structure to maintain the state of replica locations like the gossip-based approaches, they can propagate an update with less bandwidth consumption than the gossip-based approaches. Zhong, et al. [24] proposed the convergence-guaranteed random walk. A walker randomly chooses a next hop based on the Metropolis-Hastings algorithm so that the walker's probability distribution of visiting each node converges on an application-desired one. As a result, the convergence-guaranteed random walk effectively propagates an update to desired node with a few walkers. The speculative update is inspired by their approach. While the convergence-guaranteed random walk has advanced conventional random walks, it depends on the number of initial walkers and the overlay topology. At worst, it may take a long time to converge on the desired server visitation distribution.

The goal of our proposal called the *speculative update* is to overcome the shortages of the above conventional approaches. The speculative update should be attractive in P2P-based CDNs; it should not degrade the features of P2P-based CDNs such as the high scalability, the high resilience, and the low cost. The speculative update employs gossip messages to reduce the costs of building an additional structured overlay, although the dissemination time is slightly increased compared with structured overlay approaches. By leveraging the structure of P2P-based CDNs which is used for content distribution, the speculative update enables a faster update propagation than random walks and consume less bandwidth than gossip-based approaches.

3. System Model

In this section we discuss the system model on which the speculative update can be embodied. The speculative update is a methodology of propagating updates, and the key idea behind it is to infer the direction in which there will

be replicas. We believe this methodology can be embodied on P2P-based CDNs with the following two properties. The concrete implementation is dependent on the details of the underlying P2P-based CDNs and must be carefully developed for each P2P-based CDN.

- The CDN uses a structured P2P overlay to distribute replicas.
- Each server maintains a local state, which can be used as a criteria to position replicas. More precisely, each server can be located through virtual positions defined on the overlay network.

Recent P2P-based CDNs often have these properties. First, P2P-based replication systems often use metrics to position replicas on the servers close to clients. For example, in CoralCDN [4] which is a P2P-based CDN, each server monitors the number of relayed queries for each content. Then, a CoralCDN server caches the content if the number of relayed queries exceeds a threshold.

Second, recent P2P-based CDNs often use a structured overlay so that a client always locates an original content or a replica of the content. In structured networks, the overlay topology is tightly controlled and the content is placed at precisely specified locations. These systems enable a client query to be efficiently routed to the node hosting the desired content or the replica. Note that unstructured P2P-based CDNs are currently out of the scope of the speculative update.

4. ExaPeer

Before explaining our mechanism, we summarize the ExaPeer [23] architecture. In the following section, we explain the case study using ExaPeer to demonstrate the effectiveness of the speculative update. ExaPeer is a P2P-based CDN, which dynamically repositions replicas on the basis of demand fluctuations.

4.1 Basic Architecture

ExaPeer constructs an overlay network using Content Addressable Network (CAN) [29], which is a DHT, and Global Network Positioning (GNP) [30], which is a network coordinate system. GNP models the Internet as a d-dimensional space and assigns a d-dimensional coordinate to each server so that the Euclidean distance between any pair of coordinates approximates the round-trip-time (RTT) between the servers. CAN assigns each server a virtual d-dimensional coordinate and divides the d-dimensional space into *zones*. To construct a topologically-aware overlay network, ExaPeer uses coordinates calculated with GNP when CAN assigns each server a virtual coordinate. ExaPeer also allocates one server to each zone so that the server's coordinate is contained in the zone. CAN generates a key from content and maps the key to a point (P) in the coordinates space. Then CAN stores the content in the server that is responsible for the zone within which P lies.

A client obtains content from ExaPeer on the basis of the CAN protocol. A query for content is relayed through intermediate servers to the origin server which maintains a zone containing the coordinates corresponding to the content. If a server on the path hosts a replica of the content, it provides the replica to the client instead of an origin server.

4.2 Replica Placement

ExaPeer dynamically repositions replicas on the basis of demand fluctuations in a P2P fashion. Since ExaPeer relays a query so that the query gets closer to the final destination, a server that relays a query from one direction to another can figure out that there is a high-demand area in the direction from which the query comes. If there are many queries for content from one direction, a server provides a replica of the content. In ExaPeer, a server monitors the degree of Access Path Convergence (APC degree) which indicates the number of transferred queries to determine whether to host a replica. **Figure 1** illustrates ExaPeer mechanism in 2-dimensional space, in which server A is selected as the origin server. An APC degree on the server Q is three. Then, ExaPeer puts a replica on Q to deal with

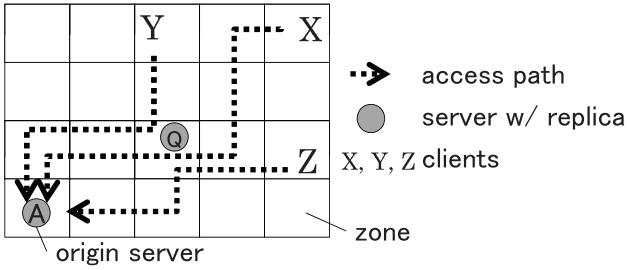


Fig. 1 Structure of ExaPeer in a 2-dimensional space.

the demand of clients X, Y, and Z.

5. Speculative Update

5.1 Overview

The speculative update accelerates the update propagation by determining the direction in which there might be replicas by exploiting the server's local state used for replica repositioning. In recent P2P-based CDNs, a server estimates the benefit of hosting a replica to determine whether to host the replica. For estimation, a server monitors the local state which represents the popularity of a content (e.g., the number of relayed queries). If the local state of a server is likely to host a replica, we can infer that the server would be in the direction in which many replicas are positioned.

Once a server determines the direction where replicas are positioned, it sends out a *speculative message* in that direction. A speculative message is relayed in the determined direction and it advertises an update to the replicas on the path of the message. To efficiently advertise an update, the speculative update generates gossip-based update messages with a small TTL, called *complementary messages*, when a speculative message reaches a replica. P2P-based CDNs tend to position several replicas on the servers close to each other on the overlay for some reasons, e.g, load balancing of the servers. Thus, we can infer that other replicas would be positioned around a replica. Complementary messages advertise an update to those replicas efficiently in a few hops.

