社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－DPS－129（2）
2006／11／16

constant-degree DHT におけるホップ数削減方式の提案と評価

加藤大志　神谷俊之
ＮＥＣ インターネットシステム研究所

P2P 技術において注目されている DHT の一方式として、固定サイズのルーティングテーブルでネットワークサイズの対数オーダのホップ数を達成する constant-degree DHT がある。本稿では、 constant-degree DHT のアルゴリズムの 1 つで de Bruijn グラフをベースにした Broose について、ホップ数を削減する改良方式を提案する。改良方式は、ルーティングテーブルの構成を変えずに可能である場合はルーティングをショートカットすることによりホップ数の平均値を下げるものであり、シミュレーション実験により有効性を確認した。

An Improvement for a constant-degree DHT and its evaluation

Daishi KATO　Toshiyuki KAMIYA
Internet Systems Research Laboratories, NEC Corporation

A constant-degree DHT is a type of a distributed hash table that has a constant-sized routing table, and achieve a logarithmetic routing hops. Among constant-degree DHTs, we focus on a de-Bruijn based DHT “Broose” and propose an improvement to reduce hop count. Our approach is to shortcut routing hops whenever possible. The simulation evaluation showed the effectivity of our approach.

1　はじめに

Peer-to-Peer ネットワークの分野では、近年 DHT (Distributed Hash Table) 技術の研究が盛んである。 DHT は、中央サーバを使わずに、ネットワークに参加するノードが協力してハッシュテーブルライクなデータベースを提供する技術である。 DHT では、ネットワークに参加するノードの上下関係がなく、ノードの参加や離脱も自由である「均一性」、大規模ネットワークでもネットワーク維持コストが低いという「効率性」の 2 つの特徴がある。

これらの特徴から、 DHT を利用すると高性能なサーバを用いずに大規模なネットワークを構築できるため、その利用価値は高い。例えば、 DNS のようにホスト発見のための利用や、 IM (Instant Messaging) や VoIP (Voice over IP) におけるユーザ発見のための利用が考えられる。また、分散ファイル共有システムにおけるファイルのロケーション発見には既に利用されている [5]。

DHT のアルゴリズムは、初期に提案された logarithmetic-degree DHT と後に提案された constant-degree DHT の 2 つの種類がある。両者の違いはルーティングテーブルの大きさに関する。 logarithmetic-degree DHT は、ネットワークサイズが N であるとき、各ノードのルーティングテーブルのサイズを $O(\log N)$ にすることで、任意のノード間のルーティングホップ数を $O(\log N)$ にするものである。一方で、 constant-degree DHT は、任意のノード間のルーティングホップ数を $O(\log N)$ にするために各ノードのルーティングテーブルのサイズが $O(1)$ で済むものである。このことから、

constant-degree DHT ではルーティングテーブルの新鮮さを保つコストが下がることが期待される。

constant-degree DHT は、グラフ理論の degree-diameter の最適性、すなわち、ある次数 (degree) とある直径 (diameter) でどれだけ多くのノードを接続できるか、を基礎に開発されたアルゴリズムである。過去に提案された constant-degree DHT で著名なものに、Viceroy [8]、Koorde [6]、CAN-D2B [1]、Broose [2] がある。これらのうち、Viceroy はバタフライトポロジを応用したもので、それ以外は de Bruijn グラフを基本にしている。我々は、Kademlia [9] の特徴を取り入れて実用性を向上させた Broose に注目し、平均ホップ数を削減する改良方式を提案する。提案方式の有効性はシミュレーション実験により確認する。

本稿の構成を以下に示す。続く 2節で Broose について説明する。次に 3節で Broose の問題点と改良案を説明し、4節で改良方式の評価を行う。最後に 5節にて本稿をまとめる。

2 Broose

Broose は de Bruijn グラフの最適性と Kademlia の実用性を兼ね備えた DHT アルゴリズムである。de Bruijn グラフを用いると、固定サイズのルーティングテーブルでネットワークサイズの対数オーダのホップ数で任意のノード間をルーティングできる。Kademlia は、ルーティングテーブルに「バケット」という概念を導入し、冗長性・柔軟性と、簡単に実装できるという特徴を実現した。Kademlia は、既にいくつかのプロジェクトで採用されている [4, 3, 5]。以下では、まず de Bruijn グラフについて説明し、次に Broose のルーティング方式を述べる。

