Vol.2013-DPS-154 No.42
Vol.2013-CSEC-60 No.42
2013/3/15

P2P システムにおけるハッシュ木の効率的な構築法の検討

樋口 太平[1,a)]　双紙 正和[1]　浅枝 智之[1]

概要：近年，中央サーバを用いずユーザ間で通信を行う Peer-to-Peer（以下 P2P）システムが注目を集めている．ここで，データの完全性を検証するデータ構造として有名なものにハッシュ木があるが，P2P システムにおけるハッシュ木の分散的な構成法については知られていない．本研究では，P2P システムにおける分散ハッシュテーブルの有名な実現法として，Chord を対象とし，その上で，ハッシュ木を実現する方法について考察する．

キーワード：P2P，Chord，ハッシュ木，データ認証

Consideration for Efficient Construction of Distributed Hash Trees on P2P Systems

Taihei Higuchi[1,a)]　Masakazu Soshi[1]　Tomoyuki Asaeda[1]

Abstract: Peer-to-Peer (P2P for short) systems and Distributed Hash Tables (DHTs for short), which are implementations of P2P systems, have attracted much attention in recent years because of their performance, scalability, and fault tolerance. Unfortunately, as far as we know, little is known about efficient construction of distributed hash trees on P2P systems. Therefore in this paper we propose several constructions of the hash trees on Chord, which is a most famous implementation of DHTs.

Keywords: P2P, Chord, hash tree, data authentication

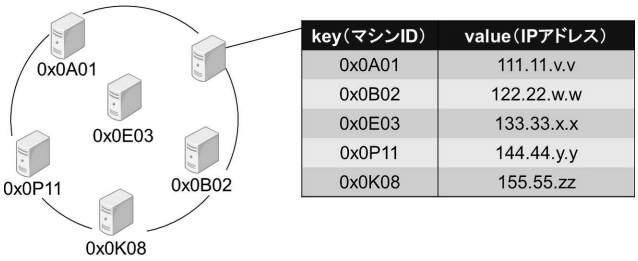
1. はじめに

近年，データの送受信に基地局となるサーバを用いず，通信端末であるピア同士で通信を行う Peer-to-Peer（以下 P2P）という通信方式が発明された．従来の基地局を介して通信を行う方式では，通信を行うクライアントが増える毎に，基地局となるサーバに負荷がかかり，処理能力の低下や，サーバ自体が故障する可能性もあった．その点，P2P は通信者間同士で通信を行うため，負荷が分散され，参加するピアを大規模に拡大することが可能である．この P2P ネットワークを実現する方法の 1 つとして Chord[1] があげられる．Chord は，システムへの負荷を分散させることで故障や障害に対して冗長性があり，従来のように中央サーバを用いないことで拡張性の高いネットワークとなっている．また分散ハッシュテーブルを実現する方法としても有効である．また，1 台以上のコンピュータで保存，処理等の作業を行う任意のデータの完全性を検証する方法の 1 つとして，ハッシュ木というものが存在する．ハッシュ木は，すでに P2P ネットワークにおいて，データが完全であるかどうか，つまり取得データが破損したり改竄されていないかを検証するのに用いられている．しかし，Chord のシステムにハッシュ木を用いた研究は存在しない．本研究では，P2P システムにおいて分散ハッシュテーブルを用いた代表的なシステムである Chord で，ハッシュ木を実現するためのアルゴリズムを提案する．

2. 関連研究

ここでは関連研究について述べる．

[1] 広島市立大学
Hiroshima City University
[a)] mw68020@wm.hiroshima-cu.ac.jp



key（マシンID）	value（IPアドレス）
0x0A01	111.11.v.v
0x0B02	122.22.w.w
0x0E03	133.33.x.x
0x0P11	144.44.y.y
0x0K08	155.55.zz

