2007 年度　修士論文

DHT における負荷分散を目的とした複製配置手法の特性改善

早稲田大学大学院・理工学研究科・情報ネットワーク専攻

3606U053-6

高木　邦孝

指導　甲藤 二郎　教授

2008 年 2 月 4 日

指導教授印	受付印

目次

第１章　序論

１．１　はじめに

近年，xDSL(x Digital Subscriber Line)，FTTH(Fiber to The Home)，CATV(Cable TV)などのブロードバンドの普及に伴い，音声やビデオなどのマルチメディアコンテンツ増加している．

現状のネットワークでは 1 対 1 の通信を行うクライアント･サーバモデルが主流であり，単一サーバであるため，クライアント数の増加に伴いサーバの負荷が増大し，中継ネットワークでボトルネックが生じてしまう．また，サーバに障害が起きた場合，サービス全体が停止してしまう．そこで解決策に Peer-to-Peer(P2P)が挙げられる．P2P ではピアと呼ばれる端末が，サーバとクライアントの両方の機能を持ちホスト上でパケットの複製・配信が行うことができる．これにより，サーバの負荷分散，帯域の有効利用が可能になる．そしてスケーラビリティや耐故障性の面でも優れている．また，P2P は検索方法の違いからクライアント・サーバモデルを融合させサーバを使用する Hybrid P2P と完全な分散環境である Pure P2P に分類できる．そしてさらに Pure P2P はルーティング方法の違いから Unstructured P2P と StructuredP2P に分類することができる．

本稿では Structured P2P の一つである分散ハッシュテーブル(DHT)に着目する．DHT では，システム共通の 1 つのハッシュ関数を用意し，ノードとコンテンツに一意の ID が割り当てられネットワークを構築する．また DHT を用いることにより検索対象のコンテンツを保有しているノードを確実に発見することが可能である．そして，非常に高速な検索を行うことができる．

１．２　研究目的

ネットワーク内における Web サービスにおいて, アクセス数が大きく異なることが確認されており，一部の人気のあるサービスにアクセスが集中してしまう．DHT ネットワークにおいても同様のことがいえ，人気のあるコンテンツを保有している特定ノードに大きな負荷がかかる問題点が発生している．本研究ではこの問題点を解決し，より公平な DHT ネットワークを構築することを目的とする．

１．３　本論文の構成

本論文では，第 2 章において研究背景として，クライアント・サーバモデル，P2P，DHT

について述べ，第 3 章では複製配置，アクセス分散に関する従来研究，第 4 章では DHT の課題を解決する提案手法について，第 5 章では提案手法の評価，そして第 6 章で本論文の総括を行い，今後の検討課題について述べる．

第２章　研究背景

２．１　クライアント・サーバモデル

現在，インターネット上のサービスモデルの主流となっているクライアント・サーバモデルではサーバとクライアントが１対１のやり取りを行うため，クライアント数の増加に伴い，サーバの負荷が増えていく．このようにサーバとクライアントの機能は明確に区別されており，ほとんどの場合，サーバの数はクライアントの数に対して圧倒的に少ない．また，近年では大容量のマルチメディアコンテンツが増加してきており，サーバにかかる負荷が更に増している．そのため，サーバを構成する計算資源，及びネットワーク資源が過負荷となり，各クライアントに提供できるサービス品質の低下を招き，利用者に多大な影響を与える可能性がる．

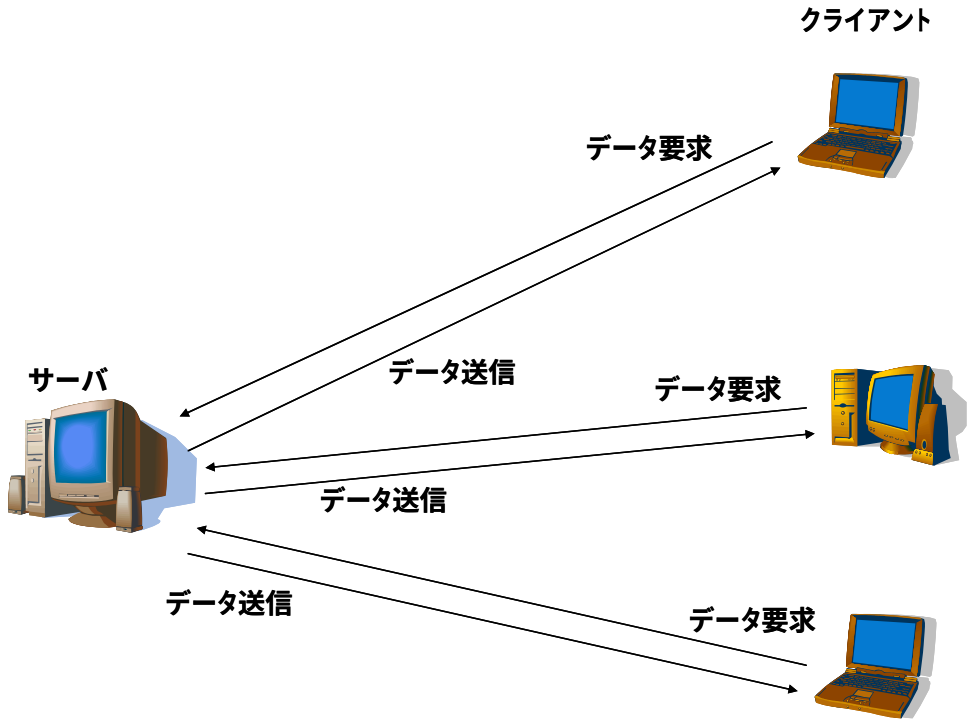


図２．１　クライアント・サーバモデル

２．２　P2P

P2P ネットワークは，クライアント・サーバモデルの問題点を解決するネットワークモデルとして注目されている．クライアント・サーバモデルのように，サービスの提供者や

