A Replica Management Protocol in a Binary Balanced Tree Structure-Based P2P Network

Hidehisa Takamizawa, Kazuhiro Saji
Department of Computer Science, Graduate School of Engineering, Gunma University, Kiryu 376-8515, Japan
Email: takami@dml.cs.gunma-u.ac.jp

Masayoshi Aritsugi
Computer Science and Electrical Engineering, Graduate School of Science and Technology, Kumamoto University, Kumamoto 860-8555, Japan
Email: aritsugi@cs.kumamoto-u.ac.jp

Abstract— **The purpose of our work is to realize a load balancing of nodes in a P2P network. A replica management protocol, which exploits replicas for balancing loads of each node managing popular data, by adapting replica partition trees to a balanced tree overlay network called BATON, for BAlanced Tree Overlay Network, is proposed for this. BATON has a load balancing mechanism where each node adjusts the number of data managed by it. However, if there are some popular data that are accessed by a large number of nodes in a network, the mechanism of BATON could fail. We propose a replica management protocol for balancing loads of both data transmission and replica management of each node. Some results of simulation in which our proposal was compared with a method without replica and another method of simple replica management are showed and the effective and weak points of our proposal are discussed.**

Index Terms— **P2P network, binary balanced tree structure, replica management, load balancing**

I. Introduction

P2P (Peer-to-Peer) networks have been applied to wide areas recently. It is required to realize load balancing of nodes in a P2P network. Since a peer can participate in or leave from a P2P network autonomously, the load of each peer must be balanced. In other words, it is supposed that there is no peer that has excessively larger load than others in a P2P network. In this paper, a replica management protocol in a binary balanced tree structure-based P2P is proposed for realizing load balancing of peers, or nodes, managing popular data that are accessed by a large number of nodes in a network.

BATON (BAlanced Tree Overlay Network) [2] is the first attempt to exploit a balanced binary tree as the P2P overlay network. Since BATON is based on the balanced binary tree structure, it can support efficient range queries easier than related conventional DHT (Distributed Hash Table)-based systems [3]–[7]. Moreover, each node can autonomously adjust the number of data it manages for balancing the number of accesses. However, if there are some data that are requested very frequently in the network, the nodes that manage such data must handle high loads. Note that such loads cannot be balanced by the function of BATON. One way to solve the problem is to create replica of such popular data and to balance the loads with them. If, however, the number of replica grows, it becomes naturally hard to manage them.

It is important to reduce the loads of nodes with popular data. Traffics in P2P networks are divided into two categories: one is download data, whose size is almost over 20 Kbytes, and the other is non-download data, e.g., messages for queries and network maintenance [8]. According to the results of [9], which measured traffics in eDonkey, the average sizes of non-download and download data are 16.7Kbytes and 2.48Mbytes, respectively, and there is a strong distinction between them. If there is a node that has a popular data or has a data that would be modified frequently, its load must become higher than those of the other nodes. We thus think that it is necessary to balance the loads of such nodes in a P2P network.

There have been studies on load balancing of nodes managing popular data by means of replicas [10]–[12]. In [10], the performance improvement of lookup with replicas was discussed and $O(1)$ lookup in an environment where queries are distributed in Zipf was proposed. In [11], creation ways of replicas according to data access rates and reallocation ways of them were discussed. However, they did not consider such environments where data are modified frequently, and thus consistency of data was not supported strictly.

In [12], a system called SCOPE (Scalable COnsistency maintenance in structured PEer-to-peer systems) was proposed for load balancing with replicas in DHT-based P2P networks. SCOPE uses RPTs (Replica Partition Trees) for strictly managing replica locations and thus support consistency of data in the network. Since SCOPE assumes P2P networks with DHTs, it must be difficult to support efficient range queries [2]. In addition, the numbers of messages were only considered as loads in [12]. As noted

This paper is based on "A P2P Protocol with Load Balancing using Replicas: A Proposal (in Japanese)," by Kazuhiro Saji and Masayoshi Aritsugi, which appeared in DBSJ Letters, vol. 5, no. 2, pp. 9–12, September 2006 [1].
Manuscript received September 17, 2008; revised January 26, 2009; accepted February 24, 2009.
Correspondence to Masayoshi Aritsugi, Graduate School of Science and Technology, Kumamoto University, 2-39-1 Kurokami, Kumamoto 860-8555, Japan.

in [8], [9], the costs for data sending should be also considered as loads. Here we thus discuss how to use and manage replicas with RPTs in BATON for load balancing based on the total data transmission costs.

In this paper, a protocol that adapts RPTs [12] to BATON [2] is proposed in order to realize load balancing by using replicas in a P2P network. Our proposal can reduce the loads of nodes managing popular data that are accessed frequently by a large number of nodes. Since our proposal is based on BATON, or a binary balanced tree structure, our proposal can easily support efficient range queries. By exploiting RPTs, replicas are managed in a distributed manner, and thus the management loads of replicas are also balanced in our proposal. The paper reports some simulation results and discusses availabilities of our proposal.

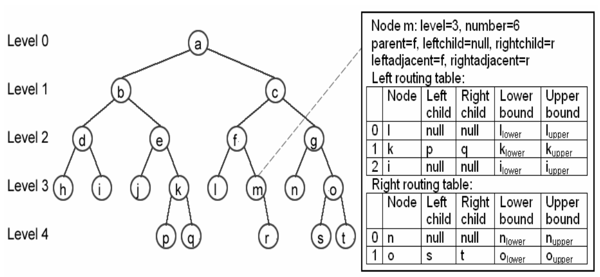
The remainder of this paper is organized as follows. Section II mentions related work and compares with our work. Section III describes briefly BATON [2] and RPTs [12]. Section IV proposes a replica management protocol that adapts RPT to BATON. Section V reports some simulation results for evaluating our proposal, and Section VI concludes this paper.

II. Related Work

In this paper, a replica management protocol is proposed for performance in a P2P network by means of managing replica locations in a distributed manner. Gnutella and Winny use replicas for retrieval performance in unstructured P2P networks, but they do not manage replica locations and thus consistency among original data and replicas is not supported. In [13], although they did not manage replica locations, they proposed a replica update propagation algorithm based on rumor spreading to reduce traffics for replica updates. However, the replica updates are processed in a probabilistic manner, and thus they did not support strictly consistency among data, either.

Ref. [14] proposed an update propagation strategy in which an n-ary tree is used for replica location management. They achieved both load balancing and delay reduction in P2P networks. However, skews of update loads of nodes would happen in their strategy if an internal node of the tree has many replicas of data.

