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

2006－DPS－126（37）
2006－CSEC－32（37）
2006／3／17

ユビキタスネットワークにおける分散ハッシュテーブルの構築と評価

徳浜元弘　　中沢実　　服部進実
金沢工業大学大学院工学研究科知的創造システム専攻

〒105-0002 東京都港区愛宕 1-3-4 愛宕東洋ビル 12F
E-mail:tokuhama@acr.kanazawa-it.ac.jp,{nakazawa,hattori}@infor.kanazawa-it.ac.jp

あらまし

本論文はユビキタスネットワークで想定されるノードの参加・離脱に対して分散ハッシュテーブルを用いた P2P システムの適用性を論じたものである．分散ハッシュテーブルはインデックス情報をノードに分散させて保持する方式で種々の方式が提案されている．今回は経路表形式と木構造形式の P2P システムに注目して，その構成と実装法について考察した．さらに，コンピュータシミュレーションを通してこれらのシステムの定常時とノードの離脱・参加時におけるノード検索の性能を計測し比較分析を行った．その結果，木構造形式のシステムがノードの参加・離脱において管理コストが少なく，ノード検索性能も低下しないことを示す．

Implementation and Its Evaluation of Distributed Hash Tables for the Ubiquitous Network.

Motohiro TOKUHAMA, Minoru NAKAZAWA, and Shimmi HATTORI
Graduate Program in Systems for Intellectual Creation, Kanazawa Institute of Technology
Atago Toyo Bldg. 12F 1-3-4 Atago, Minato-ku, Tokyo, 105-0002 Japan
E-mail: tokuhama@acr.kanazawa-it.ac.jp, {nakazawa,hattori}@infor.kanazawa-it.ac.jp

Abstract

This paper describes the applicability of the P2P system which uses the distributed hash table for participation and secession of the node assumed by ubiquitous network. The distributed hash table is a method for the decentralization of information to the node. There are various methods to form the distributed hash table.

We explain the composition and implementation method of the routing table form and the tree structure form of the P2P system. We compared the performance of node retrieval when the node of the system participates, secedes, and during its regular operation by computer simulation. As a result, it turned out that the system of the tree structure form doesn't decrease the node retrieval performance.

1. はじめに

ユビキタスネットワーク環境ではシステムを構成する多数のノード（ピア）がアドホック的に接続され，ノードの移動を伴う．このようなシステムへのノードの参加と離脱が頻繁に行われる自律型システムでは管理コストやリアルタイム処理の面で P2P(Peer-to-Peer)システムが適している．

P2P システムではネットワークのノード同士が交信し，情報の共有やコラボレーション等を実現する．代表的なシステムとして Napster[1], Gnutella[2]等がある．

本論文では P2P システムを実現するモデルとして木構造型分散ハッシュテーブルを取り上げ，実験を通して他のシステムと比較し，ノードの参加と離脱への対応に優れていることを実証する事でユビキタスネットワーク環境における分散ハッシュテーブルを考察する．

2. 分散ハッシュテーブル技術

分散ハッシュテーブル(以下 DHT と略す)は一種のインデックス情報でこれを P2P システムの各ノードに保持してノード検索に活用するケースが主流となっている．

DHT ではまず，各ノードに固有のノード ID を割り当てる．このノード ID は IP アドレスの文字列に SHA1 等のハッシュ関数を適用して算出する．また，コンテンツの格納先ノードもコンテンツのメタ情報に同一のハッシュ関数を適用してハッシュ値を算出し，この値を格納するノードと関連付ける．関連付けの方法は様々な方式が提案されているが，一つの直感的な方法としては，得られたハッシュ値に近いノード ID を持つノードにコンテンツを格納することである．

これらの DHT には，主に Chord[3]などによる経路表形式のものと Kademlia[4]による木構造形式のものがあるが，それらの評価の比較や適用分野に即した手法として論じられていないのが現状である．

そこで本稿では，動的にノードがネットワークへの参加・離脱を繰り返すユビキタスネットワーク環境を想定し，この両者の手法についてシミュレーションを通して，ノードの検索性能についての評価を行っていく．

