Vol.2013-OS-126 No.22
2013/8/1

柔軟な経路表に基づく範囲検索可能な構造化オーバレイ

宮尾 武裕[1,a)]　長尾 洋也[1,b)]　首藤 一幸[1,c)]

概要：構造化オーバレイでは，一般的にノード間の ID 距離に基づいて経路表を構築する．これらのアルゴリズムは，ノード ID の偏りが小さい状況下で経路長が短くなるよう設計されている．そのため，ノード ID の偏りが大きい時，ルーティングにおける経路長が長くなるという問題がある．そこで，我々はノード ID の偏りが大きい時でも，経路長が長くならない構造化オーバレイ FRT-Chord# を提案する．FRT-Chord# を分散ハッシュテーブル（DHT）において利用することで，範囲検索を効率よく行うことが可能となる．また，FRT-Chord# は Flexible Routing Tables（FRT）と呼ばれる構造化オーバレイ設計手法に基づき設計されているため，容易な拡張性や任意の経路表サイズ設定などの FRT 由来の特長を継承している．

A Structured Overlay for Supporting Range Queries based on Flexible Routing Tables

Takehiro Miyao[1,a)]　Hiroya Nagao[1,b)]　Kazuyuki Shudo[1,c)]

Abstract: Structured overlays construct routing tables based on differences of node IDs. Those routing algorithms are designed to achieve small hop counts on condition that node IDs are uniformly distributed. Otherwise, hop counts get larger. A new routing algorithm, FRT-Chord#, that we propose in this paper keeps hop counts small even though node IDs are not uniformly distributed. The algorithm efficiently carries out range queries when it is employed as the base of a DHT. The algorithm is based on Flexible Routing Tables (FRT), a method for designing structured overlays and it keeps desirable features derived from FRT, for example, extensibility and flexibility such as arbitrary routing table size.

1. はじめに

分散ハッシュテーブル（Distributed Hash Table，DHT）は連想配列を多数のマシンで管理する技術である．DHT では，データやノードには ID が割り当てられ，ノードは ID に基づきデータの保存を担当する ID 空間（担当領域）が決定し，その ID 空間内に存在するデータを保存する．また，たくさんのマシンを連携させるためにオーバレイネットワークと呼ばれるアプリケーションレベルの仮想ネットワークを構築する．

オーバレイの中でも，数学的・論理的なルールに基づいて構築したネットワークを構造化オーバレイと呼ぶ．今日まで，Chord [1]，Kademlia [2]，Pastry [3] といった構造化オーバレイにおけるルーティングアルゴリズムが提案されてきた．構造化オーバレイでは，ノードは隣接ノードへのポインタのリスト（経路表）を構築する．リクエストメッセージを目的のノードに到達させるために，それぞれのノードは経路表を基にメッセージの転送を繰り返す．

一般的な構造化オーバレイでは，ノードはノード間の ID 距離に基いて経路表を構築する．その際，ID 空間上においてノード ID の偏りが小さいことを前提としてアルゴリズムが設計されている．そのため，ノード ID の偏りが大きい状況で，これらのアルゴリズムを利用すると，目的のノードに到達するまでのメッセージ転送回数（経路長）が長くなる．以下に挙げる応用のためには，ノード ID の偏

[1] 東京工業大学
Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, Tokyo, JAPAN
[a)] takehiro.miyao@is.titech.ac.jp
[b)] hiroya.nagao@is.titech.ac.jp
[c)] shudo@is.titech.ac.jp

りが大きい状況でも経路長が長くならない構造化オーバレイが必要である．

ノード ID の偏りが大きい状況でも経路長が長くならない構造化オーバレイの応用のひとつに，DHT における効率的な範囲検索がある．DHT における効率的に範囲検索を行う方法として，データの連続性を保ったままデータ ID を割り当てるという方法がある．そのような方法でデータ ID を割り当てると，データ ID は ID 空間において偏りが大きくなることが想定される．そこで，leave-join などの手法を用いてデータ ID の偏りに応じてノード ID を割り当てることで，負荷分散を実現することができる．その際，経路長が長くならないように，ノード ID の偏りが大きい状況でも経路長が長くならない構造化オーバレイを利用する．

