

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

安藤 泰弘[†] 長尾 洋也[†]
宮尾 武裕[†] 首藤 一幸[†]

構造化オーバーレイの設計手法である FRT に基づいた DHT アルゴリズム FRT-2-Chord を提案する。DHT に代表される構造化オーバーレイにおいて、オーバーレイのトポロジを決定する際に考慮すべき要素に、ノード数、ID 距離、ネットワーク近接性、グループ、ノードの参加と離脱の頻度などがある。構造化オーバーレイでは、これらの要素は常に変化し、また、応用によって異なる。しかし、従来の DHT アルゴリズムは、これらの要素に対する順応性がなかったり、一部のみに着目していたため、効率のよいルーティングを行うことができなかった。FRT-2-Chord はこれらの各要素に対する順応性があるので、汎用的で効率のよいルーティングが可能である。証明と実験によって FRT-2-Chord の順応性を示す。

FRT-2-Chord: A DHT Supporting Seamless Transition between One-hop and Multi-hop with Symmetric Routing Table

YASUHIRO ANDO,[†] HIROYA NAGAO,[†] TAKEHIRO MIYAO[†]
and KAZUYUKI SHUDO[†]

We propose FRT-2-Chord a DHT, based on FRT, a method for designing routing algorithms for overlays. Structured overlays including DHTs should consider following factors when constructing their overlay topology: the number of nodes, identifier distance, proximity, groups and the frequency of joins and leaves of nodes. Existing DHTs lack adaptability to those factors or focus on part of them though they are constantly changing and heavily depend on applications. FRT-2-Chord has further adaptability and achieves efficient routing with a variety of those factors. Proofs and experimental results show it.

1. はじめに

実ネットワーク上に仮想ネットワークであるオーバーレイネットワークを構築することで、Peer-to-Peer (P2P) システムはスケーラビリティや耐故障性を達成する。オーバーレイネットワークにおけるルーティングの方式は、二つに分類される。一つは非構造化オーバーレイであり、クエリを拡散させることにより目的のノードへクエリを転送する方式である。もう一方の構造化オーバーレイは、あらかじめ定められた方法でネットワークを構築し、転送先のノードを選択を決定する。そのため、クエリが大量に拡散しないルーティングを行うことができる。

構造化オーバーレイにおいて、オーバーレイのトポロジを決定する際に考慮すべき要素に、ノード数、ID 距離、ネットワーク近接性、グループ、ノードの参加と離脱の頻度などがある。構造化オーバーレイでは、これらの要素は常に変化し、また、応用によって異なる。そのため、これらの要素に対する順応性がないと、スケーラビリティや耐故障性を実現できない。また、オーバーレイのトポロジをグラフとみなすと、ノード数は頂点数である。ID 距離、ネットワーク近接性、グループなどは経路の選択に影響する要素と一般化することができ、これらは辺の重みである。ノードの参加と離脱は時間軸に対するグラフの変化である。これらの要素以外に関する考慮も必要な場合もありうるが、グラフによる一般化を考えると、これらの要素に対する順応性がある構造化オーバーレイは十分に汎用的であると言える。そこで、これらの要素に対する順応性を実現す

[†] 東京工業大学
Tokyo Institute of Technology

るために次に述べることが求められる。

(1) **ノード数の多寡に依存しない効率**

構造化オーバーレイでは、ノード数の多寡や増減の仕方が応用や時間によって様々である。そのため、ネットワーク全体のノード数を仮定してしまうと、スケーラビリティを損なう。そこで、次のようなノード数に応じたルーティングが求められる。

(a) **multi-hop における経路長の短さ**

経路表サイズに比べてノード数 N が大きい場合、複数のノードを経由して目的ノードにたどり着く、つまり、multi-hop で目的ノードに到達する。遅延時間の短縮やバンド幅の使用率の削減などを目的として、 $O(\log N)$ などの経路長で目的ノードに到達することが求められる。

(b) **one-hop の可能性**

目的ノードのエントリを経路表が持つとき、目的ノードに直接たどり着くこと、つまり、one-hop が望ましい。また、経路表サイズよりノード数が少ない場合、全ノードを経路表が持つことで、任意の目的ノードへの one-hop が求められる。

(2) **経路の選択に影響する複数の要素への対応**

オーバーレイネットワーク上の任意のノードに対し、経路長が短いことが望まれる。また、ノード ID に限らず、ネットワーク近接性やグループなどを考慮したルーティングも同様に重要である。これらの要素が独立かつ任意の値をとる前提で、経路の選択が望まれる。

(a) **経路表の柔軟性**

ノード ID に注目するなど、一部の要素のみに注目した経路表の構築すると、各要素の独立性により、他の要素を考慮するのが難しくなる。各要素を容易に考慮することができる経路表の柔軟性が求められる。

(b) **経路表の対称性**

経路表の対称性とはノード間で互いにエントリを持つという性質である。ネットワーク近接性や同一グループに属するか否かという属性などはノード間で対称なので、経路表の対称性は重要な性質である。また、経路表の対称性によって、対称性が無い経路表に比べて経路表に含まれるノードの生存確認のコストを削減することができる。

(3) **ノードの参加と離脱の頻度に依存しない到達性**

到達性とは、目的ノードへ正しくクエリを到達させる性質である。任意の頻度、特に頻繁なノードの参加と離脱が行われる場合にも、クエリが目的ノードへ到達することが望まれる。また、到達性を保証するために必要な生存確認のコストを抑えることが求められる。