利用者といった明確な区別を行わず，各端末(ピア)が両方の役割を似ない，互いにサービスを提供しあう．クライアント・サーバモデルに比べ，負荷が分散でき，スケーラビリティや耐障害性の面で優れている．

実環境における，P2P モデルを用いたネットワークサービスとしては，データ共有や掲示板，インスタントメッセージング，データ配信などがある．データ共有サービスでは，各ピアが十進のキャッシュ領域にデータを保持し，ピア同士が互いにデータを提供することで，データを共有する．

P2P モデルを用いたネットワークサービスでは，サービスの提供者が分散するため，各ピアが要求するサービスの提供者を検索する機構が必要となる．P2P モデルを基板ネットワークサービスの形態は，サービス提供者の検索機構によって，クライアント・サーバモデルを融合させた Hybrid P2P と，完全な分散環境である，Pure P2P に分類することができる．

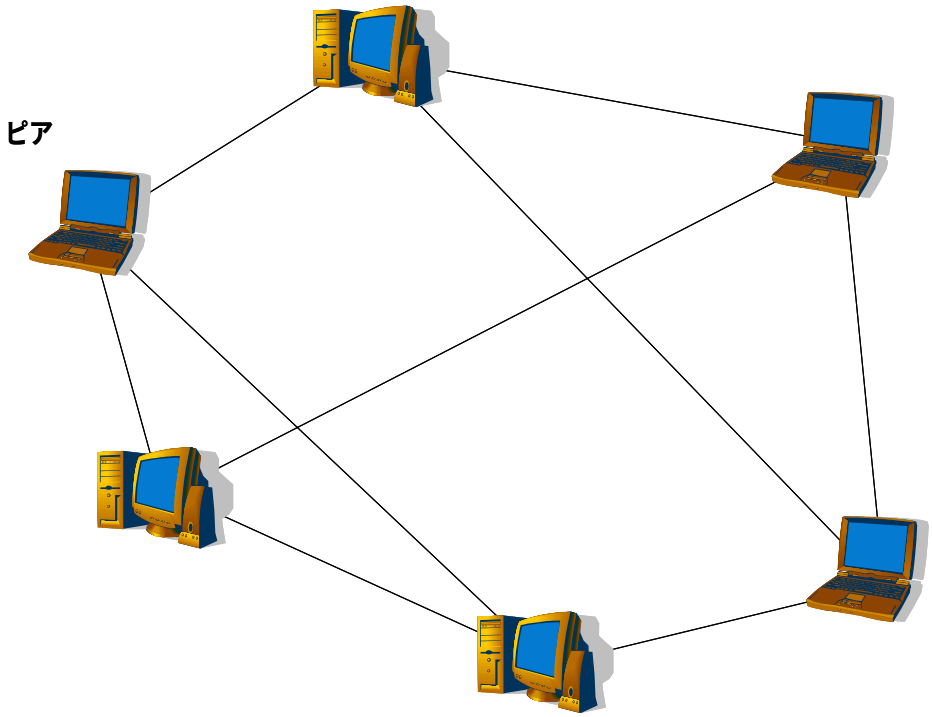


図２．２　P2P モデル

２．２．１　Hybrid P2P

Hybrid P2P では，サービス提供が可能なピアを検索するために，クライアント・サーバモデルを使用する．Hybrid P2P 型システムを適用したアプリケーションとして Napster[1]，KaZaA[3]などがある．この P2P システムでは，検索サービスを提供するサーバが，ネットワーク上の全ピアの識別子(IP アドレスなど)や，それらのピアが提供可能

なサービスを，インデックス情報として一括して管理する．ピアがサービスを検索する場合，サーバに問い合わせを行い，サービスの提供が可能なピアを発見する．サービスを提供するピアノ情報を得た後は，ピア同士の直接通信によってサービスの提供を受ける．このように Hybrid P2P 型システムでは，サーバに問い合わせを行うだけで，容易にサービスを提供するピアを発見できる．しかし，限られたサーバのみに検索の機能が集中しているため，サーバが故障や停止した場合，システム全体が停止してしまう．また，完全な分散環境ではないため，クライアント数に対するスケーラビリティを得ることができておらず，クライアント・サーバモデルの問題点を解決できていない．