他の応用として，ノード数が十分大きい数でない状況での利用がある．ノード ID の偏りが小さいことを前提とした構造化オーバレイでは，SHA-1 などの暗号学的ハッシュ関数を用いてノード ID を割り当てることが多い．そのため，ノード数が十分大きい場合では，ノード ID の偏りが小さいが，そうでない場合はノード ID の偏りが大きくなることが考えられる．また，別の応用としてノード ID の割り当てに他の要素を考慮したい状況での利用がある．範囲検索ではデータ ID の偏りに応じでノード ID が割り当てられたが，それ以外にも地理的情報などさまざまな他の要素を考慮したい状況が考えられる．ノード ID の割り当てに他の要素を考慮すると，その要素の偏りがノード ID にも影響する．このように，様々な応用において，ノード ID の偏りが大きい状況でも経路長が長くならない構造化オーバレイが必要である．

Chord$^{\#}$ [4]，Mercury [5] といったノード ID の偏りが大きい状況でも経路長が長くならない構造化オーバレイが提案されてきた．これらのアルゴリズムは効率的な範囲検索などのために利用することが可能であるが，経路表管理コストが大きい，経路表サイズを任意に設定できない，拡張が難しいといった問題がある．それらの問題点は，Flexible Routing Tables（FRT）[6] という構造化オーバレイ設計手法に基づきアルゴリズムを設計することで解決できる．

現在まで，FRT-Chord [6]，FRT-2-Chord [7] といった FRT に基づく構造化オーバレイが提案されてきた．FRT では，経路表の順序関係 $\leq_{\mathrm{RT}}$ と呼ばれる，経路表がとりうるすべてのパターンの優劣を比較する順序関係を定義することで，構造化オーバレイを設計することができる．そのようにして設計された構造化オーバレイは，容易な拡張性や任意の経路表サイズ設定などの特長を継承する．FRT には PFRT [8] や GFRT [6] といった拡張手法が存在し，これらを適用するだけで，FRT に基づく構造化オーバレイはネットワーク近接性やノードグループを考慮できるようになる．また，ネットワークが安定していたりノード数が少ない時，すべてのノードを経路表にのせることで目的ノードに直接到達するルーティングを行うことができる．

しかし，FRT-Chord や FRT-2-Chord はノード間の ID 距離に基づき経路表の順序関係 $\leq_{\mathrm{RT}}$ を定義しているため，ノード ID の偏りが大きい状況では，経路長が長くなる．そこで我々は，ノード ID の偏りが大きい状況でも経路長が長くならない構造化オーバレイ FRT-Chord$^{\#}$ を提案する．FRT-Chord$^{\#}$ では，隣接ノードの経路表に基づき経路表の順序関係 $\leq_{\mathrm{RT}}$ を定義する．隣接ノードの経路表に基いた経路表の順序関係 $\leq_{\mathrm{RT}}$ により自ノードの経路表を評価するため，ノード ID の偏りが大きい状況でも経路長が長くならない経路表を構築することができる．また，FRT-Chord$^{\#}$ は FRT が持つ望ましい特長を継承する．

2. 構造化オーバレイのルーティングアルゴリズム

この節では，まず我々が FRT-Chord$^{\#}$ を設計するにあたり ID 空間や経路表の構造などを採用した Chord [1] を説明する．次に，本研究と同様にノード ID の偏りが大きい状況でも経路長が長くならない構造化オーバレイとして Chor$^{\#}$ [4] を説明する．最後に，我々が構造化オーバレイのルーティングアルゴリズムを設計するにあたり利用した設計方法である FRT [6] を説明する．

2.1 Chord

Chord は構造化オーバレイにおけるルーティングアルゴリズムのひとつである．Chord は m ビットのリング状 ID 空間をもつ．つまり，ID x から ID y への ID 距離は，

$$d(x,y) = \begin{cases} y-x, & x < y, \\ y-x+2^m, & y \leq x, \end{cases} \quad (1)$$

と定義される．ノード n が保存するデータ ID 領域をノード n の担当領域と呼び，ノード n の担当領域は，ノード n への ID 距離が最も小さい ID の集合である．また，反対にノード n の担当領域の ID にとって，ノード n は担当ノードである．

