構造化オーバレイにおける経路表の順序関係に基づくネットワーク近接性の考慮手法

宮尾武裕[†]　長尾洋也[†]　首藤一幸[†]

Chord をはじめとする従来の分散ハッシュテーブル（DHT）では，経路表の管理をランダムに割り当てられたノード ID に基づき厳密に行っている．そのため，ノード間の通信遅延といったノード ID 以外の情報を考慮することが難しい．そこで，本研究ではノード ID 以外を考慮しやすいルーティングアルゴリズム設計手法である柔軟な経路表（FRT）に基づき，ネットワーク近接性を考慮したアルゴリズムを設計する手法（PFRT）を開発した．PFRT は，FRT における ID に関する経路表の順序関係 $\leq_{\mathrm{ID}}$ を拡張した，ネットワーク近接性を考慮した経路表の順序関係 $\leq_{\mathrm{ID+PR}}$ を用いる．PFRT に基づくルーティングアルゴリズムは，この順序関係 $\leq_{\mathrm{ID+PR}}$ に基づき経路表の改良を繰り返すことで経路表を構築していく．PFRT に基づく DHT アルゴリズム PFRT-Chord は，Chord よりルーティングにおける平均所要時間が約 40% 減少し，PFRT の有効性を示すことができた．

Proximity-aware Structured Overlays Based on an Order of Routing Tables

Takehiro Miyao,[†] Hiroya Nagao[†] and Kazuyuki Shudo[†]

Existing DHT algorithms such as Chord prescribe which nodes are held in a routing table based on the nodes' ID. The ID restriction is a major obstacle to exploitation of network proximity. We designed Proximity-aware Flexible Routing Tables (PFRT), a routing algorithm designing methodology by expanding Flexible Routing Tables (FRT). We define an order of routing tables $\leq_{\mathrm{ID+PR}}$ for PFRT based on FRT's basic order $\leq_{\mathrm{ID}}$. PFRT updates a routing table repeatedly according to order $\leq_{\mathrm{ID+PR}}$. We designed PFRT-Chord, a DHT algorithm based on PFRT. An experiment result shows that average routing latency in PFRT-Chord is about 40% less than Chord.

1. はじめに

Gnutella や BitTorrent などのさまざまな P2P アプリケーションが存在する．P2P システムはスケーラビリティや耐故障性などの点で優れている．P2P ネットワークのように，物理的に接続された実ネットワーク上に独自のトポロジで構築されたアプリケーションレベルのネットワークをオーバレイネットワークと呼ぶ．オーバレイネットワークによって，非集中であっても多数のノードをうまく連携させることができる．

オーバレイネットワークは二つに分類することができる．一つは非構造化オーバレイネットワークであり，もう一つは構造化オーバレイネットワークである．非構造化オーバレイネットワークでは，オーバレイネットワーク構築時のルールが一定の数学的ルールに従わずにネットワークを構築し，その技術は Gnutella や BitTorrent などのアプリケーションに用いられている．逆に，構造化オーバレイネットワークでは一定の数学的ルールに従いネットワークを構築する．

構造化オーバレイネットワークには Chord[1)]，Kademlia[2)]，Pastry[3)] などのアルゴリズムが存在し，近年多くの研究がなされている．これらのアルゴリズムは分散ハッシュテーブル（Distributed Hash Table, DHT）と呼ばれる．DHT とは連想配列を複数のノードで管理する技術である．非構造化オーバレイネットワークでは発見のためにフラッディング等が必要となるが，これらの DHT では，ノード数を N とすると $O(\log N)$ の経路長で効率のよいルーティングを行うことが可能である．大規模な P2P システムではノードが世界中に存在するため，一対一でのノード間の通信遅延はさまざまなものとなる．しかし，従来の DHT

† 東京工業大学
Tokyo Institute of Technology

の多くではハッシュ関数で割り当てられたノード ID に基づくルーティングを行うため，ネットワーク近接性が考慮されていない．その結果，ルーティングにおいて通信遅延が大きなノードと通信を行うことが頻繁に起こり，ルーティングにおける所要時間を小さくすることができないという問題がある．

ネットワーク近接性を考慮するための方針として，Proximity Neighbor Selection（PNS），Proximity Route Selection（PRS），Proximity Identifier Selection（PIS）が提案されてきた[4]．PNS は経路表構築時に，PRS は経路選択時に，PIS はノード ID 決定時にそれぞれネットワーク近接性を考慮する方法である．本研究では，PNS に基づいてネットワーク近接性を考慮する．

Chord，Kademlia といった従来の DHT では，ノード ID のみによって経路表に追加するノードを厳密に選択しているので，PNS に基づいてネットワーク近接性を考慮することが難しい．そこで，ノード ID 以外を考慮しやすいルーティングアルゴリズム設計手法である柔軟な経路表（Flexible Routing Tables, FRT）[5] を利用する．FRT では，ID に関する経路表の順序関係 $\leq_{\mathrm{ID}}$ に基づき経路表の改良を繰り返す．FRT における ID に関する経路表の順序関係 $\leq_{\mathrm{ID}}$ を拡張して，ネットワーク近接性を考慮した経路表の順序関係 $\leq_{\mathrm{ID+PR}}$ を定義し，$\leq_{\mathrm{ID+PR}}$ に基づき経路表の改良を繰り返すネットワーク近接性を考慮した柔軟な経路表（Proximity-aware Flexible Routing Tables，PFRT）を提案する．

