P2P-based WSN Data Sharing System using B+Tree

Nobuhiko Matsuura†, Hiroshi Mineno‡, Ken Ohta††, Norio Shiratori*, and Tadanori Mizuno**

†Graduate School of Informatics, Shizuoka University, Japan
‡Faculty of Informatics, Shizuoka University, Japan
††Research Laboratories, NTT DOCOMO, Japan
* Research Institute of Electrical Communication, Tohoku University, Japan
** Graduate School of Science and Technology, Shizuoka University, Japan
{matsuura, mineno, mizuno}@mizulab.net

Abstract - A context-aware service that uses sensing data has attracted attention, along with the development of wireless technology and sensor technology. To provide these services, the sensing data sharing system in P2P networks needs to cope with a vast amount of data. However, existing algorithms do not respond to varying the number of sensing data types. In addition, most existing algorithms cannot execute reverse key resolutions because their search algorithms need to include specific data as the key in the query. To address these issues, we propose a multi-dimensional range search algorithm in P2P networks that uses a B+tree for an efficient search with an arbitrary number of sensing data types.

Keywords: P2P, B+tree, range search, multi-dimensional search, wireless sensor network

1 INTRODUCTION

Peer-to-Peer (P2P) networks are emerging as a new paradigm for structuring large-scale distributed systems. In these systems, resources are associated with keys, and each peer is responsible for a subset of the key to guarantee scalability performance, fault-tolerance, and robustness. P2P network have been developed to have a more suitable and practical design for applications, such as those in Building Monitoring [1] and Sewer Snort [2].

One of the systems that is suitable for using P2P is a sensing data sharing system, such as WSN with P2P [3]. A context-aware service has attracted attention, along with the development of wireless technology and sensor technology. This service can offer a remarkable transformation that considers user location and conditions using sensor data. To provide these services, the system needs to manage data from wireless sensor networks (WSNs). P2P networks are thought to be the answer to cope with the vast amount of data from WSNs because of their potential.

WSNs have some dynamic properties such as varying the data values, the total number of data, and the number of data property (such as Temperature and Humidity). In other words, P2P must deal with these properties to perform as general middle-ware for a WSN data sharing system because what it takes to provide service differs depending on the service. In particular, we are sure that considering the varying numbers of data property can improve the search performance because the sensor types will likely rapidly increase as the fundamental technology is developed. However, other works in P2P networks cannot deal with the varying number of data properties. These algorithms must include specific data, such as location information, as a key in the query. Therefore, they cannot execute a reverse resolution of keys and can only use a limited number from a vast number of sensor types in the query condition. To solve this problem, we need a multi-dimensional algorithm without a special property on the data store. "Multi-dimensional" denotes the tabular form in data storage, and the query condition number dynamically increases or decreases on demand. In this paper, we extend our previous work [4] and show simulation results to demonstrate the potential of our algorithm as a WSN data sharing system.

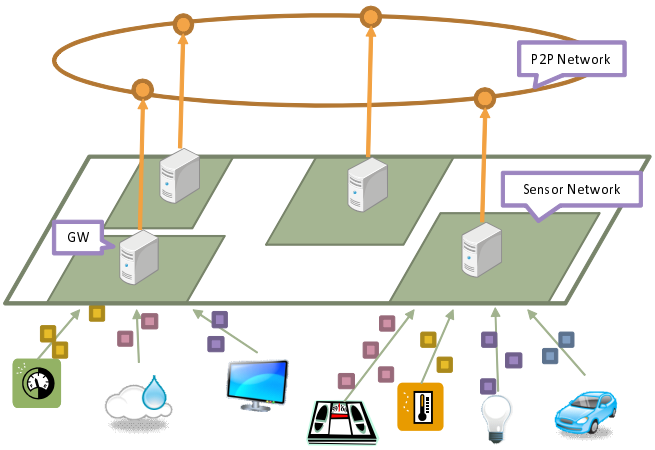


Figure 1: Sensing Data Sharing System

The rest of the paper is organized as follows: Section 2 discusses related work; Section 3 presents our previous work and the problem of bottleneck; Section 4 shows the experimental results and discussion; finally, Section 5 concludes the paper.