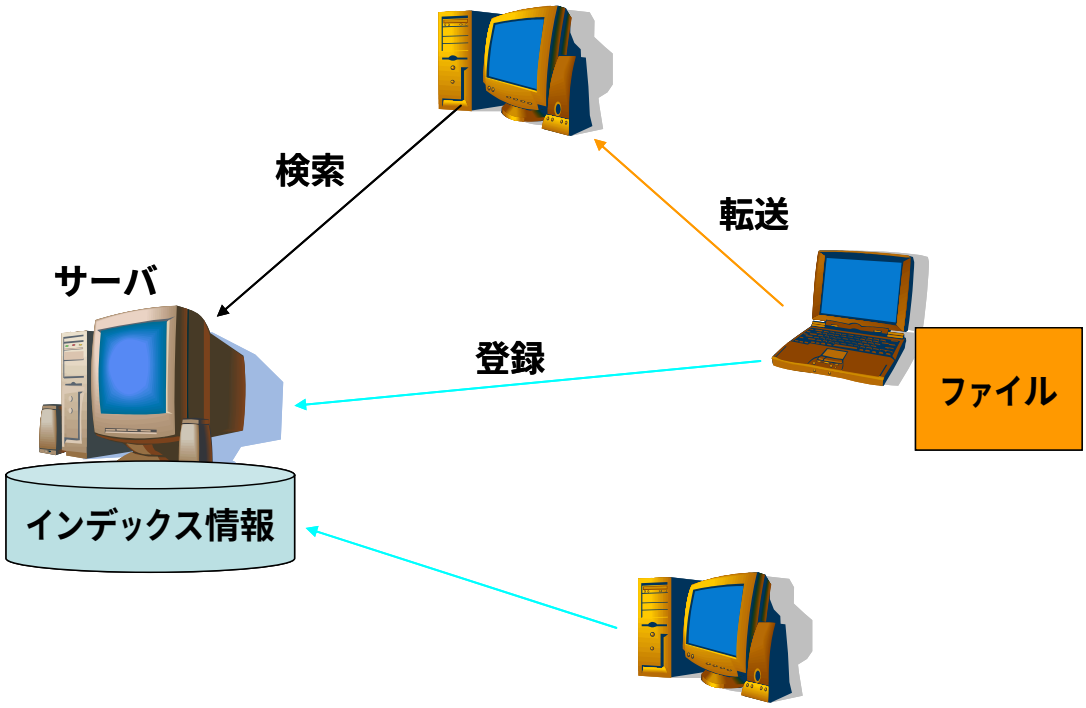


図２．３　Hybrid P2P

２．２．２　Pure P2P

Hybrid P2P 型のようにインデックス情報などを管理するサーバは存在せず，各ピアが自立分散的に動作し，サービスを提供するピアの検索を行う．この P2P システムでは，各サービスを提供可能なピアに関する情報の一部を，ネットワーク上の各ピアが管理する．また，各ピアはサービス要求を伝播するために，物理ネットワークとは独立した論理的な検索ネットワークを構成する．サービスの要求は，検索クエリを倫理ネットワーク上で隣接する他ピアへ送信する動作を，要求するサービスを提供できるピアを発見するまで繰り返すことによって実現される．そしてシステム全体が完全な分散システムとなるため，クライアント数に対するスケーラビリティや高い耐障害性を実現することができる．また，

Pure P2P は Unstructured(非構造化) P2P と Structured(構造化) P2P に分類することができる.

２．２．２．１　Unstructured P2P

Unstructured P2P 型のシステムを適用したアプリケーションの代表例として，Freenet[4]，Gnutella[2]，Winny などがある．誰を隣接ノードとするかなどのトポロジに制約がない．また存在するオブジェクトは発見できない場合もある．一般的に，効率は良くないが，柔軟な検索が可能．クエリのフラッディングにより，サーバ無しにデータを検索．そのため検索の際に巨大なネットワークトラフィックが発生する欠点がある．主な検索手法である Flooding と *k*-Walker Random Walk について次節で説明する．