Chord には，successor list，predecessor，finger table の 3 つの経路表がある．ID 空間上でノード n から時計回りに進んだとき，最初に出会うノードを successor と呼び，successor list とは時計回りに進んで出会うノードを順番に一定数 c だけ保持する経路表である．反時計回りに進んだとき，最初に出会うノードを predecessor と呼び，そのノードをエントリとして経路表に保持する．finger table とはノード n から 2^i（$i = 0, 1, ..., m-1$）の距離だけ離れた ID の担当ノードを i 番エントリとして保持する経路表である．

Chord では greedy routing と呼ばれる方法で，転送エントリを選択する．greedy routing では，目的 ID に最も近づくノードに転送するよう経路表からエントリを選択する．greedy routing を利用してメッセージ転送を繰り返す

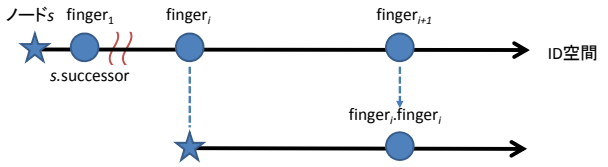

図 1 Chord# における経路表
Fig. 1 A routing table in Chord#

ことで，担当ノードの predecessor に到達する．このとき，successor にメッセージを転送することで，目的 ID の担当ノードにメッセージが到達する．

Chord では，successor list によりメッセージの目的ノードへの到達性を保証する．また，ノード ID の割り当てに SHA-1 などのハッシュ関数を用いて，ノード ID の偏りが小さい状況のもとでは，finger table および greedy routing により経路長を $O(\log N)$ に抑えることができる．

2.2 Chord#

Chord# はノード ID の偏りが大きい状況でも経路長が長くならない構造化オーバレイにおけるルーティングアルゴリズムのひとつである．ID 空間およびノードの担当領域の定義などは Chord と同様であり，経路表の構築方法が異なる．Chord# の経路表は Chord における finger table をノード ID の偏りが小さいという前提なしに利用できるように変化させたものである．finger_i を finger table の要素とし，finger_i のポインタが指すノードの経路表における finger_j を $\mathrm{finger}_i.\mathrm{finger}_j$ と表すとすると，Chord# の経路表は，

$$\mathrm{finger}_i = \begin{cases} \text{successor}, & i = 0, \\ \mathrm{finger}_{i-1}.\mathrm{finger}_{i-1}, & i > 0, \end{cases} \tag{2}$$

を満たす経路表を構築する（**図 1**）．この経路表によりノード ID の偏りが大きい状況でも，高確率で経路長が $\log N$ に抑えることができる．

2.3 FRT

Flexible Routing Tables（FRT）は，構造化オーバレイの設計手法である．FRT に基づくアルゴリズムとして FRT-Chord [6] や FRT-2-Chord [7] が存在する．また，PFRT [8] や GFRT [6] といった FRT の拡張手法があり，これらを適用するだけで，FRT に基づく構造化オーバレイはネットワーク近接性やノードグループを考慮できるようになる．

FRT では，経路表の順序関係 $\leq_{\mathrm{RT}}$ と sticky entry を設計することで，構造化オーバレイを設計することができる．経路表の順序関係 $\leq_{\mathrm{RT}}$ とは，経路表の優劣を決めるためのすべての経路表パターンの集合における全順序関係である．$E \leq_{\mathrm{RT}} F$ は，経路表 E が経路表 F より優れていることを表す．

sticky entry とは，エントリを削除する際，削除候補から除外するエントリの集合である．sticky entry は到達性保証のためやアルゴリズムの拡張のために用いられる．

2.3.1 FRT-Chord

FRT-Chord は FRT に基づき設計された構造化オーバレイである．ID 空間およびノードの担当領域の定義などは Chord と同様であり，経路表の構築方法が異なる．

ノードはそれぞれ経路表 $E = \{e\}$ を持ち，エントリはそれぞれ ID e.id およびアドレス情報 e.addr を管理する．本論文中では，e.id を e と略し，また e をそのポインタが指すノードを表すことがある．経路表は，ノード s から ID 距離が近い順に並べられている（$d(s, e_i) < d(s, e_{i+1})$）．