2 RELATED WORK

Many architectures have been developed on sensing data sharing systems using P2P. These architectures arrange Gateways (GWs) in WSNs to manage data transfers at each WSN and to automatically construct structuring P2P networks at each GW, an drown in Fig. 1. In general, structuring P2P networks are constructed using distributed hash tables (DHTs), but they have a major limitation in that they can only support an exact-match search. P2P networks need to have the exact key of a data item to store that item in the responsible node.

Table 1: NoSQL Categorization

Name	Data Model	CAP Theorem	Distribute Model	Persistence Model
Cassandra	Column-oriented	AP	Consistent Hash	Memtable/SSTable
HBase	Column-oriented	CP	Sharding	Memtable/SSTable on HDFS
CouchDB	Document-oriented	AP	Consitent Hash	Append-only B-tree
Riak	Document-oriented	AP	Consistent Hash	?
MongoDB	Document-oriented	CP	Sharding	B-tree
Tokyo Cabinet	Key-value	AP	Consistent Hash	Hash or B-tree
Voldemort	Key-value	AP	Consistent Hash	Pluggable
Redis	Key-value	CP	Consistent Hash	In-memory with background snapshots
Scalaris	Key-value	CP	Consistent Hash	In-memory only

Because the exact key is given by a hash function, the key has no order relation, and the user cannot search flexibly such as when processing a range query or a multi-dimensional query.

In P2P networks, the additional idea of flexible searching must be applied and must often use location information. Znet [5] uses Z-ordering of space-filling curves [6] to partition the 2-dimensional ID space of the location and map space ID onto corresponding nodes, and it uses a Skipgraph [7] to manage the network topology. Mill [8] uses the same partitioning and mapping technique, but it uses Chord to manage the network topology. LL-Net [9] partitions a space into 4 blocks recursively, and the quad tree is used to manage the network topology. These architectures can process a range query by using location information and also process a multi-dimensional query by adding other properties to the location ID space as an additional dimension. These P2P architectures have a limitation that involves including the location properties as a key in the query. In other words, they cannot execute a reverse resolution of the location information and also cannot deal with the varying number of properties without a reboot of the entire system because it is assumed that the number of dimensions, which represents properties, is static.

On the other hand, Skipgraph can process a flexibly search without location information. Skipgraph constructs a doubly-linked list of inserted keys at each layer according to the membership vector. The membership vector is an identifier like NodeID, and is allocated to all nodes. A certain key of the doubly-linked list in the i-layer has links between the closest key, which is defined by not only numerical distance but also i-length prefix matching of the membership vector. Skipgraph can process a range query by the sorted doubly-linked list and also process a multi-dimensional query by constructing multiple P2P networks. However, the link of Skipgraph depends on the value of key. It is difficult to maintain robustness when we use real-time and high-density data, such as sensor data.

NoSQL, which means “Not Only SQL”, is another efficient way to store and search data. NoSQL behaves as database, and the major example is key-value store, column-oriented database, and document-oriented database. Key-value store is often implemented on P2P networks. They are great hopes becoming the solution about the problem of relational database (RDB), such as scalability and availability. Today, we can select one of what can serve our purpose from a lot of NoSQL; however, we often think “What should I use?”. Table 1 shows a part of NoSQL categorization list. In the table, CAP theorem says it is impossible for a distributed computer system to simultaneously provide all of the three guarantees (Consistency, Availability, and Partition Tolerance) and a distributed system can satisfy any two of these guarantees at the same time.

There are many work for categorization and comparing NoSQL, such as [10]. Cassandra [11] and HBase [12] are the most famous systems of NoSQL. Cassandra is the Apache [13] project and has a goal: develops a highly scalable second-generation distributed database, bringing together Dynamo [14] distributed design and Bigtable [15] column-oriented data model. We can insert the data which contained key space, column family, key, column, and value in Cassandra like RDB’s record. HBase is also Apache project and behaves as the database of Hadoop [16]. This goal is the hosting of very large tables, such as X billions of rows and Y millions of columns, atop clusters of commodity hardware. We can insert the text, such as log file, in the Hadoop Distributed File System which partition the file and distribute the part of file in multiple server. It is said that Cassandra and HBase have higher availability and scalability than RDB. However, they need to construct the distributed system on static network, such as data center. Thus, they are not good way to construct the distributed system includes the home GW.