3. Chord

経路表形式の代表的な DHT を用いたシステムとして Chord がある．このシステムではノードを仮想的なリング上の空間に配置し，各ノードには経路表を準備してルーティングを行う．なお，今後の説明においてノードとメタ情報はハッシュ値で表されるものとする．また，キーはメタ情報をハッシュ化した値とする．

(1) Chord のノード空間

Chordシステムではノードはm-bitのID（識別子）を持ち，modulo 2^mの環状空間に時計回りで配置される．キーもやはりそのハッシュ値のmodulo 2^mの値をIDとし，コンテンツはキー値に等しいかより大きいノードに格納される． 図 1 にm=6 とするChordのノード空間の例を示す．図においてNはノードを，Kはキーを表す．

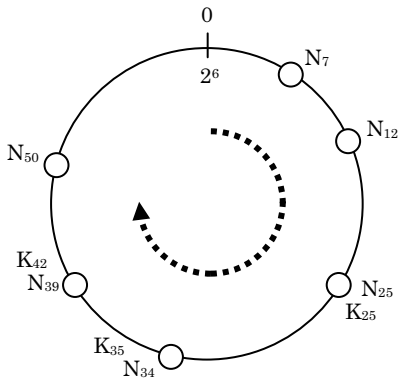

図１．Chord のノード空間の例(m=6)

(2) Chord の経路表(Finger table)

各ノードが所有する経路表はnを自身のノードIDとして，[n, $n+2^m$]の環状空間を[$n+2^k$, $n+2^{k+1}$] (k=0. . m － 1)の区間単位でm個のエントリーを構成する．各エントリーにはその区間内にある最初のノードの接続情報を格納しておく．その区間にノードが存在しない場合は次の区間上から取得される．

(3) ノード検索

経路表を利用した Chord のノード検索は次のようにして実行される．

・ 経路表より目標 ID を含む区間のエントリーに収容されているノード X を取得する
・ 目標 ID<X.ID ならノード X に対して上記の問合せを依頼する
・ 目標 ID=X.ID なら検索は成功. 目標 ID>X.ID なら失敗している．（いずれにしても検索終了）

経路表が正確に作成されていればノード検索は経路表だけで実行できる．しかし，ノードが離脱し DHT の修復がなければノード検索は失敗する．DHT の修復はコストのかかる処理であるため予め前方の数ノードの情報を格納するデータ構造を準備しておき, 経路表での検索の失敗をこのデータ構造でフォローすることが考えられている．

(4) ノードの参加と離脱

Chord ネットワークに新しくノードが参加するには, 既知ノードに新しく参加するノードの後続ノードと先行ノードを問合せ, それらにノードの新規参加を知らせる. そして参加するノードは, 経路表を構築する．

このとき, 問題はその参加が他のノードの経路表に影響を及ぼすということである. 本研究にて行った Chord のシミュレーションでも全てに近いノードで経路表を更新する必要があり, 高い負荷となった．

システムに存在するノードが離脱するときは自発的に離脱の処理を実行させることが望ましい．さもなければ Chord 環の連鎖が破綻しシステムのノード検索の性能が低下することが考えられる．

4. 木構造型 DHT モデル

Kademlia は Pastry[5]や Tapestry[6]に類似したノード探索アプローチを取るが,一貫して一つの経路選択アルゴリズムを使用するところが特徴である．

ノード ID とキーは 160 ビットのハッシュ値を使用する．これは SHA1 などのハッシュ関数の使用を前提としているためである. 本論文ではハッシュ値は一様に分布するものと仮定する. このためノード ID も一様分布の形状となる．

Kademlia のノード間の距離はノード ID 同士の排他的論理和(XOR)値で決定される．このXOR メトリックは対称性を持ち,Kademlia の参加ノードが完全に同じ分布の経路表を持つノードからのコンタクトを受け取ることを可能にする. このことは有用な経路情報を学習できることを意味している．