主に、分散ハッシュテーブル (Distributed Hash Ta-

ble, DHT) で構造化オーバーレイが利用されてきた。DHT は連想配列を複数のノードで管理する技術である。これまで、様々な DHT アルゴリズム^{1)~6)} が提案されてきたが、上述したうちの一部の要求のみに着目している。しかし、これらの全ての要求に応えなければ、スケーラビリティや耐故障性の高いシステムの実現は難しい。本研究の提案手法である FRT-2-Chord は、ルーティングアルゴリズムの設計手法である柔軟な経路表 (Flexible Routing Tables, FRT)⁶⁾ に基づいた DHT アルゴリズムである。FRT-2-Chord は上述した全ての要求に応えることが可能である。また、Chord を基にした他の DHT アルゴリズムにはなかった有用性として、複製の有効活用と管理のしやすさを備える。FRT-2-Chord の要求への対応と性質を実験や証明を通して述べる。

2. 関連研究

表 1 に、既存の DHT アルゴリズムと提案手法である FRT-2-Chord が 1 章で述べた要求に応えているか否かを示す。

Chord¹⁾, Pastry²⁾, Kademlia³⁾ は小さい経路表サイズで、 $O(\log N)$ の経路長のルーティングが可能である。ID に関する厳しい制約に従って、経路表にエントリを追加するので、経路表サイズより小さい場合でも全ノードを経路表が持つことは難しく、任意のノード間での one-hop は不可能である。EpiChord⁴⁾ や OneHop⁵⁾ は全ノードの情報を経路表に保持することを想定している。そのため、ノード数が想定より上回る場合、適用できない。FRT-Chord や EpiChord などの Chord を基にしたアルゴリズムは一度目的ノードの predecessor に到達してから、目的ノードにフォワーディングする。ただし、predecessor とは、Chord のリング状の ID 空間においてノードから反時計周り

表 1 DHT アルゴリズムと構造化オーバーレイに対する要求
Table 1 DHTs and requirements for structured overlays

	(1-a)	(1-b)	(2-a)	(2-b)	(3)
Pastry	○	×	×	○	○
Chord	○	×	×	×	○
Kademlia	○	×	×	○	△
EpiChord	×	△	×	-	○
OneHop	×	○	×	-	○
FRT-Chord	○	△	○	×	○
FRT-2-Chord	○	○	○	○	○

(1-a): multi-hop における経路長の短さ

(1-b): one-hop の可能性

(2-a): 経路表の柔軟性

(2-b): 経路表の対称性

(3): ノードの参加と離脱の頻度に依存しない到達性

に進んで最初に出会うノードである。目的ノードの predecessor がクエリ発行元ノードでない限り経路長は 2 以上である。従って、全ノードを経路表に保持していても one-hop は不可能である。

経路表の柔軟性は、2.2 節で述べる FRT を用いて初めて実現される性質である。

OneHop と EpiChord は全ノードの情報の保持を想定しているため、対称性に関する言及に意味はない。Chord, FRT-Chord の経路表は 2.1 節で述べる ID 空間を持つため、経路表は対称性を持たない。Pastry や Kademlia の経路表は対称性を持ちやすい。

Kademlia の到達性を保証するには、各ノードが全ての k-bucket に通信が可能なノードのエントリが含まれている必要がある。Chord を基にしたアルゴリズムは、リング状の ID 空間においてノードから時計回りに進んで最初に出会うノードである successor のみを維持すればよい。また、後述するが、FRT-2-Chord では successor と predecessor のみを維持すればよい。いずれも、Kademlia より到達性の保証をしやすい。

2.1 Chord

Chord は DHT アルゴリズムの一つである。Chord リングと呼ばれるリング状の ID 空間 $[0, 2^m)$ を用いる。ノードはハッシュ値である m ビットのノード ID に基づいて Chord リング上に配置される。ID x から ID y の ID 距離 $d(x, y)$ は次のように定義される。

定義 2.1 ID 距離 $d(x, y)$

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

DHT において、データは key と value のペアで管理される。key に対するハッシュ値がデータの ID であり、それによってデータを保持するノードが決まる。Chord では、ある key の ID を id とすると、id から時計回りに進んで初めて最初に出会うノードが担当ノードと呼ばれ、担当ノードに key と value を保存する。

Chord には successor list, predecessor, finger table の 3 つの経路表がある。経路表の各エントリはノードの ID と IP アドレスである。successor list は、自ノードから ID 空間のリング上を時計回りに進んで出会う定数個のノードであり、自ノードに一番近いノードを successor と呼ぶ。predecessor は、反時計回りに進んで最初に出会うノードをエントリとして保持する経路表である。次に示す stabilize と呼ばれる通信を定期的に行うことで正しい successor を維持し、到達性を保証する。

(1) 自ノード s の successor のエントリ $succ$ にメッセージを送る。

(2) s は、 $succ$ による predecessor のエントリ $pred$ の情報を含めたメッセージを受け取る。

(3) s は、 $pred$ が s と $succ$ の間にあるなら、 $pred$ を s の successor とし、 s を predecessor とするように $pred$ にメッセージを送る。そうでないなら、終了する。

finger table は自ノードから 2^{i-1} ($i = 1, 2, \dots, m$) 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。また、Chord におけるフォワーディングの方法は次の通りである。