2.1 de Bruijn グラフ

de Bruijn グラフは、各ノードから出るアーク (リンク) の数が m で、グラフの直径 (任意の 2 つのノード間の最短経路の長さの最大値) が n である場合に、総ノード数が m^n になるグラフである。このグラフでは、例えば $m = 2$ のとき、総ノード数を 2 倍にするために、グラフの直径を 1 増加するだけで済む。この特性を DHT に使うことができれば、ルーティングテーブルのサイズを固定にしつつ、ホップ数を少なくすることができる。

簡単な de Bruijn グラフを説明するために、$m = 2, n = 4$ の場合、すなわち 4-bit の ID 空間を考える。あるノードの ID を $u_1u_2u_3u_4$ とすると、そのノードから出る外向きリンクは、$0u_1u_2u_3$ と $1u_1u_2u_3$ の二つである。このリンクを使ってあるノード ID $v_1v_2v_3v_4$ から別のあるノード ID $w_1w_2w_3w_4$ にたどり着くには次のような経路を通ればよい: $v_1v_2v_3v_4 \rightarrow w_4v_1v_2v_3 \rightarrow w_3w_4v_1v_2 \rightarrow w_2w_3w_4v_1 \rightarrow w_1w_2w_3w_4$. この時のホップ数は ID 空間のビット数と同じ 4 である。ID 空間が n-bit の場合も、2 つのリンクで任意のノード ID から任意のノード ID へ n ホップでたどり着く経路がある。

2.2 Broose ルーティング

一般的に DHT では consitent hasing を実現するため、ノード ID の空間はノード数に比べると極端に大きくなっている。この空間で、ノード ID 間の距離や、ルーティング方式をどう定義するかが DHT アルゴリズムのポイントとなる。

Broose では、ノード ID の距離は Kademlia と同じ XOR 距離を使い、ルーティング方式としては de Bruijn グラフベースのものを使う。ところが、通常の de Bruijn グラフはスパースな ID 空間では作れない。そこで、ノード ID が空間上で一様に分散していることを仮定して、ノード ID の上位何ビットかを使って de Bruijn グラフを作る。すなわち、上位 d ビットだけを見れば、通常の de Bruijn グラフに見えるような、d を使うことになる。N を総ノード数とすると、d の理想値は $log(N)$ であるが、実際は、ノード ID が一様に分布することは期待できないので、d は少なめに見積り、上位 d ビットの prefix を共有するノードが複数存在するよ

うにする。ルーティングは 2 段階に行われ、最初に上位 d ビット分を、 de Bruijn グラフ型のルーティング (right shifting lookup) を行い、その後、その上位 d ビットを共有するノード内で、 one-hop ルーティング (brother lookup) を行う。 one-hop ルーティングを行うためには、完全グラフになっている必要がある。以下では我々は、 right shifting lookup に必要なルーティングテーブルの情報を *cousins* と呼び、 brother lookup に必要なルーティングテーブルの情報を *brothers* と呼ぶ。

brothers は単一のバケットで自ノードの ID (*mypeerid*) に XOR 距離が近いノードを保持する。 *brothers* バケットは、比較的多くのノードを保持する。 *cousins* は、 de Bruijn リンクのためのバケットで、 2 つある。 *mypeerid* を右に 1-bit シフトして、 MSB を 0 と 1 にした ID (cousinid0, cousinid1) にそれぞれ近いノードを保持する。 *cousins* バケットのサイズは、すべてのノードが短時間に離脱する確率が低くなる程度で、比較的小さい。最後に、 d を見積る方式を説明する。 Broose では、 *cousins* バケット中の任意のノード間で最大の prefix 長を見つけ、それに 1 を加えたものを d とする。

以上が、 Broose のルーティング方式の基本である。 Broose では、 *cousins* の逆リンクを使うルーティング (left shifting lookup) や proximity route selection や hotspot balancing も提案されているが、本稿では対象としない。

3 改良方式

本節では、 Broose のルーティング方式の問題点を指摘し、我々の改良方式を提案する。

3.1 Broose の問題点

Broose では、 right shifting lookup において $O(\log N)$ 回のホップを行い、 brother routing において $O(1)$ 回のホップを行う。 right shifting lookup におけるホップは de Bruijn グラフのリンクを辿るものであり、実際には、 $\log_2 \frac{N}{k'}$ となる。ただし、 k' は *cousins* バケットの大きさを