ノード空間はノードIDによる二分木構造になる．図 2 はKademliaのノード空間を図示した二分木で, 説明を簡略化するためにノードIDは 160 ビットではなく 4 ビットで表現している. 二分木の末端の数値はノードID=$(0101)_2$からのXOR距離を表す．点線で囲まれている部分はノードID=$(0101)_2$から$[2^i, 2^{i+1}]$の距離にあるノード群で, これらはバケットという単位で管理される．

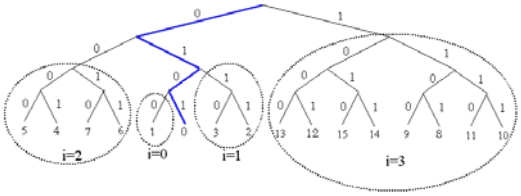

図 2.Kademlia のノード空間

4.1 DHT の構築

Kademlia の DHT は k-バケットと呼ばれる二分木構造で, その葉にノードの接続情報を収容するバケットを持つ. 各バケットには収容できるノード ID の範囲があり，エントリーの最大個数は k 個である．

k-バケットの初期状態は一つのバケットから成る．その範囲は$[0, 2^m]$であり，ノード空間全体を含む(m=160 とする). ノードにコンタクトがあると，そのコンタクトはそのノードIDを含むバケットに登録される. 初期状態ではバケットは一つしかないので, コンタクトはそのバケットに登録される．

バケットが満杯（k個）になると，その範囲が自分自身のノードを含むときにそのバケットは二分割される. そうでなければ新しいコンタクトは登録されない．図 3 にノードIDが 4 ビットで表されている場合のノードID=$(0101)_2$のバケットの分割例を示す．

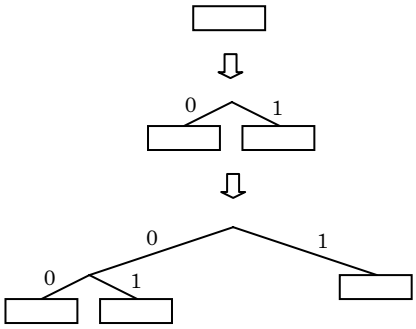

図 3. バケット分割の例

4.2 ノード検索アルゴリズム

Kademlia のノード検索は目標 ID に近い k 個のノードを返す処理である. その処理は以下のようになる.

- 目標 ID の範囲を含むバケットから k 個のノード ID を取り出す
- それらのノードに対してノード検索を再実行
- 目標 ID に等しいノード ID が返されるか, あるいはすべての ID をチェックしたら終了

ノード検索の期待値は$O(\log_2 N)$である.

4.3 ノードの参加と離脱

ノードがシステムに参加するには少なくとも一つのノード ID を知っておく必要がある. ノードの参加は次のようにして行われる.

- 参加するノード ID に近いノードを k 個得る
- それらのノードの k-バケットからノード ID を収集してプールする
- プールしたノードをバケットに登録
- 自分自身の存在を他のノードへ知らせる（バケットのリフレッシュ）

ノードの離脱に関しては特にやるべきことはない.

5. 実装

今回は Ruby 言語[7]を用いて Kademlia と Chord の実装を試みた. 以下にそれらの概要を述べる.

(1) Kademlia

シミュレーション実験用とネットワーク対応の二種類作成した. シミュレーション実験用では定常時とノードの参加･離脱時のノード検索の効率を測定した.

ネットワーク対応版は分散 Ruby を使用してネットワーク越しでノードを分散オブジェクト化し, ローカルネットワーク内でノード検索やコンテンツのダウンロード, アップロードが機能することを確認した.

(2) Chord

Chord はノード検索のシミュレーション用のみを作成した. 但し, Kademlia と同等の評価を行なうため今回はノード離脱時の DHT の修復処理は省略することとした. Chord における DHT は参加･離脱に対して, 一貫性を常に保つことが大きな負荷になるものと考える.

6. 実験と評価

次のような定常時でのノード検索とノードの離脱、再参加時でのノード検索を実行した.

(1) 定常時でのノード検索

仮想の IP アドレスを乱数で一定数（128～1024 ノード）生成しシステムに参加させた後, 総当り方式とランダム方式でのノード検索を Kademlia と Chord システムで実行し, コンタクトしたノード数を計測した.