3 THE DESIGN OF MULTI-DIMENSIONAL ALGORITHM

3.1 Overview

We are designed multi-dimensional algorithm in our previous works, and this algorithm uses a B+tree [17] for an efficient search with an arbitrary number of sensing data type. Our algorithm introduces the idea of building as many tree structures as there are properties to process a dynamic multi-dimensional range search, and these tree nodes (tree-nodes) are mapped to the node on P2P networks (P2P-nodes). The user can select an arbitrary number of trees to process a multi-dimensional search, and a conclusive result is obtained by merging all the results. Therefore, the reverse resolution of certain properties, such as location, can be done by using the

Table 2: Node Information

Node (NodeID)	temp
N_0 (5)	21
N_1 (3)	30
N_2 (1)	15
N_3 (0)	27
N_4 (4)	NULL
N_5 (2)	29
N_6 (7)	18
N_7 (6)	24

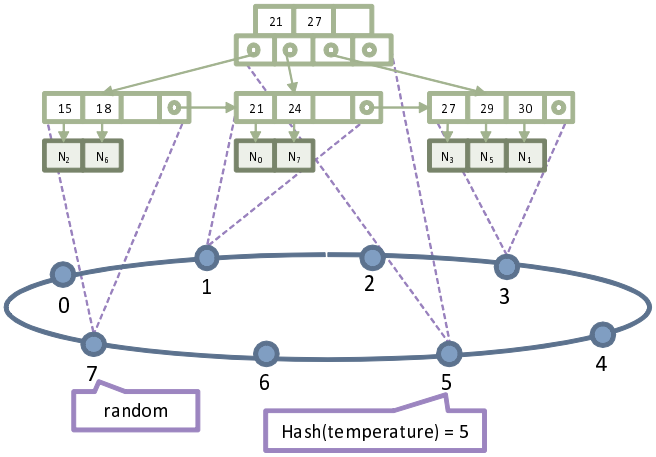


Figure 2: Mapping to P2P Networks

composition query, and the system can deal with increasing and decreasing properties while maintaining the running state.

To build a strong tree structure for frequently varying data values, such as sensor data, our algorithm uses a B+tree, which is a balanced tree, and the serch and insert order provides $O(log_t N)$ search cost, where N is the number of peers in the system and t is the constant number depending on list size of tree-nodes. The B+tree acts as a logic structure to make a comparison across property values. A network topology uses a P2P network to store the tree-nodes. Mapping from the tree-node to the P2P-node is necessary. In our algorithm, this mapping uses a hash function. Mapping is random in most tree-nodes, but the root of the tree should be uniquely known to all nodes because the search for the B+tree should obtain the root in the beginning. Therefore, the root has a mapping rule wherein the map from the tree-node to the P2P-node of $NodeID = Hash(PropertyName)$ acts as the manager node, and other node map to the P2P-node of $NodeID = Hash(Random)$. Based on this analysis, when the nodes listed in Table 2 showing a set of nodes and their properties join the P2P networks, mapping is done as in Fig. 2. Of course, this algorithm can deal with more properties by building a new tree structure.

3.2 Algorithm

If a new node join a P2P network, it should join the P2P network and insert its shared resource information, such as properties and its own NodeID, into a tree-node on the P2P-node. Join and leave algorithm is used according to the P2P algorithm. For example, the join algorithm of Chord is used when we use the Chord algorithm to construct a base P2P network. This put process shown as Algorithm 1. The node obtains a hash value based on property name and sends an insert request to the manager node that has the same hash value. The received node relays the request to the child tree-node that includes the request key between the tree ranges. On the other hand, the node creates a root for the corresponding tree if the tree-node does not exist. By repeating this recursive operation, the node in the target tree finally finds the destination leaf and inserts the data into the appropriate place.

The remove algorithm is shown as Algorithm 2, and a large part of it is constructed similarly to the put algorithm.