Figure 2 illustrates how the speculative update propagates update messages. The pseudo code of the speculative update is shown in Appendix. First, the origin server sends out a speculative message to server A and complementary messages to its neighbors. Then, a speculative message reaches server B hosting a replica (Fig. 2 (1)). B also attempts to determine the direction in which an update message should be relayed and sends another speculative message to server C (Fig. 2 (2). See the pseudo code from line 7 to 16 in Appendix A.1). At the same time, B sends out complementary messages to its neighbors (Fig. 2 (3). See the pseudo code from line 4 to line 6 in Appendix A.1). One of the complementary messages reaches server D, which hosts a replica.

The churn tolerance of the speculative update depends on the one of the structured overlay on which a CDN is built. Each server joins and leaves the CDN following the protocol of the CDN's structure such as CAN [29] and Chord [27]. For example, when a node joins or leaves CAN, to avoid redundancy it affects only few other nodes. The number of neighbors the node maintains is independent of the total number of nodes on the overlay. Thus, for a d-dimensional space, the churn affects only $O(d)$ nodes. In the case of Chord, a node joining or leaving an N-node Chord network uses $O(\log^2 N)$ messages to re-establish its finger tables [27]. The speculative update achieves about the same churn tolerance as the CDN's structure.

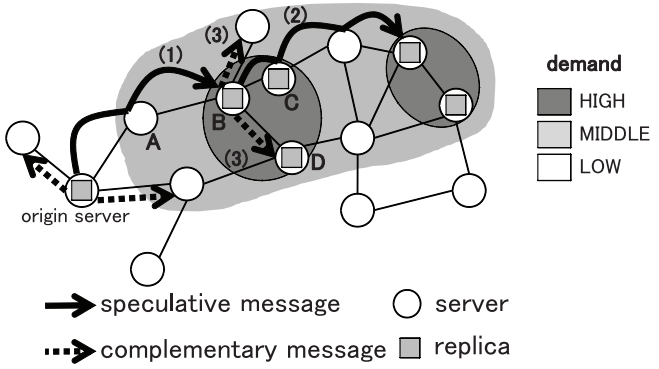


Fig. 2 Overview of the speculative update.

In the following section, we demonstrate how the speculative update generates a speculative message and a complementary message by using a case study. As described in Section 3, the implementation of the speculative update is dependent on the underlying CDN because the mechanism to position replicas differs from CDN to CDN. In this case study, we use ExaPeer described in Section 4. Although we are challenging to apply the speculative update to other P2P-based CDNs, it is included in the future work of this paper.

5.2 Speculative Message

5.2.1 Basic Mechanism

To determine the direction in which a speculative message should be relayed, the speculative update first calculates a *primitive vector*. A primitive vector roughly represents the direction of replica locations. A primitive vector is a

simple vector sum calculated by a server using the local state, that is the APC degree in ExaPeer, of the neighbor servers. In popular P2P-based CDNs, a client query often chooses different access paths even if the same client requests the same content. This is because the routing protocol has some flexibility to avoid relaying a query to a failed server or a hot spot server. Therefore, some queries reach the servers which are slightly far from the clients. The speculative update takes a cue from these queries for calculating a primitive vector; it roughly determines that replicas are positioned in a direction if a query is relayed from that direction.

In ExaPeer, a primitive vector $\boldsymbol{P}$ of server i is calculated by the function:

$$\boldsymbol{P}_i = \sum_j \alpha_j \boldsymbol{x}_{ij}, \text{ s.t. } \mid \alpha_j \boldsymbol{x}_{ij} \mid = d_j$$

where d_j is the APC degree of the neighbor server j and $\boldsymbol{x}_{ij}$ is a directional vector from server i to server j. **Figure 3** illustrates an example to calculate a primitive vector in ExaPeer. In Fig. 3, the APC degrees of servers K, L, M and N are 2, 0, 0, 1, respectively, and the directional vectors from the server A are $(0,1)$, $(-1,0)$, $(0,-1)$ and $(1,0)$, respectively. Thus, the primitive vector on the server A is set to $\boldsymbol{P}_A = (1,2)$.

Based on a primitive vector, the speculative update calculates a *target vector*. A target vector denotes the direction of replica locations on the basis of a primitive vector, but it is more accurate than a primitive vector. Once a server calculates a target vector, it also calculates an ID so that a speculative message, whose destination is the calculated ID, is relayed in the direction of a target vector on the CDN.

There are two key ideas to calculate a target vector. First, a server ignores the neighbor servers whose directional vector is the opposite direction of a primitive vector. This operation roughly adjusts a target vector in the direction of which more replicas are positioned. Second, a server applies a converting function to the values representing the local states of the neighbors, which results in reversing the order of input values. In P2P-based CDNs, almost all client queries in an area are served by the replicas closest to the clients. On the other hand, a few remaining queries take a roundabout route to avoid the replicas for some reasons. Thus, we can infer that an actual direction in which many replicas are positioned is between the vectors representing the directions from where the remaining queries come.

With these two operations, in ExaPeer, the target vector $\boldsymbol{T}$ of server i is calculated by the function:

$$\boldsymbol{T}_i = \sum_{j, \boldsymbol{P}_i \cdot \boldsymbol{x}_{ij} > 0} \beta_j \boldsymbol{x}_{ij}, \text{ s.t. } \mid \beta_j \boldsymbol{x}_{ij} \mid = f(d_j)$$

$$f(d_j) = (\max(d_k) + \min(d_k)) - d_j \quad (k = 0, 1, \cdots, N)$$

where N is the number of neighbors such that $\boldsymbol{P}_i \cdot \boldsymbol{x}_{ij} > 0$. **Figure 4** illustrates an example to calculate the target vector in ExaPeer in a 2-dimensional space. Since the primitive vector of server A is $(1,2)$, the APC degrees of L and M are

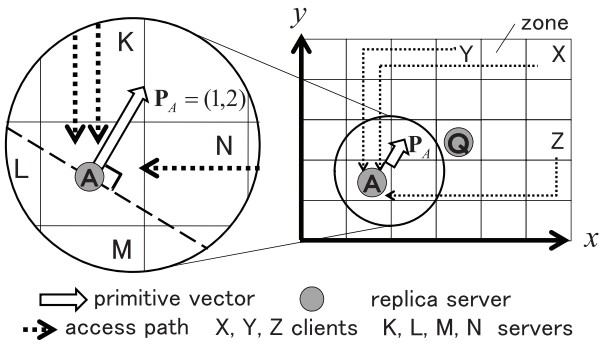


Fig. 3 Example of a primitive vector in ExaPeer.