FRT-Chord における経路表の順序関係を定義するため，まずノード n における各エントリ e_i（predecessor を除く）の最悪の短縮倍率 $r_i(E)$ を計算する．短縮倍率とは，エントリ e_i にメッセージを転送する際の転送前後の残り ID 距離の割合である．つまり，最悪の短縮倍率 $r_i(E)$ は，

$$r_i(E) = \frac{d(e_i, e_{i+1})}{d(s, e_{i+1})} \qquad (i = 1, 2, \ldots |E| - 1). \tag{3}$$

となる．

$\{r_{(i)}(E)\}$ を $\{r_i(E)\}$ を降順に並べた集合とおく．このとき，

$$E \leq_{\mathrm{ID}} F \Longleftrightarrow \{r_{(i)}(E)\} \leq_{\mathrm{dic}} \{r_{(i)}(F)\}, \tag{4}$$

となるよう，経路表の順序関係 $\leq_{\mathrm{RT}}$ を定義する．また，$\leq_{\mathrm{dic}}$ は辞書式順序関係であり，

$$\begin{aligned} &\{a_i\} <_{\mathrm{dic}} \{b_i\} \Leftrightarrow a_k < b_k \qquad (k = \min\{i \mid a_i \neq b_i\}), \\ &\{a_i\} =_{\mathrm{dic}} \{b_i\} \Leftrightarrow a_i = b_i, \\ &\{a_i\} \leq_{\mathrm{dic}} \{b_i\} \Leftrightarrow (a_i <_{\mathrm{dic}} b_i) \cup (a_i =_{\mathrm{dic}} b_i). \end{aligned} \tag{5}$$

と定義される．

FRT-Chord における sticky entry は Chord における successor list および predecessor である．つまり，$e_1, e_2, ..., e_c$ および $e_{|E|}$ である．ただし，c は successor list の大きさである．

FRT では，経路表にエントリを追加する操作を Entry Leaning と呼ぶ．FRT-Chord では，ノードが知り得たエントリをすべて経路表に追加する．

FRT では，経路表のエントリ数 $|E|$ がエントリ上限数 L を超えた時，経路表からエントリを削除する操作を Entry Filtering と呼ぶ．FRT-Chord では，次の操作を行い経路表からエントリを削除する．ただし，C は削除する可能性のあるエントリ集合である．

(1) $C \leftarrow E$
(2) C から sticky entry を取り除く．
(3) 次の式を満たす削除エントリ e_{r} を経路表 E から削除する．

$$E \setminus \{e_{\mathrm{r}}\} \leq_{\mathrm{RT}} E \setminus \{e\}, \qquad \mathrm{e} \in C \tag{6}$$

FRT-Chord では，正規化を行うことにより，$O(1)$ の計算量で削除エントリを見つけることができる．

3. FRT-Chord#

我々はノードの ID の偏りが大きい状況でも経路長が長くならない構造化オーバレイ FRT-Chord# を提案する．一般的に，ノード ID の偏りが小さいことを前提とした構造化オーバレイは ID 距離に基づき経路表を構築する．しかし，ノード ID の偏りが大きい状況でも経路長が長くならない構造化オーバレイでは，ID 距離に基づき経路表を構築することが難しい．そこで，FRT-Chord# は，ノード ID の偏りが大きい状況でも経路長が長くならないように，隣接ノードの経路表を用いて経路表を構築する．FRT-Chord# では，経路表エントリ e は ID e.id とアドレス e.addr 以外にそのノード e の経路表情報 e.RT も管理する．また，ノード e における k 番目の経路表エントリを $e.\mathrm{e_k}$ と表記する．

FRT-Chord# は FRT に基づく構造化オーバレイである．経路表の順序関係 $\leq_{\mathrm{RT}}$ および sticky entry を定義することにより設計することができる．また，FRT-Chord# は FRT に基づくアルゴリズムであるため，容易な拡張性や任意の経路表サイズ設定などの FRT 由来の特長を継承している．