(1) 目的 ID t への距離 $d(e, t)$ が最小である経路表中の ID e のノードへフォワーディングする。担当ノードの predecessor までフォワーディングを繰り返す。

(2) 担当ノードである successor へフォワーディングする。

ノード数を N としたとき、 $O(\log N)$ の経路長で担当ノードへ到達する。

2.2 FRT-Chord

FRT は、ルーティングアルゴリズムの設計手法である。FRT に基づくアルゴリズムは、ID に関する経路表の順序関係を定義し、エントリ情報学習操作とエントリ厳選操作によって経路表を構築する。

FRT-Chord は、FRT に基づいて設計された DHT アルゴリズムである。FRT-Chord の ID 空間、担当ノードの決定方法、フォワーディングの方法は Chord と同様である。

FRT-Chord では、successor list, predecessor, finger table を区別せず、各ノードは一つの経路表 E を保持する。また、経路表サイズは動的に設定可能である。 E はエントリ e_i の集合 $\{e_i\}$ である。各エントリ e_i はノード ID $e_i.id$ と IP アドレス $e_i.address$ を含む。ただし、 $e_i.id$ を e_i と略記する。また、自ノードの ID を s とすると、 $i < j \Rightarrow d(s, e_i) < d(s, e_j)$ である。 e_1 は successor, $e_{|E|}$ は predecessor である。

2.2.1 ID に関する順序関係 \leq_{ID}

FRT-Chord は経路表に含まれるノード ID の組み合わせに基づいて経路表空間上に順序関係 \leq_{ID} を定義する。 \leq_{ID} に基づいて経路表の改良ができる。 \leq_{ID} の定義にあたって、エントリ e_i にフォワーディングする際の最悪短縮倍率 $r_i(E)$ は次のように定義される。

定義 2.2 最悪短縮倍率 $r_i(E)$

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

$\{r_i(E)\}$ を降順に並べた列を $\{r_{(i)}(E)\}$ とし、次の

ように ID に関する経路表の順序関係は定義される。

定義 2.3 $r_i(E)$ による順序関係 \leq_{ID}

$$E \leq_{ID} F \Leftrightarrow \{r_{(i)}(E)\} \leq_{dic} \{r_{(i)}(F)\}$$

ただし, \leq_{dic} は辞書式順序である。

2.2.2 到達性保証操作

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 によって実現される。この操作の対象となる successor list や predecessor などのエントリを sticky entry と呼ぶ。

2.2.3 エントリ情報学習操作

エントリを経路表に追加していく操作をエントリ情報学習操作という。この操作はネットワーク参加時、他ノードとの通信時、能動学習探索時に実行され、ノードが知り得たノードをすべて経路表に追加する。能動学習探索は、現在の経路表をもとに計算された ID へのルーティングを行うことにより、必要に応じて実行することが可能である。

2.2.4 エントリ厳選操作

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、経路表の順序関係 \leq_{ID} に基づいて $|E| \leq L$ を保つようにエントリを削除する操作を行う。この操作をエントリ厳選操作という。削除エントリを効率よく見つけるために正規化間隔 S_i^E が定義される。

定義 2.4 正規化間隔 S_i^E

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

FRT-Chord の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。
- (2) C から sticky entry を除外。
- (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 を選択し、経路表から削除。

$S_{i-1}^E + S_i^E$ を昇順に保持しておくことで効率よく実行できる。また、FRT-2-Chord の厳選操作について 3.4 節で証明した削除の最適性と厳選操作の非停止性は、FRT-Chord の厳選操作においても証明されている。

3. FRT-2-Chord

FRT-2-Chord は FRT に基づいて設計した DHT アルゴリズムである。FRT-2-Chord は Chord と同様にリング状の m ビットの ID 空間を利用し、エントリ情報学習操作は FRT-Chord と同様の操作を行う。

3.1 担当ノードとフォワーディング

FRT-2-Chord の ID 距離を次のように定義する。

定義 3.1 ID 距離 $d(x, y)$

$$d(x, y) = \min\{|x - y|, 2^m - |x - y|\}$$

目的 ID を t すると、FRT-2-Chord の担当ノードは $d(t, s)$ が最小となるノード s である。目的 ID t への距離 $d(e, t)$ が最も小さい経路表中のノード e へのフォワーディングを繰り返す。ただし、 $d(e, t) = d(e', t)$ である 2 つの経路表中のエントリ e, e' が存在する場合、Chord における ID 距離を $d_{chord}(x, y)$ として、 $d_{chord}(t, e)$ と $d_{chord}(t, e')$ を比較して、小さい方の ID を持つノードへフォワーディングする。自ノードを s とし、 $d(s, t) > d(e, t)$ となる ID e のノードが経路表中に存在しなくなったら終了し、そのときの s が担当ノードである。このフォワーディングにより、経路表中に t の担当ノードのエントリが含まれていれば、one-hop が可能である。また、FRT-2-Chord のフォワーディング方法は t から見て時計回りと反時計回りの方向から t へ近づいていくため、両方向である。

