Distributed Arrays: A P2P Data Structure for Efficient Logical Arrays

Daisuke Fukuchi*, Christian Sommer*, Yuichi Sei* and Shinichi Honiden†*
*The University of Tokyo, Tokyo, Japan, †National Institute of Informatics, Tokyo, Japan
Email: {daisuke1, sommer, sei, honiden}@nii.ac.jp

***Abstract*—Distributed hash tables (DHT) are used for data management in P2P environments. However, since most hash functions ignore relations between items, DHTs are not efficient for operations on related items. In this paper, we modify a DHT into a distributed array (DA) that enables efficient operations on logical arrays. The array elements of a DA are placed in a P2P overlay network according to a simple rule such that the load is balanced and the number of messages required to access elements sequentially is reduced. The number of messages required for array operations is much smaller than that for operations on DHTs. We demonstrate this theoretically and experimentally.**

I. Introduction

Peer-to-peer (P2P) technologies are useful for constructing large systems. Since P2P systems provide functions by coordinating large numbers of nodes, they do not have to use central servers and they do not require nodes to persist. This allows for large systems. For example, some P2P file sharing systems are run by millions of users.

Distributed hash tables (DHT) [1]–[3] enable P2P systems to be used as data management systems. DHTs provide the basic functions of hash tables. Each item is represented by a (key, value) pair and is registered, referred, and deleted by using that key. The number of required messages for an operation is only $O(\log n)$, where n denotes the number of nodes (Table I).

DHTs do not consider relations between registered items. Therefore, after registering related items, executing an operation on some of those items results in a set of individual accesses with a high message cost. For example, let us assume that, out of a set of files, we want to find the first file matching certain given conditions. In this case, we access the first file and check whether it matches. If it does not, we access the next file. This process is repeated until a matching file is found. The number of messages required for each access is the same as that to access the target file from scratch. Therefore, if the first matching file is the w-th file, it requires $O(w \log n)$ messages.

TABLE I
Notation.

n	Number of nodes.
b	Number of bits for indices and IDs.
f_{r}	b-bit reverse bit order mapping.
$d(x, y)$	Distance from ID x to ID y on a circular ID space in the ascending direction.
$\gamma(x)$	Number of 1's in the upper $\log n$ bits of the b-bit binary expression of x if n is a power of 2. γ of x means the same.

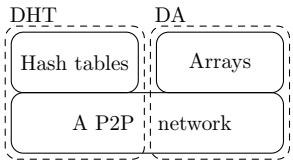


Fig. 1. Enhancement of P2P data management systems.

This paper proposes using a distributed array (DA) to support efficient operations on items stored in a logical array in a P2P environment. In the above example, by storing the sequence of files in an array, the first matching file is found in $O(w + \log n)$ messages instead of $O(w \log n)$ messages. Splitting data is another situation in which array operations are useful. In DHTs, each item is assigned to one node. Large data beyond the capacity of a single node can not be managed as one item. As such, the data has to be split into multiple items. One of the simplest data splitting methods is cutting and sequentially indexing from the front. Such indexed items are considered to be elements of a logical array. Any operation on the data can be achieved by performing array operations. For example, assume that in a file sharing system we want to retrieve information from a large log and that the log is split into parts and new information is appended to the last file. As soon as the size threshold for one file is reached, a new log file is added. Since a large number of shared files causes proportionally large loads for the retrieval process, the whole log can not be managed by one node. Therefore, we ensure that the new entry is added to the latest log and if the size of the log exceeds a threshold, a new log is added. The log as a whole is a logical array. The operation of adding an entry entails an access to the tail of the array and the addition of a new element at the tail. Retrieval is achieved by accessing all elements. Querying a specific period is done by searching for corresponding elements and accessing them.

The base P2P network of the DA can be simultaneously used for making the DHT without any additional stabilization cost. This enables DHT-based systems to select a hash table and an efficient array for each item at the same time (Fig. 1).

II. Related Work

To the best of our knowledge, there are no previous studies on efficient array operations in P2P environments. DHTs can use logical arrays if we do not have to consider the efficiency

of the array operations. Studies on range queries are relevant for making some efficient array operations.

There are a number of DHTs based on consistent hashing [4] and the distributed data location protocol developed by Plaxton et al. [5], e.g., Chord [3], CAN [1], and Pastry [2]. The P2P network of a DHT forms a logical ID space and enables stable access to a target ID in $O(\log n)$ messages (for details, see Section III). DHTs are created by computing the hash value of the key and interpreting it as the ID of a (key, value) pair.

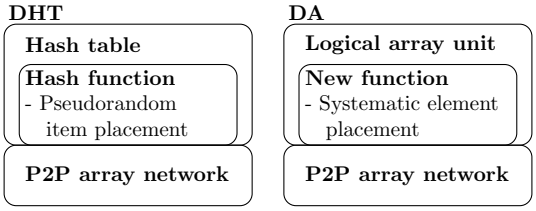
In a DHT, multiple arrays can be managed by considering the concatenation of an array name and an index to be a key. Any array operation can be achieved by manipulating elements one by one. Unfortunately, the resulting message cost is large. Access to an element is independent of other accesses because the hash function pseudo-randomly places array elements in the ID space. Accessing w target elements requires $O(w \log n)$ messages. Specific operations can be optimized. However, such optimizations are rather complex and their result is probably not efficient on top of the pseudo-random element placement. For example, assume that we want to manipulate all elements of an array. In this case, we can calculate the order to minimize the number of messages required to access all elements consecutively on the basis of the analysis in Section III-B. However, this calculation is computationally hard (by a reduction to the traveling salesman problem [6]), and even the optimal solution does not guarantee efficiency.

Studies on range queries in P2P environments focus on related items and can be applied to array operations. A range query searches for items whose target-attribute value is in a given range. A range query achieves efficient array operations like sequential access by considering an array name to be an attribute and an index to be an attribute value.