There have been many studies on replicas for performance in structured P2P networks [10]–[12]. Ref. [10] supported $O(1)$ lookup of queries in Zipf distributions with replicas in structured P2P networks, and also supported replica updates when updating their original data. Ref. [11] proposed data replication and reallocation algorithms adapting to data access rates. However, they did not consider such environments where data are modified frequently, and thus consistency of data was not supported strictly. Ref. [12] proposed a system for load balancing with replicas in DHT-based P2P networks. The system considered strict replica management by using replica-partition-trees (RPTs). However, the system assumes to



Node m: level=3, number=6
parent=f, leftchild=null, rightchild=r
leftadjacent=f, rightadjacent=r
Left routing table:

	Node	Left child	Right child	Lower bound	Upper bound
0	l	null	null	l_{lower}	l_{upper}
1	k	p	q	k_{lower}	k_{upper}
2	i	null	null	i_{lower}	i_{upper}

Right routing table:

	Node	Left child	Right child	Lower bound	Upper bound
0	n	null	null	n_{lower}	n_{upper}
1	o	s	t	o_{lower}	o_{upper}

Figure 1. BATON tree structure and routing table [2].

be built on DHT-based P2P networks, and thus it must be difficult to support efficient range queries [2].

In this paper, a replica management protocol based on RPTs [12] and BATON [2] is proposed. Our proposal realizes load balancing by using replicas in a P2P network where there are data accessed so frequently that it is impossible to support load balancing without replicas. Since replicas are not exploited in BATON [2], the load balancing function of our proposal cannot be realized by BATON itself. We adapt RPTs in SCOPE [12], which is supposed to be exploited to DHT-based P2P networks, to BATON, which is a binary balanced tree structure-based P2P network, in this paper.

III. Preliminaries

In this section, we describe only necessary parts of BATON [2] and RPTs [12] for the discussions of this paper. Please refer [2] and [12] for the full descriptions of them.

A. BATON

1) Structure: Figure 1 shows an example of BATON trees and routing table of node m.

A node in BATON tree corresponds to a node in a P2P network. As shown in the figure, each node has its $level$ and $number$ for indicating its locations in the tree. It holds links to its parent node, child nodes, and left and right adjacent nodes, and also has its right and left routing tables. Each routing table of a level L node can have at most L node ids. Because the height of a balanced binary tree in which the number of nodes is N is no greater than $1.44 \log N$ [15], the number of entries in a routing table of a node is $O(\log N)$. The j-th entry of right (left) routing table of node whose $number$ is C has the link to node whose $number$ is $C + 2^{j-1}$ (respectively $C - 2^{j-1}$), if the node exists.

Each node is assigned the range of data ids, as shown in Figure 2, and data covered by the range is stored and managed at the node. The range is from the maximum id of its left adjacent node to the minimum id of its right adjacent node. When load balancing is necessary, the data migration between two adjacent nodes occurs and as a result the ranges change. Note that the original BATON does not consider replica at all. We call a node storing the original data a primary node in this paper.

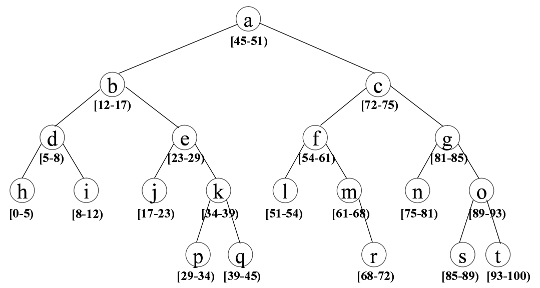


Figure 2. Example of range value [2]

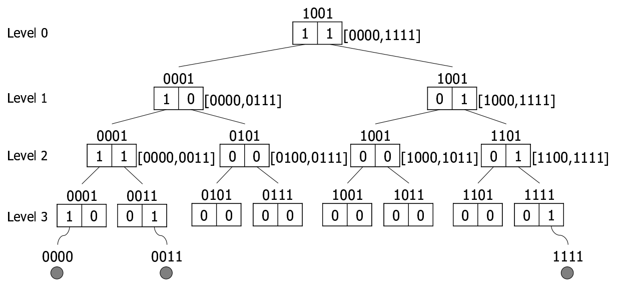


Figure 3. RPT structure [12]

Algorithm: *Search exact*(*node* n, *query* q, *value* v)

```
If ((LowerBound($n$) $\leq v$) and ($v \leq$UpperBound($n$)))
  $q$ is executed at $x$;
Else
  If (UpperBound($n$) $< v$)
    $m$=the farthest node satisfying
      condition(LowerBound($m$) $\leq v$);
    If (there exists such an $m$)
      Forward $q$ to $m$;
    Else
      If (RightChild($n$) $\neq$ null)
        Forward $q$ to RightChild($n$);
      Else
        Forward $q$ to RightAdjacentNode($n$)
      End If
    End If
  Else
    //A similar process is followed towards the left
  End If
End If
```

2) Operations to BATON:

a) Exact Match Query: When node n issues or receives exact match query for data v, the node routes the query according to the algorithm *Search exact*, where $UpperBound(n)$ and $LowerBound(n)$ stands for the maximum and minimum ids that node n stores, respectively. The routing process continues to reach the primary node of data v, and then the node sends the data to the node that initiated the query, if the data exists. If there are N nodes in the network, then the necessary steps of search exact are $O(\log N)$.

b) Range Query: When a node issues a range query, then one of data in the range is searched first. After getting the data, then the rest of the range will be obtained by navigating left and right adjacent nodes. When the whole data of the range are stored in X nodes, it costs $O(\log N + X)$ steps to process the range query.

c) Node Failure: When a node finds a fail node x, the parent node y of x should make the failure correct, i.e., node y performs the processes that are necessary when node x leaves in order to balance the whole tree and to make the tree consistency.

B. RPTs

1) Structure: In [12], a replica partition tree (RPT) was proposed for managing replicas in a distributed manner. The structure can balance loads of replica management including location management and replica update propagation among nodes in a network. Figure 3 shows an example of replica partition trees.

