*Regular Paper*

# Distributed Zone Partitioning Schemes for CAN and Its Application to the Load Balancing in Pure P2P Systems☆

Daisuke Takemoto,† Shigeaki Tagashira† and Satoshi Fujita†

In this paper, we propose several distributed zone partitioning schemes for Content-Addressable Networks (CAN), that is known as a pure peer-to-peer system based on the Distributed Hash Table (DHT). The main objective of the proposed schemes is to balance the load of nodes in the CAN system, in such a way that every node receives almost the same number of inquiries from the other nodes in the system. The result of simulations implies that, by using the proposed schemes instead of a randomized zone partitioning scheme originally implemented in the CAN system, we could reduce the response time for each inquiry to less than 75%.

## 1. Introduction

According to the recent advancement of network technologies, it emerges an increasingly strong requirement for the high quality communications over the large-scale interconnection networks. In fact, as the number of web sites serving real-time contents increases, the number and the size of data flows exchanged among remote hosts also increase, and in addition, it significantly increases the complexity of server procedures to keep (or often to improve) the quality of such data streams to be satisfactory. In general, a high complexity of server procedures will limit the scalability of distributed systems under the conventional server-client model, and it motivates the study of *fully distributed systems* such as grid computers and peer-to-peer (P2P) systems. A P2P system consists of a collection of host computers called nodes or peers, and those nodes are connected with each other by an interconnection network such as the Internet. In recent years, a lot of important services such as shared file systems and Domain Name Systems (DNS) are constructed over the P2P model, and they have been used in many application fields, such as electronic bulletin board, network auction systems, and so on.

By their logical structure, P2P systems could be classified into two categories, i.e., hybrid type or pure type. In hybrid P2P systems, retrieval of objects will be realized by sending a query message to a dedicated server who maintains a set of indices to the objects in a centralized manner, while the actual contents of the objects will be maintained by each node in a distributed manner. On the other hand, pure P2P systems do not rely on servers, and the retrieval of contents will be realized by peer nodes in a distributed manner. Examples of pure P2P systems include Gnutella [2] and FreeNet [3], where in Gnutella, indices of objects will be retrieved by using collective communications over all nodes in the system, that severely limits the scalability of the overall system.

Distributed Hash Table (DHT) is a common technique to overcome such a low scalability of the flooding-based indexing schemes, and it has been applied to many pure P2P systems such as Tapestry [11], Chord [10], P-Grid [1], and Content-Addressable Network [7]. For example, in the Content-Addressable Network (CAN) proposed by Ratnasamy *et al.*, indices to the objects are maintained through DHT in a fully distributed manner. More concretely, CAN constructs a virtual $d$-dimensional coordinate space as a table of indices, and each index that should be stored and retrieved by each node is mapped onto a point in the space by an appropriate uniform hash function (a detailed explanation of CAN will be given in Section 2). At any point in time, the entire coordinate space is dynamically partitioned among nodes in the system, in such a way that every node locally maintains indices contained in its individual portion of the space called **zone**, and thus, the store and the retrieval of an index will be realized by routing

† Hiroshima University

☆ Earlier versions of some results contained in this paper appear in D. Takemoto, S. Tagashira, S. Fujita, "Distributed Algorithms for Balanced Zone Partitioning in Content-Addressable Networks," *Proc. 10th International Conference on Parallel and Distributed Systems* (*ICPADS*).

an inquiry message to the corresponding point in the coordinate space, i.e., to the node who maintains indices corresponding to the target point.

In this paper, we focus on the zone partitioning problem in the CAN system. Although CAN could realize a scalable access to the contents of the objects [7], the load of nodes would not be well balanced when the spatial distribution of the mapped indices is not uniform and/or the access to the indices is not uniform. In this paper, we will measure the load of nodes in terms of the number of received inquiries; i.e., it is assumed to be proportional to the *access frequency* to the indices maintained by the node. We propose three schemes for solving the load balancing problem. The first one collects an accurate load information from all nodes in the system by using an expensive collective communication (as in Gnutella), the second one collects merely local information that is less accurate but much cheaper than the first one, and the third one collects an accurate information by using a *distributed heap*, that is an extension of a scheme for realizing a dynamic set in distributed networks [4]. Note that the first two schemes are based on a technique that is commonly used in the field of parallel computing, and for those schemes, there is a trade-off between the accuracy and the cost for acquiring an appropriate load information. The proposal of the third scheme is intended to overcome such a trade-off point; i.e., it intends to acquire an accurate load information with a low communication cost. The proposed schemes are evaluated and compared by simulation. The result of simulations implies that, when the number of peers is 64, the second and the third schemes exhibit almost the same performance, and by using those schemes instead of a randomized scheme that is originally implemented in the CAN system, we could reduce the response time for each inquiry to less than 75%.