図 1 DHT の例
Fig. 1 Example of DHT

2.1 分散ハッシュテーブル（Distributed Hash Table）

ある文字列を key とし，key に対応するハッシュ値 value の組を保持する方法をハッシュテーブルという．ハッシュテーブルは，文字列情報に固有の値を割り当てるため，辞書式のような文字列検索でなく数値から検索を行えるため，$O(1)$ というオーダーで高速な検索を実現できる．このハッシュテーブルをネットワーク上の複数のノードで管理する方法として，分散ハッシュテーブル（以下 DHT）が存在する．DHT は，現実のネットワーク上に仮想的に構築されたネットワークに，ハッシュ値を射影させることで実現可能である．上述したハッシュテーブルを空間上のノードが分割して管理することで，負荷を分散することが可能で，システムを大規模に拡張可能となる．**図 1** は DHT の例である．図 1 では，複数のマシンから仮想的に構築されたネットワークにおいて，各マシンが固有に持つマシン ID を key として，各マシン ID に割り当てられた IP アドレスを value とし，DHT を実現している．

2.2 Chord

ここでは，Chord の概要と主な機能について述べる．

2.2.1 概要

Chord は，DHT を実現するアルゴリズムとして，中央サーバを用いない P2P システムネットワーク上で，データ等のコンテンツを高速に検索可能な方法である．ベースのハッシュ関数として、SHA-1[2] の識別子を使用しており，2^{160} の大きさの識別子空間において，各ノードに対して $0 \leq ID \leq 2^{160} - 1$ の範囲で ID が割り当てられている．中央サーバを用いないため，システムへの負荷が一極集中することなく，参加ノードに負荷が分散され故障などに対して冗長性があり，数千数万のノードをシステムに参加させることが可能である．仮想的にネットワーク上に配置されたノードを環状に繋いで，時計回りにルーティングを行う．また，各ノードは SuccessorList や FingerTable，Predecessor といった経路情報を保持しており，これにより耐故障性や，全体のノード数が N のとき，別ノードの検索時間を $O(\log N)$ に短縮することが可能となる．

2.2.2 SuccessorList

Chord の空間内において，任意のノード k から時計回りに移動して，最初にあたるノードを，ノード k の Successor と呼ぶ．**図 2** は，Successor の一例である．図 2 の場合，ノード N15 と N26 の間にはノードが存在しないため，N15 の Successor は N26 となる．この Successor は，Chord における到達性を保障している．任意のノードがあるデータを認証するとき，各ノードが Successor を持つことで，目標のノードへ到達することが可能となる．しかし，もし Successor にあたるノードが故障などの理由により到達不可能となってしまった場合，リンクが途切れる形となり，到達性が保障されなくなる．これを回避する機能として，各ノードが Successor を複数保持する SuccessorList が存在する．SuccessorList は，自ノードの次，自ノードの次の次…といった具合に，全体のノード数が N 個の空間において，長さ $r = O(\log N)$ の分だけ，経路表に保持している．**図 3** は，SuccessorList の一例である．また，図 3 の場合のノード N15 における SuccessorList の経路表は**表 1** のようになる．

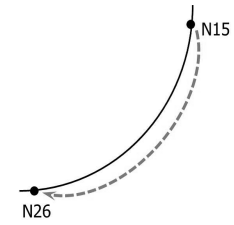


図 2 Successor の例
Fig. 2 Example of Successor

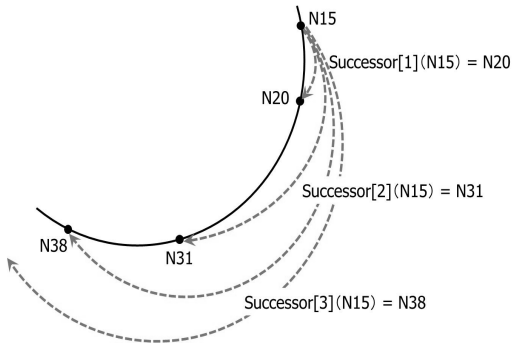


図 3 SuccessorList の例
Fig. 3 Example of SuccessorList

表 1 ノード N15 の保持する SuccessorList の経路表の例
Table 1 Example of node N15's routing table of SuccessorList

SuccessorList[i]	i 番目に経路表に保持するノード
SuccessorList[1](N15)	N20
SuccessorList[2](N15)	N31
SuccessorList[3](N15)	N38
⋮	⋮