3.1 経路表の順序関係

経路表の順序関係 $\leq_{\mathrm{RT}}$ を定義するために，まずノード s から ID x への距離を表す指標として**転送エントリ番号** $n_s(x)$ を定義する．転送エント番号 $n_s(x)$ とは，ノード s が ID x へのメッセージを転送するエントリを，経路表においてノード s から何番目に近いかを表す番号である．FRT-Chord# では，転送エントリの選択に greedy routing を利用し，また $d(s, e_i) < d(s, e_{i+1})$ を満たすようエントリの添字をつけているので，転送エントリ番号 $n_s(x)$ は，

$$n_s(x) = \max\{i \mid d(n, e_i) < x\} \tag{7}$$

と定義される．転送エントリ番号 $n_s(x)$ が小さいほど目的 ID x までが小さいことを表し，またノード s の担当領域の ID への転送エントリ番号は 0 である．

次に**転送エントリ番号減少値** $f_s(x)$ を定義する．転送エントリ番号減少値 $f_s(x)$ は，ノード n が目的 ID x へのメッセージをエントリ e_i に転送する前後における，転送エントリ番号の減少値である（**図 2**）．つまり，転送エントリ番号減少値 $f_s(x)$ は，

$$f_s(x) = n_s(x) - n_{e_i}(x) \tag{8}$$

と定義される．ノード n は e_i に転送するので，

$$n_s(x) = i \tag{9}$$

となり，転送エントリ番号減少値 $f_s(x)$ は

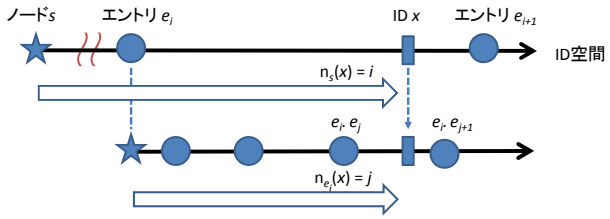

図 2 転送エントリ番号減少値
Fig. 2 Reduction value of forwarding entry

$$f_s(x) = i - n_{e_i}(x) \tag{10}$$

を満たす．転送エントリ番号減少値 $f_s(x)$ が大きいほど，転送したノードでの転送エントリ番号が小さくなるので，目的 ID x により近づけることを表す．

経路表の順序関係 $\leq_{\mathrm{RT}}$ においてエントリ e_i を評価するための評価値として，**最悪の転送エントリ番号減少値** $\hat{f}_i(E)$ を用いる．最悪の転送エントリ番号減少値 $\hat{f}_i(E)$ とは，エントリ e_i に転送する目的 ID x の中で，転送エントリ番号減少値 $f_i(E)$ の最大値である．つまり，最悪の転送エントリ番号減少値 $\hat{f}_i(E)$ は，

$$\hat{f}_i(E) = \max_x f_s(x) \tag{11}$$

である．FRT-Chord# の ID 空間はリング状であり，greedy routing を利用しているため，エントリ e_i における最悪の転送エントリ番号減少値 $\hat{f}_i(E)$ となる目的 ID は e_{i+1}.id である．よって，最悪の転送エントリ番号減少値 $\hat{f}_i(E)$ は，

$$\hat{f}_i(E) = f_s(e_{i+1}.\mathrm{id}) \tag{12}$$

となる．最悪の転送エントリ番号減少値 $\hat{f}_i(E)$ の値が大きいほど，エントリ e_i が転送する ID 空間が狭いことを意味する．また，反対に最悪の転送エントリ番号減少値 $\hat{f}_i(E)$ の値が小さいほど，エントリ e_i が転送する ID 空間が広いことを意味する．そこで，経路表の順序関係 $\leq_{\mathrm{RT}}$ は，各エントリの転送エントリ番号減少値のばらつきが小さいほど優れた経路表であると定義する．つまり，経路表の順序関係 $\leq_{\mathrm{RT}}$ は，

$$E \leq_{\mathrm{RT}} F \Leftrightarrow \{\hat{f}_{(i)}(E)\} \leq_{\mathrm{dic}} \{\hat{f}_{(i)}(F)\}, \tag{13}$$

