Vol.2011-DPS-147 No.24
Vol.2011-MBL-58 No.24
2011/6/3

構造化オーバレイにおけるバーチャルノード融合

長尾洋也[†1]　首藤一幸[†1]

DHT におけるバーチャルノードの自律的な動作を維持しつつ，マシン単位の効率を向上する手法であるバーチャルノード融合 (VNF) を提案する．VNF は，同一マシン上に存在するバーチャルノードの持つ性質を仮定し，バーチャルノードを考慮した経路表構築とフォワーディングを行う．また，VNF はマシンごとのデータ構造を利用せずに，マシン上の各バーチャルノードは経路表の管理やキャッシュの管理，各 ID へのフォワードなどの役割を分担することで，バーチャルノードを効率よく利用する．

我々は VNF に基づく DHT アルゴリズム VNF-Chord を設計および実装を行った．実験の結果から，VNF-Chord がマシンをまたぐフォワーディング回数を削減することが確かめられた．

Virtual Node Fusion for Structured Overlay

Hiroya Nagao[†1] and Kazuyuki Shudo[†1]

This paper presents *Virtual Node Fusion* (VNF), which maintain autonomy of virtual nodes and improves efficiency of them in DHT. VNF constructs suitable routing tables for using virtual nodes by assuming virtual nodes' characteristics. Each virtual node shares routing table construction, cache maintenance in a machine, and forwardings without using data structures for the machine.

We designed VNF-Chord, a VNF-based DHT, and implemented it. Experiment results show that VNF-Chord reduces the number of times of forwardings over machines.

1. はじめに

Distributed Hash Table (DHT)[1)–4)] はオーバレイネットワーク上にデータを分散保存する手法として研究されてきた．データの分散の根拠は，ハッシュ関数がランダムにノード ID とキー ID を設定することにある．しかし実際には，ハッシュ関数を利用したとしても，各ノードが担当するキー ID の領域に偏りが発生してしまうため，必ずしもデータの均等分散に十分な手法ではない．そのため，この偏りを改善する汎用な手法として，複数のノードを単一マシン上で動作させるバーチャルノードという手法が有効である[1)]．

バーチャルノードを利用する DHT では，各マシンが複数ノードを起動する．このノードを，マシンと一対一には対応しないことから，バーチャルノードと呼ぶ．バーチャルノードを利用することで，一つのマシンが複数のバーチャルノードの担当領域を合わせた領域を担当するため，結果として，一つのマシンが担当する領域の均等化が図られ，負荷分散性能が向上する[1)]．

しかし，バーチャルノードは，単一マシン上で複数ノード起動するという単純な方法で実現できる反面，非効率な点が多い．たとえば，一度マシン外へフォワードされたメッセージが再び他のマシン上のバーチャルノードを経由して自マシン上のバーチャルノードへ戻ってくる可能性がある．また，単一マシン上の各バーチャルノードが構築する経路表は完全に独立して構築されるため，その間に重複が発生してしまう問題や，マシン上の他のバーチャルノードが保持しているキャッシュを利用しない問題がある．

我々はバーチャルノードを効果的に利用する手法である**バーチャルノード融合**を提案する．

本稿では，第 2 章でバーチャルノード融合の元となる手法である柔軟な経路表の概要を示し，第 3 章で，柔軟な経路表に基づく DHT アルゴリズム FRT-Chord を紹介する．第 4 章において提案手法であるバーチャルノード融合を説明し，第 5 章においてバーチャルノード融合に基づいて設計した DHT アルゴリズム VNF-Chord を説明する．第 6 章で VNF-Chord を評価する．

2. 柔軟な経路表

柔軟な経路表（FRT）[5)] に基づくアルゴリズムは，一つの経路表 $E = \{e_i\}$ を管理する．FRT は構成しうる経路表候補間の順序関係 $\leq_{\mathrm{ID}}$ を定義し，$\leq_{\mathrm{ID}}$ を利用して経路表の改良を繰り返す．具体的に経路表管理は，到達性保証操作，エントリ学習操作，エントリ厳選操作の 3 つから構成される．

2.1 到達性保証操作

メッセージの到達性を保証するための操作を FRT では**到達性保証操作**と呼び，この操作により，メッセージの到達性を保証するために必要な経路表エントリの取得や更新を行う．