There are two kinds of P2P range query systems: placing items in a P2P network to preserve orders or differences between attribute values [7], [8] and adding management structures like a linked list to items [9]–[11]. Both approaches face problems when actual arrays are necessary.

In placing elements in a P2P network to preserve the orders or differences between indices, not a few elements are concentrated in a small area because array indices tend to be used sequentially. This causes load concentration. This is especially problematic for splitting large data into an array. Although load balancing by moving nodes or items is supported, it is not only costly; it has bad affinity to single-element operations. For example, moving light-load nodes to the neighborhood of heavy-load nodes increases the density of nodes as well as the density of elements. This increases the virtual number of nodes in a place. The number of required messages to access one of the elements increases in accordance with that virtual number.

Adding management structures to elements does not affect operations that are not conducted within these structures. For example, pointers composing PHT [11] connect elements whose indices are adjacent. Therefore, a search in a sorted array is executed by following pointers one by one or as a binary search using the DHT's access function. This is not efficient. Moreover, the DHT's access function is used to complement operations when the pointers are invalid. Therefore, performance greatly decreases when nodes join and leave frequently. Although adding generic and redundant structures like skip graphs [9] can alleviate such problems, stabilization costs and system complexity increase.



(a) Composition of the DHT. (b) Composition of the DA.

Fig. 2. P2P data structures.

The reverse bit order mapping described in Section IV-A was used before by Shao et al. [12] and in split-ordered lists [13] for other purposes than the one described here. Shao et al. reversed the bit order in converting between logical addresses and physical addresses to improve memory performance. In split-ordered lists, to construct a lock-free hash table from a lock-free linked list, each entry of a hash table is made to point to an element of a linked list at the position made from the entry's index by reversing the bit order.

III. Analysis of Distributed Hash Tables

In this section, we analyze DHTs in order to construct a DA from a DHT in Section IV.

A DHT can be divided into two layers (Fig. 2(a)). The upper layer is a hash table implemented on an array. The lower layer is an important P2P technology that we will refer to as **P2P array network**. P2P array networks have the following properties.

- Any node can access any ID in $[0, 2^b)$ in $O(\log n)$ messages. b is the number of bits of IDs and a constant parameter in the system (Table I).
- They support node joining and leaving.

We will not focus on node joining and leaving in the following. In Section VI, we experimentally demonstrate that the DA is efficient in dynamic environments as well.

A DHT uses a hash table for item placement in its ID space. Its hash function, for example SHA1 [14], entails a pseudorandom element placement that inhibits efficient array operations. Therefore, one approach to enable efficient array operations is to change the element placement scheme. Fig. 2(b) illustrates the composition of a DA in such a case. To obtain an efficient element placement scheme, we will analyze the Chord [3] network as the base P2P array network.

A. Chord network

The Chord network is a simple and useful P2P array network. It is constructed as follows:

- It uses a circular ID space $[0, 2^b)$. 0 follows after $2^b - 1$. If $x > y$, the ID range $[x, y]$ denotes $[x, 2^b) \cup [0, y]$. If

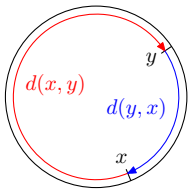


Fig. 3. Illustration of distances between ID x and ID y.

$x \geq 2^b$, ID x denotes $x \bmod 2^b$. Let $d(x, y)$ denote the *distance* from ID x to ID y in the ascending direction (clockwise in Fig. 3).

$$d(x, y) = y - x \pmod{2^b}$$

- Nodes are placed in the ID space by using a hash function. The node placed at ID x is denoted by x. For all IDs x, successor(x) denotes the first node placed at x or after x. predecessor(x) denotes the first node placed before x.
- A node x is responsible for the segment $[x, \text{successor}(x+1))$ of the ID space. Note that this is different from the original Chord where node x is responsible for the segment $(\text{predecessor}(x), x]$. The original Chord finds the successor of a target ID, whereas we want to operate an item placed at the ID.
- A node x has neighbor pointers to predecessor(x) and successor($x + 1$).
- A node x has pointers to successor($x + 2^k$) ($\forall k < b$). Such a pointer is called a *finger*, and the set of all fingers of a node is called its *finger table*.

ID access in a Chord network is conducted as follows:

- Let x denote the node executing the access process, and let y denote the target ID.

1. If $y \in [x, \text{successor}(x + 1))$, x is responsible for y. The access process stops at x.
2. Otherwise, x selects the finger x' with the minimum distance to y. The access process is then transferred to x'.

- Repeat these steps until the access process stops.

B. Analysis of Chord network

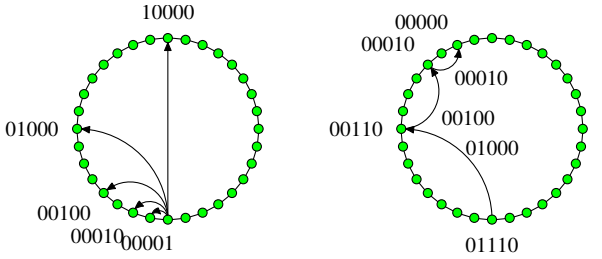
To better understand the access procedure of the Chord network, assume that n is a power of 2, i.e., $n = 2^m$ ($m \leq b$), and nodes are placed in the ID space at equal ID intervals. This is an ideal Chord network. We cope with non-ideal environments in Section IV-B and show the coping effects experimentally in Section VI (Fig. 6). For each node x, successor($x + 1$) is $x + 2^{b-m}$ and x has the following finger table (Fig. 4(a)):

$$\{x + \underbrace{0\ldots01}_{k}\underbrace{0\ldots0}_{b-k}{}_{(2)} \mid 1 \leq k \leq m\}$$

The notation $z_{(2)}$ emphasizes that z is a binary expression. In the following, we implicitly use a b-bit binary expression and call it a *bit sequence*.

The access starting at node x_0 is executed as follows:

- x_i denotes a node that is executing the access process, and y denotes the target ID.