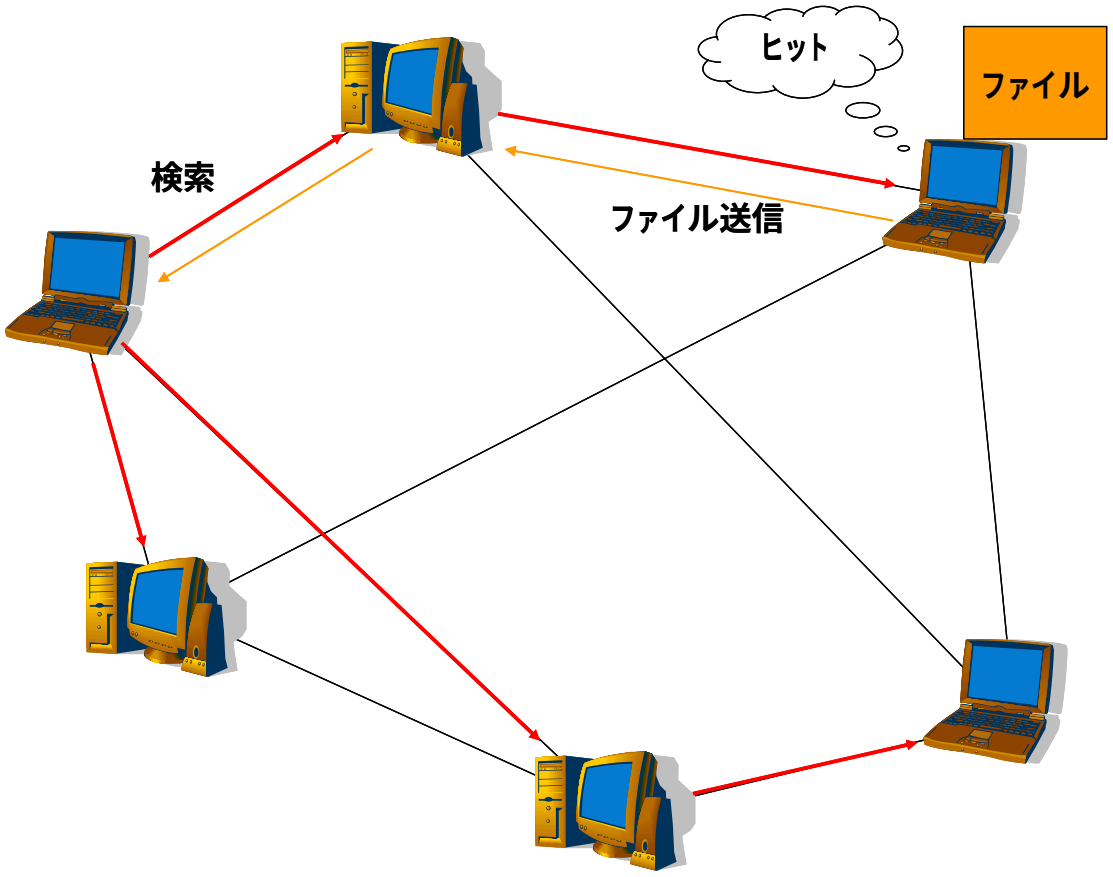


図２．４　Unstructured P2P

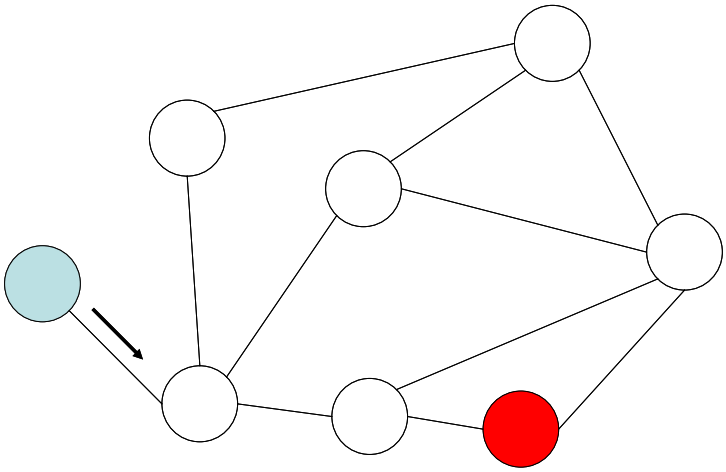
２．２．２．１．１　Flooding

Flooding[10]は，データの取得を要求するクエリを受信したノードが，そのクエリを隣接する全てのノードに転送する手法である．以下にクエリの転送手段を示す.

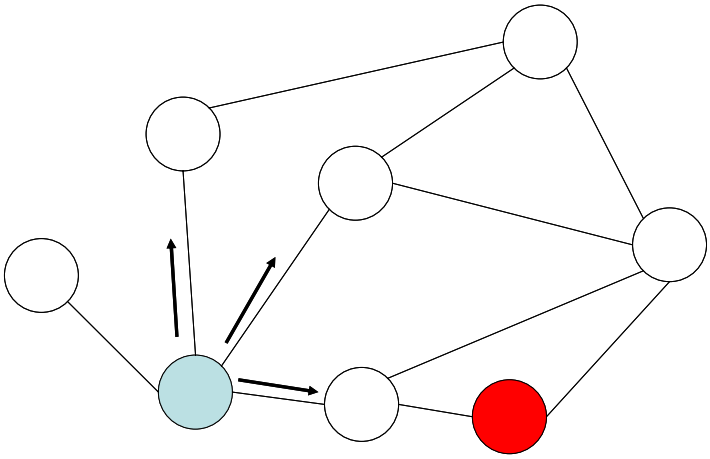
１．データを要求するノードは隣接する全てのノードに対して，クエリを送信する．
２．クエリを受け取ったノードは，次のうちいずれかの処理を行う．
・要求されているデータを保持している場合は，クエリを生成したノードに対して応答メッセージを送信する．
・要求されているデータを保持していない場合は，クエリを送信してきたノード以外の全ての隣接ノードに対して，クエリの転送を行う．
３．目的のデータが発見され，検索が成功するか，各クエリの最大転送回数を示すTTL(Time-to-Live)の値が 0 になるまで２．の処理を繰り返す．TTL 全てのクエリに設定され手織り, ノードを一つ通過するたびに 1 ずつ減少する. TTL が 0 になる前に，データが発見されれば検索が成功する．TTL が 0 になった場合には，検索終了となりクエリは破棄される．

Flooding を用いた場合の検索手順を図２．５に示す．まず図２．５(a)のように，検索を要求するノードは全ての隣接ノードにクエリを送信する．クエリを受け取ったノードは，検索対象となっているデータを保持していない場合，図２．５(b)のようにクエリを転送していたノード以外の全ての隣接ノードにクエリを転送する．図２．６(c)では，(b)と同様にクエリの転送を行っており，ここで送信された 5 つのクエリのうち一つが検索対象のデータを保持しているノードに送信され，3 ホップ目で検索が成功している．これ以降，検索に成功していなりクエリの TTL 値が 0 になるか，検索対象のデータを保持するノードを発見するまで繰り返し転送される．

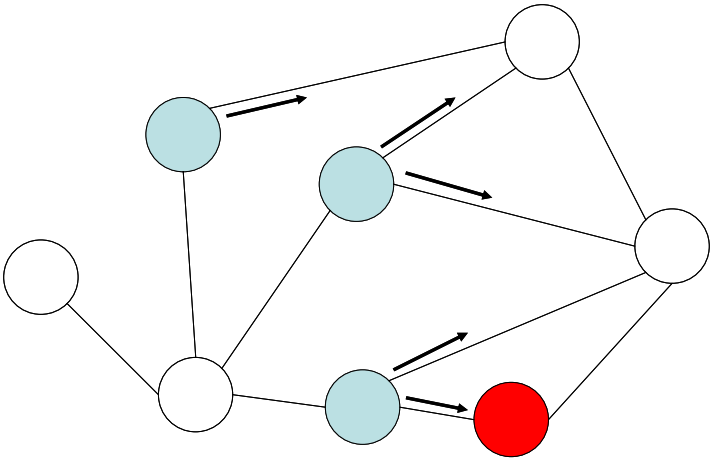
Flooding による検索では，TTL は検索の成功率に直接関わる値である．TTL 値を高い値に設定すると, 多くのノードにクエリが広まるため, 検索の成功率は高くなる. しかし，クエリがネットワーク内に大量に発生する上に，一つのクエリが検索に成功しても，他のクエリは転送され続けるため，ネットワーク内のトラフィック量が急激に増加する．そのため，P2P ネットワークのトポロジや，データの分散状況を考慮した上で，適切に TTL 値を設定する必要がある．