PFRT に基づく DHT アルゴリズムである PFRT-Chord の実装・評価を行った．PFRT-Chord は FRT に基づく DHT アルゴリズムである FRT-Chord を素直に拡張した DHT アルゴリズムである．そのため，FRT-Chord の経路表サイズの変更可能などの特長を継承している．

PFRT-Chord では，ルーティングにおける平均所要時間が Chord より約 40% 減少した．よって，PFRT はネットワーク近接性を考慮したルーティングアルゴリズムの設計手法として有効であることを示すことができた．

2. 関 連 研 究

本研究の関連研究を次に示す．

2.1 Chord

Chord[1] は DHT アルゴリズムのひとつである．m ビットのリング上の ID 空間にハッシュ関数により ID が割り当てられたノードを配置する．key と value を持つデータもまた key に基づき ID が割り当てられ，ID 空間に配置される．ID 空間上でデータ ID から時計回りに進んだとき，一番最初に出会うノードを担当ノードと呼び，担当ノードにそのデータを保存する．データを取り出すときは，目的のデータの key から ID を生成し，そのデータの担当ノードへたどり着くまで，各ノードが所持している経路表に基づきノードへの転送（ホップ）を繰り返す．

Chord には，successor list，predecessor，finger table の 3 つの経路表がある．ID 空間上で自ノード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のノード情報（エントリ）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 とは自ノード s 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．ノード x からノード y の ID 距離 $d(x, y)$ を次のように定義する．

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

Chord は，successor によってルーティングにおける目的ノードへの到達性を保証し，finger table によってルーティングにおけるノードの転送回数（経路長）を減少させる．これらの経路表により，Chord はノード数を N とすると，ルーティングにおける経路長が $O(\log N)$ で目的ノードへ到達することが可能となる．

2.2 LPRS-Chord

LPRS-Chord[6] はネットワーク近接性を考慮するように Chord を拡張した DHT アルゴリズムである．LPRS-Chord では，PNS に基づく拡張，つまり経路表構築時にネットワーク近接性を考慮している．

Chord の 3 つの経路表のうち finger table をネットワーク近接性を考慮するように拡張する．つまり，successor によってルーティングにおける目的ノードへの到達性を保証しつつ，拡張した finger table により Chord と同等の経路長を達成し，またルーティングにおける 1 回の転送（1 ホップ）あたりの平均所要時間を減少させる．その結果，LPRS-Chord におけるルーティングにおける平均所要時間が減少する．LPRS-Chord の拡張した finger table は次の通りである．

(1) 他のノードと通信を行ったとき，ノード間の通信遅延を計測する．

(2)　$2^i \leq d(s,n) < 2^{i+1}$ を満たすノード n のうち，自ノード s との通信遅延の最も小さいノードを finger table の i 番エントリとして保持する．

2.3　柔軟な経路表

柔軟な経路表（Flexible Routing Tables，FRT）[5] は，ID に関する経路表の順序関係 $\leq_{\mathrm{ID}}$ に基づき経路表の改良を繰り返すことで経路表を構築するルーティングアルゴリズム設計手法である．

2.3.1　FRT-Chord

FRT-Chord[5] は FRT に基づき設計された DHT アルゴリズムである．Chord と同様に m ビットのリング上の ID 空間を持ち，担当ノードの決定法も同じであるが，経路表の構築方法が異なる．FRT-Chord には，Chord の successor list，predecessor，finger table の 3 つの経路表を 1 つに統合した経路表 E がある．

経路表 E はエントリ e_i の集合 $\{e_i\}$ である．自ノード s の経路表エントリ e_i はノード ID e_i.id および IP アドレス e_i.address を保持している．$d(s,e_i) < d(s,e_{i+1})(i = 1,2,...,|E|-1)$ を満たしている．このとき，e_1 は successor，$\{e_i\}_{i=1,2,...,c}$ は successor list，$e_{|E|}$ は predecessor を表している．

2.3.1.1　$\leq_{\mathrm{ID}}$：ID に関する経路表の順序関係

$\leq_{\mathrm{ID}}$ は ID に関する経路表の優劣を表す順序関係である．順序関係 $\leq_{\mathrm{ID}}$ に基づき優れた経路表ほどルーティングにおける経路長が短くなる．

FRT-Chord では，自ノード s が経路表エントリ e_i へフォワーディングを行った際，残り ID 距離の短縮倍率の最悪値を $r_i(E)$ とする．

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

$\{r_i(E)\}$ を降順に並べた列を $\{r_{(i)}(E)\}$ とすると，次のように ID に関する経路表の順序関係を定義する．

$$E \leq_{\mathrm{ID}} F \Longleftrightarrow \{r_{(i)}(E)\} \leq_{\mathrm{dic}} \{r_{(i)}(F)\} \tag{3}$$