One RPT is created for one data for managing replicas of the data. Original data whose id is x is managed by node x, i.e., node x is the primary node of data x, and the node becomes the root node of the RPT of data x. Figure 3 shows an RPT with 4-bit identifier space in which replicas of data 1001 are allocated in node 0000, 0011, and 1111. The nodes 0000, 0011, and 1111 are called subscribers. Each node in an RPT has a 2-bit value called a partition vector to one data for expressing whether there is a replica in its child nodes, that is, if there is a replica in left/right child nodes then left/right bit of its partition vector is set to 1, as shown in Figure 3.

Data identifier space in an RPT is divided into partitions, and a node in the RPT is assigned to one partition. The identifiers of each level i in Figure 3 are calculated as follows:

Level i**:** The nodes of this level inherit the $4 - i$ least significant bits of 1001.

For example, nodes 0001 and 1001 are of Level 1, and nodes 0001, 0011, 0101, 0111, 1001, 1011, 1101, and 1111 are of Level 3 in the figure.

As shown in Figure 3, each node has its range to manage whether there is a replica in the range. For example, nodes 1001 of Level 0 and 0001 of Level 2 manage replica existence information in the whole identifier space and in $[0000, 0011]$, respectively.

2) Operations to RPTs:

a) Subscribe/Unsubscribe: Operation subscribe is issued when a subscriber informs the replica location to the primary node. For example, when node 0010 has a replica of data 1001 and becomes a subscriber in Figure 3, the node issues operation subscribe to node 0011. The node 0011 then sets the left of its partition vector 1. This operation continues until a node that has a value 1 in its partition vector or it reaches the root node, or the primary node.

When the replica is deleted, operation unsubscribe is issued. For example, when node 0010 deletes the replica of data 1001, the node issues operation unsubscribe to

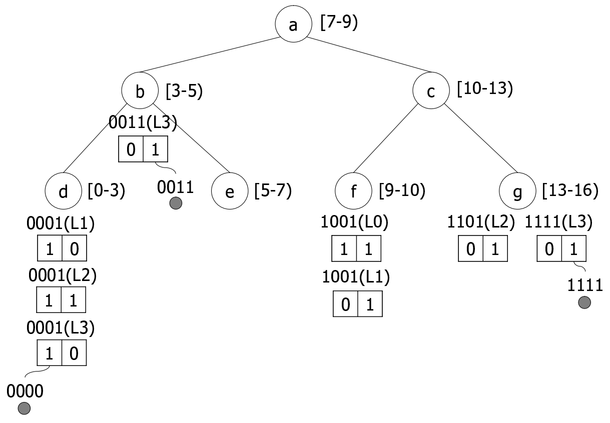


Figure 4. Our proposal's structure

node 0011. The node 0011 then sets the left of its partition vector 0. This operation continues until a node that has two 1 values in its partition vector or it reaches the root node, or the primary node.

b) Search: The primary node checks whether there is a replica or not by touching its partition vector. If there is a replica, then it forwards the operation search to its child node. The operation finally reaches the node that has a replica, and the subscriber then replies to the node that initiated the operation.

c) Update: Operation update is performed when a primary node updates the data or the node receives update request to the data. The operation is forwarded to all the subscribers for update propagation. Note that the primary node can manage all replica locations only by maintaining the partition vector, and the replica management loads can thus be balanced by RPTs.

IV. Proposal

A. Structure

A replica management protocol is proposed by adapting RPTs to BATON in order to realize load balancing of both data transmission and replica management of each node in a P2P network. Figure 4 shows an example of our proposal where the RPT for data 1001 shown in Figure 3 is adapted to a BATON with seven nodes.

In our proposal, each partition vector is assigned to the corresponding node in BATON according to the identifier of the node that manages the vector, and has its level, which is expressed at the right to the identifier of each partition vector in Figure 4. As shown in the figure, there can exist nodes which have more than one partition vector like node d. Note that when traversing partition vectors located in such a node there is no need to communication between nodes. Note also that in our proposal we do not create or delete such partition vectors that have only 0 values for reducing storage cost for partition vectors.

B. Operations to Our Proposal

1) Subscribe/Unsubscribe: For simplicity, we adopt owner replication [16] where we create a replica in a node when it retrieved and received the data.

When creating a replica in a node, the node, or the subscriber, randomly selects a data identifier from the id range that the subscriber manages, and checks whether $T = \frac{o+r}{m} + \beta$ is under a threshold value, where m is the number of ids that the subscriber manages, o and r are the total size and the numbers of data mapped to the selected id, and β is a constant. If T is over, the subscriber selects another data identifier. By keeping the value low, we can map replicas to ids uniformly distributed in the id range of the subscriber. Then, we process operation subscribe as described in III-B.2.a for registering the location in the RPT.

Because our proposal is based on BATON, each node has its data identifier range and the range would change dynamically if load balancing of the original BATON occurs. In our proposal, if migration of data mapped with a replica occurs due to the load balancing, the replica and its partition vector migrate as well. For example, when a replica is stored in node e in Figure 4, the replica is mapped to data id 5 ($= 0101_{(2)}$), 6 ($= 0110_{(2)}$), or 7 ($= 0111_{(2)}$). If load balancing occurs and the ranges of nodes e and b become $[6, 7)$ and $[3, 6)$, respectively, then the management of the data of id 6 also changes from node e to node b. In our proposal, the management of replica(s) mapped to id 6 changes in the same way.

2) Exact Match Query: Exact match query is first processed in almost the same way as described in III-A.2.a except for when a replica of the requested data is found before the process reaches the primary node. In such a case, the replica is returned to the requesting node in our proposal.

When an exact match query reaches to the primary node, our system checks whether the node is overloaded or not. If the node is not overloaded or there is no replica of the requested data in the network, then the primary node returns the original data to the requesting node, as described in III-A.2.a. On the other hand, if the node is overloaded and there is a replica, the primary node sends a request of returning the replica to a subscriber. The request sending between nodes in an RPT is processed in Algorithm *Search exact* through $O(\log N)$ nodes. Thus, the average number of hops for finding a subscriber is $O(\log^2 N)$. For load balancing of nodes managing frequently accessed data, an overloaded primary node sends a certain part of received requests to subscribers in our proposal.

If a partition vector having more than one 1 value is found in traversing an RPT, then we should select one out of them. Our proposal attempts to reduce the number of nodes traversed in the base structure, BATON. For example, when traversing the RPT shown in Figure 3 to find a replica of 1001, the partition vector of node 1001 has two 1 values. In this case, node 1001 of Level 1 is selected because they are identical and thus no real hop between nodes occurs.

However, this way may use one replica many times and, as a result, fail to load balancing. To avoid such situations, our proposal deletes a replica used a certain times.