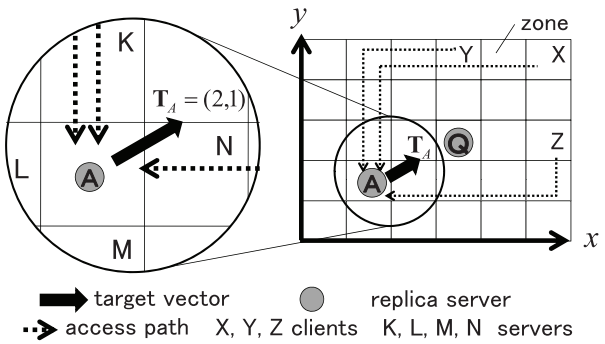


Fig. 4 Example of a target vector in ExaPeer.

ignored. The APC degrees of K and N are then converted to 1 and 2, respectively. As a result, the target vector of A is set to $\boldsymbol{T}_A = (2, 1)$. A speculative message is relayed on the basis of the protocol of the overlay, which is CAN in ExaPeer, and advertises an update to replicas on the path of the message. The speculative message whose target vector is $\boldsymbol{T}_A$ reaches server Q and advertises an update to the replica of Q.

In case of a 3-dimensional ExaPeer, used in the evaluation in Section 6, a target vector is calculated similarly. Let the coordinate of A be $(0, 0, 0)$. Suppose that the neighbors of A are K, L, M, N, O, and P and their coordinates are $(0, 1, 0)$, $(-1, 0, 0)$, $(0, -1, 0)$, $(1, 0, 0)$, $(0, 0, 1)$, and $(0, 0, -1)$, respectively. The directional vector of the neighbor is the same as its coordinate. If the APC degree of the neighbors K, L, M, N, O, and P are 1, 0, 0, 1, 2, and 0, respectively, the primitive vector of server A is $(1, 1, 2)$ and L, M and P are ignored. Among the neighbors K, N, and O, the maximum APC degree is 2 and the minimum is 1. As a result, the target vector $\boldsymbol{T}$ is calculated as follows:

$$\boldsymbol{T} = (3-1)\begin{pmatrix}1\\0\\0\end{pmatrix} + (3-1)\begin{pmatrix}0\\1\\0\end{pmatrix} + (3-2)\begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}2\\2\\1\end{pmatrix}$$

The basic mechanism of a speculative message almost correctly calculates the direction in which replicas are positioned. However, in some situations, a server calculates an incorrect target vector. In the next section, we explain how to improve the basic mechanism of a speculative message.

5.2.2 Solution for Incorrect Backtracking

A speculative message sometimes goes back to replicas on which a speculative message has already passed. **Figure 5** illustrates an example of this situation. Server B calculates the target vector in the direction of server A and sets a new destination of a speculative message on the basis of the target vector. Then, the speculative message backtracks in the direction where there are updated replicas.

To prevent a speculative message from inappropriately backtracking, when a server calculates a primitive vector, it ignores the local state of the neighbors to calculate a primitive vector if their replicas have been updated already. When a server asks its neighbors about their local states, the neighbor replies with 0 as its local state if it has already been updated. (See the pseudo code from line 2 to 8 in Appendix A.3). In Fig. 5, B calculates a primitive vector without A. Then, a primitive vector of B heads in the opposite direction of A with a high probability. In Fig. 5, a speculative message is correctly sent out to C and reaches another high-demand area.

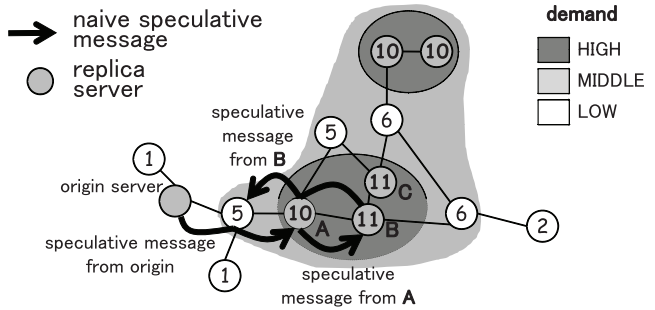


Fig. 5 Example of messages going backward. A new speculative message sometimes goes back to the areas where a speculative message has already passed. The number in each circle indicates the local state value used by the speculative update.

5.2.3 Solution for Looping

To solve this problem, a server relays a speculative message without any modification even if that server has a replica (See line 3 of the pseudo code in Appendix A.1). After having relayed the speculative message to its destination, a server with a replica sends out another speculative message heading towards the direction of the new target vector it recalculates (from line 7 to 16 in Appendix A.1). In the case of **Fig. 6**, in addition to sending a new speculative message to C, the server B relays the speculative message sent by an origin server to the server D. As a result, the speculative message can reach the replicas within another area. Although this solution slightly increases the number of speculative messages, the coverage of update propagation to replicas significantly improves.