総当り方式ではシステム内のノード同士でノード検索を実行する. ノード数をNとするとN^2回の検索が実行される. このテストは全て成功する.

ランダム方式では総当りと同様に一定数のノードを生成後に, 故意に検索をフェイルさせるためにランダムなIPアドレスの検索をN^2回実行する.

(2) ノードの離脱と再参加時でのノード検索

一定数のノードをシステムに参加させた後ノードが離脱したとき, 及び離脱後再び同じノード ID で参加させたときのノード検索テストを実行し, フェイルした割合を計測した. Kademlia ではノードの離脱時と再参加時でのテストを行い, Chord では離脱時のみテストした. なお, Kademlia ではノード検索のテスト中コンタクトしてきたノードはバケットに登録しないようにしてノード検索の評価に影響を及ぼさないよう

にしている.

6.1 定常時でのノード検索

(1)Chord

Chord環のサイズをm=40(modulo 2^{40})とし，128 ノードから 1024 ノードまで 128 単位でノード数を増やし総当りとランダム検索を実行した.

このテストは両者同様な結果となり，ノードコンタクト数は約(4/5)$\log_2$Nになっている.

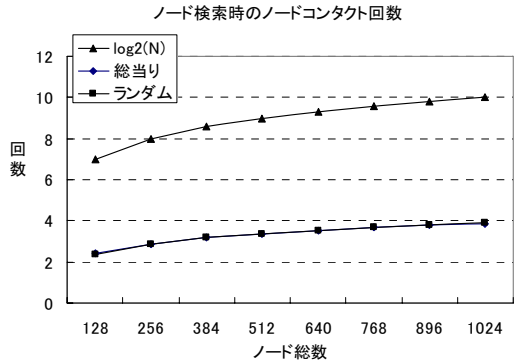

図 4. Chord のノード検索

(2)Kademlia

Kademlia でも Chord 同様 128 ノードから 1024 ノードまで 128 単位でノード数を増加させるが，k の値を 4,8,16,32 変えて総当りとランダム検索を実行した.

総当りの結果は k の値が大きいほど k-バケットに多くのノードを収容できるため，ノードコンタクト数は少なくなっている.

一方ランダムの場合はChordとは対象的に$\log_2$Nの値をかなり上回っている．kの値が大きいほどその傾向は著しい. これを改善するには総当りによる検索が$\log_2$Nを超えないという結果から，c・$\log_2$Nを超えたら検索を停止するようにするという方法が考えられる(cは定数). 但し，システムのノード数Nを求めるために定期的にノードをスキャンする処理を実行しておくようにする.

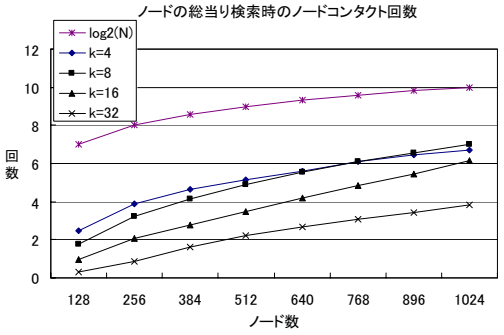

図 5. Kademlia のノード検索（総当り）

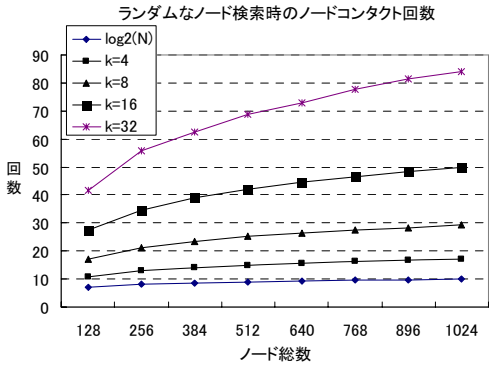

図.6 Kademlia のノード検索（ランダム）

6.2 ノードの離脱と再参加時のノード検索

初期ノードとして 1024 ノード (k=16) とするネットワークを作成してから 128 ノードずつ 896 ノードまで離脱させて，Kademlia と Chord でそれぞれ総当り検索を実行させてフェイルする割合を計測した. Kademlia では k の値が大きいときには離脱の影響は少ない. 特に k=32 の場合はフェイルしていない. Chord のデータはノード離脱のフォローを行っていないので，影響がでている.