3.2 ID に関する順序関係 \leq_{ID}

ID に関する順序関係 \leq_{ID} を定義するにあたって、最悪短縮倍率を定義する。 s の経路表を E とする。また、 $i < j \Rightarrow d_{chord}(s, e_i) < d_{chord}(s, e_j)$, $(e_i, e_j \in E)$ である。各エントリ e_i にフォワーディングする際の短縮倍率を考える。 e_i から t へ時計回りに進んだほうが近い場合と、反時計回りに進んだほうが近い場合に分ける必要がある。前者の場合、 t が e_i と e_{i+1} の中点のときに最悪値をとり、後者の場合、反時計回りの方向で t が e_i と e_{i-1} の中点のときに最悪値をとると考えられる。ただし、 e_i と e_{i+1} の中点は、 e_i から e_{i+1} へ時計回りに進んだときに途中にある方の中点であり、 e_i と e_{i-1} の中点は、 e_i から e_{i-1} へ反時計回りに進んだときに途中にある方の中点である。 e_i 時計回り方向の最悪短縮倍率を $r_i^{cw}(E)$ 、反時計回り方向の最悪短縮倍率 $r_i^{ccw}(E)$ とする。また、 e_k を s と s から時計回りに 2^{m-1} 先との間にあり、 $s + 2^{m-1}$ に最も近いエントリであるとする。 $d(e_i, e_{i+1}) = |d(s, e_i) - d(s, e_{i+1})|$, $(i \neq k)$ なので $r_i^{cw}(E)$ と $r_i^{ccw}(E)$ は次のように定義できる。

定義 3.2 時計回り方向の最悪短縮倍率 $r_i^{cw}(E)$ と反時計回り方向の最悪短縮倍率 $r_i^{ccw}(E)$

$$\begin{aligned} & i \neq k \text{ のとき,} \\ r_i^{cw}(E) &= \frac{|d(s, e_{i+1}) - d(s, e_i)|}{d(s, e_{i+1}) + d(s, e_i)} \\ & i = k \text{ のとき,} \\ r_i^{cw}(E) &= \frac{2^m - d(s, e_{i+1}) - d(s, e_i)}{2^m - |d(s, e_{i+1}) - d(s, e_i)|} \end{aligned}$$

$$i \neq k + 1 \text{ のとき,}$$

$$r_i^{\text{ccw}}(E) = \frac{|d(s, e_i) - d(s, e_{i-1})|}{d(s, e_i) + d(s, e_{i-1})}$$

$$i = k + 1 \text{ のとき,}$$

$$r_i^{\text{ccw}}(E) = \frac{2^m - d(s, e_i) - d(s, e_{i-1})}{2^m - |d(s, e_i) - d(s, e_{i-1})|}$$

定義から明らかなように, $r_i^{\text{cw}}(E) = r_{i+1}^{\text{ccw}}(E)$, ($i = 1, 2, \dots, |E| - 1$) なので, $\{r_i^{\text{cw}}(E)\} \cup \{r_i^{\text{ccw}}(E)\} = \{r_i^{\text{cw}}(E)\}$. $r_i(E) = r_i^{\text{cw}}(E)$, ($i = 1, 2, \dots, |E| - 1$) と定義する. $\{r_i(E)\}$ を降順に並べた列を $\{r_{(i)}(E)\}$ とし, ID に関する経路表の順序関係を定義する.

定義 3.3 $r_i(E)$ による順序関係 \leq_{ID}

$$E \leq_{\text{ID}} F \Leftrightarrow \{r_{(i)}(E)\} \leq_{\text{dic}} \{r_{(i)}(F)\}$$

ただし, \leq_{dic} は辞書式順序である.

3.3 到達性保証操作

FRT-2-Chord では, sticky entry に含まれるエンタリや到達性保証操作が FRT-Chord と異なる.

到達性とは, フォワーディングが終了した際に目的 ID の担当ノードにクエリを到達させる性質である. FRT-2-Chord のフォワーディングは, $d(s, t) > d(e, t)$ となる ID e のノードが経路表中に存在しなくなったら終了する. したがって, 各ノードにおいて successor と predecessor が正しいエンタリであれば, フォワーディングの終了時に担当ノードへ到達する. ノードの離脱を考えると, successor list と同様に, 自ノードから反時計回りに進んで出会う定数個のノード predecessor list も sticky entry として必要である.

また, Chord の stabilize と異なる方法で, successor と predecessor の両方のノードに対して生存確認を行い, sticky entry の更新を行う必要がある. **表 2** は FRT-2-Chord の到達性保証操作の疑似コードである. 手続き `contactWithSuccessor` において, s は s の経路表における successor $succ$ と通信を行う. 手続き `replyToPredecessor` は, predecessor 側から到達性保証に関するメッセージを受けた際に呼ばれる. 疑似コードにおいて, B の初期値は $null$ であり, メッセージ `stabilize` に返答がなかったエンタリの集合である. I はメッセージ `stabilize` の差出人 $pred$ から時計回り方向の s との間にあり, s の経路表に含まれるエンタリの集合である. また, `contactWithPredecessor` と `replyToSuccessor` も同様の操作である.