(a) Finger table. A value denotes the bit sequence of the distance to a finger.

(b) Routing of ID access. An external value denotes the bit sequence of the distance to the target ID. An internal value denotes the bit sequence of the distance to a used finger.

Fig. 4. Ideal Chord network in $b = 5$ and $n = 2^5$.

1. If all the upper m bits of $d(x_i, y)$ are 0, then $y \in [x_i, \text{successor}(x_i + 1))$ and x_i is responsible for y. The access process stops at x_i.
2. x_i finds $k \leq m$ s.t.

$$d(x_i, y) = \underbrace{0\ldots01}_{k} z_{k+1} \ldots z_{b(2)}$$

and selects the finger x_{i+1} with the minimal distance to y from x_i's finger table.

$$x_{i+1} = x_i + \underbrace{0\ldots01}_{k}\underbrace{0\ldots0}_{b-k}{}_{(2)}$$

The access process is transferred to x_{i+1}.

- Repeat these steps until the access process stops.

In step 2, the distance from the current executing node to the target ID changes from

$$d(x_i, y) = \underbrace{0\ldots01}_{k} z_{k+1} \ldots z_{b(2)}$$

to

$$d(x_{i+1}, y) = \underbrace{0\ldots00}_{k} z_{k+1} \ldots z_{b(2)} \ .$$

Step 2 eliminates the highest 1 in the bit sequence.

The access procedure in the ideal Chord network eliminates 1's in the bit sequence of $d(x_0, y)$ from above (Fig. 4(b)). Let γ_0 denote the number of 1's in the upper $\log n (= m)$ bits of $d(x_0, y)$. After step 2 has been executed γ_0 times, all the 1's in the upper $\log n$ bits of $d(x_0, y)$ are cancelled out and step 1 stops the access process at x_{γ_0}. Since $x_0, x_1, \ldots, x_{\gamma_0}$ are visited, the number of required messages is γ_0. If y is arbitrary, $d(x_0, y)$ can be an arbitrary bit pattern. Therefore, the average γ_0 is

$$\log n/2 \ . \tag{1}$$

This is the average number of required messages for an ID access in the ideal Chord network.

IV. Construction of a Distributed Array

Based on the analysis in Section III-B, we can construct a DA by introducing an element placement rule in order to execute efficient array operations. We place each element at an ID made from its index by reversing the bit order (Section IV-A). We can then cope with the discrepancy between the assumptions in the analysis and real environments by tuning the topology of the Chord network (Section IV-B).

A. Reverse bit order mapping

A placement rule for array elements is represented by a mapping f from an index space of arrays to the ID space of a P2P array network. An element indexed by x is placed in ID $f(x)$. To manage multiple arrays, an element indexed by x is mapped to $h + f(x)$ by using the hash value h of an array name. In the following, we consider only one array and omit h for the sake of readability.

An appropriate mapping has to satisfy the following requirements. First, there is a certain locality assumption. Namely, elements indexed by numbers close to each other tend to be operated on at the same time. Moreover, there are three requirements: (**1**) We want to execute array operations efficiently; (**2**) we want to balance the load among elements in order to use arrays for splitting data; (**3**) we need a simple mapping rule to optimize other operations.

The analysis in Section III-B indicates that it is effective to reduce the number of 1's in the upper $\log n$ bits of the distances between IDs at which target elements are placed. Let $\gamma : [0, 2^b) \to [0, b]$ be a function from an ID to an integer in $[0, b]$ as follows (Table I):

$$\gamma(x_1 \dots x_{b(2)}) := \sum_{k=1}^{\log n} x_k \quad (n \text{ is a power of } 2)$$

Then, by operating on target elements consecutively, a whole operation requires only a small number of messages. To enable such a placement, the following property is useful.

Property 1. *If ID x and y have the same lower $b - k$ bits ($k \le b$), $d(x, y)$ has no 1's in the lower $b - k$ bits.*

That is, by mapping $2^k (k \le b)$ or fewer indices to IDs that have the same lower $b-k$ bits, the 1's in the bit sequences of the distances between the IDs are limited to only the upper k bits.

The locality assumption implies that indices for which only lower bits are different tend to be manipulated at the same time. Since these elements tend to be operation targets at the same time, for load balancing, they should be assigned to different nodes. It is effective to place them at distant IDs, that is IDs with different upper bits. Furthermore, from Property 1, IDs having the same lower bits will satisfy Requirement 1. Therefore, the desired mapping should map indices with the same upper bits and different lower bits to IDs with different upper bits and the same lower bits. In accordance with Requirement 3, we define the reverse bit order mapping f_{r} as follows:

Definition 1 (Reverse bit order mapping f_{r})**.** *f_r is a mapping from index space $[0, 2^b)$ to ID space $[0, 2^b)$. It is defined by*

$$f_r(x_b \dots x_{1(2)}) := x_1 \dots x_{b(2)} \ .$$

In the rest of this section, we determine whether f_{r} satisfies Requirements 1 and 2.

Let $A_{k,x}$ denote a sequence of indices as follows.

$$A_{k,x} := [x2^k, (x+1)2^k) \quad (k < b \text{ and } x < 2^{b-k}) \tag{2}$$

Theorem 1. *For different indices $y \neq z$, the average value of $\gamma(f_r(y), f_r(z))$ is*

$$\begin{array}{ll} (k+1)/2 & (\textit{if } k < \log n) \\ \log n - (k-1)/2 & (\textit{if } \log n \le k < \log n + 1) \\ \log n/2 & (\textit{if } \log n + 1 \le k) \ , \end{array} \tag{3}$$

where k is the minimum k s.t. $y, z \in A_{k,x}$.