3) Range Query: As exact match query, range query in our proposal is processed in almost the same way as described in III-A.2.b. Balancing of loads caused by popular data is realized by the same way as in exact match query, i.e., if primary nodes managing data in the range have high loads then replicas are used as in IV-B.2.

4) Update: As described in III-B.2.c, this operation is performed when a primary node updates the data or the node receives update request to the data. Update in our proposal can be processed in the same way as in III-B.2.c.

A large number of replicas of popular data would be created in owner replication, and if popular data would be updated frequently, the cost for update propagation cannot be ignored. Note that it is not necessary to update all the replicas in our proposal, since replicas are used for performance. In other words, we may have to delete a part of replicas when keeping all of the replicas up-to-date degrades the system's performance.

However, no peer manages with RPTs the numbers of replicas in the network. Thus, in our proposal, when processing update propagation, operation update can be changed to operation delete in a certain amount of probability.

5) Node Failure: When node failure occurs, the operations to original BATON and RPTs are performed. If there is a subscriber of data managed by a failure node, the data can be restored in our proposal. On the other hand, as shown in Figure 4, several nodes in an RPT can be managed by a node in BATON structure. If such a node in BATON is in failure, the managed nodes in RPTs are in failure at the same time, and thus several operations in node failure to an RPT are performed simultaneously in our proposal.

6) Join and Departure: When node join or departure is occurred in BATON, data migration is also occurred. In our proposal, replicas and partition vectors also move along the data migration.

7) Data Insertion and Deletion: Operation for data insertion is processed in the same way as in BATON. On the other hand, if data deletion is issued, the data itself, its replicas, and the corresponding RPT are deleted in our proposal.

V. Simulation

A. Simulation Configuration

We compared our proposal with the original BATON in which no replica was created and a simple replica management method, in situations where there were some popular data accessed frequently. The simple replica management method was prepared for only the comparison; in this method the primary node of a data manages locations of all subscribers of the data as shown in Figure 5. When the primary node is overloaded, it sends a request to a randomly selected subscriber. A replica is created in the same way of our proposal, i.e., in owner replication [16]. When updating data, the primary node sends the updated data to subscribers directly. Since the primary node manages the locations of all subscribers in this method, it costs only one hop for navigating from the primary node to a subscriber, but the primary node can be overloaded for replica management in cases where many subscribers exist or many replica updates occur.

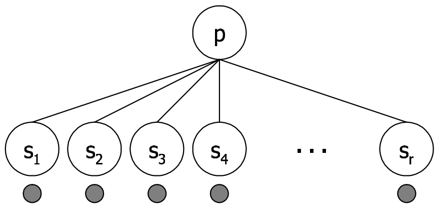


Figure 5. Example of simple replica management

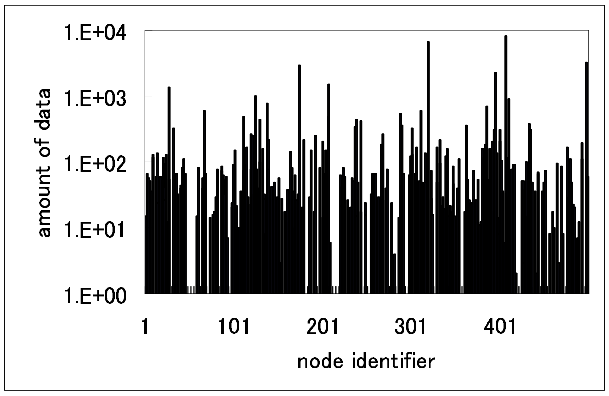
We checked the number of accesses to a data, and 75% of accesses after the fifth were forwarded to subscribers for load balancing in the simulation. To avoid skew of replica usage, we deleted replicas used ten times in this simulation. We set the size of a partition vector four, and the maximum number of replica that a node could have ten; when the capacity became full, then replica replacement occurred in LRU manner.

We run the simulation in P2P networks with 500 and 1000 nodes. The identifier spaces with 500 and 1000 nodes were $[1, 512]$ and $[1, 1023]$, respectively, and 1000 data were allocated in the spaces. Each data had an identifier randomly chosen from the identifier space. We set the size of a message 1. The sizes of original data and update data were distributed randomly in $[1, 20]$ and $[1, 50]$, respectively.

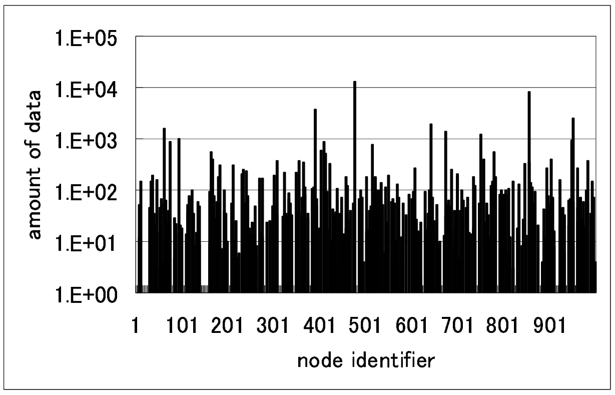
B. Search

For showing the effectiveness of our proposal in cases where there are some popular data accessed frequently by many nodes in a P2P network, we measured the amount of data transmission and hops in exact match queries. Assume exact match queries were issued by Zipfian method with parameter 1.0.

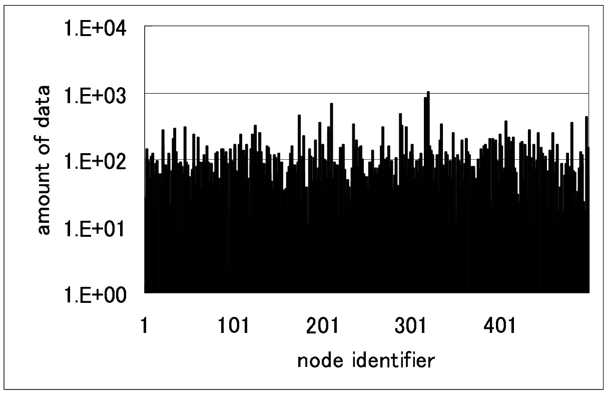
Figures 6 and 7 show the amount of data sending from each node for 5000 exact match queries in 500- and 1000-node P2P networks, respectively. In the simulation, we created another variation of our proposal in which one out of several 1 values in a partition vector was selected randomly when traversing an RPT. As shown in the figures, nodes managing popular data in no replica method, or the original BATON, were overloaded. For example, node 474 in 1000-node P2P network sent 12900 in total in the experiment. On the other hand, the other three methods, which used replicas, could balance the loads among nodes. The amount of data sent by node 474 of the simple replica management method in 1000-node P2P network was 1290. Those of our proposal with random traversal and our proposal were 1680 and 1956, respectively. Our proposal could avoid creating skew of the amount of replica sending among subscribers; the largest amounts of replicas sent by a subscriber in our proposal with random traversal and our proposal were 299 and 341, respectively.