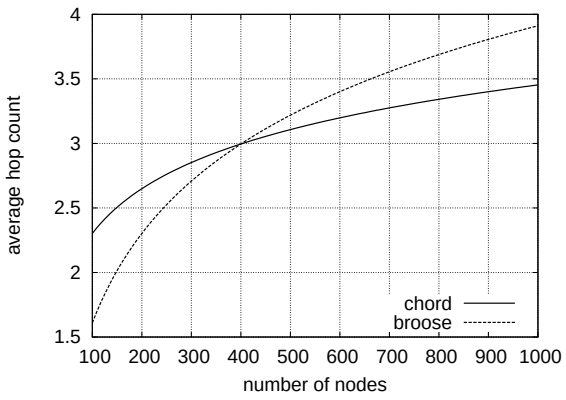

図 1: Chord と Broose のホップ数比較

示す。ここで重要なことは、このホップ数が任意のノード間で常に同じになるため、ホップ数の平均値も変わらず $\log_2 \frac{N}{k'}$ となることである。 brother routing におけるホップは *brothers* が十分に大きいことを仮定すれば 1 回で済むため、以下では right shifting lookup におけるホップに関してのみ注目する。

ところで、 logarithmetic-degree DHT の一例である Chord [11] では、ルーティングのホップ数のオーダは $O(\log N)$ で同じものの、ホップ数の分布は 1 から $\log_2 N$ まで広がりがあり、実際のホップ数の平均値は [11] での実験によると $\frac{1}{2}\log_2 N$ となっている。 Broose における k' を [2] で使われている $k' = 20$ とすると、ホップ数の平均値は $\log_2 \frac{N}{20}$ となる。 N を変数として、これらのホップ数の平均値を図示すると、図 1のようになる。この図から分かる通り、$N = 400$ の点で Chord と Broose のホップ数は逆転し、 Broose の方が大きくなってしまうという問題がある。

これを解決する一つの方法は、 Broose の right shifting lookup において一度にシフトするビット数 b を大きくすることである。例えば $b = 2$ にすれば、ホップ数の平均値は $\log_{2^2} \frac{N}{k'} = \frac{1}{2}\log_2 \frac{N}{k'}$ となり、常に Chord のそれを下回るようになる。しかしながら、この b を大きくする方法は、ルーティングテーブルも大きくなるという問題がある。そこで我々は、 b に依存しない方法でホップ数の平均値を下げる方式を提案する。

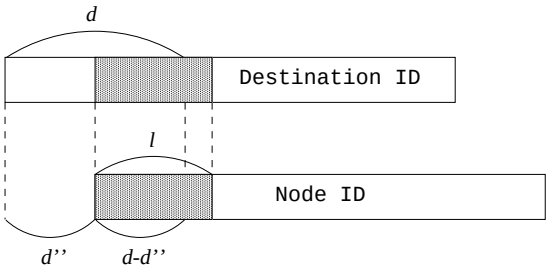

図 2: In a routing phase

3.2 改良案

我々の提案する方式は、 Broose のルーティングにおいて、可能であればショートカットしてホップ数を減らすものである。既に述べたように、 Chord ではルーティングの到達先がルーティング元から近ければホップ数が少なくなり、全体的にホップ数の分布には広がりができる。提案方式では、このホップ数の分布の広がりを Broose でも実現でき、ホップ数の平均値を下げることができるようになる。

我々は実装の容易性を考慮し、スコアによるルーティング先決定方式を提案する。以下では、スコアの計算法に関して説明する。説明のため、ある宛先 ID “Destination ID” へのルーティングの途中段階を例に考える。既に最大ホップ数は見積もられたものとし、これを d とする。また、残りのホップ数を d' とする。 Broose では、ルーティング先はホップ数を 1 つずつ減らすノードでかつ XOR 距離の小さいもの、すなわち、 Destination ID を左に $d'-1$ ビットシフトしたものと prefix 一致長が長い ID を持つノードを候補にする。提案方式では、ホップ数を 2 つ以上減らすノードの存在も想定し、次のようにスコアを計算する。

