情報処理学会論文誌 Vol. 50 No. 2 721–736 (Feb. 2009)

分散配列：効率的な論理配列を実現するP2Pデータ構造

福地大輔[†1] クリスチャン ソッメル[†1]
清雄一[†1] 本位田真一[†2,†1]

分散ハッシュテーブル（DHT）を用いることによりP2P環境にデータ管理システムを構築することができる．しかし，DHTは登録されるアイテムの関連性を無視しているため，関連する複数のアイテムに対してまとめて操作を行う場合には効率が悪い．本論文ではDHTをもとに分散配列を構築し，論理配列に対する操作を効率的に実現する．まず，理想的な環境におけるオーバレイネットワークの分析に基づき，複数要素へのアクセスに適するように，配列要素をオーバレイネットワーク上に配置する．この配置は，配列のインデクスのビット列を逆読みすることによって行われることになる．さらに，配列に対する有用な操作であるシーケンシャルアクセス，範囲アクセス，ソート済み配列用の探索の効率的なアルゴリズムを示す．また，オーバレイネットワークのトポロジを調整し，現実的な環境にも対応させる．結果，複数要素へのアクセスをともなう操作に必要なメッセージ数は，ノードの数が増えるほど，DHT上に論理配列を構築する場合と比較して減少する．これを理論と実験の両面から示す．

Distributed Arrays: A P2P Data Structure for Efficient Logical Arrays

Daisuke Fukuchi,[†1] Christian Sommer,[†1] Yuichi Sei[†1]
and Shinichi Honiden[†2,†1]

Distributed hash tables (DHT) are used for data management in peer-to-peer (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 the DHT into a distributed array that enables efficient operations on multiple logical arrays simultaneously. First, array elements are placed in an overlay network according to an easy rule by reversing the binary bit order of their indices. Next, we devise algorithms for sequential access, range access, and searching of a sorted array according to the placement in the overlay network in ideal environments. We then tune the overlay topology to adapt it to realistic environments. As a result, the number of messages required for the operations can be greatly reduced in compared to DHTs. We demonstrate this theoretically and experimentally.

1. 導　　入

P2P技術は巨大なシステムを構築する場合に有用である．P2Pシステムは多数のノードを連携させて何らかの機能を提供するように設計されるため，中央集権的な管理サーバを用意する必要がない．特定のノードがつねにシステムに存在することを仮定しないため，システムの拡大にともない，故障やメンテナンスによりノードが頻繁に欠けるようになっても対応できる．たとえば，P2Pファイル共有システムは中央サーバを持たず，数百万以上のきままなユーザによって運用されている．

分散ハッシュテーブル（DHT）[10),11),13)]の登場により，P2Pシステムをデータ管理の基盤として使用することができるようになった．DHTはハッシュテーブルの基本的な機能を提供する．すなわち，(key, value) ペアの形に表されるアイテムに対して，keyによる登録，参照，削除ができる．個々の操作の際にノード間でやりとりする必要のあるメッセージの数はノード数を n（**表1**）として $O(\log n)$ に抑えられる．

しかし，DHTの機能だけでは効率良く扱えないデータもある．DHTは登録される個々のアイテムの関連性を考慮せずにつくられている．そのため，関連のある複数のアイテムを登録し，それらに対してまとめて何らかの操作を行いたいとしても，結局は個別のアクセスを何度も行うことになる．このためのメッセージコストは小さくない．たとえば，巨大なデータを分割してDHTに保存し，その全体のハッシュ値を計算したいとする．その場合，まず，1つ目のデータ片にアクセスし，そのデータ片の分の計算を行う．その後，計算状態を次のデータ片の保持ノードに転送する．以下同様にすべてのデータ片の分の計算が終わるまで繰り返す．この際，個々の転送に必要なメッセージの数は目的のデータ片に一からアクセスする場合と変わらない．そのため，最終的に w 個のデータ片からハッシュ値を計算する場合に要するメッセージ数は $O(w \log n)$ となる．

†1 東京大学
The University of Tokyo
†2 国立情報学研究所
National Institute of Informatics

表 1　表記
Table 1　Notation.

n	ノード数.
b	ID 空間およびインデクス空間のビット数.
f	b ビットのビット列逆読み写像.
$d(x, y)$	環状 ID 空間においての ID x から ID y までの昇順方向の距離.
$\alpha(x)$，x の α	x の b ビットのビット列に現れる 1 の数.
$\gamma(x)$，x の γ	x の b ビットのビット列において上位 $\log n$ ビットに現れる 1 の数.

本論文では関連するアイテムセットとして論理配列を対象にする．論理配列用の P2P データ構造として分散配列（DA）を提案する．DA は配列に対する最も基本的な操作である**インデクスアクセス**（与えられた配列の与えられたインデクスの要素にアクセスする）に加えて，複数要素へのアクセスをともなう操作を効率的なメッセージ数により実現する．特に，以下の有用な操作に対しては，個別にアルゴリズムを示し，性能を保証する．

シーケンシャルアクセス　与えられた配列の与えられた範囲のインデクスの全要素に昇順でアクセスする．これはハッシュ値の計算や連番の鍵を用いた多重暗号化等に使える．

範囲アクセス　与えられた配列の与えられた範囲のインデクスの全要素に順序を問わずにアクセスする．これは対象要素の一括取得や何らかの合計値の計算，ソートされていない配列からの探索等に使える．

ソート済み探索　ソートされた配列の中から与えられた目標値に等しいもしくは最も近い値の要素を探す．スタックのように末尾の要素だけにデータを登録，削除する場合に現在使用中の最大のインデクスを探すことにも応用できる[⋆1]．

以降においてはこれら 3 つの操作をまとめて**配列操作**と呼ぶことにする．DA と DHT を組み合わせることによって，データごとにハッシューテーブルに登録するか効率的な論理配列に登録するかが選択可能になるため（**図 1**），P2P システムの応用範囲が広がることになる．前述の例に関していえば，分割データを配列に保存することにより，$O(w \log n)$ ではなく $O(w + \log n)$ のメッセージによるハッシュ値計算が可能になる．

以降，2 章に関連研究とその問題点を示す．3 章において DA を構築する．4 章は，DA の理論的，実験的評価である．5 章において，本論文が未対処の問題に関する考察を行う．6 章は結論である．

⋆1 分散環境において配列の末尾に対するデータの追加，削除を厳密に行うためには，最大インデクスを特定した後に，最大インデクスの要素とその次のインデクスの要素に順にアクセスしてデータの追加，削除に関して排他的にロックする等の手順が必要になる．

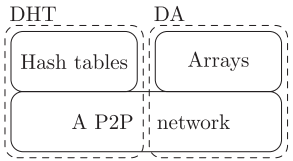


図 1　P2P データ構造の組合せ
Fig. 1　Combination of P2P data structures.

2. 関 連 研 究

我々の知る限り，P2P 環境に構築された多数の配列に対し，効率的な操作を行うことを目的とした研究は過去にはない．操作の効率を考慮しなければ，多数の配列を利用可能なものとして DHT がある．限定的になるが，配列に対する効率的な操作に応用可能なものとしてはレンジクエリ用の研究がある．

コンシステントハッシュ法[5] や Plaxton らによる分散データ配置[8] の考えをもとに Chord[13] 等の様々な DHT[10),11)] が提案されている．DHT は任意の (key, value) ペアを扱うことができるため，配列名とインデクスから key を作成することにより，DHT 上に論理配列をつくることができる．その詳細を考える前に，DHT の概要を説明しておく．DHT は 2 つの層に分割できる（**図 2**）．上層は配列上に実装されたハッシュテーブルである．下層が重要な P2P 技術であり，**配列型 P2P ネットワーク**と呼ぶことにする．配列型 P2P ネットワークは次のような機能を持つ．

- 任意の ID$\in [0, 2^b)$ へ $O(\log n)$ のメッセージ数によりアクセス可能．ただし，b はシステムに固定のパラメータである（表 1）．
- ノードの参加と離脱に対応．

本論文では注目しないため，以降においてはノードの参加，離脱に関する部分は適宜省略する．配列型 P2P ネットワークは次のように構築される．

- ノードをハッシュ関数等によって ID 空間に配置する．
- ノードは ID 空間において自分が配置された位置の周囲の ID の管理者になる．
- ノードは ID 空間における近隣のノードへのポインタを持つ．これにより，任意のノードは任意の ID の管理者へ $O(n)$ のメッセージによりアクセスできる．ただし，ここでいうポインタとは IP ネットワークにおける IP アドレスとポート番号のように相手ノードにメッセージを送るために必要となる情報のことである．

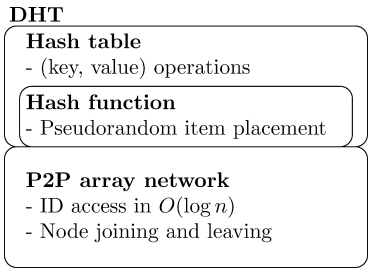


図 2　DHT の構造
Fig. 2　Structure of DHT.

