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

2006－DSM－40 (5)
2006／3／29

ハッシュ空間の複製による Chord の高速化手法の提案と検証

妙中 雄三† 山口真之介† 西野 和典† 大西 淑雅†

† 九州工業大学 情報工学部

あらまし 昨今，ネットワークの拡大やコンピュータの高性能化に伴い，コンピュータを対等な関係で相互に接続し自律的にサービスを維持する Peer to Peer 技術が注目を浴びている．しかし，接続ノード数が大幅に増加した場合に性能の低下が予測され，更にスケーラブルな P2P ネットワークが必要であると考えられる．本稿では，P2P ネットワークにおける検索手法に着目し，ノード数に影響を受けにくい検索の高速化手法を提案する．本手法は，Distributed Hash Table を用いた検索手法 Chord を基に，ノード数が増加した場合にハッシュ空間を複製しノード数を削減することで高速化を図る．本稿では，これらの設計を述べ，さらにシミュレーションによる評価を行う．

A High Speed Search Algorithm using Reproduction of Chord's Hush Space

yuzo TAENAKA†, Shin'nosuke YAMAGUCHI†, Kazunori NISHINO†, and Yoshimasa OHNISHI†

† Department of Information Engineering, Kyushu Institute of Technology

Abstract Recently, Peer-to-Peer technology that is autonomous system attracts attention. However, it is thought that the lower performance when the number of nodes increases greatly. Therefore a more scalable P2P network is necessary. In this paper, we propose the high speed search algorithm that unrelated to the number of nodes. Proposal algorithm is based on Chord using "Distributed Hush Table". When the number of nodes in P2P network increases, this algorithm attempts to reduce the number of nodes by reproducing of Chord's hush space. Then we aim at the speed-up of the search speed. In this paper, we describe these designs and evaluate it by the simulation.

1. はじめに

昨今，インターネット環境が拡大しており，接続するコンピュータも大幅に増加している．それに伴い，従来のクライアントサーバモデルではなく，コンピュータを対等な関係で相互に接続する Peer to Peer（P2P）技術が注目を浴びている．このような状況において，P2P のネットワークの性能向上のため様々な研究が行われ，Distribute Hash Table（DHT）を用いた検索手法が多く提案されている [1–3]．また，DHT の一種である Chord [2] を高速化する研究も行われている [4,5]．しかし，これらの手法は検索手法を最適化しているため，大幅なノード数増加による検索性能の低下が懸念される．

そこで本稿では，Chord を基に，ノード数が増加した場合においても一定の検索性能を有する検索の高速化手法の提案，検証を行う．

2. Chord

本章では，DHT を用いた検索手法である Chord [2] の説明を行う．なお，本手法における検索のアルゴリズムは Chord と同じ手法としている．

Chord はハッシュ関数を用い，検索を行うハッシュ空間を構成する．ハッシュとは，文字列などの入力に対し，出力として特定の数値が得られるものである．ここで得る数値は，他の入力に対する出力値と重複する可能性がほぼ無いものである．Chord では，一般に 160bit のハッシュ関数を用い，

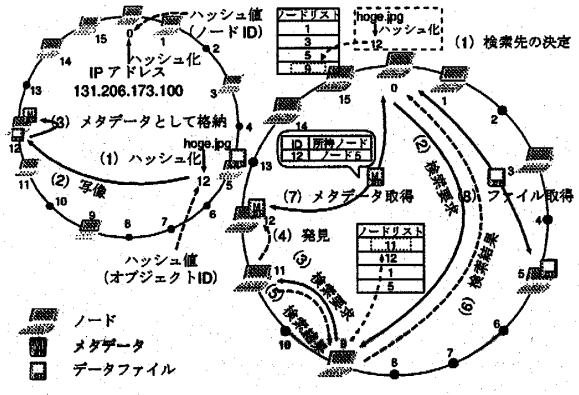

図 1 Chord のオブジェクト配置・検索例

$0 \sim 2^{160}$ の数直線の始点・終点を一致させたリングを用いる．図 1 に 4bit ハッシュ関数を用いた Chord におけるオブジェクトの配置，検索の例を示す．以降，検索対象をデータファイルに限定する．