The get algorithm is shown as Algorithm 3. The algorithm also searches tree-nodes like the put and remove algorithms, but this algorithm uses a range key and an iterative search. There are two cases of supporting range key search in com-

Algorithm 1 Put(targetHash, propertyValue, timeStamp)

Require: targetHash is closest to myNodeID
  $storedValue \leftarrow SearchLocal(targetHash)$
  if $storedValue = null$ **then**
    $PutLocal(targetHash, propertyValue, timeStamp)$
  else
    if $!storedValue.isLeaf()$ **then**
      for all c such that $storedValue.getChild()$ **do**
        if $(c.min \leq propertyValue) \&\& (propertyValue < c.next.min)$ **then**
          $Put(c.getNodeID(), propertyValue, timeStamp)$
        end if
      end for
    else
      if $storedValue.isFull()$ **then**
        $Separate(storedValue)$
      end if
      $PutLocal(targetHash, propertyValue, timeStamp)$
    end if
  end if

Algorithm 2 Remove(targetHash, propertyValue)

Require: targetHash is closest to myNodeID
  $storedValue \leftarrow SearchLocal(targetHash)$
  if $storedValue! = null$ **then**
    if $!storedValue.isLeaf()$ **then**
      for all c such that $storedValue.getChild()$ **do**
        if $(c.min \leq propertyValue) \&\& (propertyValue < c.next.min)$ **then**
          $Remove(c.getNodeID(), propertyValue)$
        end if
      end for
    else
      if $storedValue.contain(propertyValue)$ **then**
        $RemoveLocal(targetHash, propertyValue)$
      end if
    end if
  end if

Algorithm 3 Get(targetHash, min, max)

```
treeNode ← dht.get(targetHash)
result ← {}
for all e such that treeNode.getElements() do
  if treeNode.isLeaf() then
    if (min ≤ e.key)&&(e.key ≤ max) then
      result ← result ∪ e
    end if
  else
    if (min ≤ e.max)&&(e.min ≤ max) then
      r ← Get(e.hash, min, max)
      result ← result ∪ r
    end if
  end if
end for
return result
```

paring the search request with its own tree-node's information. Case 1 is a comparison of the leaf; this selects the data included between the request ranges. Case 2 is a comparison of the other place; this selects the node that includes the request range between its own tree ranges as the next candidate. The get algorithm repeats case 2 to find destination leaf-nodes, and it executes case 1 to obtain the result. We used an iterative search to repeat case 2 because a recursive search risks a fatal decrease in performance in the target environment, such as the general middle-ware for a WSN data sharing system. The tree-node needs to split the request into the same number of branches if two or more candidates are found, and then the tree-node must wait for the under-layer processing to finish or timeout in accordance with the recursive search. An iterative search can prevent this risk because it forces the node that sent the search request to wait.

3.3 The Problem of Previous Work

Actual sensor data occurs with great frequency because a sensor is generally used to obtain real-time and high-density data. The first thing to do in the these algorithms is to access the root of the tree structure, and the root tends to have a heavier workload than other nodes. Therefore, the root of the tree structure may become a bottleneck in the system. A bottleneck arises from insufficient disk space and memory, insufficient CPU capacity, and an exclusive control method.

- Insufficient disk space and memory
  When number of inserted data is over allowable number, the node frequently causes the Swapping and performance decreases. In addition, the node which too much data is concentrated greatly decreases performance in searching stored data. This is problem in a general relational database (RDB), but the Key-Value store (KVS) on P2P networks is not actually considered as problem. The KVS can easily deal with the increasing data by distributing the data to many nodes. This is called scale-out technology, therefore, it is not considered in this paper.

- Insufficient CPU capacity
  When the node receives requests, CPU use and processing time increases according to the request type and the current state, and it becomes a problem when the number of requests increases up to a certain number. A typical KVS can resolve this problem because it can distribute not only data but also the load of the processing request. However, the multi-dimensional algorithm on a KVS cannot resolve the problem well because it constructs a large tree structure. This structure can support the tabular form on KVS and is guaranteed to reach the target, but it limits the route to certain data and concentrates a local load. In particular, the nodes included in the route, such as the root, has a larger workload than other nodes.