†1 東京工業大学 大学院情報理工学研究科 数理・計算科学専攻
Dept. of Mathematical and Computing Sciences, Graduate School of Information Science and Engineering, Tokyo Institute of Technology

Vol.2011-DPS-147 No.24
Vol.2011-MBL-58 No.24
2011/6/3

2.2 エントリ情報学習操作

経路表エントリを構築するのに必要な情報（エントリ情報と呼ぶ）を入手し，すでに経路表に含まれていた場合を除き，経路表にエントリを追加することを**エントリ情報学習操作**と呼ぶ．エントリ情報学習操作を繰り返すことで，経路表のエントリ数は増加し，よりよいフォワーディング先を選択可能となる．また，経路表にシステム上の全ノードを載せることで，目的ノードに直接到達することも可能となる．

2.3 エントリ厳選操作

エントリ学習操作を繰り返すことで，経路表に含まれるエントリ数は増大し続けるため，経路表エントリの取捨選択が必要となる．経路表から $\leq_{\mathrm{ID}}$ に基づいて経路表エントリを削除する操作をエントリ厳選操作と呼ぶ．$\leq_{\mathrm{ID}}$ の考慮はこの操作に集約される．そのため，$\leq_{\mathrm{ID}}$ はエントリ厳選操作を意識して構成される．また，$\leq_{\mathrm{ID}}$ の構成を容易にするために *sticky entry* と呼ばれる，削除対象にしない経路表エントリを導入する．

経路表を E とするとき，削除候補エントリの集合を C と表記することにすると，エントリ厳選操作は以下の手順で行われる．

(1) $C \leftarrow E$
(2) C から sticky entry を取り除く．
(3) C から $\leq_{\mathrm{ID}}$ に基づいてより良い経路表が残るように削除対象を選択する．

3. FRT-Chord

この章では，FRT に基づく DHT アルゴリズム *FRT-Chord*[5] を説明する．FRT-Chord は，Chord の用いる ID 空間および担当ノードの決定方法を利用する．

3.1 順序関係 $\leq_{\mathrm{ID}}$

FRT-Chord は，ノード s が保持する経路表 $E = \{e_i\}$ に対して，経路表エントリ e_i へのフォワーディングにおける**残り距離の短縮倍率**を $\mathrm{d}(e_i, t)/\mathrm{d}(s, t)$ と定め，その最悪値 $\mathrm{r}_i(E)$ を次のように定義する．

定義 3.1 c を successor list の長さとしたとき，

$$\mathrm{r}_i(E) = \frac{\mathrm{d}(e_i, e_{i+1})}{\mathrm{d}(s, e_{i+1})}, (i = c, \ldots, |E| - 1). \tag{1}$$

ただし，距離関数 $\mathrm{d}(x, y)$ は，Chord リング上時計回りの距離として，次のように定義される．

定義 3.2

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

FRT-Chord は，$\{\mathrm{r}_i(E)\}$ を降順に並べた列 $\{\mathrm{r}_{(i)}(E)\}$ に基づいて，辞書式順序 $\leq_{\mathrm{dic}}$ を利用して次のように経路表間の順序関係を定義する．

定義 3.3

$$E \leq_{\mathrm{ID}} F \Leftrightarrow \{\mathrm{r}_{(i)}(E)\} \leq_{\mathrm{dic}} \{\mathrm{r}_{(i)}(F)\} \tag{3}$$

このとき，

$$\forall i, j \in \{c, \ldots, |E| - 1\}, \mathrm{r}_i(\tilde{E}) = \mathrm{r}_j(\tilde{E}) \tag{4}$$

を満たす経路表 $\tilde{E}$ は**最良経路表**と呼ばれ，任意の経路表候補 E に対して $\tilde{E} \leq_{\mathrm{ID}} E$ を満たす．$\leq_{\mathrm{ID}}$ をこの最良経路表からの乖離度として捉えることも出来る．

3.2 エントリ情報学習操作

FRT-Chord のエントリ学習操作は，以下の 3 つにより行う．

- 参加時に，自ノードの successor の持つ経路表に含まれる全エントリを学習する．
- あらゆる通信時に，通信相手を学習する．
- 学習するための探索（**能動学習探索**）を行い，通信相手を学習する．