Proof: Let k' be the minimum k s.t. $y, z \in A_{k,x}$ and let x' be x s.t $y, z \in A_{k',x}$. Since $A_{k,x} = A_{k-1,2x} \cup A_{k-1,2x+1}$, y and z belong to $A_{k'-1,2x'}$ or $A_{k'-1,2x'+1}$. If both y and z belong to the same interval $A_{k'-1,2x'}$ or $A_{k'-1,2x'+1}$, it contradicts the assumption of k' being minimal. Therefore, y and z belong to different intervals $A_{k'-1,2x'}$ and $A_{k'-1,2x'+1}$. Without loss of generality, $y \in A_{k'-1,2x'} \wedge z \in A_{k'-1,2x'+1}$. Let $x_b \dots x_{k'+1(2)}$ denote the $(b-k')$-bit binary expression of x'. We can write

$$\begin{aligned} y &= x_b \dots x_{k'+1} \underbrace{0 * \cdots *}_{k'}{}_{(2)} \\ z &= x_b \dots x_{k'+1} \underbrace{1 * \cdots *}_{k'}{}_{(2)} \ . \end{aligned}$$

Since the lower $b - k'$ bits of $f_{\mathrm{r}}(y)$ and $f_{\mathrm{r}}(z)$ are the same, Property 1 ensures that the lower $b-k'$ bits of $d(f_{\mathrm{r}}(y), f_{\mathrm{r}}(z))$ are 0. Furthermore, since the lower $(b - k' + 1)$-th bits of $f_{\mathrm{r}}(y)$ and $f_{\mathrm{r}}(z)$ are different, the lower $(b - k' + 1)$-th bit of $d(f_{\mathrm{r}}(y), f_{\mathrm{r}}(z))$ are 1. Since the upper $k' - 1$ bits of $f_{\mathrm{r}}(y)$ and $f_{\mathrm{r}}(z)$ are arbitrary, the upper $k' - 1$ bits of $d(f_{\mathrm{r}}(y), f_{\mathrm{r}}(z))$ can be arbitrary. Therefore, the average γ of the distance between $f_{\mathrm{r}}(y)$ and $f_{\mathrm{r}}(z)$ is as follows. If $k' < \log n$, it is

$$(k' - 1)/2 + 1 = (k' + 1)/2.$$

If $\log n \le k' < \log n + 1$, it is

$$(k' - 1)/2 + \log n - (k' - 1) = \log n - (k' - 1)/2.$$

If $\log n + 1 < k'$, it is $\log n/2$. ■

Theorem 1 ensures that we can complete a whole operation in a small number of messages by selecting indices of consecutively accessed elements from as small $A_{k,x}$ as possible.

Theorem 2. *f_r in Definition 1 maps a set of indices $A_{k,x}$ to a set of IDs at equal ID intervals.*

Proof: The set of indices in $A_{k,x}$ is represented by

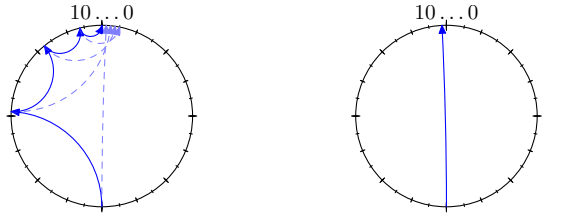
$$[x_b \dots x_{k+1} \underbrace{0 \dots 0}_{k}{}_{(2)}, x_b \dots x_{k+1} \underbrace{1 \dots 1}_{k}{}_{(2)}] \ .$$

All indices in this set have the same upper $b - k$ bits and include all 2^k bit patterns in the lower k bits. Therefore, all IDs mapped from the indices by f_{r} include all 2^k bit patterns in the upper k bits and have the same lower $b - k$ bits. After sorting the IDs, the intervals between them are equal:

$$\underbrace{0 \dots 01}_{k} \underbrace{0 \dots 0}_{b-k}{}_{(2)} \ .$$

■

Theorem 2 ensures that sequentially indexed elements are managed by different nodes among a large number of nodes. Thus, we can balance loads by splitting large data and storing the parts into elements of an array.



(a) Finger tables of successors. Since each finger represented by a dashed line points beyond the target ID, the previous finger, which points to about half the distance is used instead.

(b) Finger tables of predecessors. One transfer reaches the neighborhood of the target ID.

Fig. 5. Effect of components of finger tables on access to a 2^{b-1} distant ID.

B. Adaptation to realistic environments

In section IV-A, we assumed that n is a power of 2 and nodes are placed at equal ID intervals. However, this assumption hardly holds in real environments. The discrepancy between the ideal environment and a real one reduces the performance of a DA composed of a Chord network and reverse bit order mapping. In the following, we adapt the DA to non-ideal environments by tuning the topology of the Chord network by using predecessors instead of successors as fingers.

In Section III-B, we showed that the number of messages required in the ideal Chord network is the γ of the distance from a start node to the target ID. Theorem 1 shows that the reverse bit order mapping f_{r} makes the γ's of the distances between IDs at which indices of consecutively accessed elements are placed smaller than those in pseudorandom element placement. However, this effect does not obviously appear in real environments because of the difference between ideal fingers and real fingers. In real environments, since it is unlikely that n is a power of 2 and nodes are placed at equal ID intervals, a finger of x, successor($x+2^k$) ($k < b$), is usually placed at after $x + 2^k$. The average distance l from $x + 2^k$ to successor($x + 2^k$) is half the average node interval.

$$l = (2^b/n)/2 = 2^{b-\log n-1} = \underbrace{0\ldots0}_{\lceil\log n\rceil}\underbrace{1*\cdots*}_{b-\lceil\log n\rceil}{}_{(2)} \tag{3}$$

Here, let x denote the node executing an access process, and let y denote the target ID. Let us assume that

$$d(x, y) = 2^m \quad (m > b - \lceil\log n\rceil)\ . \tag{4}$$

The best transfer destination in x's finger table is

$$x' \approx x + 2^{m-1} + l\ .$$

The next distance from x' to the target ID is

$$d(x', y) \approx 2^{m-1} - l = \underbrace{0\ldots0}_{b-m}\ \underbrace{01\ldots1}_{m-b+\lceil\log n\rceil}\ \underbrace{1*\cdots*}_{b-\lceil\log n\rceil}{}_{(2)}\ .$$