到達性保証操作の通信のみを考えれば, より近い側のノード, つまり, successor list は自ノードの successor, predecessor list は自ノードの predecessor が

表 2 FRT-2-Chord における到達性保証操作の疑似コード
Table 2 Pseudocode for guarantee of reachability in FRT-2-Chord

(1)	<code>s.contactWithSuccessor(){</code>
(2)	<code>success = false;</code>
(3)	<code>while(success ≠ true){</code>
(4)	<code>succ = e₁;</code>
(5)	<code>send (stabilize, s, predecessor list, B) to succ;</code>
(6)	<code>if(succ is failed){</code>
(7)	<code>remove succ from E;</code>
(8)	<code>add succ to B;</code>
(9)	<code>}</code>
(10)	<code>else {</code>
(11)	<code>success = true;</code>
(12)	<code>}</code>
(13)	<code>}</code>
(14)	<code>receive (reply, succ, S, I) from succ;</code>
(15)	<code>remove successor list from E;</code>
(16)	<code>add entries in S and I to E;</code>
(17)	<code>if(succ ≠ e₁){</code>
(18)	<code>contactWithSuccessor();</code>
(19)	<code>}</code>
(20)	<code>}</code>
(21)	<code>s.replyToPredecessor(){</code>
(22)	<code>receive(stabilize, pred, P, B) from pred;</code>
(23)	<code>remove entries in B from E and I;</code>
(24)	<code>remove predecessor list not in I from E;</code>
(25)	<code>add pred and entries in P to E;</code>
(26)	<code>send (reply, s, successor list, I) to pred;</code>
(27)	<code>}</code>

正しいものを持っている*ので, 正しいリストを交換し合う (5, 14-17, 22, 24-26 行目) 必要がある.

また, `contactWithSuccessor` において B のエンタリを $succ$ に伝えて (5, 8 行目), メッセージ `stabilize` を受け取ったノードは経路表から削除 (23 行目) する必要がある. これらを行わないと, `replyToPredecessor` において, s はそれらのノードを再び $pred$ への返信メッセージ `reply` に含めてしまう. 逆に, s が predecessor と通信を行う場合も同様である. そのため, 離脱したノードの情報が経路表に残り続けてしまう. したがって, このような操作となる.

また, successor list と predecessor list の交換におけるエンタリの削除 (15, 24 行目) や B のエンタリの削除 (23 行目) を実行するノードが, 悪意のあるノードによって離脱していないエンタリを削除させられることで, 到達性が失われる場合が考えられる. この問題に対し, エンタリを削除する際に生存確認のメッセージを送る処理を追加することが解決策として

* 到達性保証操作以外の通信, つまり, 探索による通信によって, より遠い側のノードが先に正しい successor list や predecessor list を持つ場合もある. しかし, いずれは近い側のノードの経路表も到達性保証操作によって正しさが保たれる.

考えられる。

Chord の stabilize では、自ノードの predecessor が離脱して、新しい predecessor から stabilize による通信を行われても自ノードの predecessor は変わらない。従って、離脱したノードを predecessor とし続けることがある。FRT-2-Chord の到達性保証操作を Chord にも適用することで、この誤りを正すことができる。

3.4 エントリ厳選操作

厳選操作を定義するにあたって、 e_i の削除を仮定した際に更新される最悪短縮倍率 R_i^E を定義する。

定義 3.4 e_i の削除を仮定した際に更新される最悪短縮倍率 R_i^E

$$i \neq k, i \neq k+1 \text{ のとき,}$$

$$R_i^E = \frac{|d(s, e_{i+1}) - d(s, e_{i-1})|}{d(s, e_{i+1}) + d(s, e_{i-1})}$$

$$i = k \text{ または } i = k+1 \text{ のとき,}$$

$$R_i^E = \frac{2^m - d(s, e_{i+1}) - d(s, e_{i-1})}{2^m - |d(s, e_{i+1}) - d(s, e_{i-1})|}$$

FRT-2-Chord の厳選操作は次の操作である。

- (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。
- (2) C から sticky entry を除外。
- (3) C のエントリの中で R_i^E が最小となるエントリ e_i を選択し、経路表から削除。
 R_i^E を昇順に保持しておくことで効率よく操作が実行できる。また、FRT-Chord と同様に、FRT-2-Chord の厳選操作に関する次の定理が言える。

定理 3.1 削除の最適性

経路表 E からエントリ厳選操作により作成される経路表を $E - \{e_{i^*}\}$ とするとき、任意の削除対象エントリ e_i ($i = 2, 3, \dots, |E| - 1$) について、 $E - \{e_{i^*}\} \leq_{\text{ID}} E - \{e_i\}$ が成立する。

証明 削除後の経路表 E に関する最悪短縮倍率を考える。 $l^* = \min\{j | R_j^E < r_j(E)\}$ とするとき、 $\{r_{(i)}(E - \{e_{i^*}\})\}$ の先頭から $l^* + 1$ 番目までは、 $r_{(1)}(E), r_{(2)}(E), \dots, r_{(l^*-1)}(E), R_{i^*}^E, r_{(l^*)}(E)$ となる。このとき、 $l^* - 1$ 番目までは削除前と変化しない。同様に、 $l = \min\{j | R_j^E < r_j(E)\}$ とするとき、 $\{r_{(i)}(E - \{e_i\})\}$ の先頭から $l + 1$ 番目までは、 $r_{(1)}(E), r_{(2)}(E), \dots, r_{(l-1)}(E), R_i^E, r_{(l)}(E)$ となる。このとき、 $l - 1$ 番目までは削除前と変化しない。ここで、 i^* の決定方法から $R_{i^*}^E \leq R_l^E$ が成立するので、 $l^* \geq l$ 。したがって、 $i = 1, 2, \dots, l - 1$ において、 $r_{(i)}(E - \{e_{i^*}\}) = r_{(i)}(E - \{e_i\})$ が成立し、かつ $r_{(l)}(E - \{e_{i^*}\}) \leq r_{(l)}(E - \{e_i\})$ が成立する。よって、 $E - \{e_{i^*}\} \leq_{\text{ID}} E - \{e_i\}$ が成立する。□