5.3 Complementary Message

The speculative update employs the *complementary message* to improve the coverage of update propagation to replicas. When a server with a replica receives a speculative message, it sends out complementary messages with a unique mes-

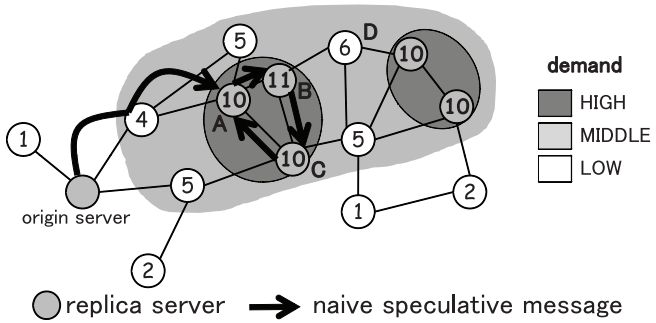


Fig. 6 Example of looping. The speculative message is relayed between A, B and C. Thus, it cannot reach replicas within another area . The number in each circle indicates the local state value used by the speculative update.

sage ID to its neighbors (See the pseudo code from line 4 to 6 in Appendix A.1). A complementary message advertises an update like a speculative message. Unlike a speculative message, however, it is relayed on the basis of the typical gossip protocols; when a server receives a complementary message, it relays the message to all neighbors except the one that delivered the incoming complementary message. When a server with a replica receives a complementary message, it relays the complementary message and sends out a speculative message (See the pseudo code in Appendix A.2). To reduce unnecessary transfers, a server receiving a complementary message with the same message ID as one it has received before, avoids relaying the message to any neighbors.

There are two important roles of a complementary message. First, it advertises an update to all replicas around an updated server which is updated with a speculative message. A speculative message quickly advertises an update to the replicas distant from an origin server. However, it is not good at advertising an update to all nearby replicas. A complementary message enables the speculative update to propagate an update to not only distant replicas but also all nearby ones.

Second, a complementary message accelerates the speculative update at the beginning of the update propagation process. In structured P2P-based CDNs, an origin server receives client queries from all directions on the logical space of the overlay. Thus, it is difficult for an origin server to send a speculative message with high accuracy. A complementary message can reach replicas missed by a speculative message from an origin server. Moreover, the servers which received the complementary message sends out more accurate speculative messages, because they receive client queries in limited directions due to the query routing protocol of the P2P-based CDN.

To limit bandwidth consumption, the speculative update assigns a complementary message a lower dissemination priority. The speculative update significantly reduces the TTL of a complementary message. Unlike the typical gossip protocols, these are realistic settings. This is because all a complementary message needs to do is to advertise an update to nearby replicas, while the typical gossip protocols have to propagate messages to all participants.

6. Evaluation

In this section, we demonstrate that the speculative update enables an efficient update propagation with several simulations. We compare the speculative update with the simple gossip-based update propagation and convergence-guaranteed random walks [24]. We apply the speculative update to ExaPeer [23], which is implemented on Overlay Weaver [31]. Overlay Weaver is a toolkit that enables to easily construct a large-scale overlay network and to emulate its behavior.

6.1 Evaluation Methodology

To evaluate the effectiveness of the speculative update, we measured four key metrics as follows.

- *The time for update propagation*, which shows the time it takes for replicas to be updated.
- *Success rate of update propagation*, which is the ratio of replicas that have received an update message.
- *The ratio of queries which reach an updated replica.* This measurement confirms queries from the clients in high-demand area reach an updated content in a short time.
- *The number of relayed messages*, which shows the overhead caused by an update propagation. The fewer update propagation messages are relayed, the lower overhead an update propagation induces.

To confirm the behavior of the speculative update, we used three data sets of

Internet measurement data to emulate the network latency. These data sets are used to calculate GNP coordinates. The following data sets are often used for evaluation purposes.

- *King*: The data set provided by the p2psim project [32]. It contains latency measurements between a set of 2,048 DNS servers.
- *Meridian*: The data set used by the Meridian project [33]. It contains pairwise RTT measurements between 2,500 DNS servers whose IP addresses are unique, spanning 6.25 million node pairs. The data were collected on May 5–13, 2004. We selected about 2,200 nodes with unique GNP coordinates.
- *PlanetLab*: We used latency measurements from 105 PlanetLab [34] nodes between September 4–22, 2008.

In the following experiments, we construct ExaPeer with a 3-dimensional GNP in a cube space, 5,000 on each side. To generate a high, localized demand area, we built client clusters on each topology. Every 300 milliseconds one client sends a query for content. 98% of the queries are sent from the clients selected by the following scheme and 2% of the queries are sent from randomly selected clients. Clients sending 98% of queries in the King, the Meridian and the PlanetLab data sets, are selected in a spherical space whose center coordinates are $(90, 90, 90)$, $(90, 90, 90)$, and $(0, -2000, 3000)$, and radii are 80, 100, and 1,500, respectively.

In this simulation, the server whose coordinate is the closest to a point $(-5000, -5000, -5000)$ is selected as an origin server. ExaPeer placed replicas on about thirty servers with the King data set, sixty servers with the Meridian data set, and ten servers with the PlanetLab data set. For the simulation, we set the TTL of a complementary message to 7 hops with the King data set and 3 hops with the Meridian data set. In this simulation, we also demonstrate that the coverage of simple gossip-based update propagation degrades if we simply set a TTL to the gossip messages. To do that, we conducted the simulation of the simple gossip-based update propagation with limited TTL as well as that without any TTL. We set this TTL to 10 hops in the simulation with the King and the Meridian data sets.

As for the convergence-guaranteed random walk, multiple independent random walkers can be used to clarify the difference caused by the number of update messages. It is expected that k independent random walkers after T steps tend to cover nearly an equal number of nodes as one random walker after $k \cdot T$ steps [24]. Thus, the origin server initiates 500 walkers. For the simulation with the PlanetLab data set, because there are very few servers compared with the other two data sets, we set the TTL of complementary messages to 2 hops. For the same reason, the convergence-guaranteed random walk initiates 6 walkers. In the same way as the simulations with the King and the Meridian data sets, we also conducted the simulations of the simple gossip-based update propagation with and without a limited TTL. We set the TTL of the simple gossip-based update messages to 3 hops.