となるよう定義される．ただし，$\{\hat{f}_i(E)\}$ を降順に並べた列を $\{\hat{f}_{(i)}(E)\}$ とする．

3.2 sticky entry

FRT-Chord# において，到達性保証のための sticky entry は FRT-Chord と同様である．

3.3 Entry Leaning

FRT において，経路表にエントリを追加する操作のことを Entry Learning と呼ぶ．ノードは他ノードと通信し

た際にそのノードのノード情報を手に入れる．FRT では，ノードがノード情報を知り得たノードをすべて経路表に追加する．

3.4 Entry Filtering

FRT において，経路表のエントリ数 $|E|$ がエントリ上限数 L を超えた時に削除する操作を Entry Filtering と呼ぶ．Entry Filtering では，経路表の順序関係 $\leq_{\mathrm{RT}}$ と sticky entry を用いて，次の手順で削除エントリ e_{r} を見つけ，経路表 E から削除する．ただし，C は削除をする可能性のあるエントリ集合である．

(1) $C \leftarrow E$
(2) C から sticky entry を取り除く．
(3) 次の式を満たす削除エントリ e_{r} を経路表 E から削除する．

$$E \setminus \{e_{\mathrm{r}}\} \leq_{\mathrm{RT}} E \setminus \{e\}, \qquad \mathrm{e} \in C \tag{14}$$

それぞれのエントリ e_i を削除した後の経路表候補 $E \setminus \{e_i\}$ の集合を経路表の候補集合と呼び，経路表の順序関係 $\leq_{\mathrm{RT}}$ に基づき優れた経路表順に経路表の候補 $E \setminus \{e_i\}$ をソートする．それにより，手順 3 において計算量 $O(1)$ で削除エントリ e_{r} を見つけることができる．要素数 L の集合をソートするための計算量は $O(L \log L)$ である．また，二つの経路表を経路表の順序関係 $\leq_{\mathrm{RT}}$ に基づき優劣を比較するための計算量は，辞書式順序関係 $\leq_{\mathrm{dic}}$ を利用しているため，$O(L)$ である．つまり，削除エントリ e_{r} を見つけるための計算量は $O(L^2 \log L)$ となる．

4. 評価

オーバレイ構築ツールキットである Overlay Weaver [9][10] 上に，FRT-Chord$^{\#}$ を実装した．一台のマシン上でエミュレーションによる実験・評価を行った．以下の環境で実験を行った．

- Overlay Weaver 0.10.1
- OS: Linux 2.6.35.10-74.fc14.x86_64
- CPU: Intel Xeon E5620 (2.40 GHz)
- Java SE 6 Update 22

ノード ID の分布が Zipf 分布（$\alpha = 0.95, 0.7$）および一様分布に従うよう，ノードに ID を割り当てた．Zipf 分布とは分布関数 $x^{-\alpha}$ に従う分布関数である．ノード ID と同様の分布に従う ID の検索を行うことで，ネットワークを構築し，経路長を計測した．

4.1 他アルゴリズムとの比較

FRT-Chord$^{\#}$ と Chord，FRT-Chord の平均経路長および 99 パーセンタイルの経路長を比較した．実験でのパラメータは以下の通りである．

- ノード数: 10,000

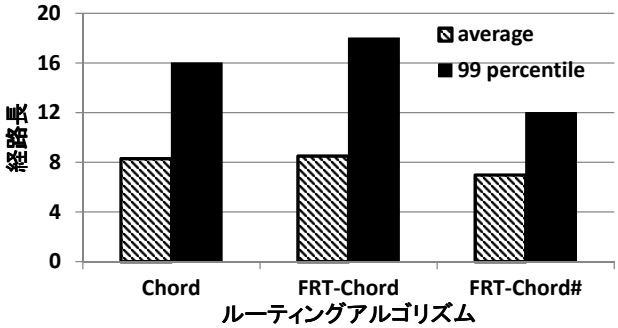

図 3 Zipf 分布（$\alpha = 0.95$) における経路長
Fig. 3 Route lengths at Zipf distribution ($\alpha = 0.95$)