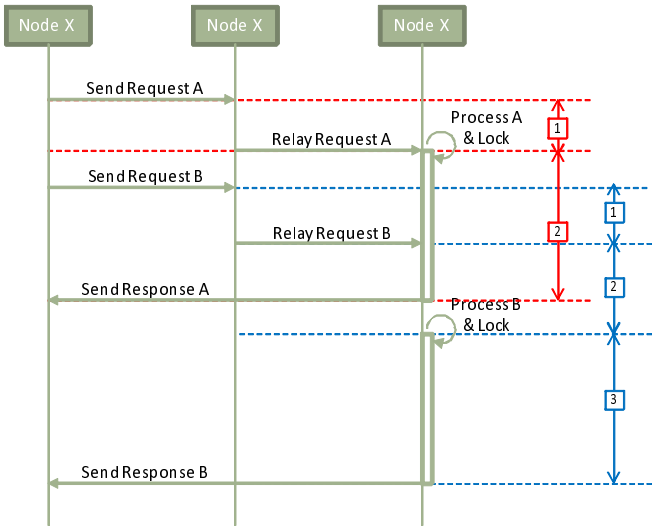


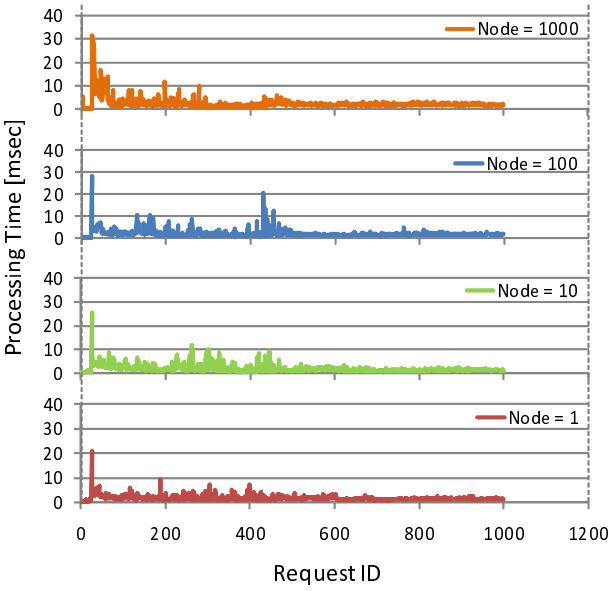
Figure 3: PutRequestSequence

- Exclusive control method
  This factor is an endemic problem of distributed systems and fatal problem of this architecture. The KVS is used by many users, and operation demand may occur around the same time. The structure breaks if operation is executed in parallel because it constructs a tree structure, as previously mentioned. To address this problem, an exclusive control method, such as lock, is needed. However, this method has a bad affects the processing time of the request. In other words, it is important to reduce the maximum number of concurrent connection to maintain throughput.

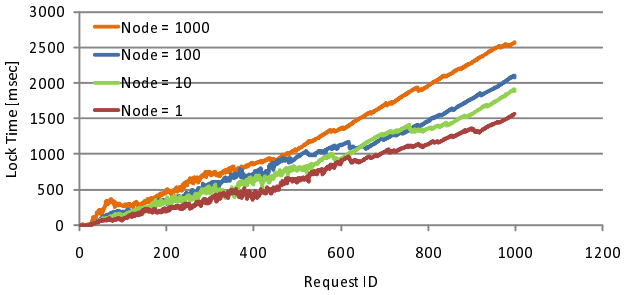
Based on the above analysis, it is necessary to evaluate the process time and the lock before being applied to the WSN data sharing system. In addition, we pay attention to the exclusive control method problem that is the biggest weakness.

4 PERFORMANCE EVALUATION

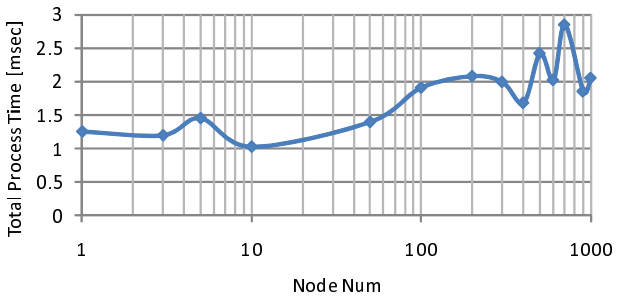
In this section, we describe the evaluation of our algorithm in delay time. There are three types of communication, lock, and processing delay time. Fig. 3 shows the sequence of processing the put request. The put request is sent from $Node_X$ to $Node_Z$. When $Node_X$ sends request A, $Node_Y$ receives



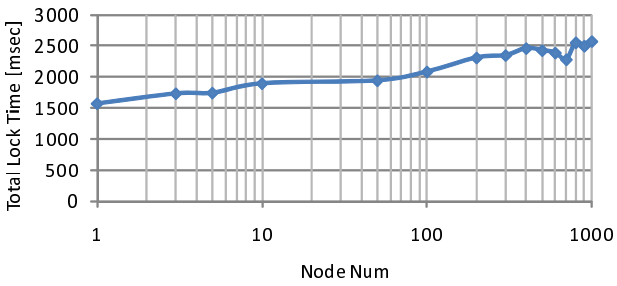
(a) Processing Time



(b) Lock Time



(c) Total Processing Time



(d) Total Lock Time

Figure 4: The Result Varying Number of Node

Table 3: Simulation Environment

Varying	Node	Data
Num Nodes	1 – 1000	100
Num Request	1000	1 – 10000
Data Distribution	Normal	Normal