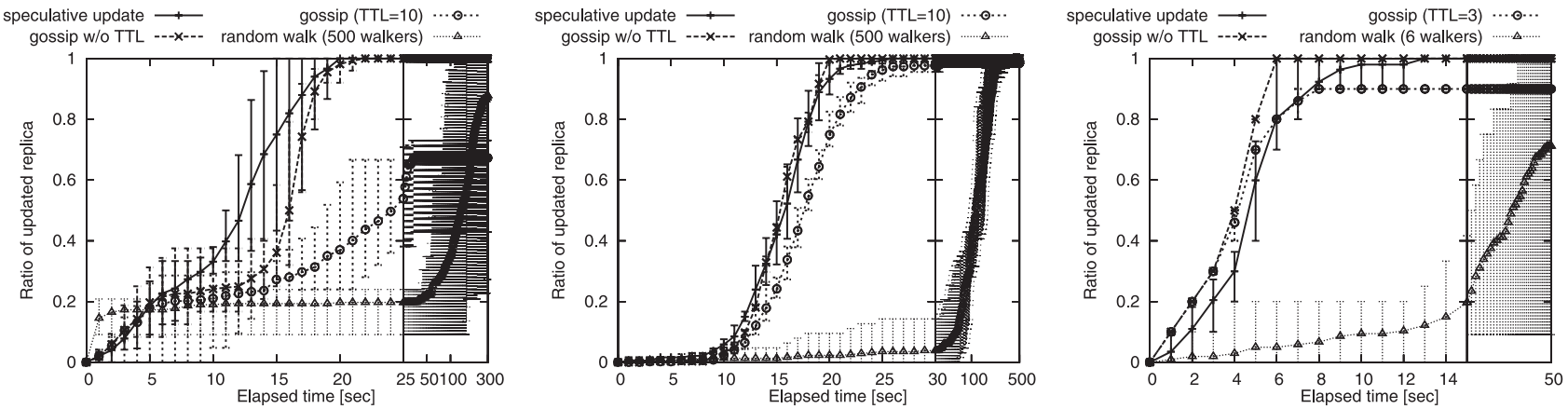
To clarify the essential difference between the mechanisms, we assume that the network condition is ideal in that the latency between servers is negligible. Due to the simulation environment, a server sends gossip-based messages to the neighbor servers one by one at regular intervals of 500 milliseconds. In a similar way, a server relays random walkers and speculative messages at regular intervals of 500 milliseconds.

6.2 Speculation Efficiency

Figure 7 shows the time for update propagation of each mechanism. In Fig. 7, the x-axis shows the elapsed time from the update initiation, and the y-axis shows the ratio of replicas which received an update message. The error bars show the maximum and the minimum of the observed values during the simulation. The result of the simulation with the King data set is shown in Fig. 7 (a). The speculative update completed an update propagation 9% faster than the simple gossip-based update propagation without any TTL and 82% faster than the convergence-guaranteed random walk.

With the Meridian data set shown in Fig. 7 (b), the speculative update completed the update propagation as fast as the simple gossip-based update propagation. The difference between the time for update propagation of the speculative update and that of the simple gossip-based update propagation is 5 seconds. Compared to the convergence-guaranteed random walk, the speculative update completed an update propagation 92% faster.

The simulation result with the PlanetLab data set is shown in Fig. 7 (c). Compared to the convergence-guaranteed random walk, the speculative update completed an update propagation 86% faster. We conducted a simulation with the



(a) King (b) Meridian (c) PlanetLab

Fig. 7 The time for update propagation. The speculative update completed an update propagation as fast as the simple gossip-based update propagation and faster than the convergence guaranteed random walk. The speculative update also propagated the update to all the replicas.

random walk producing more walkers, but the update propagation time did not improve so much. Since the simple gossip-based update propagation induces more messages than the speculative update, it completed an update propagation faster than the speculative update. However, as shown in Fig. 7 (a) and Fig. 7 (b), the speculative update achieves as fast an update propagation as the simple gossip-based update propagation when the number of servers increases.

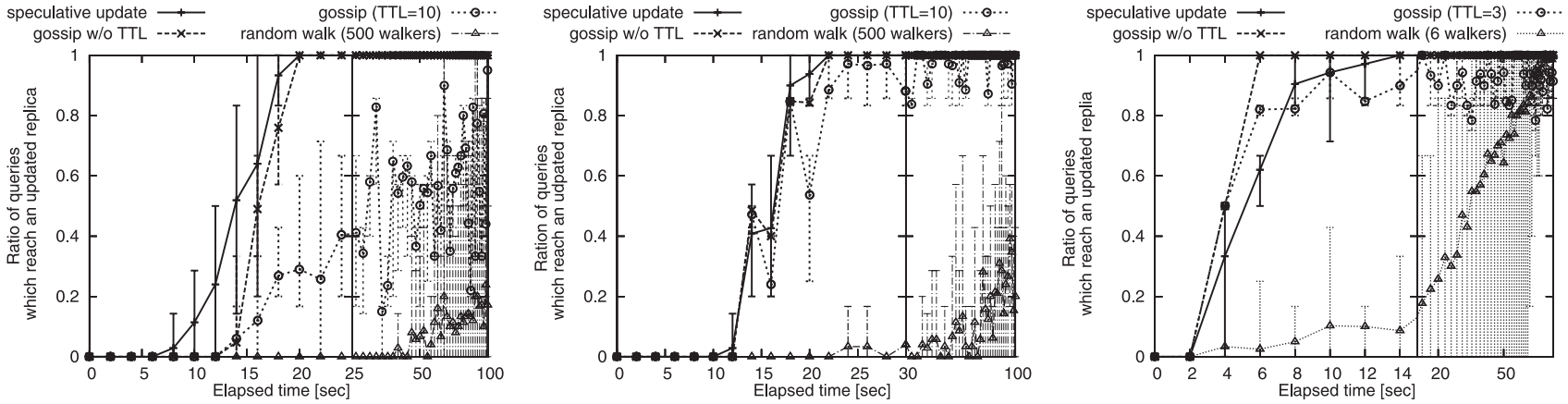
For all data sets, the convergence-guaranteed random walk takes the longest time to complete an update propagation. This is because the convergence time of the random walk is affected by the overlay topology and the number of nodes. In particular, the convergence time becomes larger if the diameter of the overlay structure is large [24]. Since CAN topology, which is the basic structure of ExaPeer, is a tori-like structure and tends to have a large diameter, the convergence time became larger.