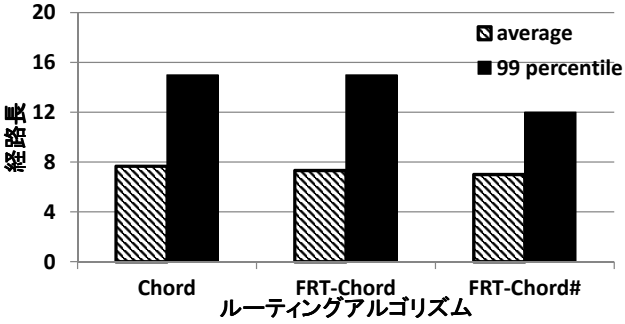

図 4 Zipf 分布（$\alpha = 0.7$) における経路長
Fig. 4 Route lengths at Zipf distribution ($\alpha = 0.7$)

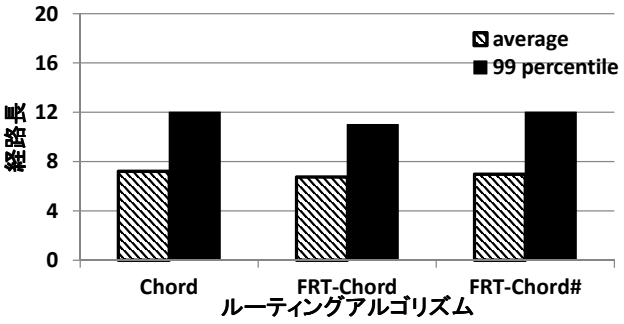

図 5 一様分布における経路長
Fig. 5 Route lengths at uniform distribution

- 経路表サイズ: 16

図 3 はノード ID が Zipf 分布（$\alpha = 0.95$）に従うときの平均経路長および 99 パーセンタイルの経路長である．FRT-Chord$^{\#}$ の平均経路長は 6.98 であり，Chord および FRT-Chord では 8.30, 8.50 である．また，99 パーセンタイルの経路長は FRT-Chord$^{\#}$ では 12 であり，Chord，FRT-Chord では 16，18 である．このことから，ノード ID の偏りが大きいとき，FRT-Chord$^{\#}$ は Chord や FRT-Chord より経路長が小さいことがわかる．

図 5 はノード ID が一様分布に従う時の平均経路長および 99 パーセンタイルの経路長である．FRT-Chord$^{\#}$ の平均経路長は 6.97 であり，Chord，FRT-Chord では 7.21，6.76 である．また，99 パーセンタイルの経路長は FRT-Chord$^{\#}$

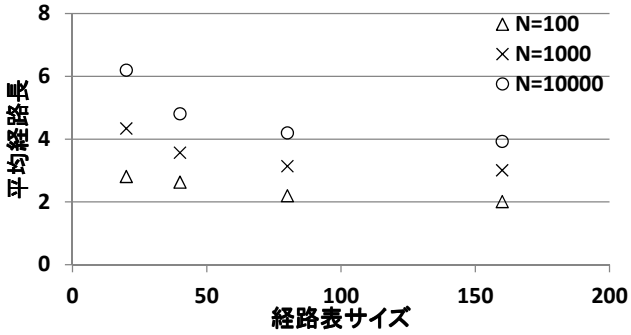

図 6 経路表サイズにおける平均経路長
Fig. 6 Route lengths in each routing table size

では 12 であり，Chord，FRT-Chord では 12，11 である．このことから，FRT-Chord# はノード ID の偏りが小さい時でも，FRT-Chord よりほとんど経路長が長くならないことがわかる．

今回の実験では，一様分布，Zipf 分布（$\alpha = 0.7$），Zipf 分布（$\alpha = 0.95$）の順にノード ID の偏りが長くなっていく．図 3，**図 4**，図 5 を見ると，Chord および FRT-Chord ではノード ID の偏りが大きくなるにつれて経路長が長くなる．その一方，FRT-Chord# では平均経路長および 99 パーセンタイルの経路長はほぼ同じ値である．つまり，FRT-Chord# はノード ID の偏り大きい状況でも経路長が長くならないということを示せた．