ただし，$\leq_{\mathrm{dic}}$ とは辞書式順序である．このとき，次の式を満たす経路表 $\tilde{E}$ を最良経路表と呼ぶ．

$$\tilde{E} \leq_{\mathrm{ID}} E \quad \text{for} \forall \mathrm{E} \tag{4}$$

2.3.1.2　到達性保証操作

FRT-Chord では，ルーティングにおける目的ノードへの到達性を保証するための操作を到達性保証操作と呼ぶ．これは Chord の stabilize 操作にあたる操作である．この操作により successor を常に最新に保ち，Chord と同等の到達性を保証する．

2.3.1.3　エントリ情報学習操作

FRT-Chord では，エントリを学習する操作をエントリ情報学習操作と呼ぶ．この操作により，ノードが知りえたエントリをすべて経路表に追加する．エントリ情報学習操作は次の場合にエントリを学習する．

- ネットワーク参加時に他のノードから経路表の全エントリを学習する．
- ノードの探索時等において通信したノードのエントリを学習する．
- 能動学習探索を行い，通信したノードのエントリを学習する．

能動学習探索は，現在の経路表より ID に関する経路表の順序関係 $\leq_{\mathrm{ID}}$ に基づき優れた経路表を構築するために，必要なエントリを学習するルーティングである．現在の経路表から計算された ID へのルーティングを行う．

2.3.1.4　エントリ厳選操作

FRT-Chord では，経路表 E のエントリ数 $|E|$ が経路表サイズ L を超えるとき，$|E| \leq L$ になるように経路表のエントリを厳選する操作をエントリ厳選操作と呼ぶ．エントリ厳選操作では，次の式を満たすエントリ e_{r} の削除を行う．

$$E - \{e_{\mathrm{r}}\} \leq_{\mathrm{ID}} E - \{e\} \quad \text{for} \forall \mathrm{e} \in \mathrm{E} \tag{5}$$

削除エントリを効率よく見つけるために，次のように正規化間隔 S_i^E を定義する．

$$S_i^E = \log \frac{d(s, e_{i+1})}{d(s, e_i)} \tag{6}$$

削除エントリは $S_{i-1}^E + S_i^E$ が最小となるエントリ e_{i} である．$S_{i-1}^E + S_i^E$ を昇順に保持しておくことで，効率よく削除エントリ探索することができる．

FRT-Chord では，Chord における successor list および predecessor にあたる経路表エントリを sticky entry と呼び，エントリ厳選操作による経路表からのエントリ削除を行わない．FRT-Chord におけるエントリ厳選操作は次の通りである．

(1)　削除候補のエントリ集合 C に経路表 E の全エントリを追加する．
(2)　C から sticky entry を除外する．
(3)　C のエントリの中で $S_{i-1}^E + S_i^E$ が最小となるエントリ e_i を経路表から削除する．

3.　PFRT：ネットワーク近接性を考慮した柔軟な経路表

ネットワーク近接性を考慮した柔軟な経路表（Proximity-aware Flexible Routing Tables，PFRT）は，ネットワーク近接性を考慮した経路表の順序関係 $\leq_{\mathrm{ID+PR}}$ に基づき経路表の改良を繰り返すことで経路表を構築するルーティングアルゴリズム設計手法である．ただし，$\leq_{\mathrm{ID+PR}}$ は，FRT の ID に関する経路表

の順序関係 $\leq_{\mathrm{ID}}$ をネットワーク近接性を考慮するように拡張した順序関係である．

3.1 PFRT-Chord

PFRT-Chord は PFRT に基づき設計された DHT アルゴリズムである．FRT-Chord を素直に拡張しているため，経路表を構築する方法以外は FRT-Chord のアルゴリズムを用いている．自ノード s の経路表エントリ e_i は ID および IP アドレスの他にノード s と e_i との間の通信遅延 e_i.latency を保持している．

3.1.1 $\leq_{\mathrm{ID+PR}}$：ネットワーク近接性を考慮した経路表の順序関係

PFRT では，経路長を短くするために ID を考慮するだけでなく，1 ホップあたりの所要時間を小さくするために自ノードとの通信遅延も考慮する．そこで，まず自ノードとの通信遅延に関する経路表の順序関係 $\leq_{\mathrm{PR}}$ を定義する．次に，$\leq_{\mathrm{ID}}$ および $\leq_{\mathrm{PR}}$ を用いることで，ネットワーク近接性を考慮した経路表の順序関係 $\leq_{\mathrm{ID+PR}}$ を定義する．