まず、あるノード (このノードの ID を “Node ID” とする) をルーティング先とした場合の残りのホップ数を d'' とする。また、宛先 ID を d'' ビット左にシフトした ID と、対象ノード ID の prefix 長を l とする (図 2)。このノードが適切なルーティング先となるためには、 l は $d-d''$ 以上でなければならない。もしこれが満たされるのであれば、 d'' が小さいほどルーティング先として適切であり、 d'' が同じ場合は、 Broose の場合と同様に l が大きいほどルーティング先として適切である。よって、残りのホップ数が d'' であるときのこのノードのスコア s は次のように計算できる。 (ただし、 m は ID のビット数とする)

$$s(d'') = \begin{cases} m(d-d'')+l & \text{if } l \geq d-d'', \\ -1 & \text{otherwise.} \end{cases} \quad (1)$$

このノードの最終的なスコアは、 d'' を 0 から $d'-1$ まで変化させたときの最大値である。よって、そのスコア *score* は次のようになる。

$$score = \max_{d''=0\ldots d'-1} s(d'') \quad (2)$$

提案方式では、ルーティングテーブルのすべてのノードに対してスコア *score* を計算して、スコアの大きいもの (1 つまたは複数) をルーティング先として決定する。

この提案方式を使うと、各ノードのルーティングテーブルを可変にできるという特徴もある。すなわち、 b をノード毎に個別に設定できることになる。 $b=2$ であれば、そのノードでは 2 ビットシフトしたルーティング先もルーティングテーブルに持っていることになり、ホップ数の削減が期待できる。また、各ノードで b を個別に設定した場合のルーティングでは、ノードの「強さ」を z としてスコア計算を以下のように変更すると、「強い」ノードを優先的に中継ノードとして使うことになり、ノードの「能力」に応じた不均一なネットワーク構成の実現が期待できる。

$$score = \max_{d''=0\ldots d'-1} s(d'') + z \quad (3)$$

ノードの「強さ」を使ったルーティングに関する詳細は本稿では扱わない。

4 評価

本節では、我々の改良方式を Broose の元の方式と比較評価する。

総ノード数	100 〜 1700
平均参加時間	60 分
平均離脱時間	20 分
平均 lookup 間隔	20 秒
ネットワーク遅延	100ms
試験時間	10 分

表 1: シミュレーションパラメータ

4.1 実験方法

2 つの方式を評価するために、シミュレーション実験を行った。シミュレーションはイベント駆動のパケットレベルネットワークシミュレータである Bamboo [10] simulator を独自に改良したものを用いた。独自に改良した内容は、 churn(ノードの参加離脱) イベントの発生と lookup イベントの発生である。このシミュレータで動作するように 2 つの方式を実装した。この実装は大部分は共通になっており、 2 つの方式の差はスコア計算に関する部分のみである。

シミュレーションに用いたパラメータを表 1に示す。 churn イベントや lookup イベントは、 p2psim [7] と同様に、指数分布に従うモデルとした。したがって、平均起動ノード数は総ノード数の 3/4 程度になる。測定データは、 lookup の成功率と、 lookup の応答時間、ネットワーク通信量である。総ノード数毎に 10 回の実験を繰り返し、データの平均値をとった。

4.2 実験結果

本実験において、 lookup の成功率の結果に関しては、 Broose の元の方式と我々の改良方式とで、明確な差はなかった。よって、残りの測定データである、ネットワーク通信量と lookup の応答時間に関して説明する。

まず、図 3にネットワーク通信量の結果を示す。横軸は総ノード数であり、ネットワークの大きさに相当する。縦軸がノードあたりの通信帯域の平均値であり、通信量とみなす。 Broose の元の方式 (original) と我々の改良方式

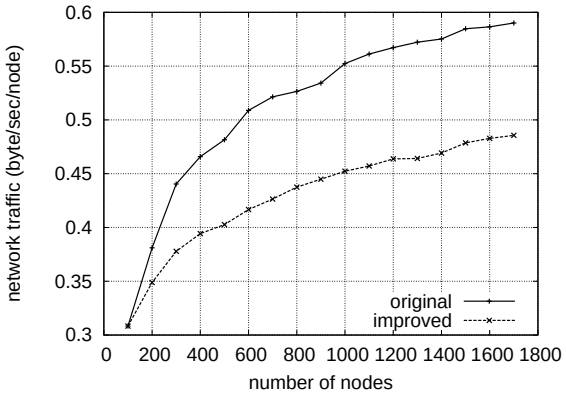

図 3: 実験結果: ネットワーク通信量

(improved) を比較すると、常に改良方式の方が通信量が少ないことが分かる。ただし、総ノード数が 100 のときは例外で、同じ値になっている。これは、 *brothers* の大きさが 100 になっているためであり、すなわち、総ノード数が 100 のときは、すべてのノードを *brothers* に保持するため right shifting lookup が起こらないからである。よって、改良方式は right shifting lookup のホップ数を削減することににのみ寄与していることが確認できた。総ノード数が 1000 のプロットに注目すると、改良方式は通信量を 18% 削減したことになる。