(a)1 ホップ目



(b)2 ホップ目



(c)3 ホップ目

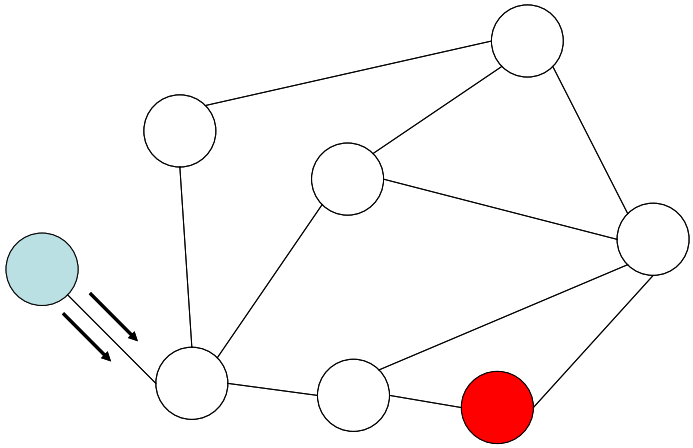
図２．５　Flooding

２．２．２．１．２　*k*-Walker Random Walk

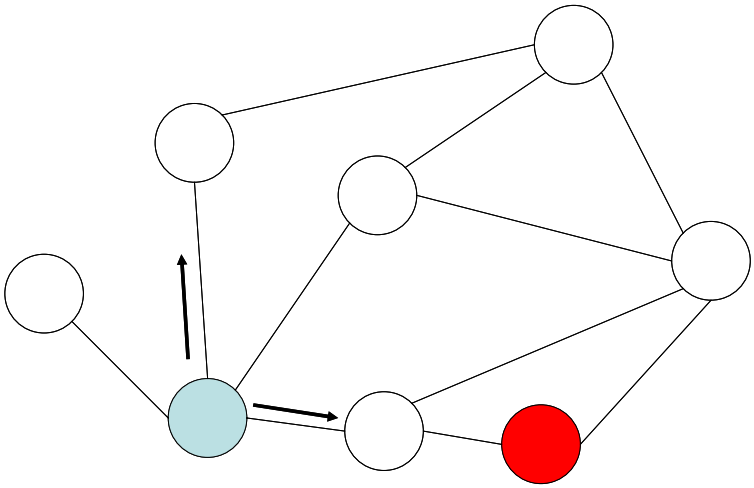
k-Walker Random Walk[10]は，Flooding のように，全ての隣接ノードにクエリを送信するのではなく，クエリ数を *k* 個に限定する．そのため，Flooding よりも P2P ネットワーク内を流れるクエリ数を大幅に減少させることができる．クエリは以下の手順で転送される．

１．データを要求するノードは，*k* 個のクエリを生成し，各クエリを隣接するノードにランダムに送信する．
２．クエリを受け取ったノードは，要求されているデータを保持していればレスポンスクエリを返し，保持していなければ隣接するノードの中から 1 台を選択しクエリを転送する．
３．２．の処理を繰り返し行い，検索が成功するか，TTL が 0 になったときに検索を終了する．

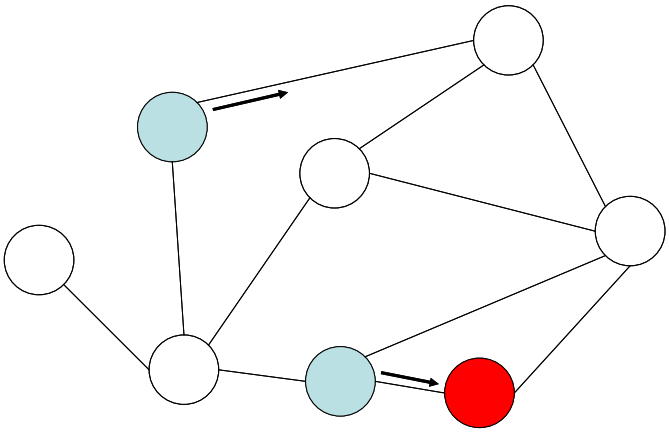
図２．６では *k*=2 の 2-Walker Random Walker により検索が行われる例である．まず，図２．６(a)では，検索要求を行うノードが隣接ノードに 2 つのクエリを送信している．この手法では図のように同じノードに重複してクエリが届くことがある．次に図２．６(b)のように，クエリを受け取ったノードは各クエリをランダムに選択された隣接ノードへ転送する．このとき，各クエリの転送先はクエリ毎に独立に決定されるため，同一のノードに 2 つのクエリが送信される可能性がある．図２．６(c)では，一つのクエリが検索対象のデータを保持するノードを発見しても検索成功となっている．この手法において，各クエリに設定されている TTL を T とすると，1 回の検索要求におけるクエリの転送回数は最大 kT 回であり，Flooding と比較して，クエリの転送にかかる処理とクエ理数を大幅に減少させることができる．一方，発生するクエリ数が少ないため，Flooding と比較して平均ホップ数は増加し，データの検索性効率は低下する問題がある．



(a)1 ホップ目



(b)2 ホップ目



(c)3 ホップ目

図２．６　k-Walker Random Walker

２．２．２．２　Structured P2P

ＤＨＴ[5,6,7,8]のネットワークがこちらに含まれる．誰を隣接ノードとするか，トポロジに制約がある．存在するオブジェクトはほぼ発見できる．一般的に効率はいいが，柔軟な検索が苦手であり基本的に全文一致検索を用いる．ノードとオブジェクトの両方にＩＤが割り当てられ，ＩＤは１６０ビットや１２８ビット用いられる場合が多い．またオブジェクトは任意の文字列，ファイル，プロセスである．ファイル名やデータそのものをハッシュすることで，得られるキーをインデックスとし，Ｐ２Ｐネットワークに参加する各ピアにそのハッシュ空間を割り当て，各ピアが自身に割り当てられた空間に対応するインデックスの管理を行うことで，リソース全体の存在の可知性や，検索の要する最大時間を保証した．このようにすることで，Unstructured　P2P の欠点であった，検索の際のネットワークトラフィックを抑えることができている．