- ノードは ID 空間において離れた位置のノードへのポインタを持つ．この構成を工夫することにより，一ノードあたり $O(\log n)$ 個のポインタによって，メッセージ数が $O(\log n)$ のアクセスを実現することができる．

DHT は配列型 P2P ネットワークにおける (key, value) ペアの ID として，key のハッシュ値を用いることにより実現されている．

DHT において，key として配列名とインデクスの連結物を用いることにより，論理配列を扱うことができる．要素へのインデクスアクセスはその連結物のハッシュ値への ID アクセスになる．配列操作は，たとえば以下のように行われる．

シーケンシャルアクセス　与えられたアクセス範囲が $[x, y]$ の場合，インデクスが x から y の要素にインデクス順にアクセスする．

範囲アクセス　与えられたアクセス範囲が $[x, y]$ の場合，インデクスが x から y の要素に ID 順にアクセスする．

ソート済み探索　インデクス空間を二分探索する．つまり，探索空間の中間のインデクスをピボットとし，それにインデクスアクセスすることを繰り返す．開始時の探索空間は探索開始ノードが自身が管理する既登録および未登録の要素のインデクスを調べることによって決められる（詳細は付録 A.1 に記す）．

しかし，これらの操作の性能は良くない．配列の要素がハッシュ関数により乱数的に ID 空間に配置されるため，必要な ID アクセスの回数を w とすると，およそ $O(w \log n)$ のメッセージが必要になる．もちろん，上記のアルゴリズムには改善の余地がある．しかし，要素が乱数的に配置されているため，アルゴリズムの改善は難しい課題となる．

P2P 環境におけるレンジクエリの研究は関連するアイテムを対象としている．レンジクエリは対象の属性を持つアイテムの中から属性値が与えられた範囲に収まるものを検索する．配列名を属性，インデクスをその値とすれば，レンジクエリ用のシステムによってシーケンシャルアクセスや範囲アクセスは効率的に行うことができる．

レンジクエリ用のシステムには大きく分けて 2 つの種類がある．専用の P2P ネットワーク上に属性値の順序を考えてアイテムを配置する方式と任意の DHT の上にツリーやトライ等の管理構造を構築する方式である．

専用の P2P ネットワークを使う方式[1),2),6)] の問題の 1 つは配列の数に対するスケーラビリティである．この方式は単一属性用のものであったり，複数属性としてもあまり属性数が多くないことや固定数であることを前提としている．そのため，データ管理システムとして不特定多数の配列を扱うためには配列や配列のグループごとに別々にシステムを作成しなければならない．これはコストが大きく，実用的ではない．登録アイテムの偏りという問題もある．配列のインデクスは連番で使用されることも多いため，順序を考えた配置をすると特に偏りやすい．これは負荷の集中を招き，要素数に対するスケーラビリティを低下させる原因となる．負荷の小さいノードを負荷の大きいノードの近くに移動させる等の方法によって負荷分散を図ることもできるが，その場合，負荷分散のコストが必要なだけでなく，アイテムの偏りに沿ってノードが偏る等の副作用が生じる．これは，インデクスアクセスやハッシュテーブルと組み合わせる際の性能の低下につながる．

DHT の上に管理構造を構築する方式[4),9)] においても，管理構造は配列ごとに必要になる．ただし，管理構造を構成するポインタは DHT の機能を使うことにより補完可能なため，積極的に維持する必要はない．そのため，専用の P2P ネットワークを使う方式と比較して容易に多数の配列を扱うことができる．しかし，ノードの出入りが激しい場合等に補完の頻度が増えると，性能は大きく低下する．また，この方式は，管理構造の形に沿わない操作に対しては効果がない．たとえば，PHT[9)] を構成するポインタは隣接する属性値のアイテムの間をつなぐ形になる．そのため，ソート済み配列に対する探索を行う場合，アイテム間ポインタを 1 つずつたどる方法か DHT の (key, value) アクセスを用いた二分探索を行うことになり，効率は良くない．

P2P 環境における配列の利用可能例として，P2P ファイルシステムの IVY[7)] や sub-network システム[3)] があげられる．IVY はファイルシステムの操作ログを DHT に保存する．sub-network システムは変更可能なアイテムを差分の集合の形に表し，アイテムごとに専用の DHT にその差分を保存する．DA はこれらの性能改善に貢献できるはずである．

3. 手　　法

DHT をもとに分散配列（DA）を構築し，論理配列に対する効率の良い操作を実現する．DHT において，ID 空間へのアイテムの配置はハッシュテーブルの役目である（図 2）．その中において使用される SHA1 等のハッシュ関数が配列要素の乱数的配置の原因である．そこでまず，新たな規則を導入し，複数要素に対する効率的なアクセスが行えるように配列要素を ID 空間に配置する．その後，その配置規則を活用するようにシステムを改良する．**図 3** が DA の構造となる．本論文では，配列要素の配置規則としてビット列逆読み写像を導入する．そして，その規則を活用した配列操作のアルゴリズムの提案と配列型 P2P ネットワークの調整を行う．

3.1 配列要素の配置

配列要素に適した配置規則を導出するため，まずは基礎とする配列型 P2P ネットワークである Chord [13] ネットワークの仕組みを分析する．そのうえで，目的に適した配置として，ビット列の逆読みを用いた手法を導入する．

3.1.1 Chord ネットワークの分析

Chord ネットワークは最も単純な配列型 P2P ネットワークであり，次のように構築される．

- 環状の ID 空間 $[0, 2^b)$ を用いる．$2^b - 1$ の次に再び 0 が続く．以降においては，$x > y$ のとき，ID 空間の領域 $[x, y]$ は $[x, 2^b) \cup [0, y]$ を表す．$x \geq 2^b$ のとき，ID x は $x \bmod 2^b$ を表す．環状 ID 空間に関して昇順方向の距離を単に距離という．$d(x, y)$ は x から y までの距離を表す（表 1，**図 4**）．
- ノードはハッシュ関数によって ID 空間に配置される．ID x に配置されたノードを x により表す．任意の ID x について，x 以降の最初のノードを successor(x)，x より前の最初のノードを predecessor(x) と呼ぶ．
- ノード x は ID 空間の領域 $[x, \text{successor}(x+1))$ を管理する[⋆1]．
- ノード x は predecessor(x) と successor$(x+1)$ へのポインタを持つ．
- ノード x は successor$(x+2^k)$（$k < b$）へのポインタを持つ．このポインタおよびポインタの指すノードを x の finger といい，x が持つすべての finger をまとめて x の finger テーブルという．

Chord ネットワークにおける ID アクセスは次のように行われる．

- x はアクセス処理を行っているノードを表す．
- y は目標の ID を表す．

1. $y \in [x, \text{successor}(x+1))$ ならば，x が y を管理する．アクセス処理は x において完了する．
2. そうでなければ，x は finger テーブルの中から y までの距離が最小となる finger x' を選ぶ．アクセス処理を x' に転送する．

- アクセス処理が完了するまで以上を繰り返す．

Chord ネットワークのアクセス手続きをさらに理解するため，$n = 2^m$（$m \leq b$）かつノードが ID 空間に等間隔に配置されている状態を考える．これを理想的な Chord ネットワークと呼ぶことにする．理想的ではない環境には 3.3 節において対応する．任意のノード x について，successor$(x+1)$ は $x + 2^{b-m}$ となり，x の finger テーブルは次のようになる

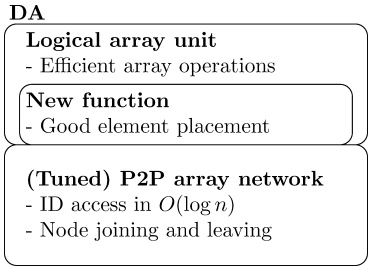


図 3　DA の構造
Fig. 3　Structure of DA.

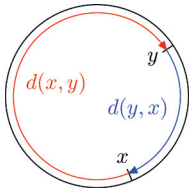
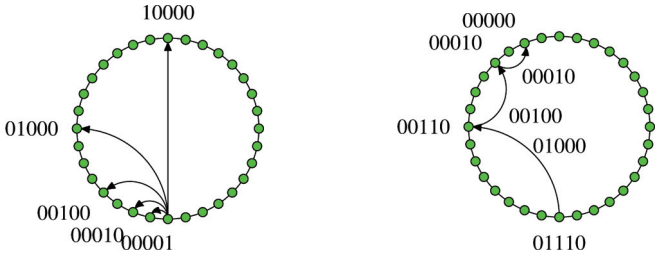


図 4　ID x，y 間の距離
Fig. 4　Distance between ID x and ID y.

⋆1 これはオリジナルの Chord とは異なる．オリジナルにおけるノード x の管理領域は $(\text{predecessor}(x), x]$ である．この違いは，オリジナルの Chord が ID y の successor を見つけることを目的としているのに対し，本論文は y にアクセスすることを目的にしていることによる．どちらにせよ，以降の論理展開には影響しない．



(a) finger テーブル．数値は finger までの距離のビット列

(b) ID アクセスの手続き．円外の数値は目標 ID までの残りの距離のビット列．円内の数値は転送先の finger までの距離のビット列

図 5　$b = 5$ かつ $n = 2^5$ の場合の理想的な Chord ネットワーク
Fig. 5　Ideal Chord network in $b = 5$ and $n = 2^5$.