学習したエントリと削除するエントリが常に一致し、経路表の改良が停止することが考えられる。このときの経路表 E を収束した経路表と呼ぶ。

定理 3.2 厳選操作の非停止性

ノード数 N のネットワークにおいて、全ノードが収束した経路表 E を持つとき、 $|E| = O(\log N)$ の条件下で経路長は $O(\log N)$ である。

証明 S_i^E を次のように定義する。

$$i = 1, 2, \dots, k - 1 \text{ のとき, } S_i^E = \log \frac{d(s, e_{i+1})}{d(s, e_i)}$$

$$i = k + 1, \dots, |E| - 1 \text{ のとき, } S_i^E = \log \frac{d(s, e_i)}{d(s, e_{i+1})}$$

$$d(s, e_{k+1}) \leq d(s, e_k) \text{ のとき,}$$

$$S_k^E = \log \frac{2^m - d(s, e_k)}{d(s, e_{k+1})}$$

$$d(s, e_{k+1}) > d(s, e_k) \text{ のとき,}$$

$$S_k^E = \log \frac{2^m - d(s, e_{k+1})}{d(s, e_k)}$$

また、時計回りの方向で e_i から e_{i+1} までの間にノードが存在する i の集合を J 、時計回りの方向で e_{i-1} から e_i までの間にノードが存在する i の集合を L 、時計回りの方向で e_{i-1} から e_{i+1} までの間に e_i 以外のノードが存在しない i の集合を P と定義する。

$r_i(E) = 1 - 2/(2^{S_i^E} + 1)$ であるため、 S_i^E と $r_i(E)$ の大小は一致する。また $R_i^E = 1 - 2/(2^{S_i^E + S_{i-1}^E} + 1)$, ($i \neq k, i \neq k + 1$) であるので、 $S_i^E + S_{i-1}^E$ と R_i^E の大小は $i \neq k$ かつ $i \neq k + 1$ のとき一致する。収束した経路表の定義から、 $j \in J, i = 2, \dots, |E| - 1, r_j^{\text{cw}}(E) = r_j(E) \leq R_i^E$ が成立する。同様に、 $l \in L, i = 2, \dots, |E| - 1, r_l^{\text{ccw}}(E) = r_{l-1}(E) \leq R_i^E$ が成立する。一般に N が大きいとき十分高い確率でノード間の距離は $2^m/N^2$ より大きく⁷⁾、明らかに $2^m > N$ である。また、 $d(s, e_k) \leq 2^{m-1}$, $d(s, e_{k+1}) < 2^{m-1}$ である。 j を固定して、次が成立する。

$$S_j^E \leq \frac{\sum_{2 \leq i \leq |E|-1, i \neq k, i \neq k+1} (S_i^E + S_{i-1}^E)}{|E| - 4}$$

$$\leq \frac{2(\sum_{1 \leq i \leq |E|-1, i \neq k} S_i^E)}{|E| - 4}$$

$$= \frac{2}{|E| - 4} \log \left(\frac{d(s, e_k) * d(s, e_{k+1})}{d(s, e_1) * d(s, e_{|E|})} \right)$$

$$< \frac{2}{|E| - 4} \log \left(\frac{N^4}{4} \right) < \frac{8}{|E| - 4} \log N$$

よって、 $|E| = 4 + (8/\log 3) \log N$ とすると、

$$r_j(E) < 1 - \frac{2}{N^{8/|E|-4} + 1} = \frac{1}{2}$$

同様にして、 $r_{l-1}(E) < 1/2$ 。ここで、残り距離を $2^m/N$ 以下にするために必要な経路長の上限を考え

る。フォワーディング先が $e_p (p \in P)$ のとき、担当ノードは e_p であり、ルーティングが終了する。それ以外の場合、つまり、 $e_i (i \in J$ または $i \in L)$ の場合を考える。残り距離は 2^{m-1} 以下であり、自ノードから目的 ID に対して時計回りにフォワーディングする際も、反時計回りにフォワーディングする際も最悪短縮倍率は常に $1/2$ より小さいので、 $\log N - 1$ 回以下のフォワーディングで残り距離が $2^m/N$ となる。自ノードから時計回りの方向、または、反時計回りの方向で、 $2^m/N$ 先までに存在するノードは、 N が大きいとき十分な確率で $O(\log N)$ 個となることが知られており¹⁾、その経路長は $O(\log N)$ となる。以上より、経路全体の経路長は $O(\log N)$ である。□

3.5 構造化オーバーレイに関する要求への対応

1 章や 2 章で触れた構造化オーバーレイに関する要求を FRT-2-Chord は全て応えている。

3.5.1 ノード数の多寡に依存しない効率

FRT-2-Chord では、ID 距離の定義に従って、担当ノードとフォワーディングの目標となるノードが一致している。そのため、経路表に担当ノードを保持しているとき、one-hop が可能である。また、経路表サイズがノード数を上回る場合、各ノードが全ノードの情報を経路表に保持でき、任意のノード間で one-hop となる。また、定理 3.2 によって、ノード数 N に対して $O(\log N)$ の経路表サイズで $O(\log N)$ の経路長となることを示した。ノード数を知る必要なしに、one-hop と multi-hop のシームレスな移行が可能である。