図 1 左では，ファイル名をハッシュ化し，ハッシュ空間に写像している．このファイルのハッシュ値をオブジェクト ID と呼ぶ．同様に，ノードは IP アドレスをハッシュ化し，ハッシュ空間に写像する．このノードのハッシュ値をノード ID と呼ぶ．また，ハッシュ空間上における任意の点 P において，時計回り方向で，最も近いノードを点 P の successor（以降 succ），半時計回りで，最も近いノードを点 P の predecessor（以降 pred）と呼ぶ．

ファイルは写像点の succ であるノードにメタデータ（オブジェクト ID，ファイル所有ノードの対）に要約して格納される．図 1 右の検索処理では，ファイル名からオブジェクト ID を計算し，各ノードが持つノードリストにおいてオブジェクト ID の pred であるノードを選択し，検索要求メッセージを送信する．これを繰り返すことで，メタデータを発見し検索が完了する．

このノードリストは，2^n 離れた点の succ であるノードを一覧している．そのため，検索要求メッセージ送信を送信する毎に目的のオブジェクト ID との距離が 1/2 になる．つまり，検索に必要なホップ数は $O(\log N)$（N: ノード数）となる．

また Chord では，このノードリストを Finger Table と呼び，Finger Table 内のノードの生存確認を行う．そのため，各ノードは定期的に確認メッセージを送信する必要がある．

3. 提案手法

3.1 概　　観

本手法では，Chord で用いるリングを分割することでリング毎のノード数を減らし，検索の高速化を図る．図 2 は，本手法の概観を示している．

従来の N ノードが存在する P2P ネットワークでは，単一の Chord のリングで構成される（図 2 左）．本手法では，N ノードが存在する Chord のリングを分割し，$N/2^i$（i: 分割回数）ノードが存在する複数のリングに分割する（図 2 右）．分割にはあらかじめ，分割を行うノード数の閾値を決定しておき，リング内のノード数が閾値以上になると分割を行う．そして，リング内のノード数が閾値未満になるまで分割を繰り返す．

また本手法では，検索を分割後の各リング内で完結させる．検索はリング内にメタデータが存在する場合に行えるため，**分割時においてメタデータの複製を行う．分割後においても，**いずれかのリングでメタデータの変更が発生した場合，リング間でメタデータを同期する．その際，分割の制御のためのノードが同期を行う．

以上の手法をとることで，リング内に存在するノード数の上限を設定することができ，ノード数が増加した場合でも検索性能が低下しないと考えられる．

3.2 設　　計

本節では，まず Chord で用いられない用語を定義し，本手法で必要な分割処理，同期処理の説明を行う．また，分割後における Finger Table の維持手法の説明を行う．

3.2.1 用語定義

- 代表ノード

分割処理やメタデータの同期処理を行うためには制御を行うノードが必要である．分割の制御やメタデータの同期処理を行うノードを代表ノードと呼ぶ．代表ノードの選択方法は，オンライン時間が最も長いノードとする．また，一度決定した代表ノードは離脱するまで変更されることは無いものとする．

- 検索リング

分割後のリングを区別するために，本稿では分割 / 検索を行うリングを検索リングと呼ぶ．この検索リングは Chord のリングと同等である．この検索リングを識別するため検索リング毎にユニークな ID を割り当て，リング ID と呼ぶ．

また各ノードは，送信するメッセージ全てに所属する検索リングのリング ID を付加し送信する．メッセージを受け取るノードは，自ノードが所属する検索リングのリング ID と等しい場合のみメッセージを受け取る方式をとる．これによ