２．２．３　Skype

2003 年に登場したＰ２Ｐを用いたインターネットを通じて無料で音声通話，チャット，ファイル転送などが可能でありながら無料ソフトである Skype について説明する．

Skype では ID を登録する際のみ Skype が用意したデータベースサーバを利用する．Skype でサーバが重要な役割を果たすのはこの場面のみで,同じ ID を登録しないために一元管理している．そしてこのサーバで IP アドレスと ID の対比をした電話帳のようなデータが作成される．しかし，このデータはユーザ数が何百万という莫大な数のために，サーバが保持するのではなく，Skype に参加しているクライアントが分散して保持している．

しかしその際に全てのクライアントにデータを細かく刻んで分散させていると効率が悪くなるので,「スーパーノード」といわれるクライアントのリーダを作る．この選ばれたスーパーノードに電話帳のデータを拡散させて, ほかのクライアントがそのデータを参照して, Skype 上でチャットや音声通信などは P2P で行う．

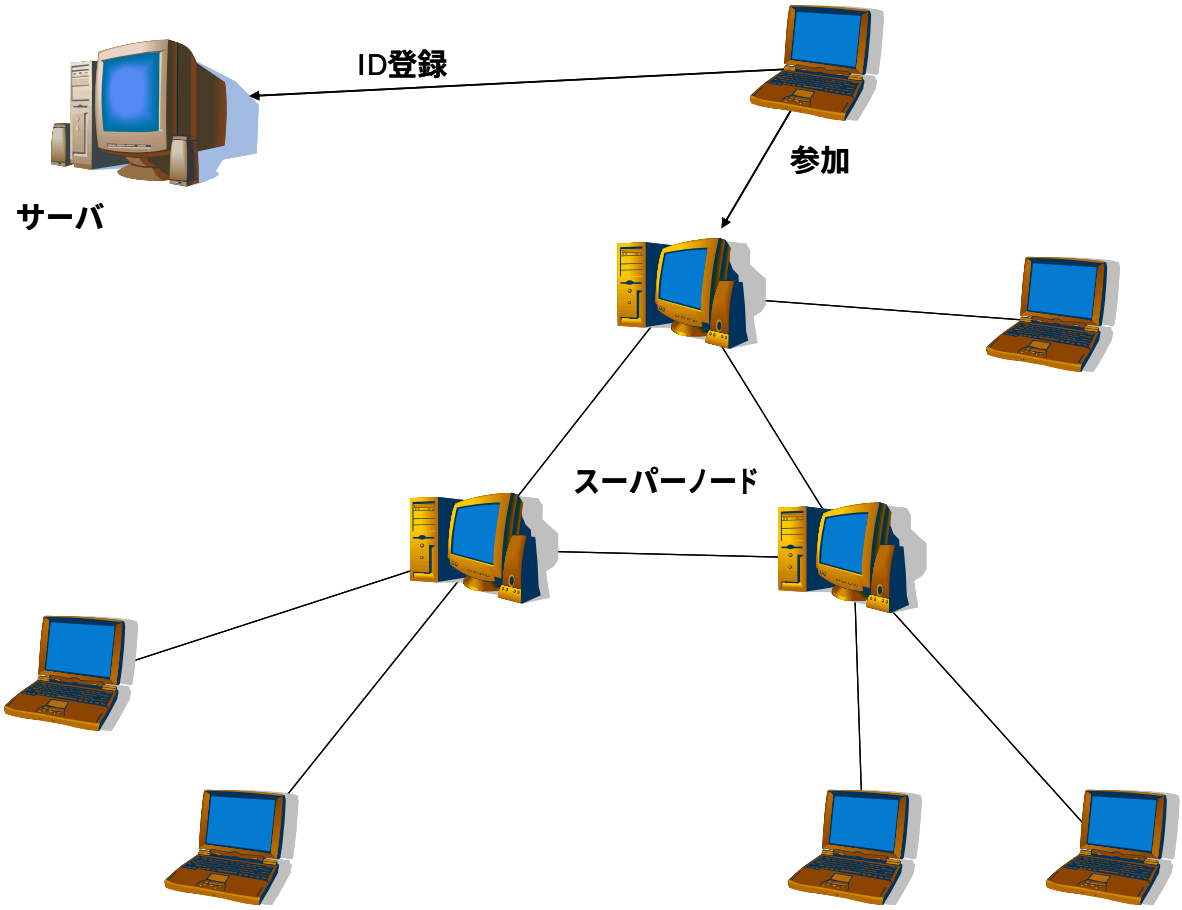


図　2．7　Skype

そして Skype では誰でも通信が可能となるようにファイアウォールや NAT が障害にならないような仕組みになっている．解決方法を以下に示す．

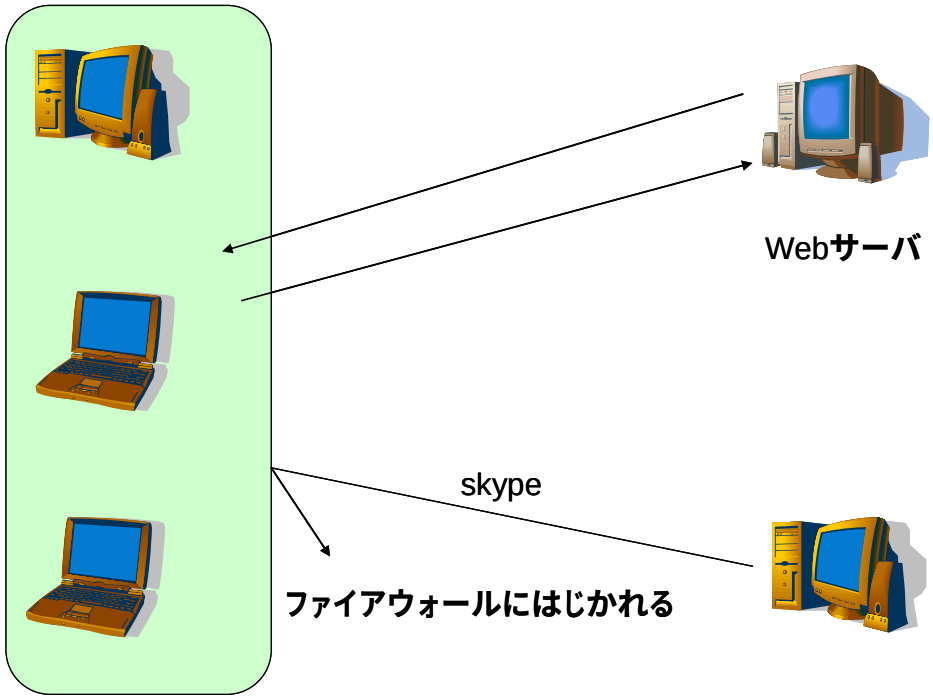


図２．８　ファイアウォール

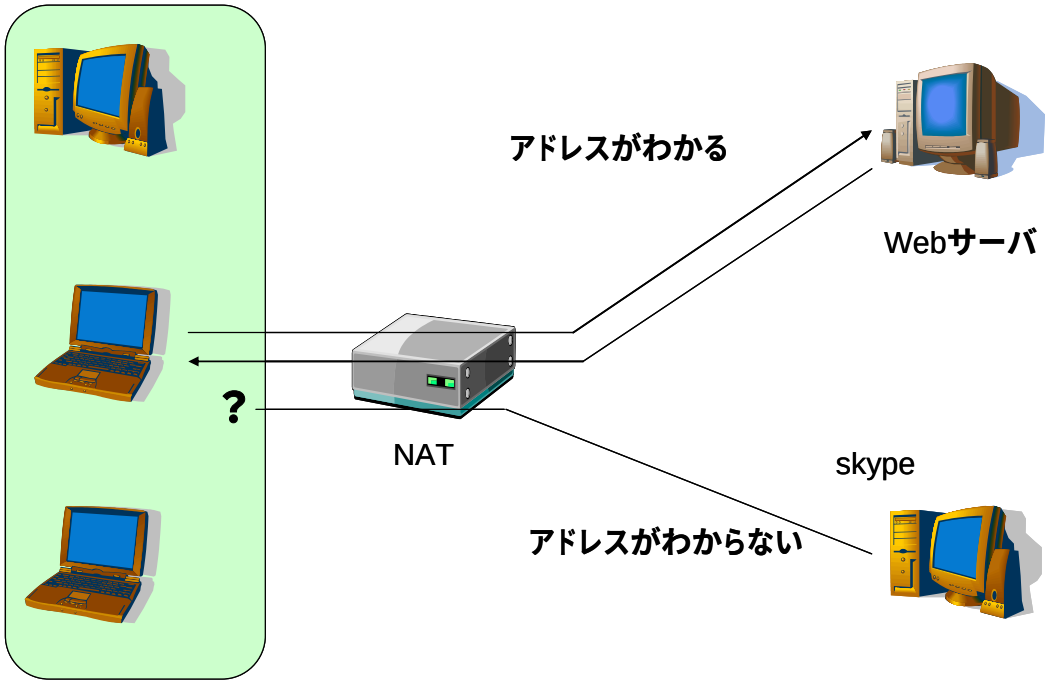


図２．９　NAT

１）片側が外部からのアクセス付加の場合

１対１で接続できない場合にはスーパーノードを使用する．外部からのアクセスを遮断

しているファイアウォール内部からの外部へのアクセスを許可するパターンが多々ある．外部にいる Skype クライアント A から内部のクライアント B へアクセスする場合，ファイアウォールや NAT が邪魔して 1 対 1 で接続ができない．そこで B と常に接続しているスーパーノードに A が B と接続したいと要求を出す．スーパーノードは B にそれを伝え，B が内部から A に接続をし，1 対 1 で接続をする．