Roughly

$$m - b + \lceil\log n\rceil - 1 \tag{5}$$

messages are required in order to set all the upper $\lceil\log n\rceil$ bits to 0. As described in Section III-B, the 1's in the bit sequence of the distance to the target ID are deleted from above by performing the access process. Therefore, a situation can occur in which only one 1 remains in the upper $\lceil\log n\rceil$ bits. This situation can be approximated as (4). The proof of Theorem 1 shows that even the last 1 is often an upper bit to operate on array elements consecutively. That is, $m \approx b$ in (4); hence, additional (5) $\approx \lceil\log n\rceil - 1$ messages are required to access a target element (Fig. 5(a)). This is inefficient.

This problem can be avoided by using predecessors instead of successors as fingers. In this approach, each finger of x points to predecessor($x + 2^k$) and the average distance from predecessor($x+2^k$) to $x+2^k$ is l. The parts of step 2 described in Section III-A and III-B have to be modified to check successor($x+1$) besids fingers in selecting transfer destinations of access processes. In the above situation, the best finger is

$$x' \approx x + 2^m - l\ .$$

The distance to the target ID is

$$d(x', y) \approx l\ .$$

Since the average node interval is $2l$, the access probably finishes at x' (Fig. 5(b)). The above shows that using predecessors instead of successors as fingers confers the advantages of Theorem 1 in realistic environments. This effect was experimentally evaluated (see Section VI and Fig. 7(a)).

V. Array Operations on a Distributed Array

We theoretically show that the DA enables efficient array operations. The measure of performance is average message complexity, i.e., the average number of required messages in order to access target elements. We use the ideal Chord network as an approximate environment for the sake of simplicity. When the need occurs, we will approximate the message complexity needed to access a target element from a node managing another element by the γ of the distance between IDs at which the elements are placed. The actual message complexity in realistic environments basically corresponds to this approximation (see Section VI). In the following, we show that the DA enables operations to be executed more efficiently than in the source DHT. We also give examples showing that more improvements are possible for specific operations.

First, let us consider the operations. Theorem 1 says that the average message complexity of the DA in order to access array elements consecutively follows (3), where k is the maximum k of $x2^{k-1}$ representing indices skipped by indices of the consecutively accessed elements. On the other hand, the average message complexity of the DHT is (1), $\log n/2$, because it does not relate to the element indices. The difference in average message complexities of the DA and DHT is

$$\begin{cases} (\log n - k - 1)/2 & (\text{if } k < \log n) \\ (k - \log n - 1)/2 & (\text{if } \log n \le k < \log n + 1) \\ 0 & (\text{if } \log n + 1 \le k)\ . \end{cases}$$

If $k < \log n - 1$, the smaller the k, the better the DA performs. If $\log n - 1 < k < \log n + 1$, although the DA performs worse than the DHT, the maximum difference is $1/2$, when $k = \log n$. That is, by selecting indices of consecutively accessed elements whose skipped indices are represented by $x2^{k-1}$ with as small k as possible, we need only a small number of messages to complete any operation. The DA performs worse than the DHT if maximum k of $x2^{k-1}$ representing indices skipped by the indices of consecutively accessed elements

frequently satisfies $\log n - 1 < k < \log n + 1$. However, if n is large enough, this situation rarely occurs. For example, if $n = 10000$, then disadvantageous k's for the DA would be 13 and 14. The indices that should not be skipped are multiples of 4096.

Next, we give examples of useful array operations – sequential access, and a search in a sorted array – to show that the DA can make these operations more efficient.

Sequential access means accessing all elements indexed by a given range in ascending order. It can be used for the example in Section I to find the first matching file in a sequence of files in an array.

DA can efficiently execute sequential accesses. For example, assume that $b = 5$ and the target range is $[7, 11]$. The distances between IDs at which consecutively accessed elements are placed are as follows:

Index		ID	Distance
$7 = 00111_{(2)}$		$11100_{(2)}$	$00110_{(2)}$
$8 = 01000_{(2)}$		$00010_{(2)}$	$10000_{(2)}$
$9 = 01001_{(2)}$	$\overset{f_r}{\mapsto}$	$10010_{(2)}$	$11000_{(2)}$
$10 = 01010_{(2)}$		$01010_{(2)}$	$10000_{(2)}$
$11 = 01011_{(2)}$		$11010_{(2)}$	

The number of 1's in the ID distances is 1 or 2. This property is analyzed in the following theorem.

Theorem 3. *The average value $\gamma(d(f_r(x), f_r(x+1)))$ is less than or equal to $3/2$.*

Proof: If x is even ($x \mod 2 \equiv 0$), that is, the target index changes from $x_b \dots x_2 0_{(2)}$ to $x_b \dots x_2 1_{(2)}$, the ID distance d is

$$d(0x_2 \dots x_{b(2)}, 1x_2 \dots x_{b(2)}) = \underbrace{10 \dots 0}_{b}{}_{(2)}$$

and $\gamma(d(f_r(x), f_r(x+1))) \le 1$.

Otherwise, if x is odd, for $k \ge 2$, one access out of 2^k accesses, if $x \mod 2^k \equiv 2^{k-1} - 1$, that is, the target index changes from

$$x_b \dots x_{k+1} \underbrace{01 \dots 1}_{k}{}_{(2)} \text{ to } x_b \dots x_{k+1} \underbrace{10 \dots 0}_{k}{}_{(2)} ,$$

the distance is

$$d(\underbrace{1 \dots 10}_{k} x_{k+1} \dots x_{b(2)}, \underbrace{0 \dots 01}_{k} x_{k+1} \dots x_{b(2)}) = \underbrace{0 \dots 011}_{k} \underbrace{0 \dots 0}_{b-k}{}_{(2)} .$$

and $\gamma(d(f_r(x), f_r(x+1))) \le 2$.