3.1.1.1 $\leq_{\mathrm{PR}}$：自ノードとの通信遅延に関する経路表の順序関係

$\leq_{\mathrm{PR}}$ は自ノードとの通信遅延に関する経路表の優劣を表す順序関係である．順序関係 $\leq_{\mathrm{PR}}$ に基づき優れた経路表ほどルーティングにおける 1 ホップあたりの平均所要時間が小さくなる．自ノードとの通信遅延に関する経路表の順序関係 $\leq_{\mathrm{PR}}$ を次のように定義する．

$$E \leq_{\mathrm{PR}} F \Longleftrightarrow l(E) \leq l(F) \tag{7}$$

ただし，$l(E)$ は経路表 E の各エントリ e_i と自ノードとの通信遅延 e_i.latency の平均値であり，次のように定義される．

$$l(E) = \frac{1}{|E|} \sum_{i=1}^{|E|} e_i.\mathrm{latency} \tag{8}$$

3.1.1.2 $\leq_{\mathrm{ID+PR}}$ の定義

$\leq_{\mathrm{ID+PR}}$ はネットワーク近接性を考慮した経路表の優劣を表す順序関係である．順序関係 $\leq_{\mathrm{ID+PR}}$ に基づき優れた経路表ほどルーティングにおける所要時間が小さくなる．経路長が減少し，1 ホップあたりの所要時間が減少するとき，ルーティングにおける所要時間は確実に減少する．そこで，$\leq_{\mathrm{ID+PR}}$ を次のように定義する．

$$E \leq_{\mathrm{ID+PR}} F \Longleftrightarrow (E \leq_{\mathrm{ID}} F) \cap (E \leq_{\mathrm{PR}} F) \tag{9}$$

$E \leq_{\mathrm{ID}} F$ および $F \leq_{\mathrm{PR}} E$ の両方を満たす経路表 E, F が存在するとき，経路表 E, F を順序関係 $\leq_{\mathrm{ID+PR}}$ に基づき比較することができない．つまり，$\leq_{\mathrm{ID+PR}}$ は半順序関係である．

3.1.2 到達性保証操作

FRT-Chord の到達性保証操作と同じである．

3.1.3 エントリ情報学習操作

PFRT-Chord ではエントリの学習を行うために，FRT-Chord のエントリ情報学習操作を拡張した操作を行う．PFRT-Chord では，経路表エントリ e_i に e_i.latency を追加しているので，エントリを学習する際に，e_i.latency を記録する．通信遅延が自然に得られない場合，自ノードとの通信遅延を計測する．エントリ情報学習操作は次の通りである．

(1) エントリ情報学習操作を行う．
(2) 手順 (1) で経路表に追加したエントリを最新追加エントリ e_{new} として記憶する．

最新追加エントリ e_{new} はエントリ厳選操作において利用される．

3.1.4 エントリ厳選操作

PFRT-Chord ではエントリを厳選するために，FRT-Chord のエントリ厳選操作を拡張した操作を行う．$\leq_{\mathrm{ID}}$ は全順序関係であるため，FRT-Chord では最善の削除エントリ e_{r} を決定することができた．しかし，$\leq_{\mathrm{ID+PR}}$ は半順序関係であるため，PFRT-Chord では一般に最善の削除エントリ e_{r} を決定することができない．そこで，まず経路表エントリのうち，順序関係 $\leq_{\mathrm{ID+PR}}$ および最新追加エントリ e_{new} を用いて削除エントリの候補集合 C_{r} に絞る．

$$C_{\mathrm{r}} = \{e \in E \mid E - \{e\} \leq_{\mathrm{ID+PR}} E - \{e_{\mathrm{new}}\}\} \tag{10}$$

次に，C_{r} の中で順序関係 $\leq_{\mathrm{ID}}$ に基づき削除するエントリ $e_{\mathrm{r}} \in C_{\mathrm{r}}$ を決定する．

$$E - \{e_{\mathrm{r}}\} \leq_{\mathrm{ID}} E - \{e\} \quad \text{for}\, \forall \mathrm{e} \in \mathrm{C_r} \tag{11}$$

$e_{\mathrm{new}} \in C_{\mathrm{r}}$ より $C_{\mathrm{r}} \neq \phi$ である．つまり，e_{r} は必ず存在する．e_{r} を削除した経路表は，最新追加エントリ e_{new} を追加する前の経路表よりネットワーク近接性を考慮した経路表として優れているか，または同じ経路表である．

エントリ厳選操作の手順は次の通りである．ただし，sticky entry のうち successor list サイズを c とする．

(1) 削除候補のエントリ集合 C に経路表 E の全エントリを追加する．
(2) C から sticky entry を除外する．
(3) ノード間の通信遅延の閾値となるエントリを閾値エントリ e_{t} とすると，

$$e_{\mathrm{t}} = \begin{cases} e_{c+1}, & (e_{\mathrm{new}} \in \{e_i\}_{i=1,2,\ldots,c}) \\ e_{|E|-1}, & (e_{\mathrm{new}} = e_{|E|}) \\ e_{\mathrm{new}}, & \text{otherwise} \end{cases} \tag{12}$$