3.5.2 経路の選択に影響する複数の要素への対応

経路表に ID による制限がないため FRT-2-Chord は経路表の柔軟性を持っている。この柔軟性は FRT に由来するものである。また、FRT-2-Chord の ID 距離は任意の ID x, y に対し、 $d(x, y) = d(y, x)$ が成り立つ。ID 距離は対称性を持ち、経路表は ID に関する順序関係によって構築されるので、経路表の対称性を持ちやすい。対称性はネットワーク近接性やグループを考慮した経路表の構築に拡張される際に有用である。

3.5.3 ノードの参加と離脱の頻度に依存しない到達性

ID 空間において任意のノードの配置で、任意のノードから任意の目的 ID に対し、目的ノードへの到達性を保証するために最低限必要なエントリの数の最大値を n とする。例えば、Chord は 1、Kademlia は ID 空間の bit 数である。 n が大きいほど、到達性保証のために必要なノードの数は大きいことになる。従って、参加や離脱によって到達性が失われやすいと考えられる。FRT-2-Chord では $n = 2$ であるので、十分な到

達性を維持できる。

3.6 その他の有用性

Chord を基に提案されてきたアルゴリズムになかった FRT-2-Chord の有用な性質について述べる。

3.6.1 複製の有効活用

一般的にノードの離脱によってデータが失われることを防ぐために複製を配置する。FRT-2-Chord では、ID t の担当ノードが最も近いノードなので、 t の複製は t に近い定数 n 個のノードに配置される方法が自然である。 t を探索する際に、フォワーディング先は t に最も近いノードなので、複製が配置されたノードにたどり着きやすい。この複製への到達によって、経路長の削減やデータを保持するべきである元々のノードへの負荷の集中が防げる。

3.6.2 複製の管理のしやすさ

複製は適切な配置で定数 n 個に保たせる必要がある。時計回りと反時計回りのそれぞれにおいて、 n 個先のノードと自ノードの midpoint より近い ID を持つデータの複製が、自ノードが持つべき複製である。successor list と predecessor list から自ノードが持つべき複製が計算でき、他ノードの join による自ノードの保持が不要な複製を削除することができる。また、適切な配置で定数個の複製を保つために、join したノードへの委譲と離脱したノードの埋め合わせをする再配布が必要である。前者は、join 操作時に行えばよい。後者は、predecessor と successor による再配布がある。predecessor による委譲は、ノード s の predecessor の predecessor list が到達性保証操作によって更新されたとき、更新によって新たに s が保持するべき複製を計算でき、 s の predecessor は s に複製を再配布することができる。successor による再配布も同様である。

4. 評価

オーバーレイネットワーク構築ツールキットである Overlay Weaver⁸⁾⁹⁾ 上に FRT-2-Chord を実装し、エミュレーションにより実験、評価を行った。実験には、Overlay Weaver 0.10.3 を用い、OS が Mac OS X 10.7、CPU が 2.3 GHz Intel Core i5、メモリが 8 GB のマシン上で Java SE 6 Update 26 を用いた。

4.1 multi-hop における経路長の短さ

全ノードをネットワークに参加させた後、ランダムに選択したノードからランダムに選択した ID への探索をノードあたり 200 回行う実験を行った。ノード数を $10^2, 10^3, 10^4$ とし、経路表サイズの上限は 160、FRT-Chord の successor list のサイズは 4、FRT-2-Chord の successor list のサイズは 4、predecessor list のサ

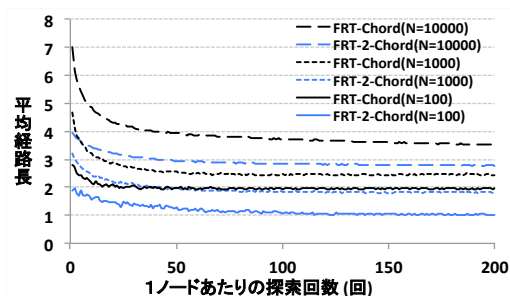


図 1 1 ノードあたりの探索回数に対する平均経路長の変化
Fig.1 Change of average path length along with the number of lookups per node.

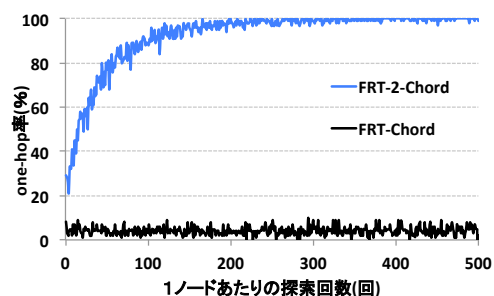


図 2 1 ノードあたりの探索回数に対する one-hop 率
Fig. 2 Rate of one-hop along with the number of lookups per node

イズは 4 である。フォワーディングは、探索を行うノードが次のフォワーディング先の問い合わせを繰り返す反復方式で行った。

図 1 は、1 ノードあたりの探索回数に対する平均経路長の変化を示している。FRT-Chord と同様に FRT-2-Chord でも探索回数に応じて平均経路長が短縮してることがわかる。