(**図 5** (a))．

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

ここで "$_{(2)}$" は二進数表記であることを示す．以降，不要な上位桁に 0 を詰めた b 桁の二進数表記を暗黙的に用い，それをビット列という．ノード x_0 が開始した ID アクセスは次のように進められる．

- x_i はアクセス処理を行っているノードを表す．
- y は目標の ID を表す．

1. $d(x_i, y)$ の上位 m ビットがすべて 0 ならば，$y \in [x_i, \mathrm{successor}(x_i + 1))$ であり，x_i が y を管理する．アクセス処理は x_i において完了する．
2. そうでなければ，次のように $d(x_i, y)$ の上位 m ビットに 1 が現れる．

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

   x_i は自分の finger テーブルから，y までの距離が最小となる以下の finger x_{i+1} を選ぶ．

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

   アクセス処理を x_{i+1} に転送する．

- アクセス処理が完了するまで以上を繰り返す．

この処理の 2 番目の手続きによって，アクセス処理を行っているノードから目標 ID までの距離は

$$d(x_i, y) = \underbrace{0\ldots01}_{k} z_{k+1} \ldots z_{b(2)} \text{ から } d(x_{i+1}, y) = \underbrace{0\ldots00}_{k} z_{k+1} \ldots z_{b(2)}$$

に変わる．これは，ビット列の最上位の 1 を消していることになる．

理想的な Chord ネットワークのアクセス手続きは $d(x_0, y)$ のビット列に現れる 1 を上位から消していく手続きである（図 5 (b)）．1 回転送することによって最上位の 1 を消すことができる．ここで，上位 $\log n$ ビットに現れる 1 の数を表す関数および略記として γ を導入する（表 1）．$\gamma_0 = \gamma(d(x_0, y))$ とすると，アクセス手続きによる γ_0 回の転送後，$d(x_0, y)$ の上位 $\log n$ ビットの 1 はすべて消える．

$$d(x_{\gamma_0}, y) = \underbrace{0\ldots0}_{\log n}\underbrace{*\cdots*}_{b-\log n}{}_{(2)} \ < \ 2^{b-\log n}$$

ノードの間隔は $2^b/n = 2^{b-\log n}$ であるため，x_{γ_0} が y を管理する．つまり，このアクセス処理は $x_0, x_1, \ldots, x_{\gamma_0}$ 間の転送を経て完了する．要するメッセージ数は，

$$\gamma_0 \tag{1}$$

である．もし，y がランダムなら，$d(x_0, y)$ の中のビットが 1 か 0 かもランダムである．その場合，要するメッセージ数 γ_0 の平均は

$$\frac{\log n}{2} \tag{2}$$

となる．これが理想的な Chord ネットワークにおける ID アクセスの性能である．

3.1.2　ビット列逆読み写像

前項の分析から，複数要素へのアクセスをともなう操作を効率良く行うためには，対象要素が配置される ID 間の距離の γ を小さくすることが有効であると分かる．そこで，そのような配列要素の配置規則を導入する．

この配置規則は，配列のインデクス空間から配列型 P2P ネットワークの ID 空間への写像として表せる．インデクス空間から ID 空間への写像を g として，インデクスが x の要素を ID $g(x)$ に配置する．多数の配列を扱う場合には，配列名のハッシュ値 h をその配列の基準 ID として，インデクスが x の要素を $h + g(x)$ に配置する．以降，簡単のために基準 ID は省略する．

複数要素へのアクセスをともなう操作を効率良く行うために，対象インデクスが写像される ID を ID 空間の狭い領域に限定することが考えられる．それにより，ID 空間において固まった少数のノードが対象要素の管理者となり，メッセージコストが抑えられると期待でき

る．しかし，本論文ではこの方針はとらない．その理由を説明する．まず，任意の ID x，y と正整数 k（$< b$）について，

$$d(x,y) = \underbrace{0\ldots0}_{b-k}\underbrace{*\cdots*}_{k}{}_{(2)} < 2^k \Leftrightarrow d(y,x) = \underbrace{1\ldots1}_{b-k}\underbrace{*\cdots*}_{k}{}_{(2)} > 2^b - 2^k$$

$$(\because d(y,x) = 2^b - d(x,y))$$

となる．$d(x,y)$ を小さくして，上位ビットに 1 が現れないようにしても，$d(y,x)$ の上位ビットにその分の 1 が現れることが分かる．つまり，ID x,y が別のノードに管理される場合，片方の ID の管理者からもう片方の ID にアクセスする際の性能は，x と y を狭い領域に限定しても，式 (2) より平均して小さくなるわけではない．もちろん，1 つのノードに管理されることになればメッセージは不要になる．しかし，複数の要素が多くの場合において 1 つのノードによって管理されるとすると，それらを個別に分散データ構造のアイテムとして扱う意味は小さい．まとめて 1 つのアイテムとして扱っても同等の結果が得られる．そのため，本論文では，各要素は個別のアイテムとして扱われるべき単位として上位層から与えられると仮定することにする．この仮定のもとでは，複数の要素はなるべく別々のノードに管理される方がよい．これは負荷分散の点からも重要である．よって，対象のインデクスを ID 空間の狭い領域に写像する方法は適さない．

求める写像は任意の操作対象のインデクスをできるだけ互いの距離の γ が小さく，かつ異なるノードに管理されるような ID に写像するべきである．そのような写像を導出するために次の自明な性質が役立つ．

補題 1. 任意の ID x,y が下位 k ビットを同じくするならば，$d(x,y)$ の下位 k ビットに 1 は現れない．

補題 1 から，2^k（$k \leq b$）個以下のインデクスを下位 $b-k$ ビットが等しい ID に写像することにより，その中の任意のインデクスが写像される ID 間の距離について，1 が現れるビットを上位 k 桁に制限することができる．たとえば，4 つのインデクスを下位 $b-2$ ビットが等しい ID $00z_3\ldots z_{b(2)}$，$01z_3\ldots z_{b(2)}$，$10z_3\ldots z_{b(2)}$，$11z_3\ldots z_{b(2)}$ に写像した場合，その中の任意の ID 間の距離は

$$**\underbrace{0\ldots0}_{b-2}{}_{(2)}$$

の形になる．1 の数は平均して 1 になるため，$n > 2$ であれば，ID 間距離の γ の平均は式 (2) より小さい．また，写像後の 4 つの ID は ID 空間を 4 等分するように配置される（**図 6**）ため，n が十分に大きい場合，別々のノードに管理されることになる．

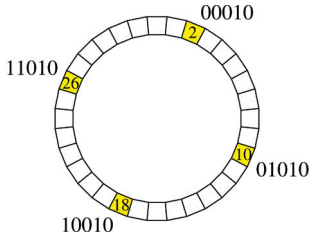


図 6　$b = 5$ のとき，下位 3 ビットを同じくする ID の例．枠内の数値が ID．枠外の数値は ID のビット列
Fig. 6　An example of IDs which have the same bit pattern in the lower 3 digits in $b = 5$.

以下に定義するビット列逆読み写像 f によって，任意の

$$[x2^k, (x+1)2^k) \quad (k < b \text{ かつ } x < 2^{b-k}) \tag{3}$$

の形のインデクスの集合を前段の例のように ID に写像することができる．

定義 1（ビット列逆読み写像 f）．f はインデクス空間 $[0, 2^b)$ から *ID* 空間 $[0, 2^b)$ への写像であり，任意の $x_i \in \{0,1\}$（$1 \leq i \leq b$）について，

$$f\left(\sum_{k=1}^{b} x_k 2^{k-1}\right) = \sum_{k=1}^{b} x_{b-(k-1)} 2^{k-1}$$

と定義する．すなわち，

$$f(x_b\ldots x_{1(2)}) = x_1\ldots x_{b(2)}$$

である．

補題 2. 定義 1 の写像 f によって式 (3) に属するインデクスが写像されるすべての ID の下位 $b-k$ ビットは等しい．

証明. 式 (3) に属する任意のインデクスは，k と $x = x_b\ldots x_{k+1(2)}$ が与えられているとき，適当な $x'_1,\ldots,x'_k \in \{0,1\}$ を用いて $x_b\ldots x_{k+1}x'_k\ldots x'_{1(2)}$ と書ける．よって，f により ID に写像すると，

$$f(x_b\ldots x_{k+1}x'_k\ldots x'_{1(2)}) = x'_1\ldots x'_k x_{k+1}\ldots x_{b(2)}$$

より，下位 $b-k$ ビットは上位から $x_{k+1},\ldots,x_b$ となる．□

f を用いることにより，インデクス間の関係として次の性質が成り立つ．

定理 1. 定義 1 の写像 f によって式 (3) に属する任意の *2* つのインデクスが写像される ID 間の距離の γ は平均して

$$\frac{\min(k, \log n)}{2}$$

となる．

証明．補題 2 から，式 (3) に属する任意の 2 つのインデクスを f によって写像すると，下位 $b-k$ ビットが等しい ID になる．補題 1 から，それらの ID 間の距離のビット列において 1 が現れるのは上位 k ビットのみである．よって，γ の平均は $k < \log n$ のとき $k/2$，$k \geq \log n$ のとき式 (2) となる．□

負荷分散等の性能に影響するインデクスの分散性として次の性質が成り立つ．

定理 2. 定義 1 の写像 f によって式 (3) のインデクス集合は ID 空間上で等間隔の ID に写像される．

証明．式 (3) のインデクス集合は

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

と表せる．この集合内のインデクスは f により次のように ID に写像される．

インデクス		ID
$x_b \dots x_{k+1} 0 \dots 00_{(2)}$		$00 \dots 0 x_{k+1} \dots x_{b(2)}$
$x_b \dots x_{k+1} 0 \dots 01_{(2)}$	$\overset{f}{\mapsto}$	$10 \dots 0 x_{k+1} \dots x_{b(2)}$
$\vdots$		$\vdots$
$x_b \dots x_{k+1} \underbrace{1 \dots 11}_{k}{}_{(2)}$		$\underbrace{11 \dots 1}_{k} x_{k+1} \dots x_{b(2)}$