2.2.3 FingerTable

Successor により，ノードからノードへの到達性は保証されたが，Chord には数万のノード数が参加することが

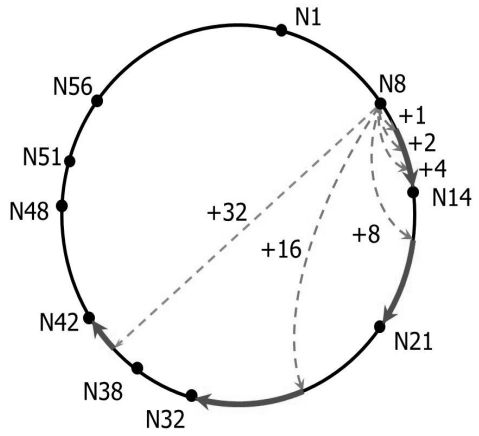


図 4 FingerTable の例
Fig. 4 Example of FingerTable

表 2 ノード N8 の保持する FingerTable の経路表の例
Table 2 Example of node N8's routing table of FingerTable

FingerTable[i]	i 番目に経路表に保持するノード
FingerTable[1](N8)	N14
FingerTable[2](N8)	N14
FingerTable[3](N8)	N14
FingerTable[4](N8)	N21
FingerTable[5](N8)	N32
FingerTable[6](N8)	N42

予想される．その際，次ノードのみを保持する Successor だけでは，N 個先のノードの経路長は N となり，大規模な利用が予想される Chord において，ノード数 N に対し計算量が $O(N)$ となる．この計算量を減らす機能として，FingerTable が存在する．FingerTable は，全体 n ビットに対して自分のノードから $+2^0, +2^1, +2^2, \cdots, +2^{n-1}$ 先のデータの担当ノードを経路表として保持する．**図 4** は，全体 6 ビット，つまり Chord に参加しているノード数が 64 個の場合の FingerTable の例である．このとき，ノード N8 の持つ FingerTable は，**表 2** のようになる．例えば，ノード N8 がノード N48 を検索する場合，Successsor だけによるルーティングでは経路長は 6 である．しかし FingerTable を用いれば，ノード N42 への経路情報が保持されているため，経路長は 2 となる．このように，自分のノードから全体の $\frac{1}{2}$ 以上経路長の離れたノードへの通信も，計算量が半分以下で検索可能となる．

2.3 ハッシュ木

任意のデータの完全性を検証するアルゴリズムとしてハッシュ木が存在する．ハッシュ木は，葉にあたる部分にデータのハッシュ値を入力し，そのハッシュ値を繰り返しハッシュ関数にかけていき，ハッシュ木を構成していくものである．**図 5** は，データ数 8 の場合のハッシュ木の構成例である．図中の k_i はデータ d_i のハッシュ値，$h(m||n)$ は，各枝の子 m と n からなるハッシュ関数を示し，|| は文字列の連結を示す．このハッシュ木の利点として，あるデータの検証のためハッシュ木を再構成する際, 全てのハッシュ値を持たなくても，一部の必要な枝だけを持っていれば再構成が可能である．この必要な枝を認証パスという．深さが H のハッシュ木において，深さ $h(=1, 2, \cdots, H)$ の枝を頂点とする部分木において，葉に検証するデータのハッシュ値が含まれていない部分木の頂点を認証パスとして保存する．ハッシュ値が含まれている方の部分木において，深さを 1 下げてさらに同様の作業を行う．これを深さ $H-1$ まで繰り返して全ての認証パスを得る．例として，データ d_3 の検証をする場合を示す．d_3 の認証パスは k_4, c_{21}, c_{12} である．最初にハッシュ木が構築されたときに得た頂点の値を r とした場合，データ d_3 は d_3 に対応したハッシュ値である k_3 と k_4 から c_{22}，枝 c_{21} と c_{22} から c_{11}，c_{11} と c_{12} から検証時点での頂点 r' を構成する．ここで r と r' を比較し，値が一致すればデータ d_3 は完全であると言える．以上の流れで，データの完全性を検証することが可能である．ハッシュ木は，木の再構築に葉のデータ全てを用いなくても，葉の数 m に対して，認証パスにより $\log m$ 個の要素で構築することが可能であり，効率のよい方法となっている．