(4) $e_{\mathrm{t}} \in C$ ならば C から $\{e \in C \mid e.\mathrm{latency} <$

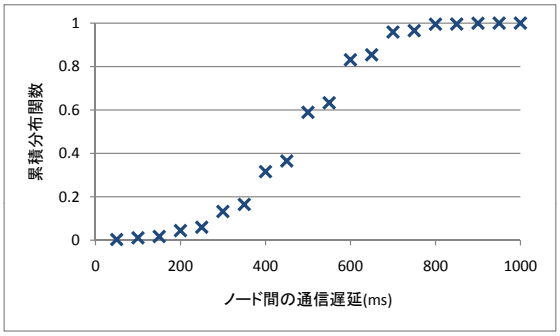

図 1　ノード間の通信遅延の分布
Fig. 1　CDF of communication latency

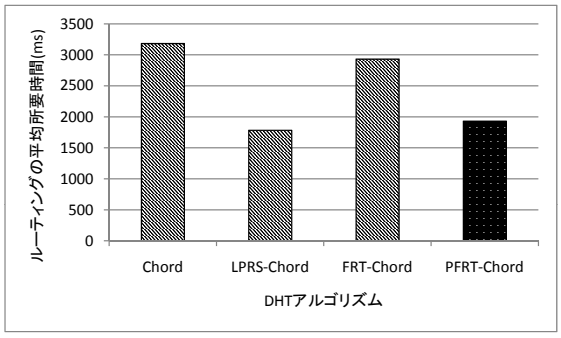

図 2　各 DHT アルゴリズムの平均所要時間
Fig. 2　Average elapsed time for routing of Chord, LPRS-Chord, FRT-Chord and PFRT-Chord

e_t.latency} を除外する．

(5)　C のエントリの中で $S^E_{i-1} + S^E_i$ が最小となるエントリ e_i を経路表から削除する．

最新追加エントリ e_{new} が sticky entry の場合は e_{new} を削除することができないので，e_{new} を追加することで sticky entry からあふれたエントリを閾値エントリ e_t にする（手順 (3)）．経路表サイズを変更したいとき，エントリ厳選操作を連続して実行する場合がある．その場合，e_{new} が経路表から削除されたことにより，経路表内に存在せず，$e_{\mathrm{new}} = e_t \notin C$ が起こりうる．このとき，自ノードとの通信遅延に基づく C からのエントリ除外を行わない（手順 (4)）．

手順 (1)，(2) および (5) は FRT-Chord のエントリ厳選操作である．PFRT-Chord により拡張した手順は手順 (3) および (4) である．$S^E_{i-1} + S^E_i$ を昇順に保持し，上から閾値エントリ e_t の通信遅延以上であるかを調べることで，削除エントリの探索をすることができる．経路表サイズを L としたとき，削除エントリの探索は $O(L)$ で終了する．

4. 評　　価

オーバレイ構築ツールキットである Overlay Weaver[7][8] 上に，PFRT-Chord を実装した．一台のマシン上でシミュレーションによる実験・評価を行った．ルーティングにおける所要時間は，全ホップの通信遅延の総和である．実験に使用したマシンは次の通りである．

- Overlay Weaver 0.10.1
- OS：Windows 7 Professional 32 bit
- CPU：Intel 2.67 GHz Core 2 Quad Q9400
- メモリ：4 GB
- Java SE 6 Update 22

4.1　ノード間の通信遅延の設定

ノード間の通信遅延を設定するために，Transit-Stub モデル（TS モデル）[9] を利用した．TS モデルにより，実ネットワークのトポロジをシミュレートした．TS モデルはトランジットノードとスタブノードの 2 種類のノードで構成されている．トランジットノード間，トランジット・スタブノード間およびスタブノード間の通信遅延を 100 ms，20 ms，5 ms とし，一対一でのノード間の通信遅延を決めた．その結果，ノード間の通信遅延の分布は図 **1** となり，通信遅延の平均値は 470 ms，最大値は 1000 ms となった．

4.2　実 験 結 果

4.2.1 節でネットワーク近接性を考慮した DHT アルゴリズムとして PFRT-Chord が有効であることを示し，4.2.2 節で想定した通りに経路表の改良がされていることを示す．また，4.2.3 節で PFRT-Chord の経路表サイズを変化させたときのルーティングにおける平均所要時間と平均経路長を調べる．これらの実験に使用したパラメータは次の通りである．

- ノード数：$N = 10000$
- 探索回数：1 ノードあたり 100 回
- successor list サイズ：$c = 4$

4.2.1　PFRT の有効性

図 **2** に，PFRT-Chord と他の DHT アルゴリズムのルーティングにおける平均所要時間を示す．他の DHT アルゴリズムと平均所要時間を比較することで，PFRT-Chord の有効性を示す．比較対象は Chord，LPRS-Chord および FRT-Chord である．

Chord および LPRS-Chord では finger table のサイズは変更できない．そこで，経路表サイズの変更可能な FRT-Chord および PFRT-Chord の経路表サイズを Chord および LPRS-Chord の経路表に載るエントリ数に合わせた．finger table に載るエントリ数は $\log N$ と計算できるので，約 13 である．successor list および predecessor を加えると Chord および LPRS-Chord の経路表に載るエントリ数は約 18 とな