インデクスの下位 k ビットは 2^k 通りのすべてのビットパターンを重複なく含む．そのため，ID の上位 k ビットも 2^k 通りのすべてのビットパターンを重複なく含む．これらの ID を昇順にソートすれば，

$$\begin{aligned} &0 \dots 00 x_{k+1} \dots x_{b(2)} \\ &0 \dots 01 x_{k+1} \dots x_{b(2)} \\ &\qquad \vdots \\ &\underbrace{1 \dots 11}_{k} x_{k+1} \dots x_{b(2)} \end{aligned}$$

となる．したがって，その間隔は等しく

$$\underbrace{0 \dots 01}_{k} \underbrace{0 \dots 0}_{b-k}{}_{(2)} \tag{4}$$

である．また，ID 空間の始点から最初の ID までの距離と最後の ID から ID 空間の終点までの距離の和も

$$\underbrace{0 \dots 0}_{k} x_{k+1} \dots x_{b(2)} - 0 + 2^b - \underbrace{1 \dots 1}_{k} x_{k+1} \dots x_{b(2)} = 2^b - \underbrace{1 \dots 1}_{k} \underbrace{0 \dots 0}_{b-k}{}_{(2)}$$
$$= \underbrace{0 \dots 01}_{k} \underbrace{0 \dots 0}_{b-k}{}_{(2)}$$

となり，等しい．□

配列のインデクスは連番で使用されることが多く，また，使用されるインデクスが動的に変わることも多いため，式 (3) の形の任意の連続するインデクスに対して，高いインデクス間のアクセス性能（定理 1）と分散性能（定理 2）を持つ f は目的に適っている．そこで，配列要素の配置にはこのビット列逆読み写像 f を用いることにする．

ビット列の逆読みは Shao ら[12] も，メモリ性能の向上のために，論理アドレスと物理アドレスの変換に用いているが，定理 1，2 のような性質に関しては言及していない．

3.2　配列操作のアルゴリズム

ビット列逆読み写像 f を用いることにより，効率の良い配列操作が可能になることを具体的にアルゴリズムをあげて示す．3.1.1 項に示したように，理想的な Chord ネットワークではアクセスに必要なメッセージ数が目標 ID までの距離の γ となる．よって，目標 ID までの距離の γ およびその上限であるビット列に現れる 1 の数（これを α により表す（表 1））を小さくするようにする．

シーケンシャルアクセス　特別なアルゴリズムは必要ない．インデクス順にアクセスする．たとえば，$b = 5$ かつアクセス範囲が $[7, 11]$ であるとする．インデクス順にアクセスすると，ID 間距離の α は次のようになる．

インデクス		ID	距離	α
$7 = 00111_{(2)}$		$11100_{(2)}$	$00110_{(2)}$	2
$8 = 01000_{(2)}$		$00010_{(2)}$	$10000_{(2)}$	1
$9 = 01001_{(2)}$	$\overset{f}{\mapsto}$	$10010_{(2)}$	$11000_{(2)}$	2
$10 = 01010_{(2)}$		$01010_{(2)}$	$10000_{(2)}$	1
$11 = 01011_{(2)}$		$11010_{(2)}$		

一般に，次の定理が成り立つ．

定理 3. 定義 1 の写像 f を用いることにより，シーケンシャルアクセスにおいて順にアクセスする ID 間距離の α は平均して

$$\frac{3}{2}$$

となる．

証明. シーケンシャルアクセスにおいて目標のインデクスは

$$\begin{aligned}
&\dots\\
&x_b \dots x_4 000_{(2)}\\
&x_b \dots x_4 001_{(2)}\\
&x_b \dots x_4 010_{(2)}\\
&x_b \dots x_4 011_{(2)}\\
&x_b \dots x_4 100_{(2)}\\
&\dots
\end{aligned}$$

のように変化していく．つまり，2 回に 1 回は $x_b \dots x_2 0_{(2)}$ から $x_b \dots x_2 1_{(2)}$ に変わり，2^k（$k \geq 2$）回に 1 回は

$$x_b \dots x_{k+1} \underbrace{01\dots1}_{k}{}_{(2)} \text{ から } x_b \dots x_{k+1} \underbrace{10\dots0}_{k}{}_{(2)}$$

に変わる．f によって ID に写像すると，2 回に 1 回は $0x_2 \dots x_{b(2)}$ から $1x_2 \dots x_{b(2)}$ に変わり，2^k（$k \geq 2$）回に 1 回は

$$\underbrace{1\dots10}_{k} x_{k+1} \dots x_{b(2)} \text{ から } \underbrace{0\dots01}_{k} x_{k+1} \dots x_{b(2)}$$

に変わる．よって，順にアクセスする際の ID 間距離は，2 回に 1 回は

$$d(0x_2 \dots x_{b(2)}, 1x_2 \dots x_{b(2)}) = \underbrace{10\dots0}_{b}{}_{(2)},$$

2^k（$k \geq 2$）回に 1 回は

$$d(\underbrace{1\dots10}_{k} x_{k+1} \dots x_{b(2)}, \underbrace{0\dots01}_{k} x_{k+1} \dots x_{b(2)}) = \underbrace{0\dots011}_{k}\underbrace{0\dots0}_{b-k}{}_{(2)}$$

となる．ID 間距離の α は $1/2$ の確率で 1，$\sum_{k\geq 2}(1/2^k) = 1/2$ の確率で 2 である．平均は

$$1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{2} = \frac{3}{2}$$

となる．　□

範囲アクセス　まず，アクセス範囲をできるだけ大きな式 (3) の形に分割する．各領域内においては ID が昇順になる順番にアクセスする．領域はインデクスに関して昇順になるように消化する．

たとえば，$b = 5$ かつアクセス範囲が $[3, 16]$ であるとする．アクセス範囲は次のように分割する．

$$[3, 16] = [3, 4) \cup [4, 8) \cup [8, 16) \cup [16, 17)$$

各領域内においては ID 順にアクセスするため，たとえば $[4, 8)$ なら，アクセス順序とそのときの ID 間距離の α は次のようになる．

インデクス		ID（昇順）	距離	α
$4 = 00100_{(2)}$		$00100_{(2)}$	$01000_{(2)}$	1
$6 = 00110_{(2)}$	$\overset{f}{\mapsto}$	$01100_{(2)}$	$01000_{(2)}$	1
$5 = 00101_{(2)}$		$10100_{(2)}$	$01000_{(2)}$	1
$7 = 00111_{(2)}$		$11100_{(2)}$		

また，$[4, 8)$ のインデクスを ID 順に並べても，最初と最後のインデクスはインデクス順で最初と最後のインデクスである 4 と 7 から変わらない．他の領域も同様になる．よって，領域をインデクス順に消化するために $[4, 8)$ の最後のインデクスから $[8, 16)$ の最初のインデクスにアクセスする場合は次のようになる．

インデクス		ID	距離	α
$7 = 00111_{(2)}$	$\overset{f}{\mapsto}$	$11100_{(2)}$	$00110_{(2)}$	2
$8 = 01000_{(2)}$		$00010_{(2)}$		

このアルゴリズムを用いることで，次の定理が成り立つことを示せる．

定理 4. 定義 1 の写像 f を用いることにより，アクセス範囲の幅が w の範囲アクセスにおいて，順にアクセスする ID 間距離の α の合計は

$$w + 2\log w$$

以下にできる．

証明. まず，式 (3) の形の領域内において ID 順にアクセスする場合の ID 間距離の α を考える．定理 2 の証明より，式 (3) の形の領域から f によって写像した ID に昇順にアクセスする場合の ID 間距離は式 (4) となり，α は 1 である．

次に，領域間のアクセスを考える．式 (3) の領域内において，インデクスを ID 順に並べ

替えると，最初と最後のインデクスは $x2^k$ と $(x+1)2^k-1$ となる．これは，インデクス順でも最初と最後のインデクスである．よって，領域をインデクス順に消化すると，式 (3) の形の領域から次の領域に移動する際に，目標インデクスは $(x+1)2^k-1$ から $(x+1)2^k$ に変わる．これはシーケンシャルであり，$(x+1)2^k-1$ の最下位ビットは 0 ではない．この場合の ID 間距離の α は 2 である（$\because$ 定理 3 の証明）．

最後に，アクセス範囲がいくつの領域に分割できるかに注目して ID 間距離の α の合計を求める．アクセス範囲が 1 つの式 (3) の形の領域の場合，その幅を w とすれば，ID 間距離の α の合計は $w-1$ である．アクセス範囲が

$$[x_b \dots x_{k+1} \underbrace{0 \dots 0}_{k}{}_{(2)}, x_b \dots x_{k+1} x'_k \dots x'_{1(2)}) \quad (\text{ただし } x'_k = 1) \tag{5}$$