能動学習探索では，0 以上 1 未満の乱数 rnd を用いて，$s + \mathrm{d}(s, e_c)(\mathrm{d}(e_{|E|})/\mathrm{d}(s, e_c))^{\mathrm{rnd}}$ で生成されるキー ID に対して探索を行う．これにより，効率よく経路表を構築することが出来る．

3.3 エントリ厳選操作

FRT-Chord は，正規化間隔 S_i^E を次のように定義する．

定義 3.4

$$S_i^E = \log \frac{\mathrm{d}(s, e_{i+1})}{\mathrm{d}(s, e_i)}, \qquad (i = 1, \ldots, |E| - 1) \tag{5}$$

そして，削除対象 e_{i^*} を，$S_{i^*-1}^E + S_{i^*}^E$ が最小となるエントリとする．この削除対象エントリは，$\{S_{i-1}^E + S_i^E\}$ を昇順に保持しておくことで，効率よく選択することが出来る．また，削除の候補である任意のエントリ e_i に対して $E \setminus \{e_{i^*}\} \leq_{ID} E \setminus \{e_i\}$ が成立し，FRT-Chord が最善の削除対象を，効率よく発見できることを意味している．

以上から，FRT-Chord の厳選操作アルゴリズムは次のように記述できる．

(1) $C \leftarrow E$

Vol.2011-DPS-147 No.24
Vol.2011-MBL-58 No.24
2011/6/3

(2) C から sticky entry である successor list および predecessor を取り除く．
(3) C から，$S^E_{i-1} + S^E_i$ が最小になる経路表エントリを削除する．

4. バーチャルノード融合

この章では，提案手法である**バーチャルノード融合**（VNF）を説明する．VNF は，バーチャルノードの利用を想定した DHT アルゴリズムの設計方式であり，バーチャルノードというアプローチの単純さを保ちつつ，バーチャルノードを効率よく利用することを目的とする．このとき VNF はマシン単位のデータ構造を構成せず，依然非集中で自律的にバーチャルノードが動作する．また，バーチャルノード間で経路表の内容や，キャッシュの情報を交換したり同期するといった新たな通信を追加することは必要なく，既存の DHT 改良手法との相性がよいと言える．

VNF は，同一マシン上のバーチャルノード同士で役割を分担し合い，マシン単位の効率を向上する．ここでの役割とは，経路表エントリを保持する役割や，マシン内キャッシュを管理する役割，各 ID へのメッセージをマシン外へフォワーディングする役割などである．

VNF は，以下に挙げるバーチャルノードの持つ特性を仮定する．

- ノードの故障はマシン単位で発生する．つまり，同じマシン上で動作しているバーチャルノードの故障は同時に発生する．したがって，あるバーチャルノードが故障したときに，同一マシン上のほかのバーチャルノードをその代わりとして利用することは出来ない．
- ノードへの通信経路の故障もマシン単位で発生する．あるバーチャルノードとの通信が不可能になった場合，同じマシン上で動作している他のバーチャルノードとの通信も不可能となる．
- 同一マシン内のフォワードは，マシンをまたぐフォワードと比較して無視できるほど小さい．
- 同一マシン上のバーチャルノード間のリソース使用量の偏りは問題にならない．同じマシン上で動作しているバーチャルノードは，互いに同じハードウェアリソースを共有しているため，バーチャルノード間におけるリソース使用量の偏りは問題にならず，マシン単位で問題となる．
- バーチャルノードの数が，全マシンで同一であるとは限らない．

我々は，VNF に基づく DHT の設計にあたり，**柔軟な経路表**（FRT）を適用する．なぜなら，FRT の持つ性質が VNF に基づく DHT を設計するのに有用だからである．

同一マシン上のバーチャルノード間の経路表エントリの重複を防ぐためには，経路表に含まれる ID の組み合わせを任意に限定し，その限定の中でよりよい経路表を構成できる FRT の性質が有効である．一方で，マシン全体でのリソース使用量を考慮した場合，経路表サイズはマシン単位で指定されるべきであり，バーチャルノードごとの経路表サイズを同一に限定する必要はない．この性質を活かすためには，F 経路表サイズを自由に，そして動的に変更できる FRT の性質が有効である．また，バーチャルノードとの通信負荷が無視できるほど小さいことを仮定するため，経路表に常に同一マシン上のバーチャルノードを載せることが好ましい．FRT では，それらのエントリを sticky として指定するだけで実現できると同時に，sticky entry と指定されたエントリの ID を考慮した経路表構築が可能である．これは，通常利用する経路表とは別に同一マシン上のバーチャルノード用の経路表を用意するより効果的なアプローチである．