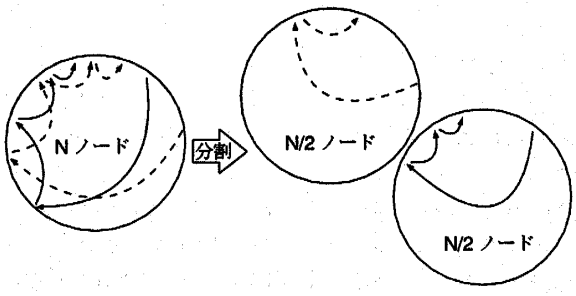

図 2 提案手法の概観

り，分割処理を速やかに行えるとともに，分割後の Finger Table の維持が容易になる．

- 管理リング

各検索リングに存在し，分割や同期処理の制御を行うノード（代表ノード）によって構成されるリングを管理リングと呼ぶ．管理リングは検索リングとは異なり分割を行うことは無く，メタデータの同期を行うために用いる唯一のリングである．

3.2.2 検索リングの分割

新規ノードの参加

代表ノードが分割処理を開始するのは，検索リング内のノード数があらかじめ決めた閾値以上になった場合である．そのため，代表ノードは，管理する検索リング内のノード数を把握しなければならない．そこで，新規ノードが P2P ネットワークに参加する時点で，代表ノードに対し JOIN メッセージを送信する方式をとる．同時に，新規ノードは所属するリング ID を特定する．なお，その他の参加処理は，Chord と同様の手法をとる．

分割処理

検索リングの分割は，分割後のリング ID を決定する処理と分割を行う処理に分けられる．リング ID を決定する処理と分割後の検索リングを図 3 に示す．図 3 では，検索リング内のノード数が閾値以上となり，代表ノードが分割処理を開始している．代表ノードが送信した移動要求メッセージを始点として，各ノードが検索リング内で移動/待機要求メッセージを succ に対し送信している．分割処理の流れは以下の通りである．この手順では，(1) ～ (3) が分割後のリング ID を決定する処理，(4) (5) が分割を行う処理となる．

（1） 代表ノードは succ に対し移動要求メッセージを送信する．

（2） 受け取ったメッセージが移動要求メッセージの場合は待機要求メッセージ，待機要求メッセージの場合は移動要求メッセージを succ に送信する．

（3） 代表ノードは移動/待機要求メッセージを受信すると自ノードのリング ID を変更する．

（4） 代表ノードが新たなリング ID を付加した各種メッセージを送信する．

（5） 各ノードは異なるリング ID（分割後のリング ID）が付加されたメッセージを受信することで分割を知り，自ノードのリング ID を変更する．

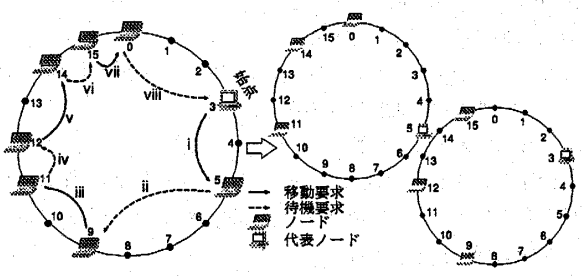

図 3 分割処理の例

3.2.3 メタデータの同期

検索リングの分割後において，いずれかの検索リングでメタデータの更新が発生する場合のために同期処理を行う．同期処理は，代表ノードが行うため，新たに各検索リングの代表ノードのみで構成する管理リングを作成する．この管理リング上で更新されたメタデータの交換を行い，検索リングへ適用することで各検索リングの同期を行う．また，検索リングにバージョン管理を採用し，どの更新が行われた状態かを把握できる方式とする．

各検索リングの代表ノードは，同期タスクを定期的に実行し，(a) 更新検索処理，(b) 更新処理の一方を実行する．図 4 に各処理の動作を示す．

(a) 更新検索処理

更新検索処理は，検索リング内でメタデータの更新が無い場合に実行する．管理リングにおいて現在のバージョン +1 を検索する．検索が成功するとメタデータの更新情報の取得・検索リングへの適用を行い，バージョン番号を増加させる．

(b) 更新処理

更新処理は検索リング内でメタデータの更新が発生している場合に実行する．検索リング内で発生した更新情報を代表ノードに渡し，検索リング内に適用せず削除する．代表ノードは管理リングにおいて現在のバージョン +1 を検索し，メタデータの更新情報を所持するノードを決定する．そのノードに更新情報を送信し，適用は更新検索処理に任せる．