Therefore, the average $\gamma(d(f_r(x), f_r(x+1)))$ is less than or equal to $3/2$. ■

Theorem 3 gives the average message complexity in order to find the first matching file in a sequence of files in an array. If the first matching file is the w-th file, the average message complexity of the whole operation is less than or equal to

$$\log n/2 + (w-1)(3/2)$$

because the average message complexity to access the first file is $\log n/2$ and the complexities of the following accesses behave according to Theorem 3.

In sequential access in the DHT, the γ of the distances between IDs at which consecutively accessed elements are placed is $\log n/2$ because the element placement is pseudorandom.

A **search in a sorted array** searches a sorted array for an element equal to or closest to a given value. It can be used for the example in Section I to search a file entry log split into elements of an array for an element corresponding to a given date.

A simple and useful search in a sorted array is the binary search. It repeatedly selects the mean index in a search space as a pivot, accesses the pivot and reduces the search space. The first search space is determined by checking registered and unregistered elements managed by the start node.

However, the binary search is not so efficient in the DA. The following simple modification greatly reduces message complexity. Change the pivot selection as follows. If the search space is

$$[x_b \dots x_{k+1} 0 x_{k-1} \dots x_{1(2)}, x_b \dots x_{k+1} 1 x'_{k-1} \dots x'_{1(2)}] , \quad (6)$$

then the pivot to specify the lower k-th bit of the search space is

$$x_b \dots x_{k+1} \underbrace{10 \dots 0}_{k}{}_{(2)} . \quad (7)$$

For example, assume that $b = 5$ and the target value v is equal to the element indexed by 7 in an array sorted in ascending order. We want to detect the index 7. If the first search space is $[3, 14]$, then $[3, 14] = [00011_{(2)}, 01110_{(2)}]$ gives a pivot $01000_{(2)} = 8$. Since the element indexed by 8 is greater than v, the next search space is $[3, 7]$. Similarly, the subsequent pivots are as follows:

Search space	Pivot
$[3, 14] = [00011_{(2)}, 01110_{(2)}]$	$01000_{(2)} = 8$
$[3, 7] = [00011_{(2)}, 00111_{(2)}]$	$00100_{(2)} = 4$
$[5, 7] = [00101_{(2)}, 00111_{(2)}]$	$00110_{(2)} = 6$
$[7, 7] = [00111_{(2)}, 00111_{(2)}]$	$00111_{(2)} = 7$

Consequently, index 7 is found. The distances between the IDs to which pivots are mapped by f_r are as follows:

Pivot		ID	Distance
$01000_{(2)}$		$00010_{(2)}$	$00010_{(2)}$
$00100_{(2)}$	$\overset{f_r}{\mapsto}$	$00100_{(2)}$	$01000_{(2)}$
$00110_{(2)}$		$01100_{(2)}$	$10000_{(2)}$
$00111_{(2)}$		$11100_{(2)}$	

1's appear in different bits of the ID distances. The analysis of this search method yields the following theorem.

Theorem 4. *For a search in a sorted array, the total number of higher order 1 bits $\gamma(\cdot)$ in the distances between all consecutively accessed pivots is less than or equal to $\log n$.*

Proof: Let p be the pivot of (7) of the search space of (6). After accessing p, the next search space is either

(<) $[x_b \dots x_{k+1} 0 x_{k-1} \dots x_{1(2)}, x_b \dots x_{k+1} \underbrace{01 \dots 1}_{k}{}_{(2)}]$

(>) $[x_b \dots x_{k+1} \underbrace{10 \dots 01}_{k}{}_{(2)}, x_b \dots x_{k+1} 1 x'_{k-1} \dots x'_{1(2)}]$

In the case of (<), the next pivot is

$$p' = x_b \dots x_{k+1} \underbrace{01 \dots 1}_{k-k'} \underbrace{10 \dots 0}_{k'}{}_{(2)} ,$$

where k' is such that the lower k-th bits are the largest different bits in the search space of $(<)$. f_{r} maps the pivots to IDs as follows:

$$f_{\mathrm{r}}(p) = \underbrace{0\ldots01}_{k} x_{k+1}\ldots x_{b(2)}$$
$$f_{\mathrm{r}}(p') = \underbrace{0\ldots01}_{k'}\underbrace{1\ldots10}_{k-k'} x_{k+1}\ldots x_{b(2)} .$$

Thus, $d(f_{\mathrm{r}}(p), f_{\mathrm{r}}(p'))$ is

$$\begin{cases} \underbrace{0\ldots01}_{k}\underbrace{0\ldots0}_{b-k}{}_{(2)} & (k-k'=1) \\ \underbrace{0\ldots01}_{k'}\underbrace{1\ldots101}_{k-k'}\underbrace{0\ldots0}_{b-k}{}_{(2)} & (k-k'>1) . \end{cases}$$

In a similar way, $d(f_{\mathrm{r}}(p), f_{\mathrm{r}}(p'))$ in the case of $(>)$ is

$$\underbrace{0\ldots01}_{k'}\underbrace{0\ldots0}_{b-k'}{}_{(2)} .$$

In both cases, 1's only appear from the lower $(b-k+1)$-th bit to the lower $(b-k'+1)$-th bit, and the number of 1's is not greater than $k-k'$. Therefore, in order to specify all bits, the sum of the $\gamma(d(f_{\mathrm{r}}(p), f_{\mathrm{r}}(p')))$ is not greater than $\log n$. ■

For the whole search operation in a sorted array, we have to access the first pivot in $\log n/2$ messages. Accordingly, the average message complexity of the whole operation is less than or equal to

$$\log n/2 + \log n = 3\log n/2 .$$

In the binary search of the DA, p and p' in the proof of Theorem 4 do not have useful properties except for the same upper $b-k$ bits. Therefore, since the bits in which 1's appear overlap, the message complexity is worse than in Theorem 4. The actual message complexity was found experimentally (Section VI).