The remainder of this paper is organized as follows. Section 2 describes an outline of CAN, and shows the result of our preliminary experiments to demonstrate the spatial imbalance of the mapped points by typical uniform hash functions. Section 3 proposes three zone partitioning schemes. The effectiveness of the proposed schemes in terms of the response time to inquiries is evaluated in Section 4. Finally, Section 5 concludes the paper.

## 2. Content-Addressable Network

### 2.1 Overview

CAN provides a mechanism for managing and retrieving objects distributed over the network, in a fully distributed manner. Consider a P2P system consisting of several nodes, and suppose that each node locally stores a set of objects. Each object has a name that will be referred to as the key of the object in what follows. The **index** of an object stored in a node is represented as a key-value pair $(K_1, V_1)$, where $K_1$ is the key of the object and $V_1$ is the unique name (e.g., IP address) of the node. The basic idea of CAN is to construct a *table of indices* to all of the objects stored in the system. The table is realized as a virtual $d$-dimensional Cartesian coordinate space on a $d$-torus, where in the following, we fix $d$ to two for simplicity; i.e., we represent each point in the space by a pair of its $X$- and $Y$-coordinate values. At any point in time, the entire coordinate space (i.e., the entire index table) is dynamically partitioned into **zones**, and those zones are assigned to the nodes in the system in such a way that every node locally maintains all indices contained in its individual zone.

The store of an index $(K_1, V_1)$ to the coordinate space (i.e., index table) is realized as follows: It first maps key $K_1$ onto a point in the space by using a uniform hash function. It then stores the index to the node corresponding to the zone containing the target point. The retrieval of an index could be done in a similar manner; that is, to retrieve an object with key $K_1$, it applies the same hash function to map the key onto a point, and then retrieves indices from that point.

If the calculated target point is maintained by the requesting node, the access to the index can be locally done. However, if it is not the case, a request for accessing the index must be routed through the CAN infrastructure until it reaches the node who maintains it. Each node in CAN is designed to learn and maintain its neighboring nodes with respect to their associated zone. This set of immediate neighbors serves as a coordinate routing table that enables routing between arbitrary points in the coordinate space. See **Fig. 1** for illustration.

### 2.2 Observation on the Zone Partition

As described above, in CAN, the entire coordinate space is divided into small portions called zones, and those zones are assigned to

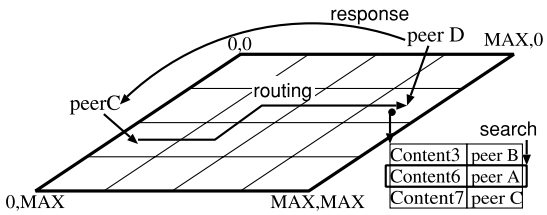


**Fig. 1** A retrieval of index in CAN; 1) When C requires content 6, it sends an inquiry to a node who maintains the index to the content via routing mechanism; 2) Upon receiving the inquiry, D returns the name of the peer who maintains the content to C.

the nodes that are currently participating to the system in a one-to-one manner. To allow CAN to grow incrementally, a new node joining the system must be assigned its own portion of the coordinate space, by splitting a zone maintained by an existing node to a half. More concretely, such a splitting proceeds as follows:

(1) The new node $u$ finds a bootstrap node $v$ that already exists in the CAN. In what follows, we refer to node $v$ as the **requester**.
(2) Node $v$ locates another node $w$ whose zone will be split into two subzones.
(3) Node $w$ splits his zone into two subzones by using an appropriate coordinate, and assigns one subzone to the new node $u$. It then notifies the fact to all neighbors of the split zone to keep the consistency of the CAN routing table.

In the method shown in the original paper [7], the locating of a target node in the second step is conducted in a random manner. More concretely, the requester randomly chooses a point in the space and sends a JOIN request destined for the point, that will eventually be received by the node who maintains indices corresponding to the point. In the following, we refer to this simple method as Random. As was mentioned previously, it is important to balance the load of nodes, that is assumed to be proportional to the access frequency, to realize a quick response to each inquiry. In Random, however, such a load balancing could be achieved merely in an "expected" sense, and there remains a non-negligible deviation on the number of received inquiries that would significantly increase the response time of the system, in the worst case.