4.1 方 式 概 要

VNF は，各マシン上に存在する各バーチャルノード v_i に対して，**マシン内担当領域** D_i を割り当てる．このとき，D_i は互いに重なることがなく，その和集合が ID 領域全体と一致する．したがって，同一マシン上のバーチャルノード同士で ID 領域を分担することになる．そして，VNF に基づくアルゴリズムでは，領域 D_i 宛のメッセージはバーチャルノード v_i を経由してマシン外にフォワードする．これは同時に，各バーチャルノード v_i が領域 D_i 宛のメッセージをフォワードすることに最適化した経路表構築を行うことを意味する．

5. VNF-Chord

この章では，VNF に基づいて設計した DHT アルゴリズム *VNF-Chord* について述べる．VNF-Chord は，FRT-Chord を拡張したアルゴリズムである．

あるバーチャルノード s と同一マシン上にある s 以外のバーチャルノードの集合を $V^s = \{v^s_i\}_{i=1,\ldots,|V^s|}$ と定義する．このとき，$i < j \Rightarrow \mathrm{d}(s, v^s_i) < \mathrm{d}(s, v^s_j)$ とし，$v^s_0 = s$ とする．また，ノード s の経路表を $E^s = \{e^s_i\}_{i=1,\ldots,|E^s|}$ と定義し，$i < j \Rightarrow \mathrm{d}(s, e^s_i) < \mathrm{d}(s, e^s_j)$ とする．

ID 空間全体を I とするとき，バーチャルノード v^s_i の**マシン内担当領域** D^s_i を次のように定義する．

$$D^s_i = \{x \in I | \mathrm{d}(v^s_i, x) < \mathrm{d}(v^s_i, v^s_{i+1})\}, \qquad (i = 0, \ldots, |V^s|) \tag{6}$$

ただし，$v^s_{|V^s|+1} = v^s_0$ とする．

Vol.2011-DPS-147 No.24
Vol.2011-MBL-58 No.24
2011/6/3

5.1 sticky entry

VNF-Chord においても FRT-Chord と同様，successor list および predecessor を sticky entry に指定し，削除対象にしない．ただしこのとき，successor list の定義を，第 4 章の仮定に即した形に変更する．

Successor list に対して設定される長さ c は，実マシン c 台が同時に故障しない限り到達性が維持されるということを意図している．したがって，同じマシン上にあるバーチャルノードが同時に故障するという仮定から，単純に自身から近い c 個のバーチャルノードを successor list にした場合，その中に同じマシン上のバーチャルノードが存在すると，c 台未満の実マシンの故障によって到達性が損なわれる可能性がある．

そこで，ノード s の successor list を，$c'(>0)$ を用いて $\{e_i^s\}_{i=1,\ldots,c'}$ の部分集合とし，c' を，$\{e_i^s\}_{i=1,\ldots,c'}$ に s の起動しているマシンを除く，互いに異なる c 台のマシンに所属するバーチャルノードを含む最小の値とする．$\{e_i^s\}_{i=1,\ldots,c'}$ に同じマシン上のバーチャルノードが複数含まれる場合は，より自ノードから近いバーチャルノード 1 つのみを successor list の要素とし，経路表エントリを節約する．

またノード s は V^s を sticky entry に指定する．これにより，自マシン上の他のバーチャルノードを経路表に保持するようになる．こうすることで，いずれ自マシン上の前バーチャルノードが学習され，経路表に V^s が存在することになるが，あくまで同一マシン上のノードであるため，参加先のノードを自マシン上のバーチャルノードとするほか，同一マシン上の全バーチャルノードのエントリ情報を参加時に与えることが望ましい．また本稿では，マシン上のバーチャルノードの数は動的に変化することはなく，起動時に自マシン上の全バーチャルマシンを起動パラメータとして与えられているものと仮定する．

5.2 到達性保証操作

到達性保証操作は，FRT-Chord と同様に stabilize を行う．Successor list の再定義により，到達性はバーチャルノードを利用しない場合と同程度保証されると考えられる．