の場合，この領域は幅 $w = x'_k x'_{k-1} \dots x'_{1(2)}$ に対して次のように $\alpha(w)$ 個の領域に分割できる．

$$\bigcup_{\{i \mid i \le k \wedge x'_i = 1\}} [x_b \dots x_{k+1} x'_k \dots x'_{i+1} \underbrace{0 \dots 0}_{i}{}_{(2)}, x_b \dots x_{k+1} x'_k \dots x'_{i+1} \underbrace{10 \dots 0}_{i}{}_{(2)})$$

よって，α の合計は

$$1(w-1-(\alpha(w)-1)) + 2(\alpha(w)-1) = w + \alpha(w) - 2 \le w + \log w - 1$$

となる．アクセス範囲が

$$(x_b \dots x_{k+1} x_k \dots x_{1(2)}, x_b \dots x_{k+1} \underbrace{1 \dots 1}_{k}{}_{(2)}] \quad (\text{ただし } x_k = 0) \tag{6}$$

の場合，この領域は幅 $w = \bar{x}_k \bar{x}_{k-1} \dots \bar{x}_{1(2)}$（ただし $\bar{x}_i = 1 - x_i$）に対して次のように $\alpha(w)$ 個の領域に分割できる．

$$\bigcup_{\{i \mid i \le k \wedge x_i = 0\}} (x_b \dots x_{k+1} x_k \dots x_{i+1} \underbrace{01 \dots 1}_{i}{}_{(2)}, x_b \dots x_{k+1} x_k \dots x_{i+1} \underbrace{11 \dots 1}_{i}{}_{(2)}]$$

ID 間距離の α の合計は式 (5) の場合と同様となる．アクセス範囲が

$$(x_b \dots x_{k+1} 0 x_{k-1} \dots x_{1(2)}, x_b \dots x_{k+1} 1 x'_{k-1} \dots x'_{1(2)}) \tag{7}$$

の場合，まず，次のように式 (6) と式 (5) の形の領域に分割できる．

$$\begin{aligned} &(x_b \dots x_{k+1} 0 x_{k-1} \dots x_{1(2)}, x_b \dots x_{k+1} \underbrace{01 \dots 1}_{k}{}_{(2)}] \quad \cup \\ &[x_b \dots x_{k+1} \underbrace{10 \dots 0}_{k}{}_{(2)}, x_b \dots x_{k+1} 1 x'_{k-1} \dots x'_{1(2)}) \end{aligned} \tag{8}$$

式 (7)，式 (8) の 1 つ目の領域，式 (8) の 2 つ目の領域の幅をそれぞれ w，w_1，w_2 とすると，ID 間距離の α の合計は

$$\begin{aligned} &1(w-1-(\alpha(w_1)+\alpha(w_2)-1)) + 2(\alpha(w_1)+\alpha(w_2)-1) \\ &= w + \alpha(w_1) + \alpha(w_2) - 2 \\ &\le w + 2\log w \end{aligned}$$

である．□

ソート済み探索　二分探索のピボットの選び方を変更する．探索空間が

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

と表せるとき，k 桁目のビットを特定するために，ピボットとして

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

を選ぶ．

たとえば，$b=5$ かつ探索する値 v が昇順にソートされた配列のインデクス 6 と 7 の要素の値に挟まれるとする．正しい探索であれば，値が v に最も近い要素のインデクスとして 6 か 7 が特定される．最初の探索空間が $[3,14]$ であるとすると，まず，$[3,14] = [00011_{(2)}, 01110_{(2)}]$ より，ピボットとして $01000_{(2)} = 8$ にアクセスする．インデクスが 8 の要素の値は v より大きいため，探索空間は $[3,7]$ となる．以降も同様にして，次のようにピボットにアクセスする．

探索空間		ピボット
$[3,14]$	$= [00011_{(2)}, 01110_{(2)}]$	$01000_{(2)} = 8$
$[3,7]$	$= [00011_{(2)}, 00111_{(2)}]$	$00100_{(2)} = 4$
$[5,7]$	$= [00101_{(2)}, 00111_{(2)}]$	$00110_{(2)} = 6$
$[7,7]$	$= [00111_{(2)}, 00111_{(2)}]$	$00111_{(2)} = 7$

最後にアクセスするインデクス 7 の要素の値は v より大きいため，値が上から v に最も近い要素のインデクスが 7 であることが分かる．以上のピボットを f によって ID に写像すると，ID 間距離の α は次のようになる．

ピボット		ID	距離	α
$01000_{(2)}$		$00010_{(2)}$	$00010_{(2)}$	1
$00100_{(2)}$	$\overset{f}{\mapsto}$	$00100_{(2)}$	$01000_{(2)}$	1
$00110_{(2)}$		$01100_{(2)}$	$10000_{(2)}$	1
$00111_{(2)}$		$11100_{(2)}$		

このアルゴリズムを用いることで，次の定理が成り立つことを示せる．

定理 5. 定義 1 の写像 f を用いる場合，ソート済み探索において順にアクセスする ID 間の距離の γ の合計は

$\log n$

以下にできる．

証明．探索空間が

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

のとき，ピボットは

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

である．p にアクセスした後，次の探索空間は

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

$[x_b \dots x_{k+1} \underbrace{10 \dots 01}_{k}{}_{(2)}, x_b \dots x_{k+1} 1 x'_{k-1} \dots x'_{1(2)}]$ (9)

となる．次のピボットは

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

である．ただし，k' は式 (9) 内の両端のインデクスの間で異なる最上位のビットの桁位置を表す．これらのピボットを f により ID に写像すると，

$f(p) = \underbrace{0 \dots 01}_{k} x_{k+1} \dots x_{b(2)}$

$f(p') = \underbrace{0 \dots 01}_{k'} \underbrace{1 \dots 10}_{k-k'} x_{k+1} \dots x_{b(2)}$ または $\underbrace{0 \dots 01}_{k'} \underbrace{0 \dots 01}_{k-k'} x_{k+1} \dots x_{b(2)}$

となる．よって，ID 間距離は

$$d(f(p), f(p')) = \begin{cases} \underbrace{0 \dots 01}_{k} \underbrace{0 \dots 0}_{b-k}{}_{(2)} & (k - k' = 1) \\ \underbrace{0 \dots 01}_{k'} \underbrace{1 \dots 101}_{k-k'} \underbrace{0 \dots 0}_{b-k}{}_{(2)} & (k - k' > 1) \end{cases}$$

または

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

である．この γ は p と p' で特定するビットの桁位置の差 $k - k'$ を超えない．また，1 が現れるビットの桁位置は $b - k + 1$ から $b - k' + 1$ の間だけである．よって，最終的に全ビットを特定するまでに必要となる ID 間距離の γ の合計は

$\log n$

を超えない． □

3.3　現実的な環境への対応

この節において，n が 2 の冪乗かつノードが等間隔に配置されているという仮定の成り立たない環境に対応する．具体的には，Chord ネットワークの finger テーブルを successor ではなく predecessor によって構成するようにする．

3.1.1 項において，理想的な Chord ネットワークにおいて要するメッセージ数はアクセス処理中のノードから目標 ID までの距離の γ であることを示した．定理 1，3，4，5 は，ビット列逆読み写像 f を用いることによって，配列に対する操作の際にアクセスすることになる ID 間の距離の γ が相対的，絶対的に小さくなることを示す．しかし，Chord ネットワークと f および 3.2 節のアルゴリズムの組合せは現実の環境においては十分な性能を発揮しない．その原因は理想的な環境における finger テーブルと現実の finger テーブルとの違いにある．現実の環境において，ノード数が 2 の冪乗であったり，ノードが等間隔に配置されていたりすることは稀である．そのため，x の finger である successor$(x + 2^m)$ が正確に $x + 2^m$ になることも稀である．通常は少し先に位置することになる．そのときの $x + 2^m$ から successor$(x + 2^m)$ までの距離の平均 l はノードの平均間隔の半分である．

$$l = \frac{1}{2}\frac{2^b}{n} = 2^{b - \log n - 1}$$

ここで，x をアクセス処理中のノード，y を目標の ID とする．

$d(x, y) = 2^k$　（ただし $k \geq \lceil \log l \rceil$） (10)

と仮定すると，x の finger テーブルの中の最適な転送先 x' は

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

となる．目標 ID までの次の距離は

$d(x', y) \approx 2^{k-1} - l = \underbrace{0 \dots 0}_{b-k} \underbrace{01 \dots 1}_{k - \lceil \log l \rceil} \underbrace{* \cdots *}_{\lceil \log l \rceil}{}_{(2)}$

である．$b - \lceil \log l \rceil > \log n$ ($\because \log l = b - \log n - 1$) より，上位 $\log n$ ビットをすべて 0 にするには，さらにおよそ

$\log n - (b - k + 1)$ (11)

回の転送が必要となる．また，3.1.1 項で説明したように，目標 ID までの距離のビット列に現れる 1 は転送のたびに上位から消えていく．したがって，アクセス処理を進めるうちに，目

(a) successor からなる finger テーブル．点線により示される finger は目標 ID を少しだけ超えてしまうため，その約半分しか進むことのできない 1 つ手前の finger が使用される

(b) predecessor からなる finger テーブル．1 回の転送によって目標 ID の少し手前まで進むことができる

図 7　2^{b-1} 先の ID へのアクセスにおける finger テーブルの構成の影響
Fig. 7　Effect of components of finger tables in access to the 2^{b-1} distant ID.