3.2.4 Finger Table の維持

本手法では，検索リングの分割処理後も，succ，pred を除き，分割前の全ての Figner Table を継続して使用する．そのため，分割後 Finger Table 内に存在する他のリング ID のノード情報を速やかに削除する必要がある．そこで，受け取ったメッセージに付加されているリング ID が所属するリング ID と異なり Finger Table に含まれている場合，Finger Table より削除する．

また，定期的な Finger Table の維持処理においても，メッセージに付加されているリング ID が異なるリング ID で

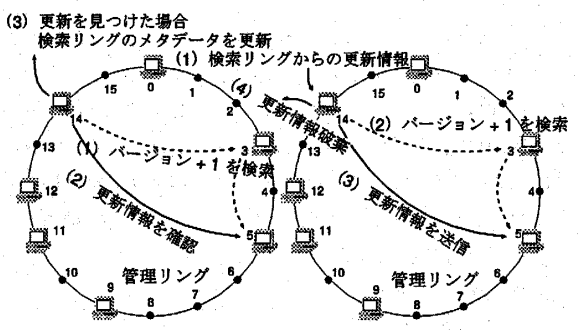

図 4 左：更新検索処理，右：更新処理の動作例

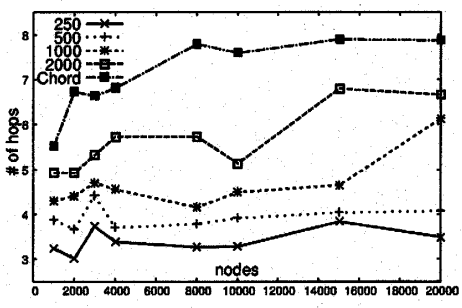

図 5　検索ホップ数の平均値

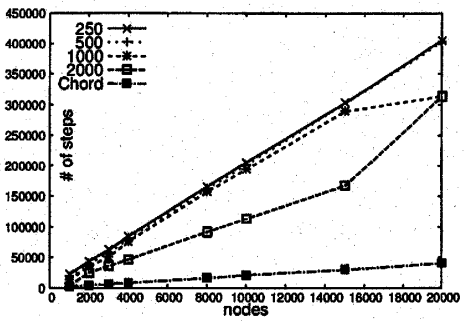

図 6　必要ステップ数

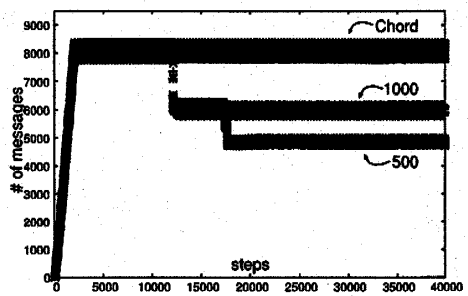

図 7　メッセージ量

あった場合は拒否されるため，誤ったノード情報を Finger Table に追加してしまうことを防止することができる．

4. 評　　価

3 章で述べた設計の評価をシミュレーションを用いて行う．シミュレーションは PlanetSim [6] を用いて行った．本章では，シミュレーション条件，シミュレーションで得られた結果を述べる．

4.1 シミュレーション条件

以下の条件のもと，シミュレーションを実施した．

- ノード

P2P ネットワークに参加するノードのノード ID はランダムに選択し，偏りが無いものとした．また，ノードの故障・離脱は無く，ノード数が増加する場合のみを想定している．

- メタデータ

メタデータはあらかじめ十分分散したものを用意しておくため，シミュレーション毎に変化する事はない．

- 検索

検索は，あらかじめ用意しておいたメタデータからランダムに選択し，合計 105 回の検索を行う．検索を行うノードは検索の試行毎に P2P ネットワーク内に存在するノードからランダムに選択することとした．

- 実行単位

シミュレーションの実行は，ステップ単位で行う．1 ステップとは，P2P ネットワーク内のノードに対するメッセージを全て各ノードに届け．処理を行うことを表している．