Figure 7 also shows that the speculative update enables the update propagation to all replicas even if we assign complementary messages a small TTL. The speculative update advertised an update to all the replicas, while the simple gossip-based update propagation with a limited TTL could not. The simple gossip-based update propagation with TTL = 10 hops propagated an update to 67% of the replicas with the King data set and 98% of the replicas with the Meridian data set. Similarly, the simple gossip-based update propagation with a TTL = 3 reached 90% of the replicas with the PlanetLab data set. This is because the speculative update uses the gossip protocol to propagate an update to the nearby replicas to assist a speculative message. For the speculative update to advertise the update to all replicas, complementary messages' TTL must be carefully set according to the network size. We carefully selected the TTL in the simulation.

6.3 Ratio of Queries which Reach an Updated Replica

Figure 8 shows the ratio of queries which reach an updated replica. The x-axis shows the elapsed time from the update initiation, and the y-axis shows the ratio of client queries which reach an updated replica in a high-demand area. The error bars show the maximum and minimum values throughout the simulations.

The speculative update enables a rapid update propagation to popular replicas as the simple gossip-based update propagation. In the simulation with the King data set, the speculative update enabled the queries from clients in a high-demand



(a) King (b) Meridian (c) PlanetLab

Fig. 8 The ratio of queries which reach an updated replica. The speculative update propagated an update to popular replicas as fast as the simple gossip-based update propagation and faster than the convergence-guaranteed random walk.

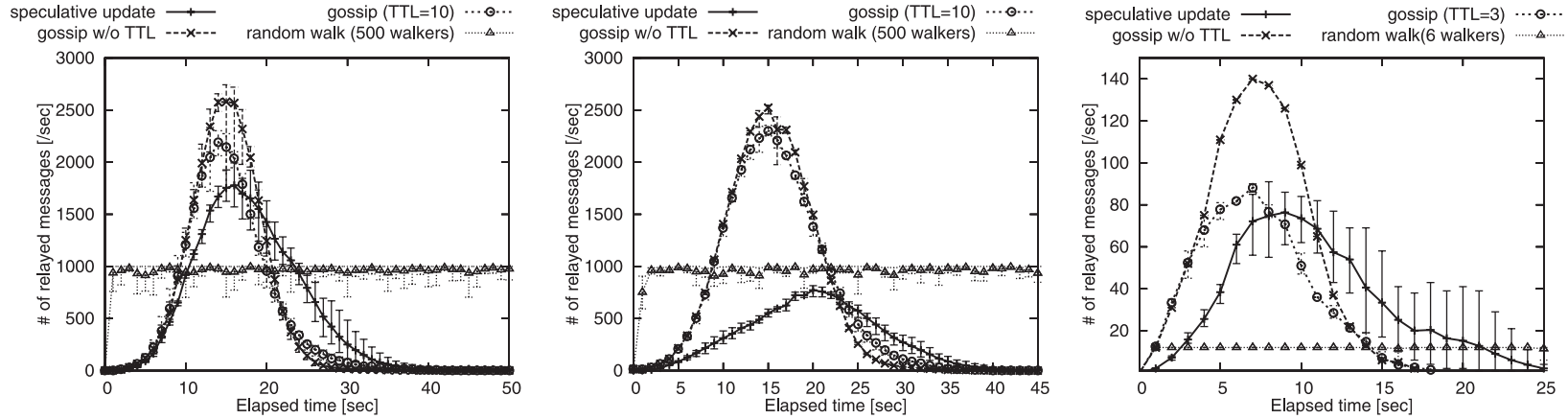
area to reach an updated replica as fast as the simple gossip-based update propagation and 86% faster than the convergence-guaranteed random walk. Similarly, in the simulation with the Meridian data set, the speculative update and the simple gossip-based update propagation made the time for the queries to reach an updated replica shorter than the convergence-guaranteed random walk by 88%.

In the case of the simulation with the PlanetLab data set, the speculative update made the time for the queries to reach an updated replica 84% shorter than the convergence-guaranteed random walk with 6 walkers. Compared with the simple gossip-based update propagation without any TTL, the speculative update took 42% longer time than the simple gossip-based update propagation. Since there are fewer servers in the PlanetLab data set, the simple gossip-based update propagation without TTL reached all servers within a few hops. Although the speculative update took more 7 seconds to complete update propagation, the bandwidth consumption is reduced by 45% compared to the simple gossip-based update propagation as shown in Section 6.4.

6.4 Number of Relayed Messages

To evaluate the overhead of the speculative update, we measured the number of relayed messages per second on the network. **Figure 9** (a) shows the result with the King data set, Fig. 9 (b) shows the result with the Meridian data set and Fig. 9 (c) shows the result with the PlanetLab data set, respectively. With the speculative update, servers relayed fewer messages during an update propagation than the simple gossip-based update propagation. In the case of the simulation with the simple gossip-based update propagation, the relayed messages rapidly increased because each server propagates update messages to all its neighbors. On the other hand, in the simulation with the speculative update, the largest number of relayed messages per second was smaller than that of the simple gossip-based update propagation without any TTL by 31% with the King data set, by 69% with the Meridian data set and by 45% with the PlanetLab data set, respectively.