To verify the above conjecture, we conducted a preliminary experiment to evaluate the spatial imbalance of the mapped points under typical CAN settings. In the experiment, we first extract 230,000 words from English dictionary as a set of keys, and map them into the 2-dimensional space by using a hash function that is a function actually used in JXTA [5]. We then partition the entire space into $2^{14}$ small zones of an equal size, and count the number of keys mapped onto each zone. The result is summarized in **Fig. 2**, where the horizontal axis represents the number of keys associated with a zone, and the vertical axis represents the number of such zones containing given number of keys.

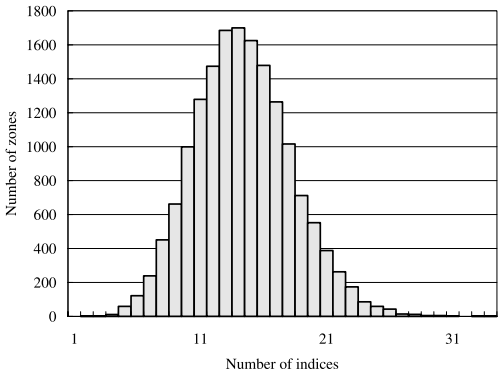


**Fig. 2** Distribution of the number of keys in each zone (the total number of zones is $2^{14}$ and total number of keys is 230,000).

As is shown in the figure, the minimum and the maximum number of keys in a zone are two and 34, respectively, i.e., a zone contains 17 times more keys than another zone, although every zone has an equal size in the given coordinate space. A similar result could be observed when we replace the hash function by other ones such as SHA-1 ☆. Hence, we can conclude that although it might balance the load of nodes in an expected sense, there remains a non-negligible deviation (17 times, in the above case) even when each zone has an equal size and each key is accessed with the same frequency (as will be shown in Section 4, an imbalance of access frequency freqeucntly occurs if there is an access locality to the stored keys). The above observations justify the importance of the problem of load balancing to realize a quick retrieval of indices, since an overloaded node would become a bottleneck, and it can easily degrade the overall performance.

☆ SHA-1 is a hash function commonly used for authentification and digital signature [6].

## 3. Quick Location of Target Node

In this section, we propose three node locating schemes to find a target node in the zone partitioning scheme. All the schemes proposed here try to collect *load information* distributed over the CAN system. The first and the third schemes collect a global information (i.e., global maxima), whereas the second one tries to collect merely local information (i.e., local maxima). The difference of the first and the third ones is the access time and the time lag before reflecting the change of load status to the global information; i.e., in the first scheme, it takes a time proportional to the diameter of CAN to find a target node, although the change of status immediately reflects to the global information. On the other hand, in the third scheme, both of the update and the retrieval of the global information take relatively short time, that is $O(\log n)$ in an amortized sense.

### 3.1 Multicast-Discover Method

The first method is based on the flooding of inquiry messages to the all nodes in the system. In Ref. 8), a multicast protocol for CAN is proposed and evaluated. In our first method, called Multicast-Discover method (MD), a global load information is collected by using this multicast protocol, and a node with a heaviest load is selected as the target node. A requester who wants to find a target node broadcasts an inquiry message to all the other nodes by using the multicast protocol given in Ref. 8). Upon receiving an inquiry message, each node returns its access frequency to the requester by using the broadcast route in the reverse direction. After collecting them, the requester identifies a node with a heaviest load, and selects it as the target zone to be split into two halves.

The multicast protocol given in Ref. 8) proceeds as in a simple X-Y routing in the 2-dimensional case; i.e., the requester sends a message to two neighbors in X-dimension, that will be forwarded to the succeeding ones until they meet with each other at the opposite side of the torus; and after receiving a message via X-dimension, it sends copies of the message to two neighbors in Y-dimension until they meet with each other. Thus, it takes a time proportional to the diameter of CAN and the total number of exchanged messages is proportional to the number of nodes in the system, although it could always find a node with a heaviest load, at the time when the inquiry message is received by the node.