次に、図 4に lookup の応答時間結果を示す。横軸は総ノード数であり、縦軸が応答時間の平均値である。図 3の通信量と同様に、改良方式は常に Broose の元の方式より応答時間が短いことが分かる。総ノード数が 1000 のプロットでは、 34% の削減になっている。

4.3 考察

以上の実験により、我々の改良方式は Broose の元の方式と比較して通信量と応答時間をともに削減することができその有効性を確認した。両方式においては、ルーティングテーブルの構成法は変わらないため、両方式の差はホップ数の差だけである考えられる。ホップ数が削減されることで、応答時間が短縮し、かつ、通信量が減少した。通信量の削減割合が応答時間の削減割合より小さかった理由は、応答時間が

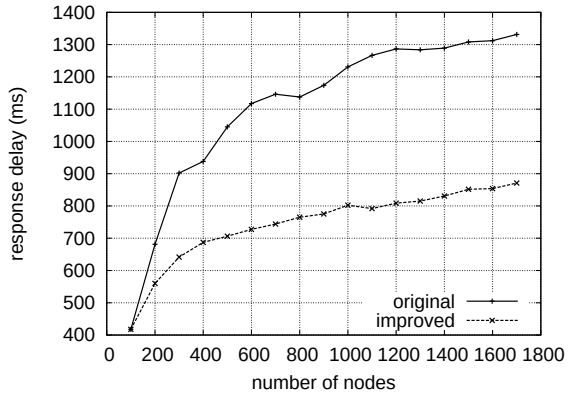

図 4: 実験結果: lookup 応答時間

lookup 時のみの測定データであり、ホップ数の削減の影響を完全に受けるのに対して、通信量は lookup 以外の定常的に起こる通信も含むため、ホップ数の削減の影響が部分的であるためであると考えられる。

本実験の別の側面として重要な点を述べる。この実験は、DHT アルゴリズムにおける通信量や応答時間などのコストをオーダで議論しているだけでは、実際的な差が見えないことを明らかにした。今回の例では、スコア計算式のみが違う 2 つの方式が、アルゴリズムのオーダレベルでの機能が同じであるにも関わらず、シミュレーションレベルでは 20% 近くの差となった。場合によっては、アルゴリズムのオーダレベルで劣性な方式でも限られた条件でのシミュレーションレベルの評価では優性になる可能性もある。今後 DHT アルゴリズムを改良を進める上ではオーダレベルの議論だけではなく、シミュレーションレベルでの議論により差を明らかにする方法をとるべきだと我々は考えている。

5 おわりに

本稿では、constant-degree DHT の一つである Broose について、ホップ数を削減する改良方式を提案した。改良方式はスコア計算によって実現したため、その実装が容易であるとともに拡張性があるという利点を持つ。この改良方式と元の方式をシミュレーション実験によって比較評価し、改良方式の有効性を確認した。さらに本実験を通して、アルゴリズムのオーダだけの議論ではなく実際の数値をベースに性能評価することが重要であることも確認した。

参考文献

[1] Pierre Fraigniaud and Philippe Gauron. The content-addressable network d2b. Technical Report 1349, CNRS Universie de Paris Sud, January 2003.

[2] A. Gai and L. Viennot. Broose: a practical distributed hashtable based on the de-brujin topology. In *Proceedings of the Fourth IEEE International Conference on Peer-to-Peer computing (IEEE P2P 04)*, 2004.

[3] http://azureus.sourceforge.net/.

[4] http://www.bittorrent.org/.

[5] http://www.emule-project.net/.

[6] Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, 2003.

[7] Jinyang Li, Jeremy Stribling, Thomer M. Gil, Robert Morris, and M. Frans Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS04)*, San Diego, CA, February 2004.

[8] D. MALKHI, M. NAOR, and D. RATAJCZAK. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st annual ACM symposium on Principles of distributed computing*. ACM Press, 2002.

[9] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of IPTPS02, Cambridge, USA*, March 2002.

[10] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling Churn in a DHT. In *Proceedings of the 2004 USENIX Technical Conference, Boston, MA, USA*, June 2004.

[11] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pp. 149–160, 2001.