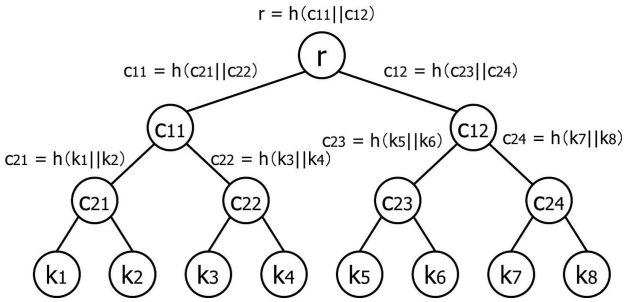


図 5 ハッシュ木の例
Fig. 5 Example of hash tree

3. 提案方式

ここでは，Chord 上にハッシュ木を構築する具体的な方法を示していく．

3.1 研究目的

Chord では，中央サーバを用いないピュア P2P 方式を採用しており，参加ノードがネットワーク上にあるデータを管理している．そのため，ネットワーク上にあるデータが欠損したり不完全でないかを第三者が検証しないため，データの完全性を検証することが重要となってくる．通常，ユーザ間でのデータ認証では公開鍵証明書を用いた認証が主であるが，計算量が大きい側面があり，数千数万のノードが参加することが想定される Chord においては，システム全体への負荷が懸念される．そこで，本研究では証

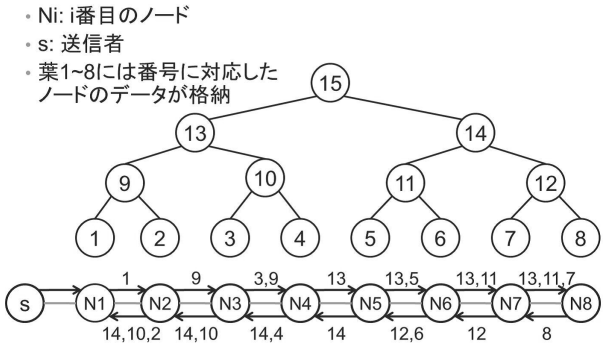


図 6 2 パスハッシュ木構成法によるハッシュ木の構築例
Fig. 6 Example of hash tree construction with 2-pass hash tree method

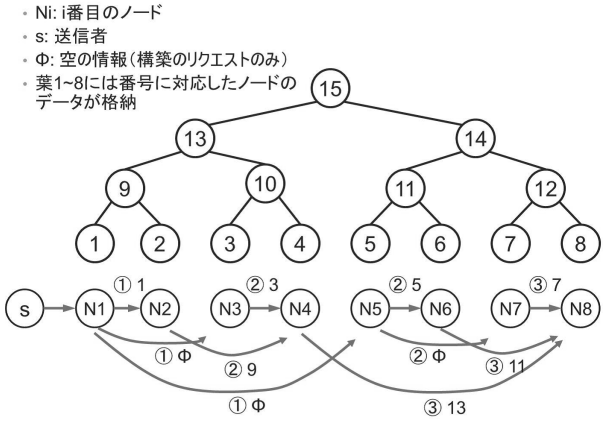


図 7 提案手法 2 によるハッシュ木の構築例
Fig. 7 Example of hash tree construction with Method 2

明書を用いたデータ認証に代わり，アルゴリズムが容易で計算量を抑えることが期待できるハッシュ木を用いたデータ認証を提案する．

3.2 提案手法

提案手法では，ハッシュ木を構築するため，その頂点 r を導出する単純のためハッシュ木を構築するリクエストを出すノードをノード N1 の 1 つと想定し，どこかのノードで頂点 r が導出できるような認証パスが受信されれば良いとする．

3.2.1 手法 1—2 パスハッシュ木構成法を利用するプロトコル

ハッシュ木を構築する方法として，Chan らが提案した 2 パスハッシュ木構成法 [3] が存在する．これは，直線的なトポロジを有する P2P ネットワークにおいて，各ノードが隣接したノードとのみ通信可能であると仮定した上で，ハッシュ木を構築する方法である．**図 6** は，ノード数を 8 とした場合の 2 パスハッシュ木構成法の概略図である．ある送信者 s はまずノード N1 にハッシュ木を構築するリクエストをメッセージとして送信する．ノード N1 は隣接したノードである N2 へ，認証パスとなる葉 1 をメッセージと共に送信する．N1 から葉 1 を受け取った N2 は，N3 の認証パスを計算し，メッセージと共に N3 へ送信する．同様に隣接したノードへ認証パスとメッセージを送信していくことで，ノード N8 でハッシュ木の頂点にあたる頂点 15 が導出される．この 2 パスハッシュ木構成法を Chord に組み込む．Chord には，Successor と呼ばれる隣接したノードへの経路情報を保持しており，これを用いて 2 パスハッシュ木構成法を実現する．