5.3 エントリ学習操作

あるノード s は，sticky entry を除く任意の経路表エントリ e が $e \in D^s$ を満たすように経路表を構築する．つまり，学習したエントリ e を経路表に追加した結果に sticky entry にならず，かつ $e \notin D^s$ を満たすときは，経路表に追加しない．

学習方法は FRT-Chord と同様，その手段を問わないが，能動学習探索を利用する場合，経路表に追加しないエントリを学習することを避けるため，探索先 ID の生成方法を変更する．ノード s における能動学習探索の探索先 ID は，0 以上 1 未満の乱数 rnd を用いて，$s+\mathrm{d}(s,e_{c'}^s)(\mathrm{d}(s,v_1^s)/\mathrm{d}(s,e_{c'}^s)^{\mathrm{rnd}}$ によって生成される．ただし，$e_{c'}^s$ は，ノード s の successor list の末尾エントリである．

5.4 経路表サイズ

エントリ学習操作により，経路表サイズを経路表エントリ数が上回ろうとするとき，エントリ厳選操作が行われる．

VNF は第 4 章の仮定から，経路表サイズ L をマシンに対して与えられるべきパラメータと位置づけ，経路表サイズ L を同一マシン上のバーチャルノード間で共有する．こうすることで，バーチャルノードごとのマシン内担当領域の違いを吸収することが出来る．このとき，第 4 章で述べたように，バーチャルノードという手法の単純さを保つため，経路表サイズの分担の正確性より，経路表の中身を交換しあうなどの新しい通信を追加で定義しないことを重要視する．

VNF-Chord では，マシン上の全バーチャルノードの持つ経路表を一つの経路表と見なした場合の最良経路表を求め，そのときの各バーチャルノードの経路表エントリ数をそのバーチャルノードの経路表サイズとする．

以下，バーチャルノード s の経路表サイズ L_0 を決定する．まず，次のように変数を定義する．

- 各バーチャルノード v_i^s の successor list の末尾を $e_{c'_i}^{v_i^s}$ とする．
- L_i を v_i^s の経路表サイズとする．つまり，$L_i = |E^{v_i^s}|$.
- $E_{\mathrm{far}}^{v_i^s}$ を，経路表 $E^{v_i^s}$ のうちの，同一マシン上のバーチャルノードおよび predecessor を表す集合とする．このとき，predecessor が同一マシン上のバーチャルノードである可能性に注意する．
- $E_{\mathrm{near}}^{v_i^s} = E^{v_i^s} \setminus E_{\mathrm{far}}^{v_i^s}$
- $L'_i = |E_{\mathrm{near}}^{v_i^s}|$

すると，$c'_i < L'_i$ のとき，経路表エントリ $e_j^{v_i^s}(j = c'_i, \ldots, L'_i)$ へメッセージをフォワードするときの残り距離短縮倍率の最悪値 $\mathrm{r}_j^{v_i^s}$ は，FRT-Chord 同様，以下のように定義できる．

$$\mathrm{r}_j^{v_i^s} = \frac{\mathrm{d}(e_j^{v_i^s}, e_{j+1}^{v_i^s})}{\mathrm{d}(v_i^s, e_{j+1}^{v_i^s})}, \qquad (j = c'_i, \ldots, L'_i) \tag{7}$$

したがって，定義 3.2 から次が成立する．

Vol.2011-DPS-147 No.24
Vol.2011-MBL-58 No.24
2011/6/3

$$\prod_{j=c'_i}^{L'_i}(1-\mathrm{r}_j^{v_i^s}) = \frac{\mathrm{d}(v_i^s, e_{c'_i}^{v_i^s})}{\mathrm{d}(v_i^s, e_{L'_i+1}^{v_i^s})} \tag{8}$$

$$\prod_{i=0}^{|V^s|}\prod_{j=c'_i}^{L'_i}(1-\mathrm{r}_j^{v_i^s}) = \prod_{i=0}^{|V^s|}\frac{\mathrm{d}(v_i^s, e_{c'_i}^{v_i^s})}{\mathrm{d}(v_i^s, e_{L'_i+1}^{v_i^s})} \tag{9}$$