- 実行手順

シミュレーションでは，処理を P2P ネットワークの作成，メタデータの格納，検索の 3 段階に分けて行う．ネットワークの作成は，全ノードの Finger Table が安定し，検索リングの分割が完了した時点を表している．また，メタデータの格納はアプリケーションがメタデータを検索リングに格納した状態を表しており，全ての検索リングにおける同期処理が完了していることは保証していない．

4.2 結　　果

4.2.1 検索ホップ数

図 5 に P2P ネットワーク内のノード数に対する検索ホップ数の平均値を示す．また，分割を行う閾値を変更した場合の結果も示している．

図 5 より，閾値が小さく分割回数が多い検索リングであるほど，検索ホップ数の平均値が減少している事がわかる．よって，Chord と比較すると提案手法が高速に検索を行えると言える．

4.2.2 必要ステップ数

図 6 に P2P ネットワーク内のノード数に応じて，検索リングの分割が全て完了するまでに必要なステップ数を示す．Chord は分割を行わないため，Finger Table が安定し検索が行えるまでに必要なステップ数を示している．なお，図 6 は，ネットワークの作成に必要なステップ数のみを示し，メタデータの格納や検索に必要なステップ数を含んでいない．

図 6 では，ノード数が増加するに従い，本手法が Chord に比べ，大幅にステップ数が増加している事がわかる．ノード数増加に伴うステップ数の増加量も Chord に比べると大きく，分割処理に多くステップが必要な事がわかる．また，閾値によって分割完了までに必要なステップ数が異なるのは，分割回数が異なるためである．

しかし，検索は Chord のステップ数以下で行う事ができるため，性能の劣化にはならない．

4.2.3 メッセージ量

図 7 に，全ノード数が 1,000 の P2P ネットワーク内における全てのメッセージ量を示す．なお，図 7 は，データ格納や検索処理を含んでいない．

図 7 では，全ノード数が 1,000 であるため，閾値 500 が 2

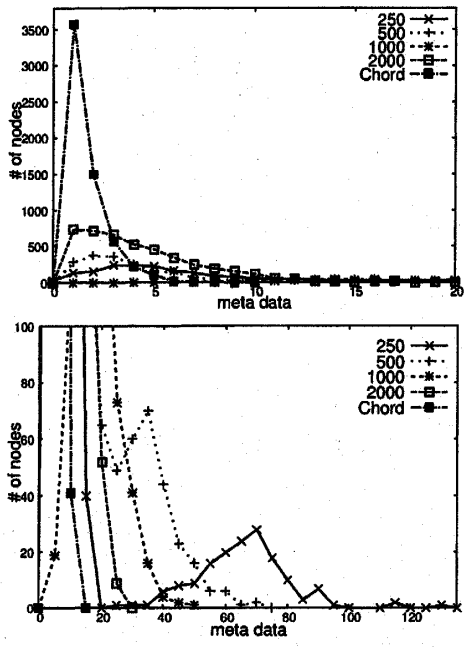

図 8　データ所持量分布

回，1,000 が 1 回の分割を行っている．この分割を行う際にメッセージ量が減少し，減少量が 1 回目の分割から分割毎に約 1/2 となっている事がわかる．また，分割処理の際に必要な移動 / 待機要求メッセージに伴うオーバーヘッドも殆ど見られない．よって，提案手法は Chord と比較すると，P2P ネットワークを維持するために必要なメッセージ量を削減できると言える．

4.2.4　メタデータ分布

図 8 にアプリケーションがメタデータを格納した時点における，メタデータの所持量の分布を示す．図 8 の上・下はそれぞれスケール異なるものである．条件は，P2P ネットワーク内の全ノード数を 5,000，格納するメタデータ量を 10,000 個として実行している．

図 8 上では，Chord がメタデータを 1 個持つノードが多く存在している事がわかる．よって Chord では，メタデータが広く分散し，ノード単位の負荷も低いと言える．一方で，提案手法では，閾値が小さく分割回数が多いほど多くのメタデータを持つノードが存在し（図 8 下），閾値が大きいほど Chord に近い分布になっている事がわかる（図 8 上）．これは，分割によって検索リング内のノード数が減少するにも関わらず，メタデータを全て複製するためである．よって提案手法においては，メタデータが一定であれば閾値が小さくなり分割回数が増加するほどノード単位の負荷が高くなると言える．