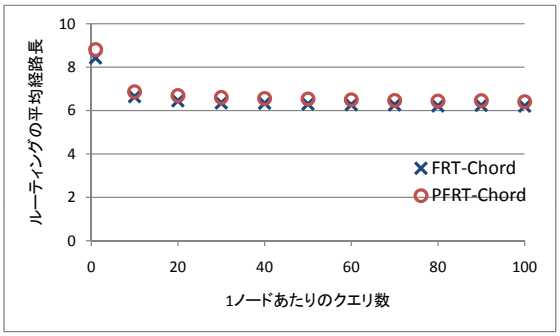

図 3　実行クエリ数と平均経路長の関係
Fig. 3　Correlation between number of lookups and average route length

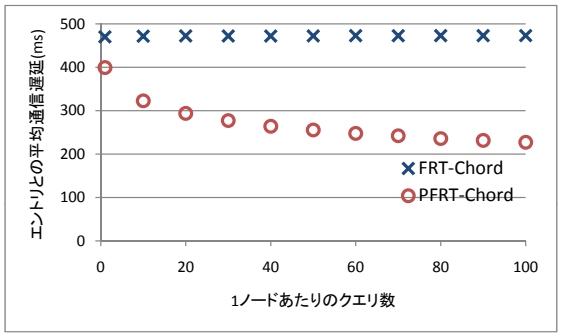

図 4　実行クエリ数と経路表エントリの平均通信遅延の関係
Fig. 4　Correlation between number of lookups and average communication latency to entry node

る．PFRT-Chord が有利にならないようにするために，FRT-Chord および PFRT-Chord では経路表サイズ $L = 16$ とした．

PFRT-Chord でのルーティングにおける平均所要時間は 1930 ms であった．まず，Chord と比較する．Chord の平均所要時間は 3180 ms であることから，PFRT-Chord により約 40 % 減少した．この結果から，PFRT-Chord はネットワーク近接性を考慮した DHT アルゴリズムとして有効であるといえる．

次に FRT-Chord と比較する．FRT-Chord の平均所要時間は 2930 ms であることから，約 35 % 減少した．この結果から，PFRT は FRT に基づく DHT アルゴリズムをネットワーク近接性を考慮するように拡張する手法として有効であるといえる．

最後に LPRS-Chord と比較する．LPRS-Chord の平均所要時間は 1780 ms であることから，約 8 % 増加した．しかし，平均所要時間の差は約 150 ms であり，これはノード間の通信遅延の平均値の 3 分の 1 以下に過ぎない．また，LPRS-Chord の経路表は finger table であり，経路表サイズは変更できないが，一方 PFRT-Chord は FRT の特長を引き継ぎ，経路表サイズの変更が可能なので，経路表サイズを大きくすることで平均所要時間を短縮することも容易である．さらに，PFRT-Chord は他にも FRT の特長を引き継いでいるため，LPRS-Chord より優れた DHT アルゴリズムであるといえる．

4.2.2　PFRT-Chord の経路表改良の効果

PFRT-Chord では，$\leq_{\mathrm{ID+PR}}$ に基づき経路表の改良を繰り返しているので，経路表を改良するごとにルーティングにおける平均経路長が減少し，各エントリの自ノードとの通信遅延の平均値（以下，平均通信遅延）が小さくなると想定している．そこで，経路表を改良するごとに平均経路長および平均通信遅延の変化を確認する．図 **3** および図 **4** に，各クエリ実行回数に対する平均経路長および平均通信遅延を示す．1 ノードあたり 10 クエリ実行する毎に平均経路長および平均通信遅延を測定した．

まず，図 3 からルーティングにおける平均経路長の変化を確認する．PFRT-Chord では，実行したクエリ数が増加するにつれて，平均経路長が減少している．よって，エントリ厳選操作によって平均経路長が減少していることがわかる．また，10 番目のクエリを実行するまでの間に平均経路長が大きく減少し，それ以降のクエリでは緩やかに減少し続け，100 番目のクエリを実行するときにはほとんど減少していない．つまり，PFRT-Chord の経路表は収束していないが，少ないクエリ数で収束後の経路表に大きく近づくことができる．さらに，FRT-Chord とほぼ同じ変化であるので，PFRT-Chord に拡張することによる平均経路長の損失がほとんどないといえる．

次に，図 4 から平均通信遅延の変化を確認する．PFRT-Chord では，実行したクエリ数が増加するにつれて平均通信遅延が減少している．このことから，エントリ厳選操作によって平均通信遅延が減少していることがわかる．また，10 番目のクエリを実行するまでの間に平均通信遅延が大きく減少し，それ以降のクエリでも減少を続けている．つまり，PFRT-Chord の経路表は平均通信遅延が収束するために，平均経路長が収束するよりたくさんのクエリを実行する必要があるが，少ないクエリ数で収束後の経路表に大きく近づき，クエリ数を増やすことでより経路表をより改良することができる．さらに，FRT-Chord より常に平均通信遅延が小さく，100 番目のクエリを実行するときに約 50 % 減少しているので，PFRT-Chord に拡張することで大きく減少させることができるといえる．