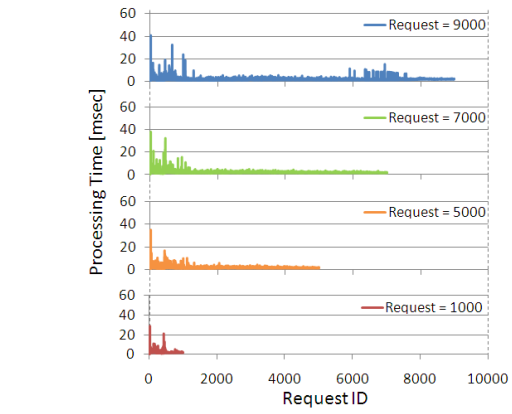
the request once and relays it to $Node_Z$. The communication and processing time occur in this sequence. Communication time is the time between sending and receiving of the request No.1 in the figure. Processing time is the time between processing of the request No.3. Lock time occurs in processing of request B. Request B is sent like request A, and lock time occurs by waiting for the processing of the request B. Lock time is the time between receiving and the start of processing No.2.

We evaluated the lock and processing times when a request is sent to one root node just around the same time. We did not evaluate the communication time because we wanted to consider only root performance. The evaluation was conducted with simulations using Overlay Weaver [18] in the conditions listed in Table 3. The list size of the tree-node in our algorithm was adjusted in the experiment to 25, and this algorithm used Chord to construct the base P2P networks.

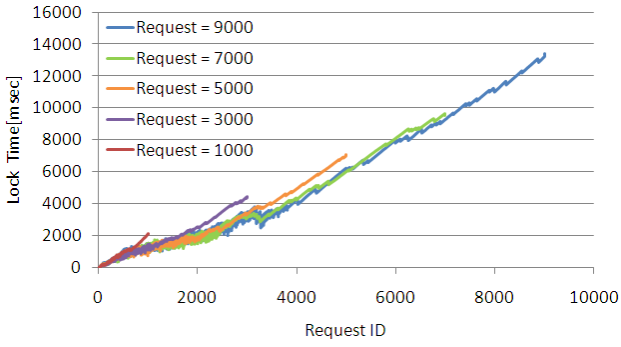
Fig. 4(a), 4(b) show the history of processing each request that has a sequential unique ID, where Node is the number of node in a P2P network. Fig. 4(c), 4(d) show the total time for processing the last request at each node. Fig. 5(a), 5(b) also show the history of processing each request that has a sequential unique ID, where Request is the number of simultaneously inserted requests. Fig. 5(c), 5(d) show the total time for processing the last request at each node.

Fig. 4(a) has place where processing time suddenly increases. The reason is that the B+tree structure must divide a full leaf, create new a leaf to store the leaf information, and increase the number of layers on the tree structure in addition to normal operation. These additional operations occur when the number of leaves becomes more than 25^i in an ideal environment, where 25 is adjusted as the leaf size. These additional operations also affect Fig. 4(b). The part near the origin point in Fig. 4(b) has a very low delay time, and the time suddenly increases when requests ID exceede 25 because the root can process the up to 25 request in only oneself. When the Fig. 4(c), 4(d) is considered, the effect of node number is small because the order is $O(logN)$, where N is the number of nodes. The reason of this order is that we constructed the tree structure on a P2P networks with Chord algorithm. The Chord algorithm search cost is $O(logN)$, and we used Chord search method to find the appropriate child node on the tree structure, and the effect of the Chord search order appeared in the result. We are certain that our idea, building a large structure on P2P networks to handle a multi-dimensional as our work, have enough scalability.