### 3.2 Gradient Method

The second method, called Gradient Method (GR), is a quasi-optimal scheme based on the hill climbing method; i.e., the requester tries to find a local maxima in terms of the access frequency by using a local search method. Let next be a local register to point a node with a high access frequency (note that each node has its own next register). The landscape for the hill climbing is constructed as follows: 1) every node exchanges its local information with its all neighbors for every $T$ seconds; and 2) if none of the neighbors have a higher access frequency than itself, then it sets next to point itself, and otherwise, to a neighboring node with a highest access frequency. In the following, we denote the GR with maintenance interval $T$ seconds as GR($T$).

**Figure 3** illustrates an outline of the retrieval of a target node under GR. The retrieval is conducted based on the collection of next registers that are locally and asynchronously updated by each node. More concretely, the requester issues an inquiry message, and the message will be handled by each intermediate node in the following manner:

- If the next register of the current node points itself and those of all neighbors point to the current node, the current node is selected as the target node, and thus, it directly sends back its IP address to the requester.
- Otherwise, it forwards the received inquiry to a neighbor as follows:

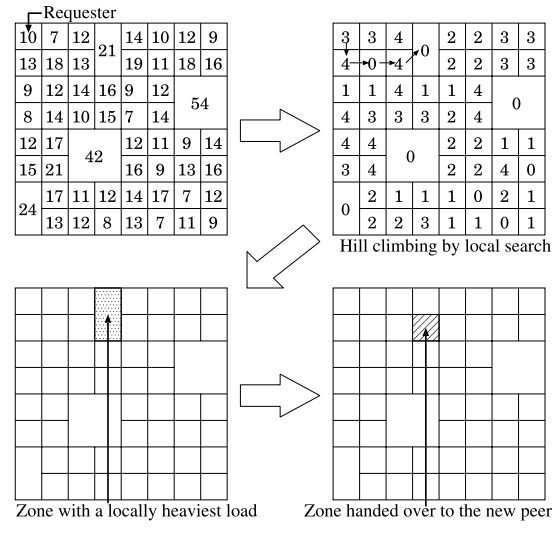


**Fig. 3** Overview of the Gradient Method.

- If the next register does not point itself, then it forwards the message to the node pointed by the register.
- Otherwise, it forwards the message to the node pointed by a neighbor node that does not point to the current node.

The time required for finding a target node under GR depends on the spatial distribution of heavily loaded nodes; i.e., it takes a constant time in the best case, and it takes a time proportional to the diameter of CAN in the worst case.

### 3.3 Distributed Heap Method

The third method, called Distributed Heap method (DH), is based on a distributed realization of *self-adjusting heaps*, that is well known as an efficient data structure to find an item with a maximum key in an ordered set. In particular, we will adopt *top-down skew heap* [9] as the underlying data structure for DH because of its simplicity [☆]. Recall that the design of DH is intended to realize a quick locating of a node with a highest access frequency, with a low communication cost.

Top-down skew heap is a data structure that supports the following six operations [9]:

- Makeheap($h$): Create a new, empty heap, named $h$.
- Insert($i, h$): Insert item $i$ into heap $h$, not previously containing $i$.
- Deletemax($h$): Delete and return an item of maximum key from heap $h$; if $h$ is empty return **null**.
- Findmax($h$): Return but do not delete an item of maximum key from heap $h$; if $h$ is empty return **null**.
- Delete($i, h$): Delete item $i$ from heap $h$.
- Meld($h_1, h_2$): Return the heap formed by taking the union of disjoint heaps $h_1$ and $h_2$. This operation destroys $h_1$ and $h_2$.

In the application to our load balancing problem, we may associate each item in the heap to a node in CAN and a key attached to the item to the access frequency of the corresponding node (recall that we are trying to find a node with a highest access frequency). Since the top-down skew heap is an out-tree with out-degree at most two, the (static) structure of the heap could be relatively easily realized in a distributed manner by preparing two local registers, say left and right, to store the pointer to the left and the right children in the tree, respectively.

Let $U$ be the set of root vertices of the current heaps. Since the above six operations always start from the root of a heap, in DH, we prepare another data structure to maintain set $U$ in a distributed manner, that has been proposed by the authors in Ref. 4). Note that dynamic set is a data structure that supports the following three operations; i.e., Find that finds a node in set $U$, Insert that inserts the caller to set $U$, and Delete that deletes the caller from set $U$. In Ref. 4), we proposed a distributed algorithm to realize such a dynamic set, that is based on the technique of path compression and lazy evaluation, and showed that it completes each of the above three operations in $O(\log N)$ time in an amortized sense, where $N$ is the number of nodes in the system.