PFRT-Chord では，経路表を改良するごとに，ルー

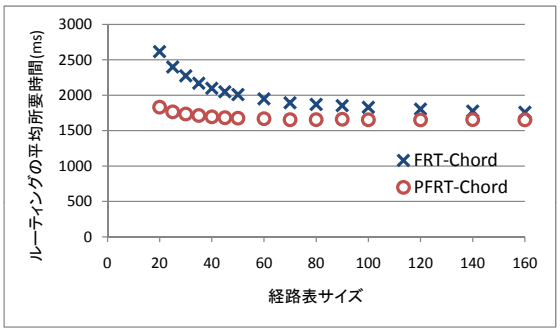

図 5　経路表サイズと平均所要時間の関係
Fig. 5　Correlation between averageelapsed time for routing and size of routing table

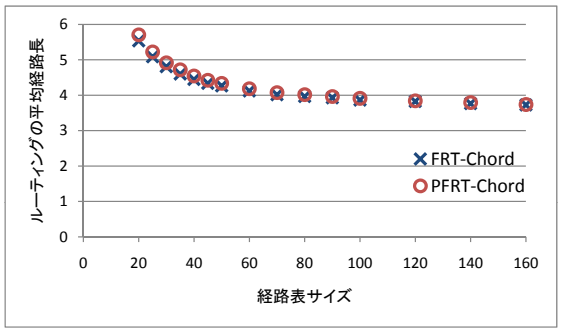

図 6　経路表サイズと平均経路長の関係
Fig. 6　Correlation between average route length and size of routing table

ティングにおける平均経路長および平均通信遅延が減少していることから，$\leq_{\mathrm{ID+PR}}$ に基づき想定通りに経路表が改良されていることを確認することができた．

4.2.3　経路表サイズの変更

PFRT-Chord は FRT の特長を継承しているため，経路表サイズの変更が可能である．そこで，図 **5** および図 **6** に，経路表サイズを変化させたときのルーティングにおける平均所要時間および平均経路長を示す．FRT-Chord および PFRT-Chord の経路表サイズを 20 から 160 まで変化させ，平均所要時間および平均経路長を測定した．

図 5 より，PFRT-Chord は経路表サイズが 25 のとき，ルーティングにおける平均所要時間が 1760 ms であることがわかる．4.2.1 節で実験を行った LPRS-Chord は 1780 ms であるため，PFRT-Chord は経路表サイズが 25 を超えると LPRS-Chord より平均所要時間が小さくなる．LPRS-Chord では，finger table に載るエントリが約 13 であっても，配列といった単純な実装では 160 エントリ分のメモリを消費する．一方，PFRT-Chord では，経路表サイズが 25 のときは 25 エントリ分だけメモリを消費する．つまり，平均所要時間が同等なとき，PFRT-Chord の方がメモリの消費量が小さいといえる．また，FRT-Chord が LPRS-Chord と同等の平均所要時間になるためには，経路表サイズが 140 必要である．このことからも FRT-Chord を PFRT-Chord に拡張することによるルーティングにおける平均所要時間の短縮効果が確認できる．

PFRT-Chord および FRT-Chord のルーティングにおける平均所要時間を比較する．経路表サイズを大きくするにつれて，FRT-Chord および PFRT-Chord の平均所要時間は小さくなるが，FRT-Chord の減少量に比べ，PFRT-Chord の減少量が小さい．経路表サイズが 20 のときは PFRT-Chord の平均所要時間は FRT-Chord に比べ約 30 % 減少していたが，経路表サイズが 100 のときは約 10 % の減少である．よって，経路表サイズを大きくするにつれて，FRT-Chord を PFRT-Chord に拡張することによるルーティングにおける平均所要時間の短縮効果が小さくなっている．

図 6 から，PFRT-Chord および FRT-Chord のルーティングにおける平均経路長を比較する．経路表サイズが変化しても，PFRT-Chord の平均経路長は FRT-Chord とほとんど同じ変化をしていることがわかる．よって，経路表サイズを変化させても，FRT-Chord を PFRT-Chord に拡張することによる平均経路長の損失がない．

PFRT-Chord および FRT-Chord の経路表サイズを大きくしたとき，平均経路長の差がほとんどないにもかかわらず，平均所要時間の差が小さくなっていることがわかる．つまり，経路表サイズが大きくなるにつれて，PFRT-Chord の 1 ホップあたりの平均所要時間が増加しているといえる．これには 2 つの原因が存在する．1 つ目の原因は，平均経路長が短くなることにより，目的ノードに到達するために successor list にあるノードへの転送回数が経路長全体に対して増加するからである．successor list は自ノードとの通信遅延を考慮していないノードをエントリとして保持しているため，1 ホップあたりの平均所要時間は FRT-Chord および PFRT-Chord ともに等しい．2 つ目の原因は，PFRT-Chord は自ノードとの通信遅延だけでなく，ノード ID も考慮して経路表を構築しているので，経路表サイズが大きくなることにつれ，自ノードとの通信遅延の大きいノードをエントリとして保持することが増えるからである．そのため，経路表エントリの自ノードとの通信遅延の平均値の増加し，1 ホップあたりの平均所要時間が増加する．