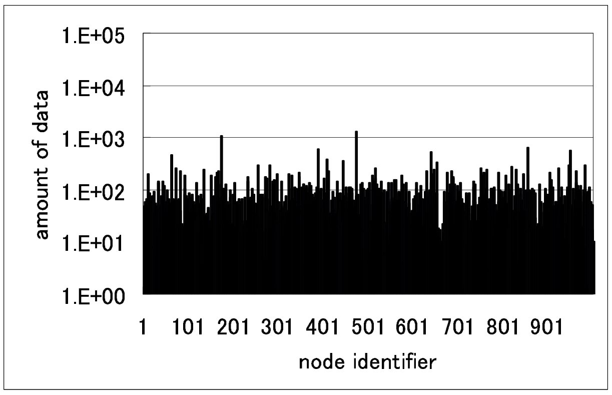
(a) No replica



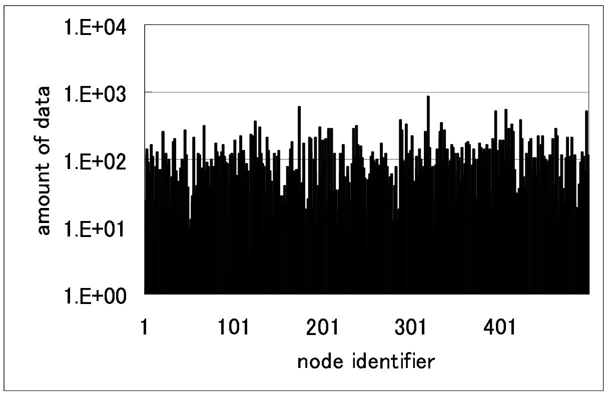
(a) No replica



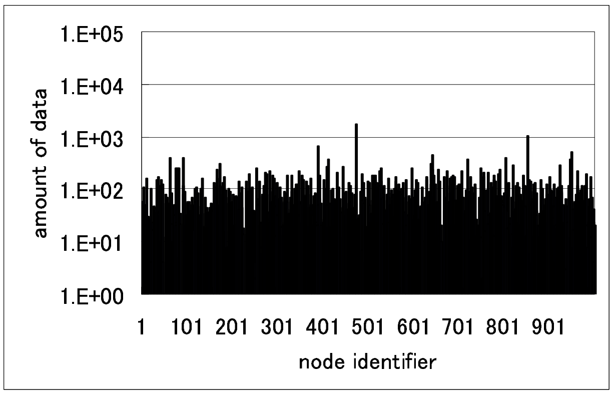
(b) Simple replica management



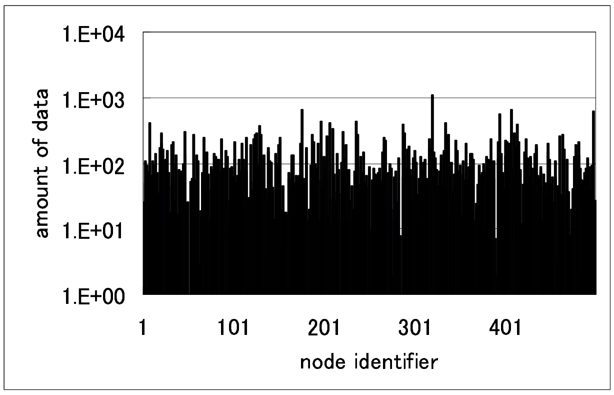
(b) Simple replica management



(c) Our proposal with random traversal through RPTs

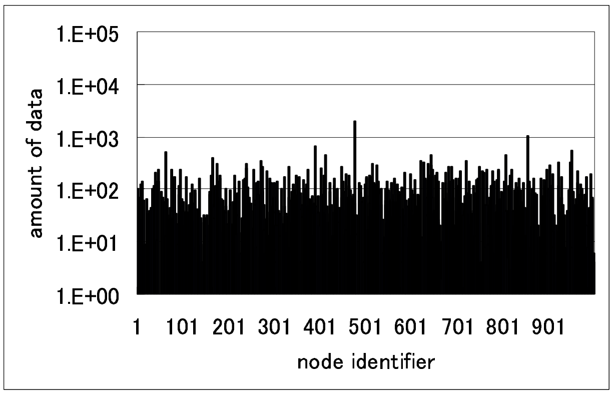


(c) Our proposal with random traversal through RPTs



(d) Our proposal

Figure 6. Amount of data sent by each node (500 nodes)



(d) Our proposal

Figure 7. Amount of data sent by each node (1000 nodes)

TABLE I.
COMPARISON OF THE NUMBERS OF HOPS

	# nodes	# hops	
		avg.	max.
No replica	500	4.50	14
	1000	5.05	13
Simple repl. management	500	3.40	14
	1000	4.24	13
Our proposal (with random traversal in RPTs)	500	4.52	28
	1000	5.99	33
Our proposal	500	4.37	28
	1000	5.81	31

TABLE II.
COMPARISON OF REPLICA MANAGEMENT COSTS

	# nodes	avg.	std. deviation	max.	# replicas $\times 10^3$
simple repl. management	500	7.86	29.2	370	3.93
	1000	4.56	26.3	489	4.56
Our proposal	500	18.1	5.85	35	3.91
	1000	9.40	4.00	27	4.36

Table I compares the numbers of hops in the four methods. The numbers of hops in the simple replica management method were slightly smaller than those in no replica method because the usage of replica in processing queries can reduce the costs. On the other hand, the numbers of hops in our proposal were larger than those in the simple replica management method. This is mainly because it costs only one hop in the simple replica management method to get a subscriber from the primary node but costs $O(height\ of\ RPTs)$ hops in our proposal. However, as we will see the next subsection, the simple replica management method could not balance the costs of replica management because of the simple structure.