We also compared the cumulative number of message creation. What we mean by message creation is the number of times that a server creates a new update message heading towards a new target. The speculative update reduced the cumulative number of message creation compared with the simple gossip-based update propagation without any TTL by 5% with King data set, by 55% with the Meridian data set, and 26% with the PlanetLab data set, respectively. In partic-



(a) King (b) Meridian (c) PlanetLab

Fig. 9 # of relayed messages per second. The speculative update induced fewer messages than the simple gossip-based update propagation.

ular, with the simple gossip-based update propagation, the number of message creation rapidly increased. The reason for these results is that a speculative message enables complementary messages to reach more replicas with a small TTL. Once a speculative message reaches a replica, the speculative update causes a server with a replica to start propagating an update with complementary messages. The complementary message generates an increase of relayed messages, but it helps the P2P-based CDNs to propagate an update quickly.

In the simulation with the convergence-guaranteed random walk, the number of relayed messages converged to a constant value because only the origin server created update messages. With the King and the PlanetLab data sets, the number of relayed messages with the convergence-guaranteed random walk was smaller than that with the speculative update. These results represent a property of the convergence-guaranteed random walk in that it limits the bandwidth consumption. However, as we mentioned in Section 6.2, the convergence-guaranteed random walk takes longer to complete update propagation due to the dependence on the network topology. With the Meridian data set, the speculative update induces 22% fewer messages than the convergence-guaranteed random walk, because the speculative update uses complementary messages with a limited TTL.

7. Conclusion

In this paper, we have proposed *the speculative update*, which is a novel update propagation mechanism for structured P2P-based CDNs. By leveraging the characteristics of P2P-based CDNs, the speculative update determines the direction in which many replicas are positioned and rapidly propagates an update in that direction. The speculative update improves the coverage of an update propagation by sending complementary messages based on the gossip protocol. Simulation results demonstrate that the speculative update propagates an update to all replicas up to 92% faster than the convergence-guaranteed random walk and require up to 69% less network bandwidth than the simple gossip-based update propagation. For future work, we plan to evaluate the speculative update on the real Internet. We also plan to evaluate it on other P2P-based CDNs.

Acknowledgments This work was partially supported by MEXT Grant-in-Aid for Scientific Research on Priority Areas (New IT Infrastructure for the Information-explosion Era).

References

1) Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R. and Weihl, B.: Globally distributed content delivery, *IEEE Internet Computing*, Vol.6, No.5, pp.50–58 (2002).
2) BitTorrent, Inc.: BitTorrent. http://bittorrent.com
3) Deshpande, M., Amit, A., Chang, M., Venkatasubramanian, N. and Mehrotra, S.: Flashback: A peer-to-peer Web server for flash crowds, *Proc. IEEE ICDCS*, 8 pages (2007).
4) Freedman, M.J., Freudenthal, E. and Mazières, D.: Democratizing content publication with Coral, *Proc. USENIX Symp. on NSDI*, pp.239–252 (2004).
5) Gkantsidis, C. and Rodriguez, P.R.: Network coding for large scale content distribution, *Proc. IEEE INFOCOM*, Vol.4, pp.2235–2245 (2005).
6) Chun, B., Wu, P., Weatherspoon, H. and Kubiatowicz, J.: ChunkCast: An anycast service for large content distribution, *Proc. IPTPS*, 6 pages (2006).
7) Peterson, R.S. and Sirer, E.G.: Antfarm: Efficient content distribution with managed swarms, *Proc. USENIX Symp. on NSDI*, pp.107–122 (2009).
8) Sirivianos, M., Park, J.H., Yang, X. and Jarecki, S.: Dandelion: Cooperative content distribution with robust incentives, *Proc. USENIX Annual Tech. Conf.*, pp.157–170 (2007).
9) Pierre, G. and van Steen, M.: Globule: A collaborative content delivery network, *IEEE Communications Magazine*, Vol.44, No.8, pp.127–133 (2006).
10) Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C. and Zhao, B.: OceanStore: An architecture for global-scale persistent storage, *Proc. ACM ASPLOS*, pp.190–201 (2000).
11) Li, Z., Xie, G. and Li, Z.: Locality-aware consistency maintenance for heterogeneous P2P systems, *Proc. IEEE IPDPS*, 10 pages (2007).
12) Ramaswamy, L., Liu, L. and Iyengar, A.: Cache clouds: Cooperative caching of dynamic documents in edge networks, *Proc. IEEE ICDCS*, pp.229–238 (2005).
13) Deshpande, M., Xing, B., Lazardis, I., Hore, B., Venkatasubramanian, N. and Mehrotra, S.: CREW: A gossip-based flash-dissemination system, *Proc. IEEE ICDCS*, 8 pages (2006).
14) Eugster, P.T., Guerraoui, R., Handurukande, S.B., Kouznetsov, P. and Kermarrec, A.-M.: Lightweight probabilistic broadcast, *ACM Trans. Comput. Syst.*, Vol.21, No.4, pp.341–374 (2003).
15) Li, H.C., Clement, A., Wong, E.L., Napper, J., Roy, I., Alvisi, L. and Dahlin, M.: BAR gossip, *Proc. USENIX OSDI*, pp.191–204 (2006).
16) Van Renesse, R., Birman, K.P. and Vogels, W.: Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining, *ACM Trans. Comput. Syst.*, Vol.21, No.2, pp.164–206 (2003).
17) Bharambe, A.R., Agrawal, M. and Seshan, S.: Mercury: Supporting scalable multi-attribute range queries, *Proc. ACM SIGCOMM*, pp.353–366 (2004).
18) Cooper, B.F.: Quickly routing searches without having to move content, *Proc. IPTPS*, pp.163–172 (2005).
19) Gkantsidis, C., Mihail, M. and Saberi, A.: Random walks in peer-to-peer networks, *Proc. IEEE INFOCOM*, Vol.1, pp.120–130 (2004).
20) Loguinov, D., Kumar, A., Rai, V. and Ganesh, S.: Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience, *Proc. ACM SIGCOMM*, pp.395–406 (2003).
21) Lv, Q., Cao, P., Cohen, E., Li, K. and Shenker, S.: Search and replication in unstructured peer-to-peer networks, *Proc. ACM Int'l Conf. on Supercomputing*, pp.84–95 (2002).
22) Tsoumakos, D. and Roussopoulos, N.: Adaptive probabilistic search for peer-to-peer networks, *Proc. IEEE Int'l Conf. on P2P*, pp.102–109 (2003).
23) Asahara, M., Shimada, A., Yamada, H. and Kono, K.: Finding candidate spots for replica servers based on demand fluctuation, *Proc. IEEE ICPADS*, 10 pages (2007).
24) Zhong, M., Shen, K. and Seiferas, J.: The convergence-guaranteed random walk and its applications in peer-to-peer networks, *IEEE Trans. Comput.*, Vol.57, No.5, pp.619–633 (2008).
25) Jung, J., Krishnamurthy, B. and Rabinovich, M.: Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites, *Proc. ACM WWW*, pp.293–304 (2002).
26) Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D. and Kubiatowicz, J.D.: Tapestry: A resilient global-scale overlay for service deployment, *IEEE Journal on Selected Areas in Communications*, Vol.22, No.1, pp.41–53 (2004).
27) Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, F.M., Dabek, F. and Balakrishnan, H.: Chord: A scalable peer-to-peer lookup protocol for Internet applications, *IEEE/ACM Trans. Networking*, Vol.11, No.1, pp.17–32 (2003).
28) Datta, A., Hauswirth, M. and Aberer, K.: Updates in highly unreliable, replicated peer-to-peer systems, *Proc. IEEE ICDCS*, pp.76–85 (2003).
29) Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S.: A scalable content-addressable network, *Proc. ACM SIGCOMM*, pp.161–172 (2001).
30) Eugene Ng, T. S. and Zhang, H.: Predicting Internet network distance with coordinates-based approaches, *Proc. IEEE INFOCOM*, Vol.1, pp.170–179 (2002).
31) Shudo, K., Tanaka, Y. and Sekiguchi, S.: Overlay Weaver: An overlay construction toolkit, *Comput. Communications*, Vol.31, No.2, pp.402–412 (2008).
32) Gil, T.M., Kaashoek, M.F., Li, J., Morris, R. and Stribling, J.: p2psim. http://pdos.csail.mit.edu/p2psim
33) Wong, B., Slivkins, A. and Sirer, E.G.: Meridian: A lightweight network location service without virtual coordinates, *Proc. ACM SIGCOMM*, pp.85–96 (2005).
34) Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson,