表 3 は 1 ノードあたり 150 回から 200 回探索時における FRT-Chord と FRT-2-Chord の平均経路長を集計し、ノード数ごとに求めたものである。また、 $A-B$ は FRT-Chord の平均経路長の値から FRT-2-Chord の平均経路長の値を引いた差である。FRT-Chord より FRT-2-Chord のほうが multi-hop において短い経路長であることを示している。

FRT-2-Chord のエントリ厳選操作を繰り返すことで、経路表が改良されていると言える。また、FRT-2-Chord は、FRT-Chord より経路長が短い。

4.2 one-hop の実現

4.1 節の実験でノード数が 10^2 の場合、経路表サイズがノード数を上回るため、経路表が全ノードを持つことが可能である。図 2 は 1 ノードあたりの探索回数に対する one-hop の回数を集計し、それをノード数で割って 100 を乗じた one-hop 率を計算し、まとめた図である。one-hop 率の計測においては、1 ノードあたり 500 回探索を行った。今回の実験において、1 ノードあたり 500 回探索後では 95 % 以上の探索が

表 3 1 ノードあたり 150 回から 200 回探索時における FRT-Chord と FRT-2-Chord の平均経路長の比較

Table 3 Comparing average path length from 150 to 200 lookups per node for FRT-Chord and FRT-2-Chord

ノード数	FRT-Chord(A)	FRT-2-Chord(B)	$A - B$
100	1.958	1.035	0.923
1000	2.458	1.825	0.633
10000	3.565	2.788	0.777

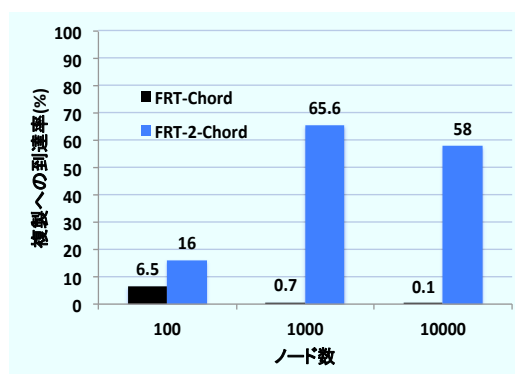


図 3 複製への到達率
Fig. 3 Rate of reaching replicas

one-hop であり、任意のノード間で one-hop であるとはいえない。しかし、1 ノードあたり、さらに 700 回探索したところ、one-hop 率は 100 % を維持した。

4.3 複製への到達

全ノードをネットワークに参加させた後、ランダムに選択したノードからランダムに選択した ID を持つデータの put をノードあたり 100 回行い、その後、それらのデータの ID の中からランダムに選択した ID への探索をノードあたり 100 回行った。複製数は 8 として実験を行った。複製数とは、担当ノードを含めた複製を持たせるノードの数である。

図 3 は、1 ノードあたり 100 回のデータ探索における複製への到達率をノード数ごとにまとめた図である。複製への到達率は、担当ノード以外のノードに保存されている目的データへ到達した回数を全探索回数で割って 100 を乗じた値である。FRT-2-Chord において、ノード数が 10^3 , 10^4 のとき、複製への到達がされやすいことがわかる。ノード数が 10^2 のときは、one-hop が頻繁にされるため、複製への到達率は小さくなっていると考えられる。今回の実験における

複製数は必ずしも適切なものとは言えないが、FRT-2-Chordにおいて、複製への到達がしやすいことが確認できた。

5. ま と め

FRTに基づいて設計したDHTアルゴリズムFRT-2-Chordを設計し、実験や証明を行った。FRT-2-Chordは、ノード数、ID距離、ネットワーク近接性、グループ、ノードの参加と離脱の頻度に関する順応性があるアルゴリズムであることを示すことができた。

今後の課題は、実際のP2Pシステムに適用をするためにFRT-2-Chordを拡張することである。特に、経路の選択に影響する要素に対する考慮に関して、FRTの性質や経路表の対称性を持っているが、それは、拡張性があるという段階なので、拡張する。

謝辞 本研究は科研費(22680005)および独立行政法人情報通信機構(NICT)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」の助成を受けたものである。

参 考 文 献

- 1) Stoica, I., Morris, R., Karger, D., Kaashoek, M. and Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications, *ACM SIGCOMM Computer Communication Review*, Vol. 31, No. 4, pp. 149–160 (2001).
- 2) Rowstron, A. and Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, *IFIP/ACM Middleware 2001*, pp. 329–350 (2001).
- 3) Maymounkov, P. and Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric, *IPTPS 2002*, pp. 53–65 (2002).
- 4) Leong, B., Liskov, B. and Demaine, E.: Epi-chord: parallelizing the Chord lookup algorithm with reactive routing state management, *Computer communications*, Vol. 29, No. 9, pp. 1243–1259 (2006).
- 5) Fonseca, P., Rodrigues, R., Gupta, A. and Liskov, B.: Full-Information Lookups for Peer-to-Peer Overlays, *IEEE Transactions on Parallel and Distributed Systems*, pp. 1339–1351 (2009).
- 6) Nagao, H. and Shudo, K.: Flexible routing tables: Designing routing algorithms for overlays based on a total order on a routing table set, *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, IEEE, pp. 72–81

- (2011).
- 7) Cordasco, G. and Sala, A.: 2-Chord halved, *HOT-P2P'05*, pp. 72–79 (2005).
- 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) 首藤一幸: *Overlay Weaver: An Overlay Construction Toolkit*. <http://overlayweaver.sourceforge.net/>.