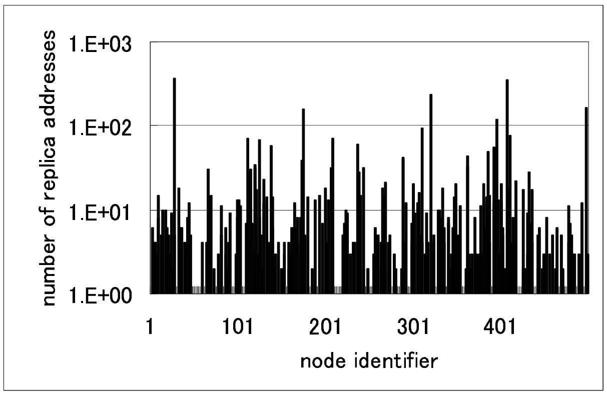
C. Replica Management Costs

Figures 8 and 9 show the replica management costs at each node in 500- and 1000-node P2P networks, respectively. The values are the numbers of subscribers' locations and partition vectors managed at each node in the simple replica management method and in our proposal, respectively, after finishing the search operations in Section V-B. We describe some statistical information obtained from the figures in Table II.

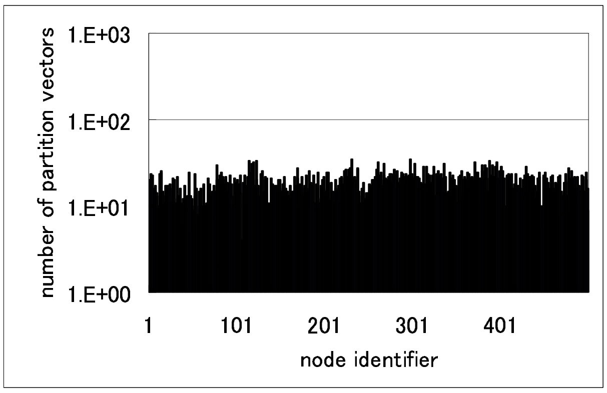
It should be noted that our proposal could realize balancing of the costs among nodes, although the total cost becomes larger than that of the simple replica management method. In contrast, as we wrote before, the simple replica management method could not balance the costs of replica management; some nodes in the network had excessively larger loads than others because of its simple structure.

D. Update

Here we compare the amount of data sent by each node in cases where update operations occur. In this simulation, an update operation occurred after fifty exact match

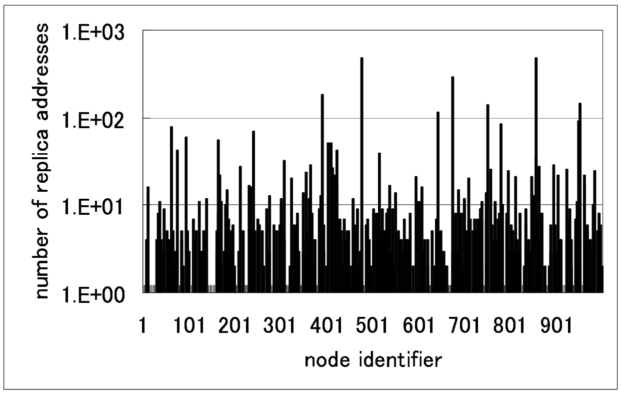


(a) Simple replica management

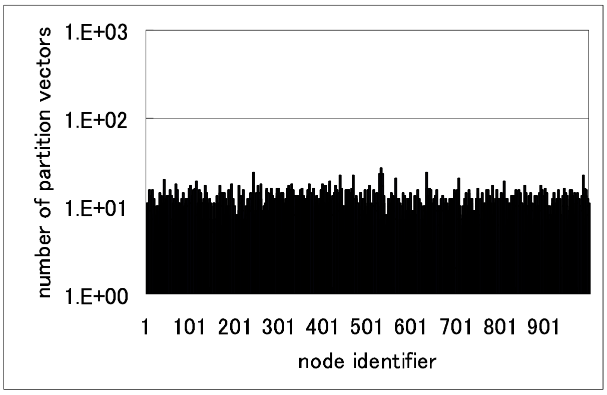


(b) Our proposal

Figure 8. Replica management costs (500 nodes)

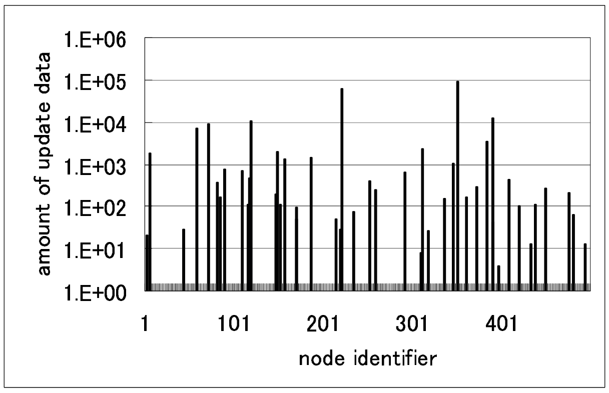


(a) Simple replica management

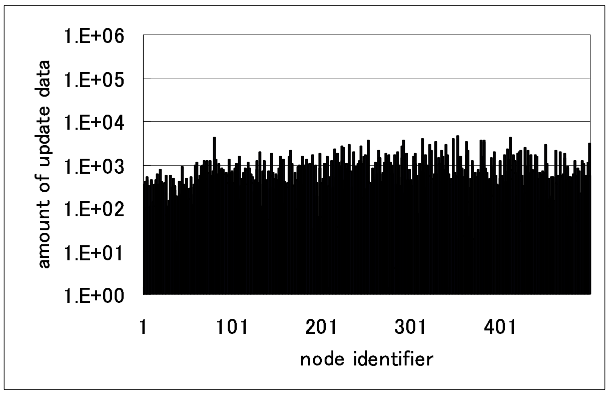


(b) Our proposal

Figure 9. Replica management costs (1000 nodes)