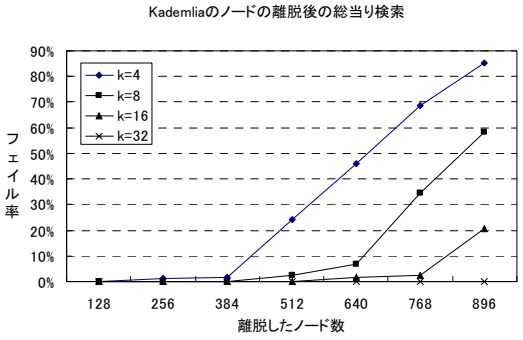

図 7. Kademlia のノード離脱後のノード検索

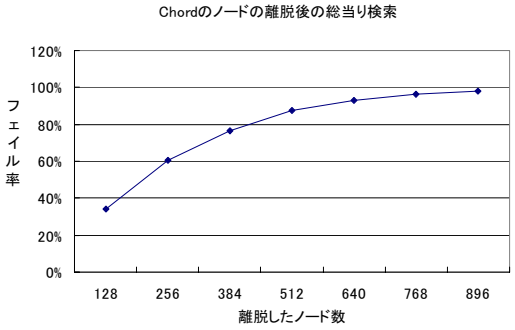

図 8. Chord のノード離脱後のノード検索

次の計測は Kademlia でノードの離脱後同じノード ID を使って再参加させて総当り検索を実行する. フェイル率は数パーセントであまり影響ない.

なお, Chord の場合はノード離脱後の再参加の処理を実装していないのでこのテストは実施していない.

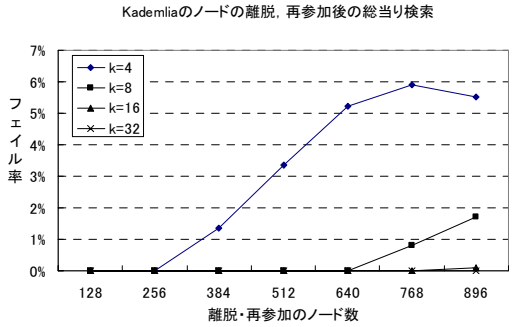

図 9. Kademlia のノード再参加時のノード検索

7. おわりに

本論文ではノードの離脱と再参加が頻繁に繰り返されるユビキタス環境において二分木構造の DHT を持つ Kademlia が Chord と比べて P2P システムとして適切であることを示した.

Kademlia はスケーラビリティがあり，ノード検索が高速で, ノードの離脱と再参加時でもノード検索はフェイルしにくいという特性を持つ. また, バケットサイズが大きいほどノード検索の性能がよいが, ランダムな検索では逆に効率が悪いという結果も明らかにした. これについては改善案を提示した.

今後はバケットリフレッシュやコンテンツの再発行時におけるトラフィックの軽減化の検討を行い，その実装化を図っていきたい.

参考文献

[1] Napster, http://www.napster.com/

[2] Gnutella, http://www.gnutella.com/

[3] Ion Stoica, et al., Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications IEEE/ACM Transactions on Networking, Vol. 11, No. 1, pp. 17-32, Feb 2003.

[4] Maymounkov, P., and Mazieres, D., Kademlia: A peer-to-peer information system based on the XOR metric. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems, Springer-Verlag version, Cambridge, MA (Mar. 2002).

[5] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.

[6] Ben Y. Zhao, et al., Tapestry: A Resilient Global-scale Overlay for Service Deployment, IEEE Journal on Selected Areas in Communications, January 2004, Vol. 22, No. 1.

[7] オブジェクト指向スクリプト言語 Ruby, http://www.ruby-lang.org/ja/