By using the above two data structures, the load balancing under DH proceeds as follows:

- When a node newly joins the CAN system, it first inserts itself to the data structure for the dynamic set by setting the next pointer of the node to any node contained in the data structure [4]. Note that this step can be omitted if the node has been in the system in the past.
- It then executes Find to locate the root of a heap; and executes Deletemax to locate and delete a node with a heaviest load in the heap (see **Fig. 4** for illustration).
- After being associated with a split zone, the new and the located nodes execute Insert to insert themselves to the heap, with a key corresponding to the access frequency of their newly associated zone.
- The node located by the requester (i.e., the node with a heaviest load) executes Delete to delete itself from $U$, and a node who becomes the new root of heap $h$ executes Insert to insert itself to $U$.
- Finally, when a node leaves the CAN, it deletes itself from the heap by executing Delete.

[☆] We could not find any other data structure suitable to our purpose in the literature; e.g., $d$-heap, that is based on the balanced $d$-ary trees, fixes the number of items in advance, and does not support "melding" of two disjoint heaps, that will be required in distributed environments since links in the heap are expected to be disconnected frequently due to unexpected faults of nodes and communication links. On the other hand, in Fibonacci heap and the other variants of binomial heaps, we must keep an "unbounded" number of children in each vertex, that is very expensive in general.

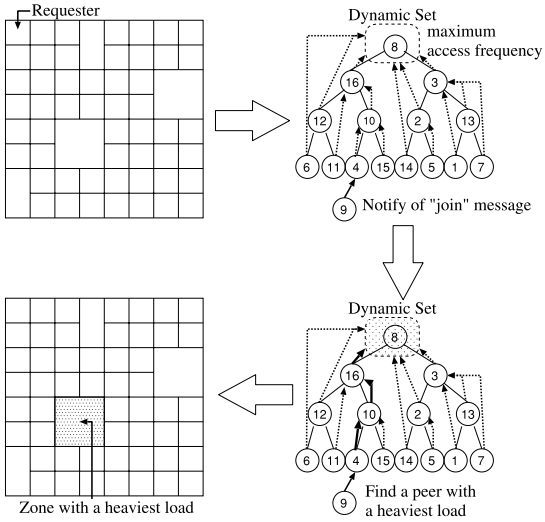


**Fig. 4** Overview of the Distributed Heap method.

As was mentioned previously, an amortized cost for finding a node in the dynamic set could be bounded as $O(\log N)$ [4]; in addition, we can complete each heap operation in $O(\log N)$ time in an amortized sense [9], as well. Hence, in DH, the locating and the splitting of a zone with a heaviest load takes $O(\log N)$ time, in an amortized sense. In the following, in order to reduce the maintenance cost, we suppose that each node *flushes* the change of its load status for every $T$ seconds (i.e., each node repeats deletion and insertion to the heap for every $T$ seconds), and denote the resulting scheme as DH($T$).

## 4. Simulation

In this section, we evaluate the performance of the proposed schemes by simulation. In particular, we measure the response time required for accessing the index of a requested object, that will be referred to as the **access time** hereafter, and the overall communication cost including the maintenance cost. Note that the access time is the summation of the turn around communication time to the target node maintaining the requested index, and the transaction time of the inquiry, including the waiting time in the ready queue.

Recall that for the first two schemes MD and GR, there is an apparent trade-off between the accuracy and the cost for acquiring an appropriate load information, and the proposal of the third scheme DH was intended to overcome such a trade-off point. The main objective of simulation is to examine the impact of such a trade-off to the access time, and to verify that the communication cost for DH could be bounded by a small value compared with other schemes.

### 4.1 Simulation Environment

Let $V = \{v_1, v_2, \ldots, v_N\}$ be the set of nodes to be considered in the simulation. In this paper, we consider a situation in which: 1) the CAN system initially contains a single node in $V$ called *anchor*, and 2) the other nodes in $V$ repeat JOIN and LEAVE operations in a dynamic and concurrent manner, where LEAVE is an operation executed when a node wants to leave CAN. In addition, each node participating with CAN repeatedly tries to access an object stored in the system, and issues an inquiry message to find the index of the object accordingly.