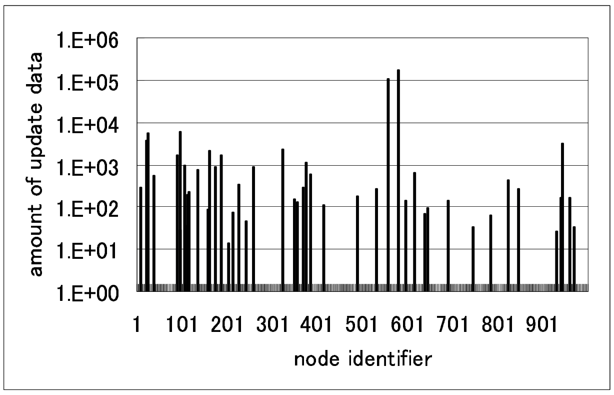


(a) Simple replica management

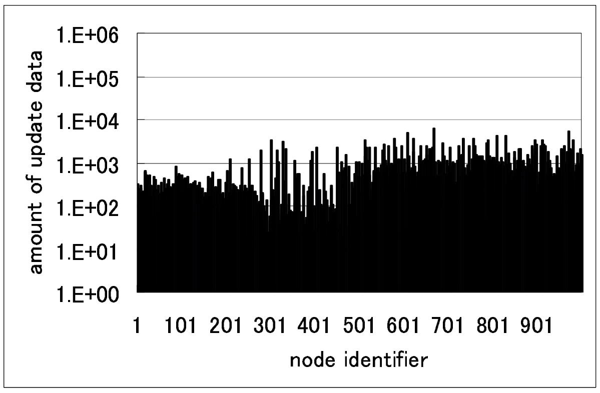


(b) Our proposal

Figure 10. Amount of update data sent by each node in complete update propagation (500 nodes)



(a) Simple replica management



(b) Our proposal

Figure 11. Amount of update data sent by each node in complete update propagation (1000 nodes)

TABLE III.
COMPARISON OF AMOUNT OF UPDATE DATA SENT BY EACH NODE IN COMPLETE UPDATE PROPAGATION

	# nodes	Data transmission costs		
		avg. $\times 10^2$	std. dev. $\times 10^3$	max. $\times 10^3$
Simple repl. management	500	4.12	4.85	88.4
	1000	3.16	6.84	169
Our proposal	500	7.89	0.747	4.65
	1000	4.87	0.706	6.46

TABLE IV.
COMPARISON OF AMOUNT OF UPDATE DATA SENT BY EACH NODE IN PARTIAL UPDATE PROPAGATION

	# nodes	Data transmission costs		
		avg. $\times 10^2$	std. dev. $\times 10^3$	max. $\times 10^3$
Simple repl. management	500	1.23	1.24	20.9
	1000	0.671	1.07	24.2
Our proposal	500	2.72	0.338	2.71
	1000	1.45	0.278	4.37

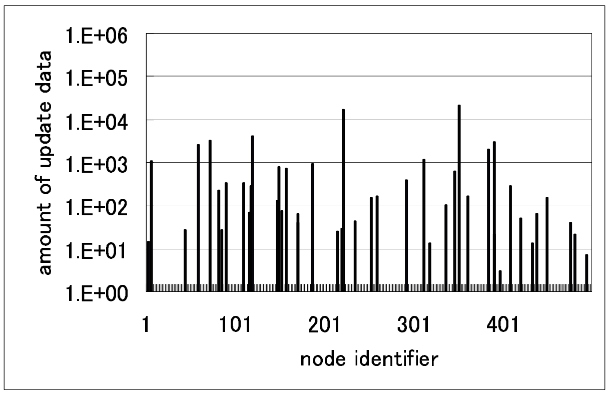
queries, and we measured the amount after 5000 exact match queries and 100 update operations in total. Assume update operations to a data were issued by Zipfian method with parameter 1.0.

We run the simulation in two cases: one was to propagate update information to all subscribers completely and the other was to change a part of update operations into delete operations during update propagation.

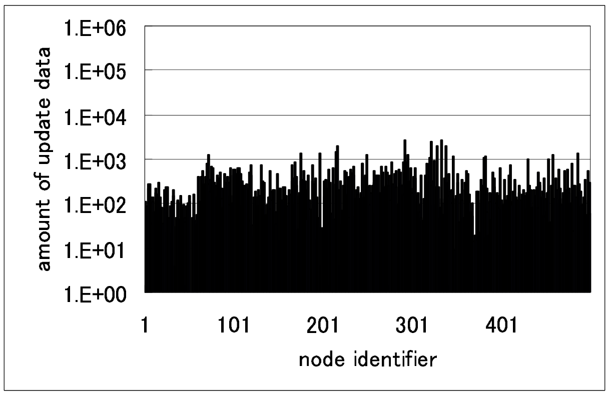
1) Complete Update Propagation: Figures 10 and 11 show the amount of data for update operations sent by each node in 500- and 1000-node P2P networks, respectively, in complete update propagation cases. We describe some statistical information obtained from the figures in Table III.

Note that our proposal could also realize load balancing in update operations by comparing with the simple replica management method. However, the average costs of our proposal were larger than those of the simple replica management method because of the differences of the number of hops between a primary node and subscribers as shown in Table I.

2) Partial Update Propagation: Figures 12 and 13 show the amount of data for update operations sent by each node in 500- and 1000-node P2P networks, respectively, in partial update propagation cases where a node changes 10% of update operations into delete operations in our proposal. Because the height of an RPT in the simulation was five, the possibility that a replica was updated was $0.9^5 = 0.59049$. In other words, about 60% replicas were updated and the rest were deleted. Thus, in the simple replica management method, a primary node sent update operations to 60% of its subscribers and delete operations to the rest. We describe some statistical information obtained from the figures in Table IV.

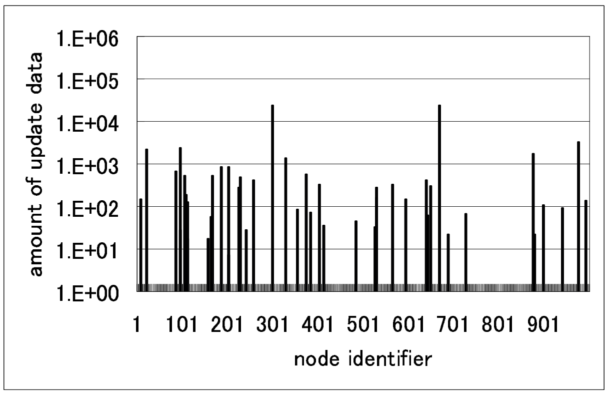


(a) Simple replica management

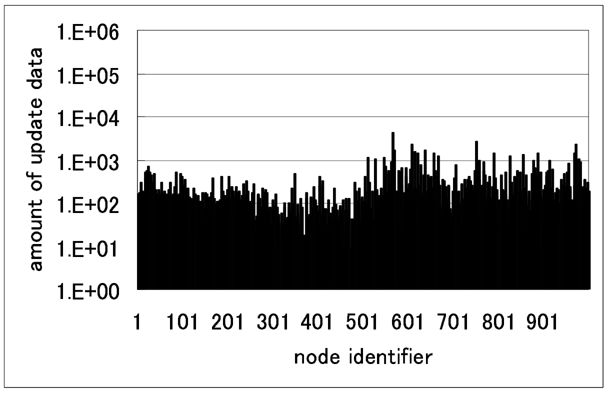


(b) Our proposal

Figure 12. Amount of update data sent by each node in partial update propagation (500 nodes)



(a) Simple replica management



(b) Our proposal

Figure 13. Amount of update data sent by each node in partial update propagation (1000 nodes)

We could reduce the costs compared with the previous results by changing some update operations into delete operations. However, the search costs would increase because the total number of replicas reduces. The analysis and optimization around update and search costs will be included in our future work.