5. 関連研究

PChord [4] は，Chord の Finger Table に加えて IP レイヤにおける距離を考慮した proximity list という経路表を用いることで検索の高速化を図る．proximity list を作成・維持するためのメッセージは必要なく，検索要求メッセージや Chord の維持メッセージより RTT 値を計算し，RTT 値の低いノードを一覧している．しかし，proximity list が利用される場合は，Finger Table 内のノードより目的のオブジェクト ID に近いノードが含まれている場合のみである．そのため，proximity list による検索速度の高速化が望めない状況が存在する．一方，本手法では，検索リング内のノード数を減少させることで高速化を図っているため，常に検索速度の向上を実現できる．

EpiChord [5] は，Chord の検索クエリを並列に送信することで検索の高速化を行う．Finger Table 内から目的のオブジェクト ID の前後 r ノードに対し検索要求メッセージを送信し，応答の受信毎に，応答に基づいた検索要求メッセージを非同期に送信する．しかし，複数の検索要求メッセージにより検索の高速化を図っているため，メッセージ数の増加というデメリットが生じる．また，Finger Table の維持メッセージは，検索要求メッセージを監視・利用する事で削減している．検索要求メッセージが少ない場合は，Chord と同様に生存確認のためのメッセージを送信する．よって，十分な検索要求メッセージが存在する場合はメッセージ量が大幅に削減されるが，検索要求メッセージが不足する場合は Chord と同等のメッセージ量である．

一方，本手法では，検索リング内のノード数が少ないため，Finger Table 内に目的にオブジェクト ID に十分近いノードが存在し，少ないホップ数で高速に検索を行える．また，メッセージ量も EpiChord では検索要求メッセージが十分に存在する場合のみ削減できるのに対し，本手法は分割を行っているため，常に削減することに成功している．

6. 考　　察

6.1　分割に必要なメッセージ量

本手法では，分割処理を完了するまでに必要なステップ数が増加する．この原因として以下の 2 点が考えられる．

- 移動 / 待機要求メッセージ

分割処理において，移動 / 待機要求メッセージを交互に送信し，検索リングを一周する．この際，1 ノードの経由毎に 1 ステップ必要であり，ノード数が増加するに伴いステップ数も同様に増加しているためであると考えられる．

- Finger Table の安定化処理

分割処理を行うためには succ，pred が安定している必要がある．そのため，分割毎に安定になるまでのステップが必要となり，複数回分割を行うと無視できないステップ数になると考えられる．

今後は，消費ステップ数を低下させる手法を検討する必要がある．設計では，代表ノードが succ に対して移動要求メッ

セージを送信するのみであるが，succ 以外の Finger Table 内のノードへ並列に送信する手法が考えられる．この手法を用いる事で m bit のハッシュ空間において，N ノード存在しているリングをメッセージが 1 周するまでに約 N ステップ必要であったものが，ノード分布が一様であるとすると $(2^m - 2^{m-1})/2^m \times N$ ステップ程度に減少すると考えられる．

6.2 メッセージ量減少

Chord と比較すると検索リングを分割することのみでメッセージ量の削減を実現している．減少したメッセージには **Finger Table の維持メッセージ量の割合が多いことが考え**られる．これは，Chord におけるメッセージ量のほとんどが Finger Table や succ・pred の維持に用いられるためである．Finger Table は自ノードから 2^i 離れた点の succ を一覧したものであるため，ノード数が減少することで重複したノードが存在することとなる．これに伴い，各ノードが Finger Table 内のノードの生存確認を行う維持メッセージが減少することになり，検索リングのメッセージ量が減少していると考えられる．