標 ID までの距離の上位 $\log n$ ビットに 1 が 1 つだけの状態になる．この状態は式 (10) に近似できる．また，定理 1，3，4，5 の証明から，配列の操作において目標 ID までの距離に 1 が現れるのは上位ビットだけであることが多い．その場合，$k \approx b$ であり，(式 (11)) $\approx \log n$ となる．これは効率が悪い（**図 7** (a)）．

finger テーブルを successor ではなく predecessor により構成することにより，この問題は回避できる．その場合，x の finger は predecessor$(x + 2^m)$ となり，$x + 2^m$ よりも平均して l だけ手前に位置する．これにともない，アクセス手続きも 1 カ所変更する．元のアクセス手続きにおいては転送先を finger テーブルの中から選び出していたが，それを finger テーブルと successor$(x+1)$ の中から選ぶようにする．以上により，上記と同じ状況において，最適な転送先 x' は

$$x' \approx x + 2^k - l$$

となる．目標 ID までの次の距離は

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

である．平均ノード間隔が $2l$ であるため，アクセス処理は x' において完了する可能性が高い（図 7 (b)）．

このように，finger テーブルを predecessor によって構成することにより，配列に対する操作において ID 間距離の γ が小さいという特徴を現実的な環境に対しても活かすことができる．実際の効果は 4.2 節の実験によって検証する．

4. 評　　価

理論と実験の両面から DA の性能を評価する．評価する性能は目標となるインデクスすべてにアクセスするために要するメッセージ数の平均である．これは操作の速度やトラフィックの指標であり，小さいほど良い．インデクスアクセスと配列操作について，DA および DHT 上の論理配列を比較する．DHT としては Chord を用いる．

4.1 理　　論

理想的な Chord ネットワークにおけるメッセージ数を表す式 (1) を用いて近似する．適宜，γ_0 をアクセスする ID 間の距離の γ と見なす．また，α が γ の上界であることも考慮する．シーケンシャルアクセス，範囲アクセスにおいては，1 つのノードが連続して目標となる複数の要素を管理することはないと仮定する．

まず，DA について考える．ランダムなノードから始められるインデクスアクセスはランダムなノードからの ID アクセスのため，性能は式 (2) から，

$$\frac{\log n}{2} \tag{12}$$

である．任意のインデクスの管理者から任意のインデクスへのアクセス（以降はインデクス間アクセスと呼ぶ）は定理 1 において，$x' = x2^k$ かつ $w = 2^k$ とおくことにより，インデクスの選択範囲が $[x', x' + w)$ のときの性能は

$$\frac{\min(\log w, \log n)}{2} = \frac{\log(\min(w, n))}{2} \tag{13}$$

となる．アクセス範囲の幅が w のときのシーケンシャルアクセスの性能は，定理 3 から，

$$\frac{3}{2}(w - 1) + \frac{\log n}{2} \tag{14}$$

以下である．最後の項はランダムな開始ノードからの最初のアクセスに必要なメッセージ数を意味する．アクセス範囲の幅が w のときの範囲アクセスの性能は，定理 4 から，

$$w + 2\log w + \frac{\log n}{2} \tag{15}$$

以下である．ソート済み探索の性能は，定理 5 から，

$$\log n + \frac{\log n}{2} = \frac{3}{2}\log n \tag{16}$$

以下となる．

次に DHT について考える．インデクスアクセス（インデクス間アクセス含む）はランダムなノードからの ID アクセスになるため，性能は式 (12) と同じである．シーケンシャルアクセスは乱数的な ID への ID アクセスによって構成される．したがって，アクセス範囲の幅が w のときの性能は，

$$w\frac{\log n}{2} \tag{17}$$

である．範囲アクセスには環状 ID 空間に沿って 1 周する間に，対象のインデクスに対応するすべての ID にアクセスする方法を用いる．アクセス範囲の幅が w のとき，アクセス対象となる ID の平均間隔は

$$\frac{2^b}{w} = 2^{b-\log w}$$

である．そのため，順にアクセスする際の ID 間距離の上位 $\lceil \log w \rceil - 1$ ビットは平均して 0 であると考えられる．よって，性能は

$$w\frac{\log n - (\lceil \log w \rceil - 1)}{2} \leq \frac{w}{2}\left(\log \frac{n}{w} + 1\right) \tag{18}$$

である．ソート済み探索は開始時の探索空間 $[x, x+w)$ を二分探索する．1 つのノードが管理する ID 空間の領域の平均幅は

$$l = \frac{2^b}{n}$$

であり，その中の ID に写像されるインデクスの平均間隔は

$$\frac{2^b}{l} = n$$

となる．これが開始時の探索空間の幅 w の平均になる．よって，要するピボットの数は平均 $\log n$ である．性能は

$$\log n \cdot \frac{\log n}{2} = \frac{\log^2 n}{2} \tag{19}$$

となる．

以上をまとめたものが**表 2** である．DA のインデクスアクセスの性能は DHT のそれと同等である．インデクス間アクセスはインデクスの選択範囲の幅が n より小さいとき，DA の方が性能が良い．シーケンシャルアクセスとソート済み探索は，DA の方が n への依存性が低下した分だけ性能が良い．範囲アクセスは n がアクセス範囲の幅よりずっと大きい場合には DA の方が有利であるが，そうでない場合には DHT の方が有利になりうる．

表 2　理論的性能．理想的な環境において目標とするすべての要素にアクセスするために要する平均メッセージ数
Table 2　Theoretical average number of required messages.

	DA	（式番号）	DHT	（式番号）
インデクスアクセス	$O(\log n)$	(12)	$O(\log n)$	(12)
インデクス間アクセス	$O(\log(\min(w, n)))$	(13)	$O(\log n)$	(12)
シーケンシャルアクセス	$O(w + \log n)$	(14)	$O(w \log n)$	(17)
範囲アクセス	$O(w + \log n)$	(15)	$O(w \log (n/w))$	(18)
ソート済み探索	$O(\log n)$	(16)	$O(\log^2 n)$	(19)

インデクス間アクセスの w は対象となるインデクスの選択範囲の幅
シーケンシャルアクセスおよび範囲アクセスの w はアクセス範囲の幅

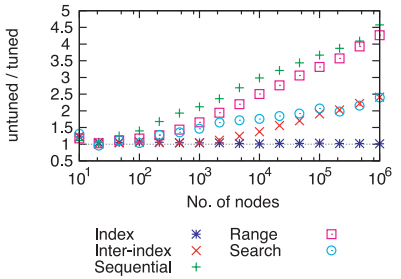
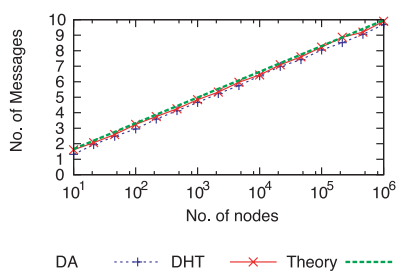


図 8　理想的ではない環境における 3.3 節の調整の効果．1,000 回の試行に対して要したメッセージ数の平均値の比（未調整の DA/DA）．各操作ごとの設定は図 9 (a)，9 (b)，9 (d)，9 (f)，9 (h) と同じ
Fig. 8　Ratio of untuned DA to tuned DA in average No. of messages in 1,000 simulations.

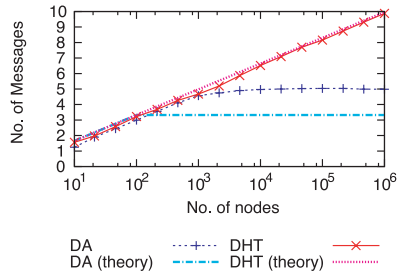
4.2 実　　験

ここまでの分析を検証するために自作のシミュレータに DA と DHT を実装し，実験を行った．Chord ネットワークは 3.1.1 項に示したように構築する．ただし，DA には 3.3 節における調整を施す．$b = 64$ とし，ノードはノード番号 $\in [0, n)$ の SHA1 によるハッシュ値の上位 64 ビットを使って ID 空間に配置する．これは理想的な Chord ネットワークではない．ただし，ノードの参加，離脱は考えない．つまり，ネットワークは静的かつ finger テーブルは完全な状態である．DHT のアイテム配置は配列名とインデクスの連結物の SHA1 によるハッシュ値の上位 64 ビットを用いて行う．各操作の開始ノードはランダムに選ぶ．

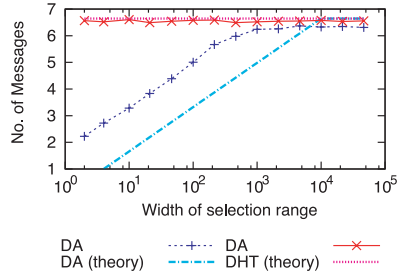
まず，3.3 節における調整の効果として，DA において Chord ネットワークを調整した場合と未調整の場合との比較結果を**図 8** に示す．縦軸は未調整の場合と調整済みの場合の性