3.2.2 手法 2—ステップ数重視型プロトコル

Chord 上の各ノードは Successor，FingerTable により経路情報を保持しているため，特定の参加ノードにメッセージのブロードキャストが可能である．これらの経路情報を用いて，ハッシュ木の構築にかかるメッセージ送信のステップ数を少なくするようなプロトコルを提案する．**図 7** は，ノード数 8 の場合の例である．このプロトコルでは，まず初めにノード N1 から N2，N3，N5 へメッセージをブロードキャストする．ただし，図 7 からも分かるようにノード N1 の持つ値を認証パスとして用いるノードは N2 のみであるため，N3，N5 に送るメッセージには N1 の持つ値を送らず，空の情報 ϕ を送っている．この時点でメッセージを受け取った N3，N5 はハッシュ木を作り始めることができるため，続く 2 ステップ目で，複数のノードから別の複数のノードへメッセージを送ることができる．図 7 では，ノード N2 からノード N4 へ，ノード N3 からノード N4 へ，ノード N5 からノード N6，N7 へ同時にメッセージを送信している．3 ステップ目で，ノード N4，N6，N7 からノード N8 へメッセージを送信することでハッシュ木を構築するのに必要な認証パスがノード N8 に揃ったため，ノード N8 にてハッシュ木を構築し，頂点 15 を得ることができる．

3.2.3 手法 3—メッセージ数重視型プロトコル

メッセージ数はシステム全体の負荷に影響している．そのため，メッセージが増えればシステムに負荷がかかり，Chord 全体のパフォーマンスに影響を与える可能性が考えられるため，メッセージは少ない方が望ましいと考えられる．一方，ステップ数はハッシュ木を構築する時間に影響している．そのため，通常新規のノードが参加してきた場合，その都度ハッシュ木を構築し直すことで更新すればよいが，ステップ数があまりに多いと，メッセージを送受信している間に別のノードが参加してきた場合に，ノードの位置に応じて計算した認証パスが誤ったノードに送信されたり，また，本来メッセージを受け取るノードではない，別のノードから誤った認証パスを受信してハッシュ木を構築してしまい，システム全体でエラーを起こす可能性がある．しかし，手法 1 は，隣接ノードへ 1 つずつメッセージを送信していく方法で，ステップ数はかかるがメッセージ数の少ない手法であり，また，手法 2 は，Chord の持つ経路情報を活かしたメッセージの送信方法で，メッセージ

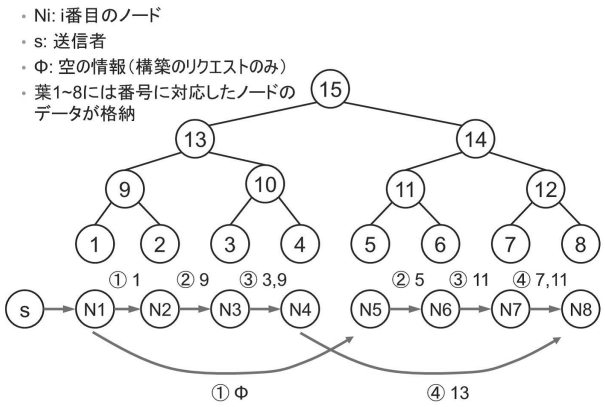


図 8 提案手法 3 によるハッシュ木の構築
Fig. 8 Example of hash tree construction with Method 3

数は多くなるがステップ数の少ない手法である．つまり，メッセージ数とステップ数はトレードオフな関係にある．そこで Chord の経路情報を活かしたうえで，提案手法 2 よりメッセージ数が抑えられるような手法を提案する．**図 8** は，ノード数 8 の場合の例である．これは，メッセージ数を抑えることのできる 2 パスハッシュ木構成法を 2 つに分けて行うような方式になっており，