VI. Conclusion

In this paper, a protocol that adapts RPTs [12] to an overlay network based on the balanced binary tree structure BATON [2] was proposed in order to realize load balancing by using replicas in a P2P network where some popular data exist. Some simulation results show that our proposal can realize load balancing of data transmission costs and also of replica management costs of nodes.

As we wrote before, we need to analyze and optimize around update and search operations. In [17], several replica management methods were evaluated in DHT-based networks under churn. We need to evaluate our proposal in such environments. We adopt owner replication and LRU in replica creation and reallocation in this work. On the other hand, [10] and [11] studied replica allocations to improve lookup performance in DHT-based networks. Also, [18] studied a resource replication strategy for multimedia data. We think we need to extend our proposal to have more sophisticated replica allocations in environments with more complex data like multimedia data. Jagadish et al., who are the researchers of BATON, proposed a general P2P framework allowing us to handle multi-dimensional data [19]. We intend to integrate the results into our proposal.

References

[1] K. Saji and M. Aritsugi, "A P2P protocol with load balancing using replicas: A proposal (in Japanese)," *DBSJ Letters*, vol. 5, no. 2, September 2006, pp. 9–12. [Online]. Available: http://www.dbsj.org/Japanese/DBSJLetters/vol5/no2/saji.pdf

[2] H. V. Jagadish, B. C. Ooi, and Q. H. Vu, "BATON: A balanced tree structure for peer-to-peer networks," in *Proc. 31st International Conference on Very Large Data Bases (VLDB 2005)*, 2005, pp. 661–672. [Online]. Available: http://www.vldb2005.org/program/paper/thu/p661-jagadish.pdf

[3] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in P2P systems," *Commun. ACM*, vol. 46, no. 2, February 2003, pp. 43–48.

[4] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications." *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, 2003, pp. 17–32.

[5] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001, pp. 329–350.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content-addressable network,” in *Proc. 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM ’01)*, 2001, pp. 161–172.

[7] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao, “Distributed object location in a dynamic network,” in *Proc. 14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA ’02)*, 2002, pp. 41–52.

[8] L. Plissonneau, J.-L. Costeux, and P. Brown, “Analysis of peer-to-peer traffic on ADSL,” in *Proc. 6th International Workshop on Passive and Active Network Measurement (PAM 2005)*, ser. Lecture Notes in Computer Science, vol. 3431. Springer, 2005, pp. 69–82.

[9] K. Tutschku, “A measurement-based traffic profile of the eDonkey filesharing service,” in *Proc. 5th International Workshop on Passive and Active Network Measurement (PAM 2004)*, ser. Lecture Notes in Computer Science, vol. 3015. Springer, 2004, pp. 12–21.

[10] V. Ramasubramanian and E. G. Sirer, “Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays,” in *Proc. 1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004, pp. 99–112.

[11] J. Kangasharju, K. W. Ross, and D. A. Turner, “Adaptive content management in structured P2P communities,” in *InfoScale ’06: Proc. 1st International Conference on Scalable Information Systems*. ACM, 2006, p. 24.

[12] X. Chen, S. Ren, H. Wang, and X. Zhang, “SCOPE: Scalable consistency maintenance in structured P2P systems,” in *Proc. 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, 2005, pp. 1502–1513.

[13] A. Datta, M. Hauswirth, and K. Aberer, “Updates in highly unreliable, replicated peer-to-peer systems,” in *Proc. 23rd International Conference on Distributed Computing Systems (ICDCS’03)*, 2003, p. 76.

[14] T. Hara, M. Nakadori, W. Uchida, K. Maeda, and S. Nishio, “Update propagation based on tree structure in peer-to-peer networks,” in *Proc. 2005 ACS/IEEE International Conference on Computer Systems and Applications (AICCSA’05)*, 2005, pp. 40–I.

[15] D. E. Knuth, *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

[16] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, “Search and replication in unstructured peer-to-peer networks,” in *Proc. 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS ’02)*, 2002, pp. 258–259.

[17] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod, “Performance evaluation of replication strategies in DHTs under churn,” in *MUM ’07: Proc. 6th International Conference on Mobile and Ubiquitous Multimedia*. New York, NY, USA: ACM, 2007, pp. 90–97.

[18] L. Rong, “Multimedia resource replication strategy for a pervasive peer-to-peer environment,” *Journal of Computers (JCP)*, vol. 3, no. 4, 2008, pp. 9–15. [Online]. Available: http://www.academypublisher.com/jcp/vol03/no04/jcp03040915.pdf

[19] H. V. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, and A. Zhou, “VBI-Tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes,” in *Proc. 22nd International Conference on Data Engineering (ICDE’06)*, 2006, p. 34.

Hidehisa Takamizawa received his B.E. and M.E. Degrees in Computer Science from Gunma University, Japan, in 2001 and 2003, respectively. He is presently with InterDesign Technologies, Inc., Japan, and a Ph.D. student at Gunma University, Japan. His research interests include database systems and security. He is a member of IPSJ.

Kazuhiro Saji received his B.E. and M.E. Degrees in Computer Science from Gunma University, Japan, in 2005 and 2007, respectively. He is presently with ACCESS CO., LTD., Japan.

Masayoshi Aritsugi received his B.E. and D.E. Degrees in Computer Science and Communication Engineering from Kyushu University, Japan, in 1991 and 1996, respectively. From 1996 to 2007, he was with the Department of Computer Science, Gunma University, Japan. Since 2007, he has been a Professor at the Graduate School of Science and Technology, Kumamoto University, Japan. His research interests include database systems and parallel/distributed data processing. He is a member of IPSJ, IEICE, ACM, IEEE-CS, and DBSJ.