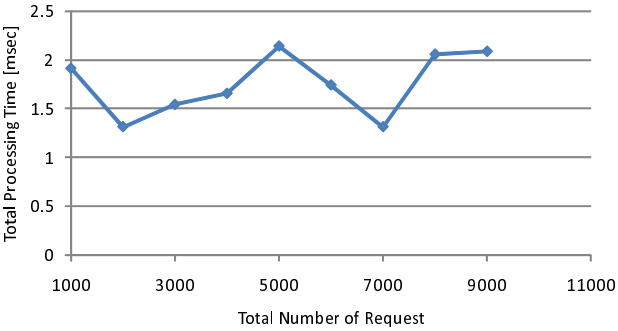
Fig. 5(a) shows a similar change as in Fig. 4(a), and the change in Fig. 5(c) is flat because rapid change occurs by dividing and creating leaves when the leaf is full. In Fig. 5(b), the total number of varying requests does not affect the lock time, and the result linearly increases, as shown in Fig. 5(d).



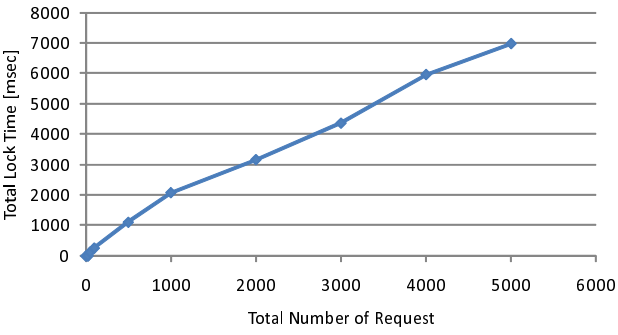
(a) Processing Time



(b) Lock Time



(c) Total Processing Time



(d) Total Lock Time

Figure 5: The Result Varying Number of Request

Because the request goes into a queue once, even if the system receives any number of request at any time. On the other hand, Fig. 5(d) shows throughput that can process x request by y msec from point of view of the system side. We can obtain the throughput that guarantees to process as soon as receiving request from the intersection which figure with expression $y = 1000$, and throughput is about 500 requests per second. This throughput is not too few because the traditional RDB's default number of concurrent connections is from 100 to 500 on demand and the some systems often remain at the default number.

5 CONCLUSION

We designed a multi-dimensional search algorithm for P2P networks using a B+tree. Our novel P2P index structure is well suited for applications, such as a sensing data sharing systems by supporting range and multi-dimensional queries. This structure is in accordance with the basic key idea that a tree structure, such as a temperature-tree, builds on P2P networks for each property.

We evaluated the performance of the proposed algorithm using simulations. From the simulation results, we found that the proposed algorithm has scalability because the processing and lock times are order $O(logN)$. In addition, the throughput that guarantees to immediately process the request is about 500 requests per second.

In the future, we will aim at the performance gain of the algorithm. We can improve the performance using the replication or cache algorithm. The replication may distribute the workload of the root and decrease the lock time. Cache algorithm may dramatically improve the performance because the number of request relays becomes 1 when the node has visited by search algorithm in the past. Additionally, both algorithm can become churn tolerant improvement techniques. Churn will cause a serious problem for the P2P put algorithm: The data of node is removed due to the continuous process of node arrival and departure. To use replication and cache algorithm, at least one node can hold inserted data and the system can recover from churn damage by using the data.

REFERENCES

[1] M. Ceriotti, L. Mottola, G. Picco, A. Murphy, S. Guna, M. Corra, D. Zonta and P. Zanon, "Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment," Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN'09) (2009).

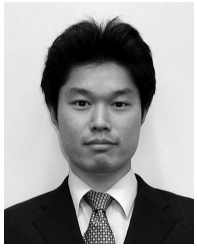
[2] J. Kim, J. Lim, J. Friedman, U. Lee, L. Vieira, D. Rosso, M. Gerla and M. Srivastava, "SewerSnort: A Drifting Sensor for In-situ Sewer Gas Monitoring," Proceedings of the 6th Annual IEEE communications society conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON'09) (2009).