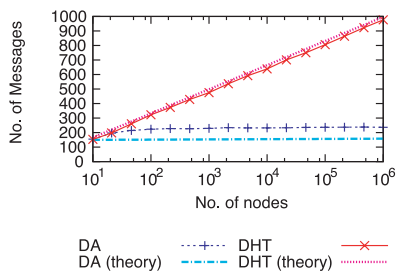
(a) インデクスアクセス．登録要素のインデクスは $[0, 1000)$



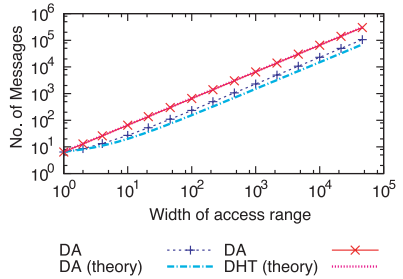
(b) インデクス間アクセス．登録要素のインデクスは $[0, 1000)$．対象インデクスの選択範囲の幅は 100



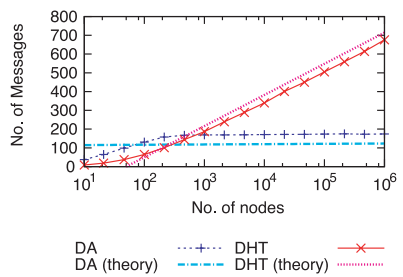
(c) インデクス間アクセス．ノード数は 10000．登録要素のインデクスは $[0, 100000)$



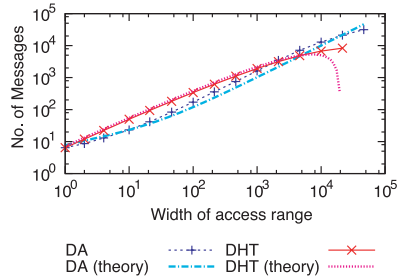
(d) シーケンシャルアクセス．登録要素のインデクスは $[0, 1000)$．アクセス範囲の幅は 100



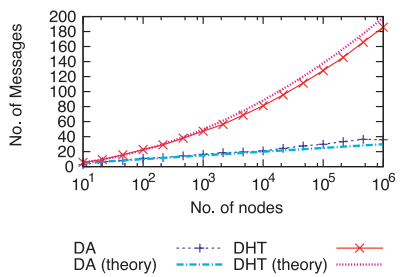
(e) シーケンシャルアクセス．ノード数は 10000．登録要素のインデクスは $[0, 100000)$



(f) 範囲アクセス．登録要素のインデクスは $[0, 1000)$．アクセス範囲の幅は 100



(g) 範囲アクセス．ノード数は 10000．登録要素のインデクスは $[0, 100000)$



(h) ソート済み探索．登録要素のインデクスは $[0, 1000)$

図 9　静的ではあるが理想的ではない環境において 1,000 回の試行に対して要したメッセージ数の平均値
Fig. 9　Average No. of messages in 1,000 simulations.

能比であり，値が大きいほど調整の効果が大きい．図 8 から，配列への操作において，調整の効果がおおよそ $\log n$ に比例することが分かる．これは，未調整の場合の追加メッセージ数を表す式 (11) が $\log n$ を含むことと一致する．また，ソート済み探索よりも，シーケンシャルアクセス，範囲アクセスの方が効果が大きいことも分かる．これはシーケンシャルアクセス，範囲アクセスでは式 (11) の k が何度も b に近い値になる可能性がある（∵ 定理 3, 4 の証明）のに対し，ソート済み探索では k が同じ値になることがない（∵ 定理 5 の証明）からである．

インデクスアクセスの結果を示す**図 9** (a) から DA のインデクスアクセスの性能が DHT と比較して低下しないことが分かる．インデクスアクセスの性能は ID アクセスの性能でもあるため，DA の配列型 P2P ネットワークが DHT の下層としても問題なく機能することを意味する．

図 9 (b) と図 9 (c) はインデクス間アクセスの結果を示す．DA の実験結果は理論と比較するとあまり良くない．これは，DA の理論性能（式 (13)）が対象とするインデクスの選択範囲として式 (3) の形を仮定して導かれているのに対し，実験では幅 w 以外はランダムな選択範囲を用いているためである．ただし，w が n の 10 分の 1 より小さいときは DHT より DA の方が効率が良い．よって，インデクスの選択範囲を n と比較して十分に小さくすることにより，その中の任意のインデクスに任意の順でアクセスする操作を効率良く行うことができる．

シーケンシャルアクセスのメッセージ数を示す図 9 (d) から，DA のシーケンシャルアクセスが n にほとんど依存しないことが分かる．理論値と比較すると，約 100 メッセージ（1

メッセージ/1 インデクス）を余分に要している．これは 3.3 節の調整後も理想的な環境と現実的なシミュレーション環境の差が完全には埋まっていないことを示す．DHT の結果は理論をよく再現している．図 9 (e) はアクセス範囲の幅 w を変化させた場合のメッセージ数であり，w の影響が理論の傾向と一致することが分かる．

範囲アクセスの結果を示した図 9 (f) においては，n がアクセス範囲の幅 w の 2 倍以上になると DA の範囲アクセスもほとんど n に依存しなくなることが分かる．しかし，n が w の 10 倍未満のときには DA は DHT より劣っている．w を変化させた場合の図 9 (g) においても，w が n の 10 分の 1 になるあたりにおいて，DA と DHT の性能が逆転している．これは，w が n に対して大きくなるにつれて，DA の範囲アクセスでは中継ノードや別の目標インデクスの管理者として同じノードを複数回訪ずれる可能性が大きくなるためである．DHT では，アルゴリズム上，同じノードを訪ずれることはない．この点はアルゴリズムの改良や切替え等の改善が必要となる．

図 9 (h) はソート済み探索のメッセージ数である．結果は DHT よりも DA の方が良い．

以上より，n への依存性が低下するという理論どおり，n が大きいほど DA が DHT に対して優位になることが分かる．

5. 考　　察

本研究では，論理配列向けに DHT のアイテム配置ルールを変更し，それに合わせてシステムを改良する方針をとった．具体的には，理想的な Chord ネットワークの分析をもとにビット列逆読み写像を導入し，それに合わせて配列操作のアルゴリズムを考案した．さらに，現実的な環境に対応するため，Chord ネットワークを調整した．この方針は本研究に限定されるものではない．様々な配列型 P2P ネットワークにビット列逆読み写像や他の配列要素の配置規則とアルゴリズムを組み合わせることにより，ほかにも DA を構築できる可能性がある．

P2P 環境の特徴として，ノードの参加，離脱が他の分散システムより激しいことがあげられるが，そこまで考慮しなかった．しかし，本手法の DA は動的な環境に対しても有効であると考えられる．

まず，Chord 上に論理配列を構築する場合と比較する．この場合，DA に対するノードの出入りの影響は致命的にはならないと考えられる．その理由は動的な環境に対応するために必要な維持手続きに関するものとアクセス手続きに関するものに分けられる．まず，維持手続きに関する理由として，DA の配列型 P2P ネットワークが Chord のそれとほぼ等しく，維持手続きもほぼ同じものが使えることがあげられる．ノード x の finger を更新するための ID $x+2^m$ へのアクセス処理が $x+2^m$ の管理者に到達した際に，successor$(x+2^m)$ へのポインタを返すか，predecessor$(x+2^m)$ へのポインタを返すかだけの違いである．アクセス手続きに関しては，環境が動的になった影響は finger テーブルの誤情報として現れる．finger としてよりふさわしいノードが出現することにより，finger が最適ではなくなることの影響は元々小さく，DA に対してのみ影響が特に大きくなるとは考えにくい．ノードの離脱により，finger が存在しないノードを指すようになる場合，アクセスする ID 間の距離の γ が小さいという DA の特徴が大きく関係してくる可能性がある．アクセス処理中のノード x から目標 ID y までの距離が

$$\underbrace{0\ldots0}_{b-k}\underbrace{1*\cdots*}_{k}{}_{(2)}$$

であるとする．i 回転送に失敗し，$i+1$ 個目の転送先候補である finger

$$x' \approx x + \underbrace{0\ldots0}_{b-k+i}\underbrace{10\ldots0}_{k-i}{}_{(2)}$$

への転送が成功した場合，$d(x',y)$ はおおよそ $d(x,y)$ のビット列において $k-i$ 桁目以上で最下位の 1 を 0 にし，$k-i$ 桁目からその 1 までの 0 を 1 に変えたものとなる．たとえば，

$$\begin{aligned} d(x,y) &= 0\ldots0100*\cdots*{}_{(2)} \\ &\quad\downarrow\ 2\text{ 回失敗の後，成功} \\ d(x',y) &\approx 0\ldots0011*\cdots*{}_{(2)}\ . \end{aligned}$$

このような場合，y にアクセスするためには，新たに現れた 1 も消す必要がある．そのため，元の $d(x,y)$ に 0 が多い方が悪影響が大きい．この影響がどの程度かは詳しく検証する必要があるが，以下のように予想できる．finger が存在しない確率が低い場合は，転送に失敗する回数は静的環境における転送回数に依存する．つまり，余分に要するメッセージ数のオーダは表 2 に示したとおりになるため，DA の優位性はなくならない．finger が存在しない確率が増すにつれ，DA にしろ DHT にしろ性能は低下していく．その際，DA においては範囲アクセスやソート済み探索のアルゴリズムを静的かつ理想的な Chord ネットワークに合わせて作成しているため，それらのアルゴリズムと DHT 用のアルゴリズムの優位性が逆転する．ただし，逆転する条件を発見し，操作を開始するノードがアルゴリズムを選択することにより DHT 未満の性能になることは防げる．