6.2.1 分割を繰り返すことによる影響

本手法では分割毎にメッセージの減少量が約 1/2 になる．これは，Finger Table における「近いノードを密に持つ」という性質によるものだと考えられる．検索リングを分割すると，Finger Table 内に重複するノードが現れ，エントリ数が減少する．Finger Table は自ノードに近いノードを密に含んでいるため，一度目の分割で近いノードが複数，重複する．更に分割を繰り返すと，Finger Table 内の近いノードで重複が現れなくなる．また，遠いノードは，重複が発生するほど密に持っていないため，重複は発生しない．よって，検索リングの分割を繰り返すと，Finger Table 内において，一度の分割で影響を受けるノード数が減少するためだと考えられる．

6.3 ノード毎の負荷と検索速度

図 8 より，分割の閾値が小さいほどノード毎の負荷が増加し，分割の閾値が大きいほどノード毎の負荷が減少するという結果が得られた．また，図 5 より，閾値が小さいほど検索の高速化性能を向上でき，閾値が大きいほど高速化性能が低下するという結果が得られた．よって，ノード負荷と検索速度はトレードオフの関係にあると言える．

しかし，メタデータが少ない環境においては，閾値が低い状態が最適であり，一方でメタデータが多い環境では閾値が大きい状態が最適であると考えられる．本手法では，閾値を固定的に設定しているため，ノード負荷の調節が行えない．そこで，メタデータの量とノード数の関係を考慮した閾値の決定手法が必要であると考えられる．

本稿におけるシミュレーション規模においては，閾値 2,000 程度が適していると考えられ，ノード毎のデータ所持量の平均を 5 個程度に設定すると最適だと考えられる．しかし，そのためにはメタデータ量の把握やメタデータ量の変更時における処理などを検討する必要がある．

6.4 代表ノードの負荷

本手法では，分割やメタデータの同期に代表ノードを用いている．そのため，連続したノード参加やメタデータの格納が発生した場合に負荷が集中すると考えられる．そこで，代表ノードを複数設定する事による負荷分散や，代表ノード以外のノードで自律的に処理を分散するなどの改善が今後の課題である．

6.5 ノード数減少時の処理

本稿では，ノードの参加のみを考慮し設計を行ったため，ノードの離脱や故障を考慮していない．また，ノードの離脱による検索リング内のノード数減少も考慮していない．そこで，ノードの突然離脱も考慮し，ノード数が減少した場合の処理を設計する必要がある．検索リング内のノード数が大幅に減少するとノード負荷が増加することが考えられるため，検索リングの併合処理やノード数の同期処理を検討する必要がある．

7. ま と め

本稿では，Chord のリングを分割する事によって検索リング毎のノード数を減少させ，検索の高速化を実現した．検索リング毎のノード数を減少させることで P2P ネットワーク内のメッセージ量の削減も行えた．これにより，本手法は Chord と比較すると，検索・維持性能が良いと言える．しかし，分割を完了するまでのステップ数が多い，ノードの離脱を考慮していないなど検討すべき課題も存在する．

また，シミュレーションの結果より，検索の高速化（閾値）とノード負荷（メタデータ所持量）がトレードオフの関係であり，今後はメタデータ量とノード数の関係に応じた閾値を設定する機構を設計する必要があると考えられる．

謝辞

本研究の一部は，九州工業大学平成 17 年度研究戦略経費および科学研究費補助金（若手研究（B）16700072）によるものである．

参 考 文 献

[1] Sylvia Ratnasamy, et al., *A Scalable Content-Addressable Network*, ACM SIGCOMM 2001, pp.161-172, August 2001.

[2] Ion Stoica, et al., *Chord: A Scalable Peer-to-peer Lookup service for internet Applications*, ACM SIGCOMM 2001, pp.27-31, August 2001.

[3] Antony Rowstron, et al., *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*, IFIP/ACM, pp. 329-350, November 2001.

[4] Feng Hong, et al., *PChord: improvement on Chord to Achieve Better Routing Efficiency by Exploiting Proximity*, ICDCSW 2005, pp. 806-811, June 2005.

[5] Ben Leong, et al., *EpiChord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management*, ICON 2004, pp. 270-276, November 2004.

[6] PlanetSim: An Overlay Network Simulation Framework, *http://ants.etse.urv.es/planetsim/*