More concretely, a node who is not contained in CAN issues a JOIN message according to a Poisson distribution with mean $\lambda_{join}$, that will be varied from 50 to 250 seconds, and the time duration before issuing a LEAVE message follows an exponential distribution with mean $\lambda_{hold}$, that will be fixed to 200 seconds in the simulation. Note that it corresponds to a situation in which each node issues JOIN and LEAVE operations very frequently (i.e., zone partitioning takes place frequently), and it intends to clarify the difference of maintenance costs incurred by the node locating schemes. In addition to that, each node who is participating with the CAN system issues an inquiry message according to a Poisson distribution with mean $\lambda_{access}$, that is fixed to 3 seconds to realize a heavily loaded situation.

We suppose that each node in $V$ stores individual objects. The number of objects held by a node is assumed to follow a normal distribution with mean 20 and distribution 2. Each object stored in a node is associated with a key, and the association (i.e., the binding of a name to the object) is carried out only when the holder of the object issues its first JOIN message. Keys and their access probability are determined as follows. First, we randomly extract 100,000 words from English dictionary ☆. Let $D = \{w_1, w_2, \ldots, w_{|D|}\}$ be the set of extracted words. We then associate an access probability to each word in $D$, according to Zipf's first law [12]; i.e., the $i^{th}$ word $w_i$ in $D$ is associated with an access probability $p_i = 1/(i \times Q)$, where $Q \stackrel{\text{def}}{=} \sum_{i=1}^{|D|}(1/i)$. In the experiments, we

☆ We used an online version of Webster's Second International containing 234,936 words.

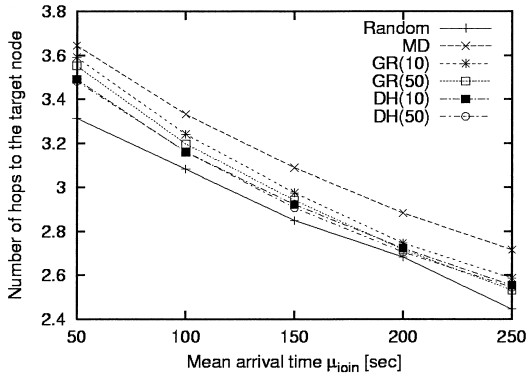


**Fig. 5** Average access time of an object under each scheme ($|V| = 64$, $\lambda_{hold} = 200$, and $\lambda_{access} = 3$).

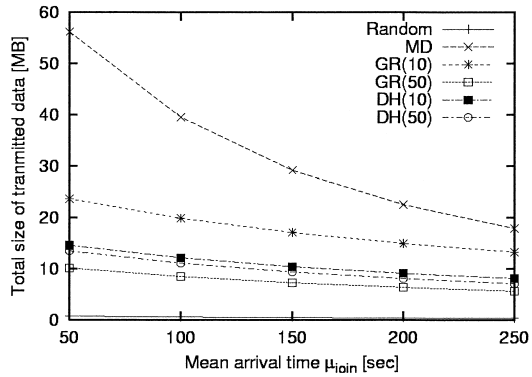


**Fig. 6** Distribution of access frequency under each scheme ($|V| = 64$, $\lambda_{hold} = 200$, and $\lambda_{access} = 3$).

used an arbitrary order of words in set $D$. Note that $\sum_{i=1}^{|D|} p_i = 1$ holds by definition.

We use the same hash function with Section 2.2, where the range of the function is fixed to $256 \times 256$. In addition, to clarify the difference of node locating schemes, we assume that the message transmission between any pair of nodes takes 1 second, and the transaction of an inquiry takes 1 second on any node. The proposal and evaluation of a zone partitioning scheme taking into account the heterogeneity of the underlying physical network is left as a future problem.

### 4.2 Effect of Load Balancing

In this subsection, we evaluate the performance of the proposed schemes by varying the frequency of JOIN messages. More concretely, we measure the access time, load distribution, and the overall communication cost by varying mean arrival time $\lambda_{join}$ from 50 to 250 seconds. The other parameters are fixed as $|V| = 64$, $\lambda_{hold} = 200$ [sec] and $\lambda_{access} = 3$ [sec].

#### 4.2.1 Access Time

**Figure 5** summarizes the average access time under each scheme. As is shown in the figure, the average access time monotonically decreases as increasing the mean arrival time $\lambda_{join}$, and the access time under Random could be reduced to about 75% by using the proposed schemes; i.e., we can really observe the effect of load balancing by the schemes. In addition, MD exhibits a slightly worse performance than the other two schemes, and the difference of the maintenance interval $T$ in GR and DH does not significantly affect the average access time.