In the DHT, in order to search a sorted array, we use a binary search in the first search space $[x, x+w)$. The average width of the ID segment managed by one node is $2^b/n$. Accordingly, the average interval of indices mapped to the ID segment is $2^b/(2^b/n) = n$. This is the average of w. The average number of required pivots is $\log n$. Since IDs at which elements indexed by pivots are placed are pseudorandom, the overall performance is

$$\log n(\log n/2) = \log^2 n/2 .$$

VI. Evaluation

To verify the analysis of the previous sections, we implemented the DA and DHT in our simulator and compared their performances. The performance measure was the same as in Section V, the average message complexity.

A. Setting

The Chord network was constructed in the way described in Section III-A, except for fingers in the DA. Fingers in the DA were composed of predecessors as described in Section IV-B. We set $b = 64$ and placed each node in the ID space by referring to the upper 64 bits of the SHA1 hash value of its node number in $[0, n)$. This is not the ideal Chord network. Each array element in the DHT was placed at the ID that had the upper 64 bits of the SHA1 hash value of the concatenation of its array name and its index. We used the average of 1000 tests in which the start nodes and target elements for operations were selected at random.

In order to evaluate dynamic environments, we simulated a situation in which rn nodes (we call r the join-leave ratio) leave and join while finger tables are updated. The node set containing $(1+r)n$ nodes was randomly partitioned into 3 groups: rn leaving nodes (**Group 1**), rn joining nodes (**Group 2**) and the remaining $(1-r)n$ nodes (**Group 3**). Groups 2 and 3 had n nodes in total, and only the n nodes were active in simulations. Fingers were selected from Groups 1 and 3. Neighbor pointers were selected from Groups 2 and 3. If a finger pointed to a node in Group 1, transfers using this finger failed. A failed transfer was retried any number of times by using the next better pointer. Since neighbor pointers were correct, all access processes finished at some time.

In the following, we use **Inter-element access** to denote an access from a node managing an element to another element. This access was used to access elements consecutively.

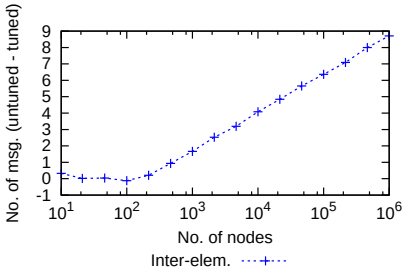


Fig. 6. Inter-element accesses of the DA in non-ideal environments. The ordinate is the difference in average message complexities as a result of the tuning described in Section IV-B ((untuned DA) − (tuned DA)). Parameters are the same as in Fig. 7(a).

B. Results and interpretation

Fig. 6 shows the effect of the tuning described in Section IV-B. The figure plots the difference in inter-element accesses between the tuned and the untuned DA. The ordinate is the difference in the average message complexity of the tuned case from that of the untuned case. The bigger the value, the better the tuning effect. The result shows that tuning effects are proportional to $\log n$. This is consistent with the theoretical value of $m - b + \lceil \log n \rceil - 1$ in (5).

Fig. 7 shows the results of inter-element access. Fig. 7(a) shows that the DA performance is independent of n for large enough n. Fig. 7(b) shows that the DA performs better as the range that we select target indices from becomes smaller. These results indicate that the DA can greatly reduce message complexity, although they are not completely consistent with Theorem 1 because the tuning described in Section IV-B does not erase all the differences between an ideal environment and realistic ones. Fig. 7(c) shows that the average message complexity of the DA is lower than that of the DHT even in the case of frequent node joining and leaving.

Fig. 8 shows the results for sequential accesses. Fig. 8(a) shows that the average message complexity of the DA is almost independent of n; the average message complexity per target element is constant. Fig. 8(b) shows that the value is close to the theoretical one. Fig. 8(c) are results in dynamic environments, showing that the DA performs better than the DHT.

Fig. 9 shows the results of a search in a sorted array. Fig. 9(a) shows that the DA improves the binary search a little. Furthermore, the pivot selection described in Section V greatly improves the average message complexity. In Fig. 9(b) showing the results in dynamic environments, the pivot selection performs much better than the binary search of the DHT.

Fig. 10 shows that the average message complexities to access a single element in the DA and in the DHT are the same. Since a single-element access is an ID access of the base P2P array network, this result proves that the P2P array network of the DA can be used instead of the P2P array network of the DHT.

The above results indicate that the DA greatly reduces message complexity, as expected. They also show that the DA is effective when node joining and leaving are frequent.

VII. Conclusion

We constructed a distributed array (DA) that enables efficient operations for logical arrays managed in P2P environments. DAs are useful in situations in which we want to manage a large database by splitting and sequentially indexing it. Since the P2P overlay network of the DA can be used for that of a distributed hash table (DHT) without additional stabilization costs, the DA can enable DHT-based systems to support a hash table and an efficient logical array for each item at the same time.

We showed that the number of required messages to manipulate array elements consecutively can be reduced in many cases by modifying the item placement rule of DHTs. We devised a reverse bit order mapping based on an analysis of the DHT overlay network. This approach is not limited to this research. We may be able to construct other DAs by combining reverse bit order mapping or other item placement rules and various P2P networks.

The DA may be able to support useful operations besides sequential access and searches in a sorted array.

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker, "A scalable content-addressable network," in *SIGCOMM*, 2001, pp. 161–172.

[2] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware*, ser. Lecture Notes in Computer Science, R. Guerraoui, Ed., vol. 2218. Springer, 2001, pp. 329–350.

[3] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM*, 2001, pp. 149–160.

[4] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *STOC*, 1997, pp. 654–663.

[5] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," *Theory Comput. Syst.*, vol. 32, no. 3, pp. 241–280, 1999.

[6] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds., 1972, pp. 85–103.