[3] G. Gutierrez, B. Mejias, P. Roy, D. Velasco and J. Torres, "WSN and P2P: A Self-Managing Marriage," Proceedings of the 2nd IEEE International Conference on

Self-Adaptive and Self-Organizing Systems Workshops (SASOW'08) (2008).
[4] N. Matsuura, H. Mineno, N. Ishikawa and T. Mizuno, "Evaluation of B+Tree-based Multi-dimensional Range Search Algorithm for P2P Networks," Proceedings of the 20th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT'10) (2010).
[5] Y. Shu, B. Ooi, K. Tan and A. Zhou, "Supporting Multi-dimensional Range Queries in Peer-to-Peer System," Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing (P2P'05) (2005).
[6] J. Wierum, "Logarithmic Path-Length in Space-Filling Curves," Proceedings of the 14th Canadian Conference on Computational Geometry (CCCG'02) (2002).
[7] J. Aspnes and G. Shah, "Skip Graphs," ACM Transactions on Algorithms (2007).
[8] S. Matsuura, K. Fujikawa and H. Sunahara, "Mill: Scalable Area Management for P2P Network based on Geographical Location," Proceedings of the 12t Annual Scientific Conference on Web technology, New Media, Communications and Telematics theory, Methods, Tools and Applications (Euromedia'06) (2006).
[9] Y. Kaneko, K. Harumoto, S. Fukumura, S. Shimojo and S. Nishio, "A location-based peer-to-peer network for context-aware services in a ubiquitous environment," Proceedings of the Symposium on Applications and the Internet Workshops (SAINT'05) (2005).
[10] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan and R. Sears, "Benchmarking cloud serving systems with YCSB," Proceedings of the 1st ACM symposium on Cloud computing (SoCC'10) (2010).
[11] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," Proceedings of the 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware (LADIS'09) (2009).
[12] HBase, http://hbase.apache.org/.
[13] The Apache Software Foundation, http://www.apache.org/.
[14] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogels, "Dynamo: Amazon's Highly Available Key-value Store," Proceedings of the 21st ACM SIGOPS symposium on Operating systems principles (SOSP'07) (2007).
[15] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes and R. Gruber, "Bigtable: A Distributed Storage System for Structured Data," Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06) (2006).
[16] Hadoop, http://hadoop.apache.org/.
[17] D. Comer, "The Ubiquitous B-Tree," ACM Computing Surveys (1979).
[18] Overlay Weaver, http://overlayweaver.sourceforge.net/.

(Received August 24, 2010)
(Revised April 26, 2011)

Nobuhiko Matsuura received the B.I. degree in Informatics from Shizuoka University, Japan in 2010. He is currently working towards the M.I. degree at Shizuoka University. In 2010, he received Best paper award at the International Workshop on Informatics. His current research interests include Internetworking, distributed systems, Peer-to-Peer networks, and databases.



Hiroshi Mineno received his B.E. and M.E. degrees from Shizuoka University, Japan in 1997 and 1999, respectively. In 2006, he received the Ph.D. degree in Information Science and Electrical Engineering from Kyushu University, Japan. Between 1999 and 2002 he was a researcher in the NTT Service Integration Laboratories. In 2002, he joined the Department of Computer Science of Shizuoka University as an Assistant Professor. His research interests include sensor networks as well as heterogeneous network convergence. He is a member of IEEE, ACM, IEICE, IPSJ and Informatics Society.

Ken Ohta received B.E., M.E. and Ph.D. degree from Shizuoka University, Shizuoka, Japan, in 1994, 1996 and 1998. In 1999, he joined NTT DoCoMo, Inc. His current research includes mobile terminal architecture, mobile application, and mobile security. He is a member of IEICE.

Norio Shiratori received his doctoral degree from Tohoku University in 1977. After that he served as an Associate Professor and Research Associate at Research Institute of Electrical Communication (RIEC), Tohoku University. He was also the Professor of Information Engineering at Tohoku University from 1990 to 1993. Now he is the President of IPSJ (Information Processing Society of Japan) and served as an IFIP representative of Japan. Now, he is an Emeritus and Research Professor at RIEC, Tohoku University, Japan. His research interests include Ubiquitous and Symbiosis computing. He is the fellow of IEE, IPSJ and IEICE.

Tadanori Mizuno received the B.E. degree in industrial engineering from the Nagoya Institute of Technology in 1968 and received the Ph.D. degree in computer science from Kyushu University, Japan, in 1987. In 1968, he joined Mitsubishi Electric Corp. Since 1993, he is a Professor of Shizuoka University, Japan. Now, he is a Professor of graduate school of Science and technology of Shizuoka University. His research interests include mobile computing, distributed computing, computer networks, broadcast communication and computing, and protocol engineering. He is a member of IEEE, ACM, IEICE, IPSJ and Informatics Society.