#### 4.2.2 Load Distribution

Clearly, a main reason of the above phenomena is due to the effect of the load balancing. See **Fig. 6**. This figure illustrates the distribution of access frequency when $\lambda_{join}$ is fixed to 150, where the horizontal axis represents the number of accesses and the vertical axis represents the number of zones that receives the given number of accesses during the simulation. As is shown in the figure, a broad distribution in Random can really be concentrated to a narrow area by taking into account the load balancing; e.g., by using DH(50) instead of Random, the standard deviation of the distribution reduces from 22.94 to 8.08, and in addition, the ratio of the maximum to the minimum number of accesses reduces from 176 (=176/1) to 2.76 (= 80/29). However, we could not find a significant difference among proposed schemes from the figure at least in the sense of the access distribution, that would require another reason for the explanation of the phenomena shown in Fig. 5.

#### 4.2.3 Hop Counts

Another possible explanation could be that by the number of hops that must be traversed before an inquiry reaching the target node. **Figure 7** compares the average number of hops under each scheme. As is shown in the figure, the number of hops under MD is greater than that under the other methods, and it well explains the increase of the access time under MD. In MD, a node with a heaviest load will always be selected as the zone to be split. Hence, a subregion with higher access frequency will be partitioned into smaller zones, whereas a subregion with lower access frequency will be partitioned into larger zones (recall that there is a non-negligible imbalance on the spatial locality of mapped points as is shown in Fig. 2). In

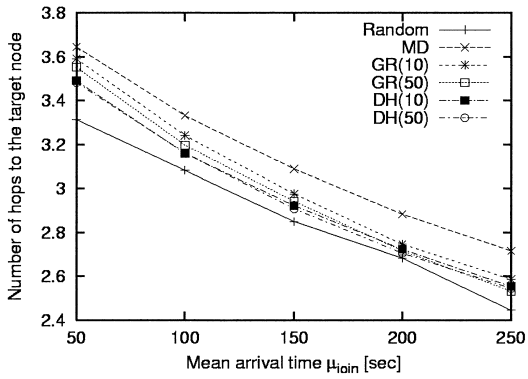


**Fig. 7** Average number of hops to the target node.

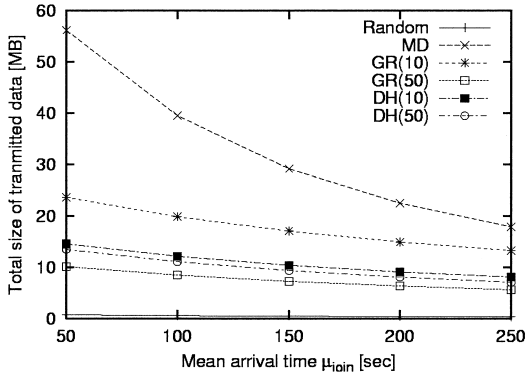


**Fig. 8** The total size of the transmitted data.

addition, since the actual splitting of the selected zone can take place only after collecting the load information from the all nodes in the system under MD, there is a very long time lag before reflecting the decision to the global state; i.e., a heavily loaded node will be split by many requesters that would eventually generate many zones with unnecessarily small sizes. Such an unnecessary imbalance of zone sizes will significantly increase the average access time, since we are assuming that the partitioning is conducted based on the access frequency, and a region split into many small zones will be accessed with a high probability. It should be worth noting that such an imbalance will be relaxed under the other two schemes GR and DH, that is probably due to the low communication cost to find a target node to be split into subzones.

#### 4.2.4 Communication Cost

Finally, we evaluate the overall communication cost under each scheme. **Figure 8** summarizes the result. The horizontal axis of the figure represents parameter $\lambda_{join}$ and the vertical axis represents the total size of the exchanged messages. Note that since the length of each message could be bounded by a fixed value, the figure indicates the difference of exchanged number of messages, as well.

From the figure, we can observe that the communication cost for Random is bounded by a very small value, that is clearly because of the low maintenance cost of the scheme. In contrast, MD requires much more communications compared with the other methods (more than two times when $\lambda_{join} = 50$). The communication cost for GR and DH depends on the maintenance interval, the cost for GR reduces to a half by increasing the interval from 10 seconds to 50 seconds, whereas the cost for DH reduces to 95% by a similar reduction. This phenomenon implies that the maintenance cost of DH is relatively small compared with GR; i.e., the cost for inquiry and zone partitioning dominates the overall cost of the scheme.