[7] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: supporting scalable multi-attribute range queries," in *SIGCOMM*, R. Yavatkar, E. W. Zegura, and J. Rexford, Eds. ACM, 2004, pp. 353–366.

[8] D. Li, X. Lu, B. Wang, J. Su, J. Cao, K. C. C. Chan, and H. V. Leong, "Delay-bounded range queries in DHT-based peer-to-peer systems," in *ICDCS*. IEEE Computer Society, 2006, p. 64.

[9] J. Aspnes and G. Shah, "Skip graphs," *ACM Transactions on Algorithms*, vol. 3, no. 4, 2007.

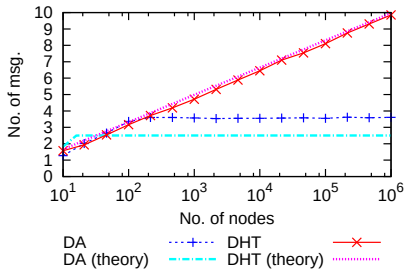
[10] J. Gao and P. Steenkiste, "An adaptive protocol for efficient support of range queries in DHT-based systems," in *ICNP*. IEEE Computer Society, 2004, pp. 239–250.

[11] S. Ramabhadran and J. M. Hellerstein, "Prefix hash tree: An indexing data structure over distributed hash tables," Tech. Rep., 2004.

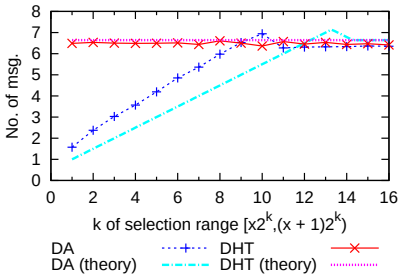
[12] J. Shao and B. T. Davis, "The bit-reversal SDRAM address mapping," in *SCOPES*, ser. ACM International Conference Proceeding Series, K. M. Kavi and R. Cytron, Eds., vol. 136, 2005, pp. 62–71.

[13] O. Shalev and N. Shavit, "Split-ordered lists: Lock-free extensible hash tables," *J. ACM*, vol. 53, no. 3, pp. 379–405, 2006.

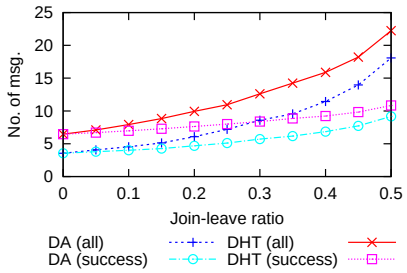
[14] D. Eastlake and P. Jones, "RFC 3174: US secure hash algorithm 1 (SHA1)," 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3174.txt



(a)

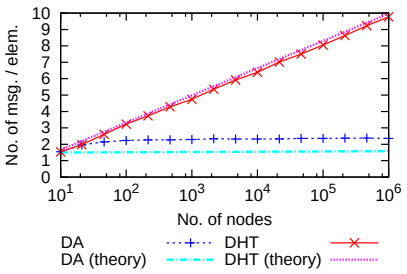


(b) The abscissa is minimum k s.t. target indices belong to $[x2^k, (x+1)2^k)$.

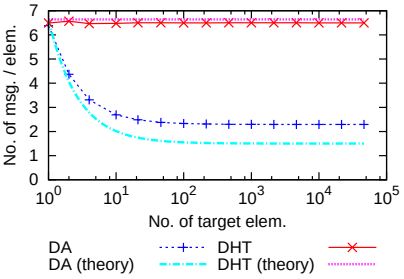


(c) "success" means the number of success transfers. "all" means the sum of "success" and the number of failed transfers.

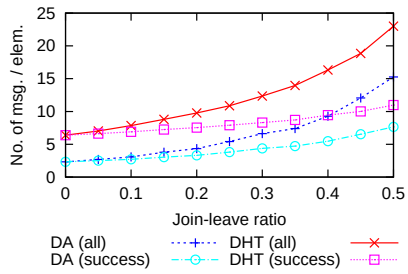
Fig. 7. Inter-element accesses. For each graph, the number of nodes is 10000, k of the index selection range is 4, and the join-leave ratio is 0.0.



(a)

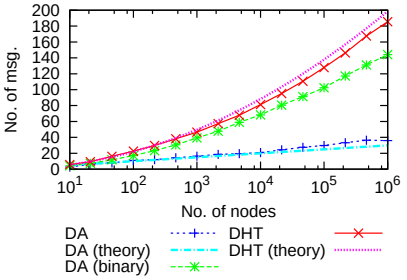


(b)

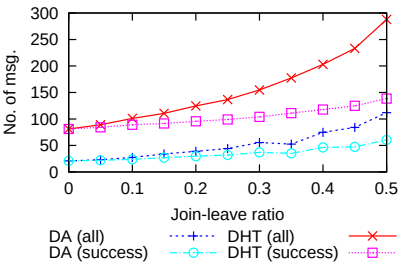


(c) "success" means the number of success transfers. "all" means the sum of "success" and the number of failed transfers.

Fig. 8. Sequential accesses. The ordinate is the average message complexity per target element. For each graph, the number of nodes is 10000, the number of target elements is 100, and the join-leave ratio is 0.0.



(a) "DA (binary)" means the binary search of the DA.



(b) "success" means the number of success transfers. "all" means the sum of "success" and the number of failed transfers.

Fig. 9. Searching a sorted array. The number of nodes is 10000, and the join-leave ratio is 0.0.

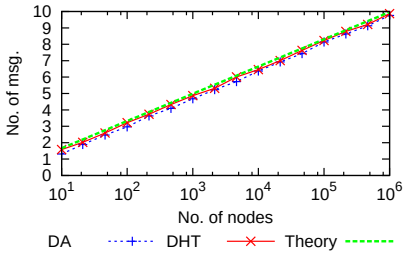


Fig. 10. Single element access. The join-leave ratio is 0.