次に，レンジクエリ用のシステムにより配列を扱う場合との比較を考える．レンジクエリ用のシステムにおいて，不特定多数の配列に対する効率的な配列操作を行うためには，配列

に比例した数の P2P ネットワークや DHT 上の管理構造を維持する必要がある．それに対し，DA は，基盤とする 1 つの配列型 P2P ネットワークのみを維持することにより，不特定多数の配列に対応できる．そのため，DA の方が動的環境への適応力は高いといえる．

動的環境の影響を正確に検証することは将来の課題である．

6. 結　　論

分散配列（DA）を構築し，P2P 環境において複数要素へのアクセスをともなう操作を効率良く行うことができる論理配列を実現した．DA は分散ハッシュテーブルとともに 1 つの P2P オーバレイネットワーク上に構築できる．そのため，効率的なハッシュテーブルと論理配列をデータごとに使い分けることが可能になる．P2P データ管理システムへの DA の応用が期待される．

参 考 文 献

1) Aspnes, J. and Shah, G.: Skip graphs, *ACM Trans. Algorithms*, Vol.3, No.4 (2007).
2) Bharambe, A.R., Agrawal, M. and Seshan, S.: Mercury: Supporting scalable multi-attribute range queries, *SIGCOMM*, Yavatkar, R., Zegura, E.W. and Rexford, J. (Eds.), ACM, pp.353–366 (2004).
3) Fukuchi, D., Sei, Y. and Honiden, S.: Managing Difference-Based Objects with Sub-networks in Peer-to-Peer Environments, *OTM Workshops* (*2*), Meersman, R., Tari, Z. and Herrero, P. (Eds.), Lecture Notes in Computer Science, Vol.4806, pp.1001–1010, Springer (2007).
4) Gao, J. and Steenkiste, P.: An Adaptive Protocol for Efficient Support of Range Queries in DHT-Based Systems, *ICNP*, IEEE Computer Society, pp.239–250 (2004).
5) Karger, D.R., Lehman, E., Leighton, F.T., Panigrahy, R., Levine, M.S. and Lewin, D.: Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web, *STOC*, pp.654–663 (1997).
6) Li, D., Lu, X., Wang, B., Su, J., Cao, J., Chan, K.C.C. and Leong, H.V.: Delay-Bounded Range Queries in DHT-based Peer-to-Peer Systems, *ICDCS*, IEEE Computer Society, p.64 (2006).
7) Muthitacharoen, A., Morris, R., Gil, T.M. and Chen, B.: Ivy: A Read/Write Peer-to-Peer File System, *OSDI* (2002).
8) Plaxton, C.G., Rajaraman, R. and Richa, A.W.: Accessing Nearby Copies of Replicated Objects in a Distributed Environment, *Theory Comput. Syst.*, Vol.32, No.3, pp.241–280 (1999).
9) Ramabhadran, S. and Hellerstein, J.M.: Prefix Hash Tree: An Indexing Data Structure Over Distributed Hash Tables, Technical Report (2004).
10) Ratnasamy, S., Francis, P., Handley, M., Karp, R.M. and Shenker, S.: A scalable content-addressable network, *SIGCOMM*, pp.161–172 (2001).
11) Rowstron, A.I.T. and Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, *Middleware*, Guerraoui, R. (Ed.), Lecture Notes in Computer Science, Vol.2218, pp.329–350, Springer (2001).
12) Shao, J. and Davis, B.T.: The Bit-reversal SDRAM Address Mapping, *SCOPES*, Kavi, K.M. and Cytron, R. (Eds.), ACM International Conference Proceeding Series, Vol.136, pp.62–71 (2005).
13) Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F. and Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications, *SIGCOMM*, pp.149–160 (2001).

付　　録

A.1　ソート済み探索開始時の探索空間の求め方

ソート済み探索における探索開始時の探索空間 (x, y) の求め方を示す．前提として，対象の配列は昇順にソートされているとする．

探索開始ノードが探索値 v 以下の値の要素を管理している場合，その中の最大のインデクスが x である．同様に，v 以上の値の要素を管理している場合，その中の最小のインデクスが y である．v 以下の値の要素を 1 つも管理していない場合，x は -1 である．v 以上の値の要素を 1 つも管理していない場合，以下のように自分が管理するはずのまだ登録されていない要素を見つけることにより y を決めることができる．

要素の配置に一般のハッシュ関数を用いる場合，インデクスが $x+1, x+2, \ldots$ の要素の ID を順に計算する．自分が管理者となる最初の ID にあたったとき，そのインデクスが y である．4.1 節の DHT におけるソート済み探索の評価より，ノードが管理するインデクスの平均間隔は n となるため，n に関する平均計算量は $O(n)$ である．

定義 1 のビット列逆読み写像 f を用いる場合は $O(1)$ により y を求めることができる．探索空間の下端 $x+1$ を $x_b \ldots x_{1(2)}$ と書くことにする．

ノードが管理する ID 空間の領域が

$$[z_1 \ldots z_m \underbrace{0 \ldots 0}_{b-m}{}_{(2)}, z_1 \ldots z_m \underbrace{1 \ldots 1}_{b-m}{}_{(2)}] \tag{20}$$

の場合，$y' = x_b \ldots x_{m+1} z_m \ldots z_{1(2)}$ とおく．$y' > x$ のとき，y' が求める y である．$y' \leq x$ のときは $y'' = y' + 2^m$ とおく．$y'' \leq 2^b$ ならば，y'' が求める y である．$y'' > 2^b$ ならば，

2^b が求める y である．以上が正しいことは f によって式 (20) に属する ID に写像されるインデクスが

$$\underbrace{*\cdots*}_{b-m} z_m \ldots z_{1(2)}$$

の形のすべてのインデクスであることから分かる．

管理する ID 空間の領域が

$$[z_1 \ldots z_{m-1} 0 \underbrace{0 \ldots 0}_{b-m}{}_{(2)}, z_1 \ldots z_{m-1} 1 z'_{m+1} \ldots z'_{b(2)}) \tag{21}$$

の場合，$z = z_1 \ldots z_{m-1} 0_{(2)}$ とおく．式 (20) の場合と同様にして $A = [z2^{b-m}, (z+1)2^{b-m})$ と $B = [(z+1)2^{b-m}, (z+2)2^{b-m})$ からそれぞれ y の候補を求める．それを y' と y'' とする．$y'' < y'$ かつ $f(y'') \in$ (式 (21)) ならば y'' が求める y である．そうでない場合は y' が求める y である．以上が正しいことは

$A \subset$ (式 (21)) $\subset A \cup B$

であり，f によって A と B に写像されるインデクスの間隔が等しく 2^m であることから分かる．

管理する ID 空間の領域が

$$(z_1 \ldots z_{m-1} 0 z_{m+1} \ldots z_{b(2)}, z_1 \ldots z_{m-1} 1 \underbrace{1 \ldots 1}_{}{}_{(2)}] \tag{22}$$

の場合は式 (21) の場合と同様である．

管理する ID 空間の領域が

$$(z_1 \ldots z_{m-1} 0 z_{m+1} \ldots z_{b(2)}, z_1 \ldots z_{m-1} 1 z'_{m+1} \ldots z'_{b(2)})$$

の場合，この領域は式 (22) と式 (21) の領域に分けられる．それぞれから y の候補を上述の方法で求めると，小さい方が求める y である．

管理する ID 空間の領域が $2^b - 1$ と 0 を含む場合はその間で区切り，それぞれから y の候補を上述の方法で求める．その小さい方が求める y である．

(平成 20 年 5 月 19 日受付)
(平成 20 年 11 月 5 日採録)

福地　大輔

2007 年東京大学理学部情報科学科卒業．同年東京大学大学院情報理工学系研究科コンピュータ科学専攻修士課程進学，現在に至る．文部科学省国立情報学研究所アーキテクチャ研究系リサーチアシスタント．P2P 技術の研究に従事．

クリスチャン　ソッメル

2006 年スイス連邦工科大学チューリッヒ校コンピュータ科学修士号．2007 年東京大学大学院情報理工学系研究科コンピュータ科学専攻博士課程進学，現在に至る．2008 年より東大フェローシップ．文部科学省国立情報学研究所アーキテクチャ研究系リサーチアシスタント．理論系コンピュータ科学，主にアルゴリズムとグラフ理論の研究に従事．

清　　雄一（学生会員）

2004 年東京大学工学部システム創成学科卒業．2006 年東京大学大学院情報理工学系研究科コンピュータ科学専攻修士課程修了．同年同博士課程進学，現在に至る．文部科学省国立情報学研究所アーキテクチャ研究系リサーチアシスタント．エージェント技術，センサネットワークの研究に従事．

本位田真一（正会員）

1978 年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て 2000 年より国立情報学研究所教授，2004 年より同研究所アーキテクチャ科学研究系研究主幹を併任，現在に至る．2001 年より東京大学大学院情報理工学系研究科教授を兼任，現在に至る．2002 年 5 月〜2003 年 1 月英国 UCL ならびに Imperial College 客員研究員．2005 年度パリ第 6 大学招聘教授．早稲田大学客員教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．ソフトウェア工学，エージェント技術，ユビキタスコンピューティングの研究に従事．IEEE，ACM 等各会員，日本ソフトウェア科学会理事，情報処理学会理事を歴任．日本学術会議連携会員．