## 5. Concluding Remarks

In this paper, we studied the zone partitioning problem in the Content-Addressable Network, and proposed three schemes MD, GR, and DH for solving the problem. The performance of the schemes is evaluated by simulation. The result of simulations implies that, when the number of peers is 64, GR and DH exhibit almost the same performance, and by using those schemes, we could reduce the access time of the original scheme to less than 75%. In addition, although we could not demonstrate the superiority of DH to GR, we could verify that the communication cost under DH is bounded as small as that under GR.

The access time of indices in CAN depends on several factors in addition to the accuracy of load information that has been considered in this paper. Such factors include the heterogeneity of the underlying physical network, routing protocols, the performance of local caching protocols, and so on. Hence, as a future work, we should extend the proposed schemes to include those factors, after examining the effect of the schemes under more realistic environments, such as P2P systems consisting of more than 10,000 nodes and physical networks with a heterogeneous structure.

**Acknowledgments** This research was partially supported by Grant-in-Aid for Scientific Research (C) 13680417, and Priority Areas (B)(2) 14085204.

## References


1) Aberer, K.: P-Grid: A Self-Organizing Access Structure for P2P Information Systems, *Proc. 6th Int'l Conf. on CoopIS 2001*, pp.179-194 (Sept. 2001).
2) Gnutella. http://gnutella.wego.com/
3) Clarke, I., Sandberg, O., Wiley, B. and Hong, T.W.: Freenet: A Distributed Anonymous Information Storage and Retrieval System, *ICSI Workshop on Design Issues in Anonymity and Unobsevability*, pp.46–66 (July 2000).
4) Fujita, S. and Yamashita, M.: Maintaining a Dynamic Set of Processors in a Distributed System, Distributed Algorithms, *Proc. 10th International Workshop, WDAG '96*, LNCS, Vol.1151, pp.220–233 (1996).
5) Gong, L.: ProjectJXTA: A Technology Overview, Sun Microsystems Inc (Apr. 2001).
6) U.S. Department of Commerce. Secure Hash Standard. FIPS PUB 180-1, 17 (Apr. 1995).
7) Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S.: A Scalable Content-Addressable Network, *Proc. SIGCOMM 2001*, pp.161–172 (Aug. 2001).
8) Ratnasamy, S., Handley, M., Karp, R. and Shenker, S.: Application-level multicast using content-addressable networks, *Proc. 3rd Int'l Workshop on Networked Group Communication*, pp.14–29 (Nov. 2001).
9) Sleator, D.D. and Tarjan, R.E.: Self adjusting heaps, *SIAM J. on Computing*, Vol.15, No.1, pp.52–69 (Feb. 1986).
10) Stoica, I., Morris, R., Karger, D., Kaashoek, F. and Balakrishnan, H.: Chord : A Scalable Peer-to-peer Lookup Service for Internet Applications, *Proc. SIGCOMM 2001*, pp.149–160 (Aug. 2001).
11) Zhao, B., Kubiatowicz, J. and Joseph, A.: Tapestry : An Infrastructure for Fault-tolerant Wide-area Location and Routing, Technical Report, UCB/CSD-01-1141 (2000).
12) Zipf, G.K.: *Human Behavior and Principle of Least Effort*, Addison-Wesley, Boston (1949).


(Received May 13, 2004)
(Accepted November 1, 2004)
(Online version of this article can be found in the IPSJ Digital Courier, Vol.1, pp.54–62.)

**Daisuke Takemoto** received the B. Eng. degree from Hiroshima University, Japan, in 2003. He is currently working towards the M.E. degree in Hiroshima University. His current research interests include peer-to-peer system and network architecture.

**Shigeaki Tagashira** received the B. Eng. degree from Ryukoku University, Japan, in 1996; and the M. Eng. and D. Eng. degrees in Information Science from Nara Institute of Science and Technology (NAIST), Japan, in 1998 and 2000, respectively. Since 2000 he has been a research associate of Information Science at Hiroshima University, Japan. His current research interests include mobile computing and system software. He is a member of Institute of Electrical and Electronics Engineers Computer Society (IEEE CS).

**Satoshi Fujita** received the B.E. degree in electrical engineering, M.E. degree in systems engineering, and Dr.E. degree in information engineering from Hiroshima University in 1985, 1987, and 1990, respectively. He is an Associate Professor at Graduate School of Engineering, Hiroshima University. His research interests include communication algorithms on interconnection networks, parallel algorithms, graph algorithms, and parallel computer systems. He is a member of IEICE, SIAM Japan, IEEE Computer Society, and SIAM.