N1 → N2 → N3 → N4 → N8

N1 → N5 → N6 → N7 → N8

というように，2 つの 2 パスハッシュ木構成法を並行して構築することで提案手法 1 よりステップ数を軽減し，かつ提案手法 2 よりメッセージ数を軽減できるようになっている．

4. 評価

各提案手法でかかったメッセージ数とステップ数について評価する．**表 3** は，各提案手法によるメッセージ数とステップ数をまとめた表である．表中の N は Chord に参加しているノード数である．

5. 考察

表 4 は，各ノード数における提案手法 1，2，3 のメッセージ数であり，**表 5** は，各ノード数における提案手法 1，2，3 のステップ数である．表 4 から，ノード数が小さいとき，メッセージ数に大きな違いは見られないが，ノード数が大きくなるにつれメッセージ数の差は増えていき，ノード数が 1024 のときの最も差の大きい提案手法 1 と提案手法 2 において，メッセージ数の差は 511 にもなる．しかし，表 5 を見ると，提案手法 1 と提案手法 2，3 とのステップ数の差は非常に大きく開いており，数万のノードの参加が想定される Chord において，提案手法 1 は計算時間が大きくかかってしまうため適していないと考えられる．また，提案手法 2 と提案手法 3 を比較すると，ステップ数は差が 1 しかないにも関わらず，ノード数が 1024 のときのメッセージ数の差は提案手法 3 の方が提案手法 2 より 256 も少ないことが分かる．しかし，ノード数 1024 のときの提案手法 1 と提案手法 3 のメッセージ数の差は 255 と決して差が小さくはないため，メッセージを重視した提案手法 3 においては，さらにメッセージ数を少なくできるような改良が必要である．また，上記の 3 つの提案手法では，各ノードは自分以外の他ノードのデータを持っていない状況を想定してプロトコルを提案した．しかし，あるノードに複数のデータを集めておけば，以上の 3 つの手法よりステップ数，メッセージ数をともに減少させることが予想される．

表 3 各提案手法におけるメッセージ数，ステップ数の評価
Table 3 Evaluation of the number of messages and steps in our proposed methods

	メッセージ数	ステップ数
提案手法 1	$N-1$	$N-1$
提案手法 2	$\frac{3}{2}N-2$	$\log N$
提案手法 3	$\frac{5}{4}N-2$	$(\log N)+1$

表 4 各提案手法におけるメッセージ数
Table 4 The number of messages in our proposed methods

参加ノード数	提案手法 1	提案手法 2	提案手法 3
8	7	10	8
16	15	22	18
32	31	46	38
64	63	94	78
128	127	190	158
256	255	382	318
512	511	766	638
1024	1023	1534	1278

表 5 各提案手法におけるステップ数
Table 5 The number of steps in our proposed methods

参加ノード数	提案手法 1	提案手法 2	提案手法 3
8	7	3	4
16	15	4	5
32	31	5	6
64	63	6	7
128	127	7	8
256	255	8	9
512	511	9	10
1024	1023	10	11

6. まとめ

中央サーバを用いない P2P ネットワークにおいて代表的な Chord を用いて，ハッシュ木による効率的な認証方法を提案した．本研究では，ハッシュ木の構築時間に影響するステップ数と，システム全体の負荷に影響するメッセージ数を考慮したハッシュ木の構築方法を提案した．2 パスハッシュ木構築法を用いたプロトコルとステップ数を重視したプロトコル，メッセージ数を重視したプロトコルの 3

つの手法を提案した.

参考文献

[1] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M., Dabek, F. and Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications, *Networking, IEEE/ACM Transactions on*, Vol. 11, No. 1, pp. 17–32 (2003).
[2] Eastlake, 3rd, D. and Jones, P.: US Secure Hash Algorithm 1 (SHA1) (2001).
[3] Chan, H. and Perrig, A.: Round-Efficient Broadcast Authentication Protocols for Fixed Topology Classes, *Security and Privacy (SP), 2010 IEEE Symposium on*, IEEE, pp. 257–272 (2010).