L., Roscoe, T., Spalink, T. and Wawrzoniak, M.: Operating systems support for planetary-scale network services, *Proc. USENIX Symp. on NSDI*, pp.253–266 (2004).

Appendix

The pseudo-code of the speculative update is shown below.

A.1 Speculative Message Handler

```
1  // At server i
2  if i receives a speculative message from k then
3    Relay the speculative message to the message's destination
4    for all neighbor j such that j != k do
5      Send a complementary message to j
6    end for
7    if server i has a replica then
8      // Create a new speculative message
9      for all neighbor j do
10       Send request(C, V) to j to ask j's local state s_j
11       // C denotes the name of updated content
12       // V denotes the version number of the content
13       // In ExaPeer, s_j is the APC degree of j
14     end for
15     Calculate direction D_i
16     // In ExaPeer, calculate a primitive vector and a target vector
17     Send the new speculative message toward D_i
18     // In ExaPeer, calculate an ID which denotes the direction of the target vector on
19     // CAN. The new speculative message whose destination is the calculated ID
20     // is relayed in the direction of the target vector.
21   end if
22 end if
```

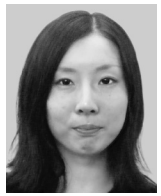
A.2 Complementary Message Handler

```
1  // At server i
2  if i receives a complementary message from k then
3    for all neighbor j such that j != k do
4      Relay the complementary message to j
5    end for
6    if i has a replica then
7      // Create a new speculative message
8      for all neighbor j do
9        Send request(C, V) to j to ask j's local state s_j
10       // In ExaPeer, s_j is the APC degree of j
11     end for
12     Calculate direction D_i
13     // In ExaPeer, calculate a primitive vector and a target vector
14     Send the speculative message toward D_i
15     // In ExaPeer, calculate an ID which denotes the direction of the target vector on
16     // CAN. The speculative message whose destination is the calculated ID is
17     // relayed in the direction of the target vector.
18   end if
19 end if
```

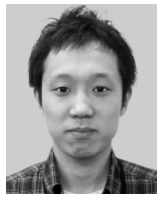
A.3 Reply about Local State

```
1  // At server j
2  if j gets request(C, V) which asks its local state then
3    // Compare V with j's version V'
4    if V ≤ V' then
5      // j is already updated
6      return 0
7    else
8      return s_j
9    end if
10 end if
```

(Received January 26, 2010)
(Accepted May 14, 2010)



Ai Hayakawa received her B.E. and M.E. degrees from Keio University in 2008 and 2010, respectively. Since 2010 she works for Nomura Research Institute, Ltd. Her research interests include Peer-to-Peer systems and content distribution networks.



Masato Asahara received his B.E. degree from University of Electro-communications in 2005 and M.E. degree from Keio University in 2007, respectively. He was a Ph.D. student in Keio University from 2007 to 2010. Since 2010 he works for Service Platforms Research Labs., NEC Corp. His research interests include Peer-to-Peer systems, distributed systems, middleware and operating systems. He is a member of IPSJ, IEEE/CS, ACM and USENIX.

Kenji Kono received his B.Sc. degree in 1993, M.Sc. degree in 1995, and Ph.D. degree in 2000, all in computer science from the University of Tokyo. He is an associate professor of the Department of Information and Computer Science at Keio University. His research interests include operating systems, system software, and Internet security. He is a member of IEEE/CS, ACM and USENIX.

Toshinori Kojima received his B.E. and M.E. degrees from Keio University in 2008 and 2010, respectively. Since 2010 he works for Research and Development Headquarters, NTT DATA CORPORATION. His research interests include network coordinates and distributed hash tables.