したがって，$\max_{i,j}\{\mathrm{r}_j^{v_i^s}\}$ が最小となるのは各項 $(1-\mathrm{r}_j^{v_i^s})$ が等しくなるときであるから，その最小値は次のようになる．

$$\mathrm{r}_j^{v_i^s} = 1 - \left\{\prod_{i=0}^{|V^s|}\frac{\mathrm{d}(v_i^s, e_{c'_i}^{v_i^s})}{\mathrm{d}(v_i^s, e_{L'_i+1}^{v_i^s})}\right\}^{\left(\sum_{i=0}^{|V^s|}(L'_i-c'_i+1)\right)^{-1}} \tag{10}$$

またこのとき，式 8 の各項も等しいので，式 11 が成立する．

$$\mathrm{r}_j^{v_i^s} = 1 - \left\{\frac{\mathrm{d}(v_i^s, e_{c'_i}^{v_i^s})}{\mathrm{d}(v_i^s, e_{L'_i+1}^{v_i^s})}\right\}^{(L'_i-c'_i+1)^{-1}} \tag{11}$$

ここで，マシン全体の経路表サイズ L は，マシン外への経路表エントリの上限に相当すると考え，次の数式が成立すると考える．

$$L = \sum_{i=0}^{|V^s|}(|E \setminus V^s|) \tag{12}$$

したがって，式 10 より，最良経路表におけるノード s の経路表サイズ L_0 は，式 13 のようになる．

$$L_0 = |E^s_{\mathrm{far}}| + c'_0 - 1 + \left(L - \sum_{i=0}^{|V^s|}|E_{\mathrm{far}}^{v_i^s}| + (|V^s|+1)^2 - \sum_{i=0}^{|V^s|}c'_i\right)\left\{\frac{\ln\frac{\mathrm{d}(s,v_1^s)}{\mathrm{d}(s,e^s_{c'_0})}}{\sum_{i=0}^{|V^s|}\ln\frac{\mathrm{d}(v_i^s,e_{L'_i+1}^{v_i^s})}{\mathrm{d}(v_i^s,e_{c'_i}^{v_i^s})}}\right\} \tag{13}$$

このとき，他のノードの経路表を知る必要がある項があるが，次のように近似することで L_0 の近似値 $\tilde{L}_0$ を s の経路表だけから算出することが出来る．

$$c'_i \sim c'_0 \tag{14}$$

$$\mathrm{d}(v_i^s, e_{c'_i}^{v_i^s}) \sim \mathrm{d}(s, e^s_{c'_0}) \tag{15}$$

また，predecessor が s と同じマシン上に存在する可能性は非常に小さいと考え，次のように扱う．

$$|E_{\mathrm{far}}^{v_i^s}| = |V^s|, \qquad (i = 1, \ldots, |V^s|) \tag{16}$$

結果，近似値 $\tilde{L}_0$ は，$e_{L'_i+1}^{v_i^s} = v_{i+1}^s$ に気を付ければ最終的に次のようになる．

$$\tilde{L}_0 = |E^s_{\mathrm{far}}| + c'_0 - 1 + \left(L - |E^s_{\mathrm{far}}| + 2|V^s| + 1 - (|V^s|+1)c'_0\right)\left\{\frac{\ln\frac{\mathrm{d}(s,v_1^s)}{\mathrm{d}(s,e^s_{c'_0})}}{\sum_{i=0}^{|V^s|}\ln\frac{\mathrm{d}(v_i^s,v_{i+1}^s)}{\mathrm{d}(s,e^s_{c'_0})}}\right\} \tag{17}$$

また，successor list の末尾 $e^s_{c'_0}$ が v_1^s 以降に存在する場合，D^s に存在するノードはすべて successor list に含まれるノードが存在するマシン上にあるため，これ以上経路表にエントリを追加する必要がない．そこで，経路表サイズを例外的に小さくし，現在の sticky entry の数を経路表サイズとする．

5.5 エントリ厳選操作

エントリ厳選操作は，次の手順で行われる．

(1) $C \leftarrow E$

(2) C から sticky entry である successor list，predecessor，同一マシン上のバーチャルノードを取り除く．

(3) C から，$S_{i-1}^E + S_i^E$ が最小になる経路表エントリを削除する．

これは，FRT-Chord 全く同様であり，エントリ学習操作の変更のみで経路表を変更していることを意味する．

5.6 フォワーディングアルゴリズム

VNF-Chord は，メッセージを次のようにフォワードする．

$$E^s.\mathrm{forward}(t) = \begin{cases} e \in E^s\ (\forall f \in E^s, \mathrm{d}(e,t) \le \mathrm{d}(f,t)), & (\mathrm{d}(s,t) \le \mathrm{d}(v_1^s,t)) \\ v \in V^s\ (\forall w \in V^s, \mathrm{d}(v,t) \le \mathrm{d}(w,t)), & (otherwise) \end{cases} \tag{18}$$