5. ま と め

ノードID以外を考慮しやすいルーティングアルゴリズム設計手法である柔軟な経路表（FRT）を拡張した，ネットワーク近接性を考慮した柔軟な経路表（PFRT）を提案した．PFRTは，ネットワーク近接性を考慮した経路表の順序関係 $\leq_{\mathrm{ID+PR}}$ に基づき経路表の改良を繰り返すことで経路表を構築するルーティングアルゴリズム設計手法である．また，PFRTに基づき設計されたDHTアルゴリズムであるPFRT-Chordを提案し，実験・評価を行った．

PFRT-ChordはFRTに基づき設計されたDHTアルゴリズムであるFRT-Chordの素直な拡張であるため，FRT-Chordの特長（たとえば，経路表サイズの変更可能など）を継承している．また，拡張を行うことによるデメリットを最小限に抑えている．そのため，ルーティングにおける平均経路長はFRT-Chordよりほとんど増加しない．

PFRT-Chordはルーティングにおける平均所要時間がChordより約40 %減少し，FRT-Chordより約35 %減少した．また，LPRS-Chordよりわずかに増加した．LPRS-Chordと同等の平均所要時間にするためには，PFRT-Chordの経路表サイズを25にする必要がある．LPRS-Chordでは，finger tableに載るエントリ数が約13であっても，配列といった単純な実装では160エントリ分のメモリを消費するが，PFRT-Chordでは設定した経路表サイズ分だけのメモリが消費されるため，PFRT-ChordはLPRS-Chordよりメモリ消費量が少ない．

PFRT-Chordではネットワーク近接性を考慮した経路表の順序関係 $\leq_{\mathrm{ID+PR}}$ に基づく経路表の改良，つまりIDに関する経路表の順序関係 $\leq_{\mathrm{ID}}$ および自ノードとの通信遅延に関する経路表の順序関係 $\leq_{\mathrm{PR}}$ の両方において優れている経路表に改良している．しかし，順序関係 $\leq_{\mathrm{ID}}$ または $\leq_{\mathrm{PR}}$ のどちらか一方のみが優れている経路表に改良した場合でも，ルーティングにおける平均所要時間が減少することが考えられる．それらの経路表への改良が行われないことによる損失を分析することで，さらに良い順序関係 $\leq_{\mathrm{ID+PR}}$ を定義することができる．

本研究では，順序関係 $\leq_{\mathrm{PR}}$ を用いることで経路表エントリの自ノードとの通信遅延を考慮し，ネットワーク近接性を考慮した経路表の順序関係 $\leq_{\mathrm{ID+PR}}$ を定義したが，ID以外の情報（たとえばネットワークに生存している時間）を考慮するために，順序関係 $\leq_{\mathrm{EX}}$ を用いて本研究と同じ方式で経路表の順序関係 $\leq_{\mathrm{ID+EX}}$ を定義することが可能である．このような，単独または複数のID以外の情報を考慮するように柔軟な経路表を拡張することが課題である．

謝辞　本研究は科研費（22680005）の助成を受けたものである．また，本研究成果の一部は，独立行政法人情報通信機構（NICT）の委託研究「新世代ネットワークを支えるネットワーク仮想化基盤技術の研究開発」により得られたものです．

参 考 文 献

1) Stoica, I., Morris, R., Karger, D., Kaashoek, F. and Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, *Proc. ACM SIGCOMM '01*, pp. 149–160 (2001).
2) Maymounkov, P. and Mazieres, D.: Kademlia: A Peer-to-peer Information Systems Based on the XOR Metric, *Proc. IPTPS 2002*, 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*, pp. 329–350 (2001).
4) Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S. and Stoica, I.: The Impact of DHT Routing Geometry on Resilience and Proximity, *Proc. ACM SIGCOMM '03*, pp. 381–394 (2003).
5) 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*, pp. 72–81 (2011).
6) Zhang, H., Goel, A. and Govindan, R.: Incrementally Improving Lookup Latency in Distributed Hash Table Systems, *Proc. ACM SIGMETRICS '03* (2003).
7) 首藤 一幸: Overlay Weaver, http://overlayweaver.sourceforge.net/.
8) Shudo, K., Tanaka, Y. and Sekiguchi, S.: Overlay Weaver: An Overlay Construction Toolkit, *Computer Communications*, Vol. 31, No. 2, pp. 402–412 (2008).
9) Zegura, E. W., Calvert, K. L. and Bhattacarjee, S.: How to Model an Internetwork, *Proc. IEEE INFOCOM '96*, pp. 592–602 (1996).