4.2 経路表サイズの変更

FRT に基づくアルゴリズムに継承される特長のひとつに任意の経路表サイズ設定がある．この節では，それぞれの経路表サイズにおける平均経路長を比較した．

図 6 は経路表サイズを変化させた時の FRT-Chord# の平均経路長である．ノード数が 10,000 の結果に着目する．経路表サイズが 20 のとき，平均経路長は 6.20 である．経路表サイズが大きくなるにつれて，平均経路長は減少し，経路表サイズが 160 のとき，3.93 となる．このことから，経路表サイズを大きくすることで，経路長が短くなることを確認した．

FRT には，また，ノード数が経路表サイズより小さい場合にはすべてのノードを経路表にのせられる，という特長がある．図 6 における，ノード数が 100 の結果に着目する．経路表サイズが 20 のとき，平均経路長は 2.81 である．経路表サイズが大きくなるにつれて，平均経路長は減少し，経路表サイズがノード数を超えた時，平均経路長は 2.00 となる．FRT-Chord# の担当ノードの定義は Chord と同じであるため，経路表にすべてのノードがのっているとき，ほとんどの場合で経路長が 2 となる．このことから，ノード数が経路表サイズより小さい時，すべてのノードが経路表にのっていることを確認した．

5. まとめ

我々はノード ID の偏りが大きい状況でも経路長が長くならない構造化オーバレイ FRT-Chord# を提案した．ノード ID の偏りが大きい状況でも経路長が長くならない構造化オーバレイは，DHT における効率的な範囲検索をはじめ，さまざまな応用が存在する．また，FRT-Chord# は FRT に基づく構造化オーバレイであるため，容易な拡張性や任意の経路表サイズ設定などの特長を継承している．

実験により，ノード ID の偏りが大きい時，FRT-Chord# は FRT-Chord より経路長が短縮していること，また，FRT-Chord# はノード ID の偏りが小さい時でも，FRT-Chord と経路長がほぼ同じであることを確認した．さらに，ノード ID の偏りによって FRT-Chord# の経路長が長くならないことを確認した．

謝辞 本研究は JSPS 科研費 24650025，25700008 の助成を受けたものである．

参考文献

[1] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F. and Balakrishnan, H.: Chord: A scalable peer-to-peer lookup protocol for Internet applications, *Networking, IEEE/ACM Transactions on*, Vol. 11, No. 1, pp. 17–32 (2003).
[2] Maymounkov, P. and Mazieres, D.: Kademlia: A peer-to-peer information systems based on the XOR metric, *Proc. IPTPS ' 02*, MA, USA, pp. 53–65 (2002).
[3] Rowstron, A. and Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, *Proc. IFIP/ACM Middleware 2001*, Heidelberg, Germany, pp. 329–350 (2001).
[4] Datta, A., Hauswirth, M., John, R., Schmidt, R. and Aberer, K.: Range queries in trie-structured overlays, *Proc. IEEE P2P '05*, IEEE, pp. 57–66 (2005).
[5] Bharambe, A. R., Agrawal, M. and Seshan, S.: Mercury: Supporting scalable multi-attribute range queries, *ACM SIGCOMM Computer Communication Review*, Vol. 34, No. 4, ACM, pp. 353–366 (2004).
[6] Nagao, H. and Shudo, K.: Flexible Routing Tables: Designing routing algorithms for overlays based on a total order on a routing table set, *Proc. IEEE P2P '11*, Kyoto, Japan, pp. 72–81 (2011).
[7] 安藤泰弘, 長尾洋也, 宮尾武裕, 首藤一幸：FRT-2-Chord: one-hop と multi-hop のシームレスな移行が可能かつ経路表に対称性を持つ DHT アルゴリズム，情報処理学会論文誌 コンピューティングシステム (ACS)， Vol. 5, No. 5, pp. 66–75 (2012).
[8] Miyao, T., Nagao, H. and Shudo, K.: A method for designing proximity-aware routing algorithms for structured overlays, *Proc. IEEE ISCC '13*, Split, Croatia (2013).
[9] 首藤 一幸：Overlay Weaver, http://overlayweaver.sourceforge.net/.
[10] Shudo, K., Tanaka, Y. and Sekiguchi, S.: Overlay Weaver: An overlay construction toolkit, *Comput. Comm.*, Vol. 31, No. 2, pp. 402–412 (2008).