Vol.2011-DPS-147 No.24
Vol.2011-MBL-58 No.24
2011/6/3

マシン上の各バーチャルノード間に十分ノードが存在するときは，単なる greedy routing となる．しかし，successor list が同一マシン上のバーチャルノードを越える場合のみ，例外的なフォワードを行う．

6. 評　　価

6.1 利　　点

第 5 章で説明した VNF-Chord には，以下のような利点がある．

- Successor list に，同一マシン上のバーチャルノードが複数存在した場合に耐障害性が低下することを防ぐ．
- 同一マシン上のバーチャルノード間の経路表エントリの重複を避け，マシン単位で効率よく経路表エントリを保持できる．また，このとき，各バーチャルノード間で経路表の内容に関する通信を行わないため，バーチャルノードを利用することによるプログラムの複雑性の増加を避けることが出来る．
- 経路表サイズをバーチャルノード単位ではなく，マシン単位で指定し，大きいサイズの経路表が必要なバーチャルノードに対して大きい経路表サイズを割り当てることが出来る．
- マシン内からマシン外へ $t(\in D_i^s)$ 宛のメッセージをフォワードするとき，必ずそのメッセージは v_i^s を経由するため，v_i^s が $t(\in D_i^s)$ に対するキャッシュ管理を担当することで，各バーチャルノードは自然とキャッシュがマシン内に存在するかどうかを調べてからマシン外と通信することが可能となる．また，s の起動しているマシン全体のキャッシュサイズを C^s とするとき，v_i^s のキャッシュサイズ C_i^s は，D_i^s に含まれる ID の数 $\mathrm{d}(v_i^s, v_{i+1}^s)$ に比例するように $C_i^s = C^s(\mathrm{d}(v_i^s, v_{i+1}^s)/2^m)$ と設定することで，マシン全体のキャッシュサイズを，メッセージをマシン外へフォワードする割合を反映して各バーチャルノードに割り当てることが出来る．また，このキャッシュサイズに関しても，各バーチャルノードが調整するための通信を行う必要がない．
- 自マシン上のバーチャルノードの数は自身の経路表から判断でき，他のマシン上のバーチャルノードの数に依存せず，マシンごとに異なるバーチャルノードを自由な数だけ起動することが出来る．

6.2 経路長の評価

我々は VNF-Chord を Overlay Weaver[6),7)] 上に実装し，計算機 1 台上でエミュレーション機能を用いて実験を行い評価した．

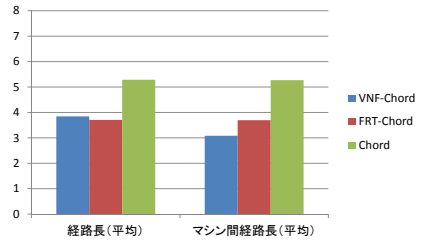

図 1　平均経路長と平均マシン間経路長 ($M = 250$)
Fig. 1　Average path length and average path length over machines($M = 250$).

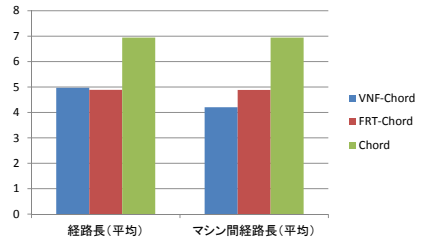

図 2　平均経路長と平均マシン間経路長 ($M = 2500$)
Fig. 2　Average path length and average path length over machines($M = 2500$).

本手法は，マシン間を横断する経路長の短縮が目的ではない．しかし，本手法を利用することでマシン間の経路長を短縮できることが予想される．そこで，その短縮を確認することが本実験の目的である．

実験では，マシンあたりのバーチャルノードの数を 4 ノードとし，実マシンの数 M を 250 台および 2500 台とした．各バーチャルノードが 50 回の能動学習探索を実行した後，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 10^4 回の探索を行い，1 回の探索における経路長および，マシンをまたぐフォワーディングの回数（マシン間経路長と呼ぶ）を計測した．また，10^4 回の探索が終了した時点における各バーチャルノードの経路表に含まれるエントリ数を測定した．

マシンあたりの経路表サイズ L を 160 として実行したとき，実マシンの数が 250 台の場合と 2500 台の場合とで，バーチャルノードあたりの経路表サイズの平均がそれぞれ約 43.7 と約 46.1 となった．そこで，比較対象として，FRT-Chord のバーチャルノードあたりの経路表サイズを実マシンが 250 台の場合に 44 とし，実マシンが 2500 台の場合に 47 に設定して同様の実験を行った．また，Chord において経路表を自由に設定することは出来ないが，参考とするため，同様の実験を行った．このとき，実マシンが 250 台の場合には経路表サイズが約 19.3，実マシンが 2500 台の場合には経路表サイズが約 22.6 となった．

図 1 および図 2 は，それぞれ実マシンが 250 台および 2500 台の場合の実験結果である．

この実験結果において，VNF-Chord は FRT-Chord と比較して経路長が長くなっているが，その反面，マシン間経路長は短縮されている．これは，同一マシン上のバーチャルノード間のフォワーディングがマシンをまたぐフォワーディングと比較して無視できるほど小さいということを仮定した場合，VNF-Chord のほうがより低コストに担当ノードまでフォ

Vol.2011-DPS-147 No.24
Vol.2011-MBL-58 No.24
2011/6/3

ワーディングを行えることを意味している．

この短縮は，フォワード先が同一マシン上のバーチャルノードであることによるものであり，目的 ID に近づくほどそのようなフォワードは発生しにくくなる．実際，この効果が発生するのは主に経路における最初のフォワード時である．したがって，この短縮効果は限定的なものであるが，DHT が本来から備える自律的な動作を維持しているため，その他の手法を適用しやすい形を保っていると考えられ他の手法との組合せが期待される．

7. まとめと今後の課題

本稿では，DHT におけるバーチャルノードの自律的な動作を維持しつつ，マシン単位の効率を向上する手法であるバーチャルノード融合（VNF）および，それを利用した具体的な DHT アルゴリズム VNF-Chord を提案した．VNF では，同一マシン上の各バーチャルノードが経路表の管理やキャッシュの管理，各 ID へのフォワードなどの役割を分担することで，バーチャルノードを効率よく利用する．また，VNF-Chord は，柔軟な経路表（FRT）を利用して構成した VNF ベースのアルゴリズムであり，FRT の持つ経路表サイズを自由に設定する機能や，経路表に含まれるノード ID に限定を加えた状態であってもノード ID を優れた状態へ改良する機能，特定の経路表エントリを経路表に保持し続ける機能などがアルゴリズムの構成に有効である．

VNF-Chord の実装および実験を行った結果，マシン間をまたぐフォワード回数を減らす効果が確認された．

今後，VNF-Chord におけるキャッシュの効果の測定や，より詳細な評価を行う予定である．また，本稿で仮定したマシン上のバーチャルノード数が変化せず，起動時に同一マシン上のバーチャルノードを知っているという仮定を取り除いた場合に，どのように自マシン上のバーチャルノードを扱うべきかを検討する予定である．

謝辞　本研究は科研費（22680005）の助成を受けたものである．

参 考 文 献

1) Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, *Proc. ACM SIGCOMM*, pp.149–160 (2001).
2) Maymounkov, P. and Mazières, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric, *Proc. IPTPS '02*, pp.53–65 (2002).
3) Zhao, B.Y., Kubiatowicz, J.D. and Joseph, A.D.: Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing, Technical Report UCB/CSD-01-1141, EECS Department, University of California, Berkeley (2001).
4) 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).
5) 長尾洋也，首藤一幸：柔軟な経路表：経路表空間上の順序関係を利用したオーバレイネットワークルーティング方式，Proc. 先進的計算基盤システムシンポジウム (SACSIS2011)（採録決定）(2011).
6) Shudo, K., Tanaka, Y. and Sekiguchi, S.: Overlay Weaver: An overlay construction toolkit, *Computer Communications*, Vol.31, No.2, pp.402–412 (2008).
7) Shudo, K.: Overlay Weaver: An Overlay Construction Toolkit, `http://overlayweaver.sourceforge.net/`.