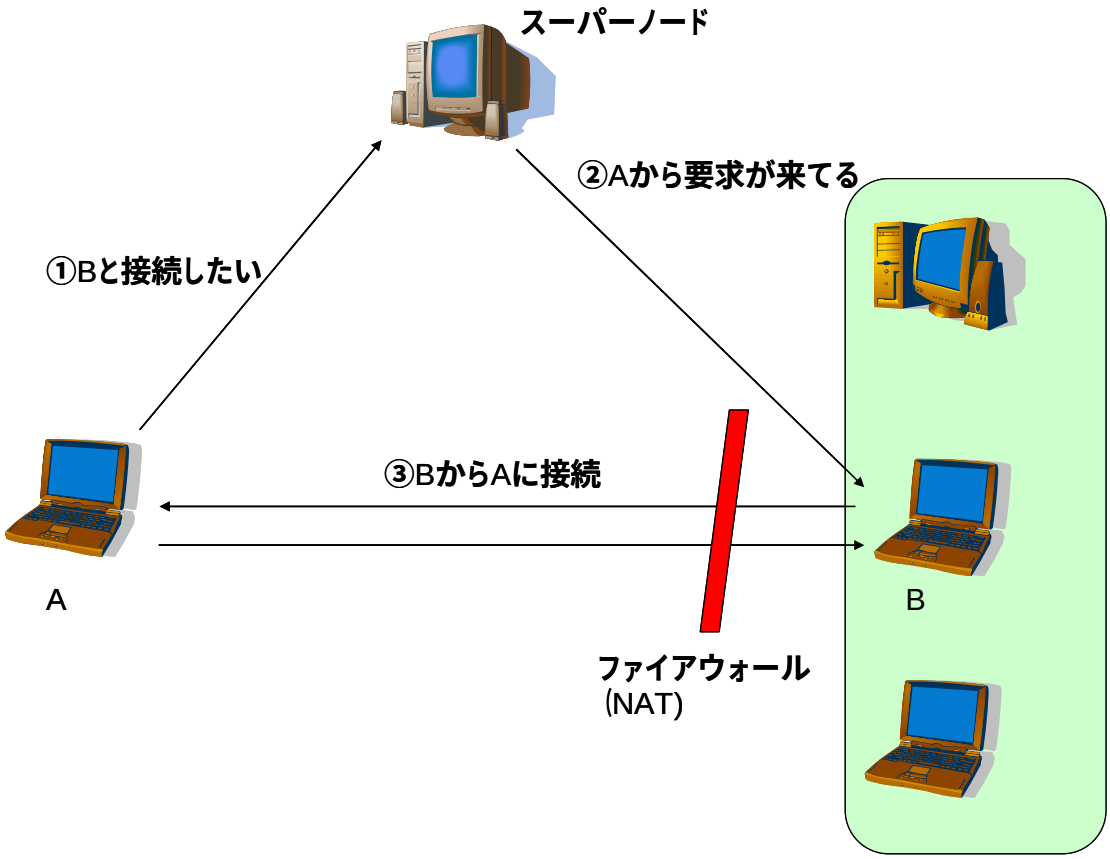


図２．１０　スーパーノードを使用

２）両側が外部からのアクセス負荷の場合

スーパーノードを使用する．それぞれのファイアウォール・NAT の内側にいるクライアントでもスーパーノードとは接続している．クライアント A と B が接続を試みる時，それぞれ同じスーパーノードに接続をし，スーパーノードに発信のタイミング管理をしてもらいながら，A と B が同時に「A から B に接続したい」，「B から A に接続したい」とそれぞれ接続パケットを送る．

ファイアウォールや NAT は内側の A から外部の B に接続要求を出したのだから B のパケットを通過させる．そのためそれぞれの接続要求がファイアウォール・NAT を超えて互いに接続をする．これを UDP hole punching と呼ぶ．

3）1，2で解決できない場合

お互いスーパーノードとは確実に接続しているので，お互いのクライアントがスーパーノードを経由してデータをやり取りする．この方法をリレーノードと呼ぶ．

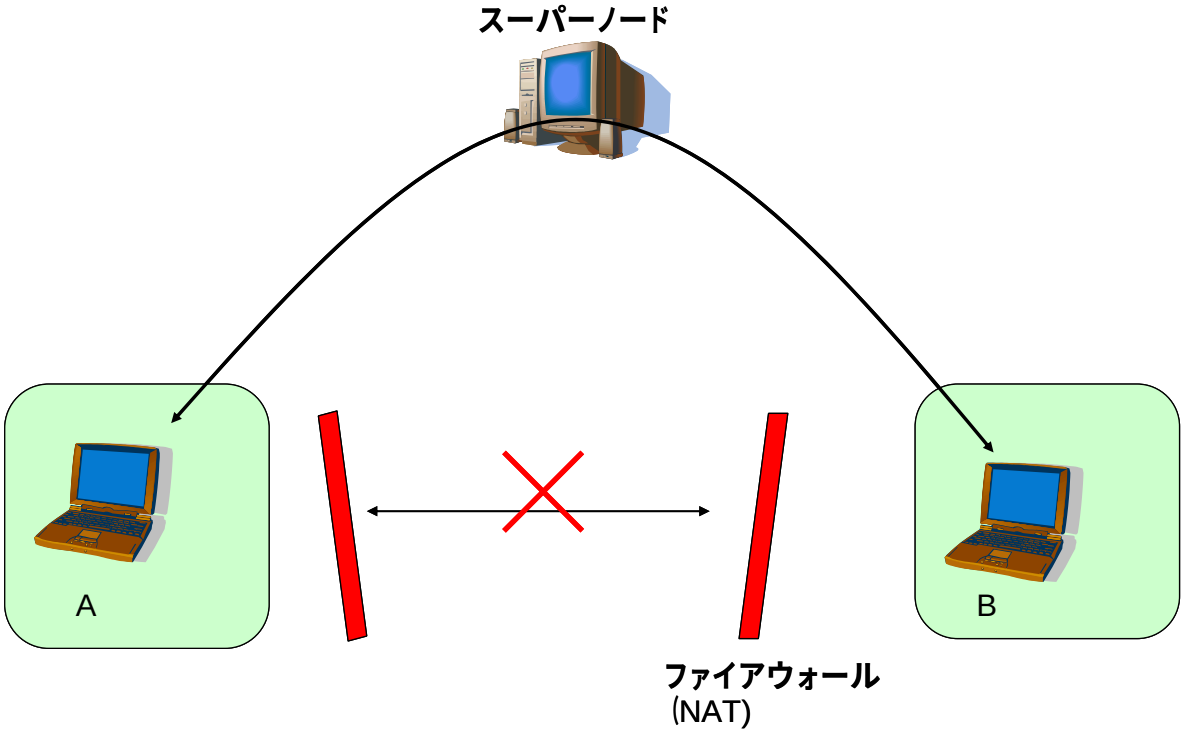


図２．１１　スーパーノードが完全中継

２．３　DHT(Distributed Hash Table)

DHT では各データに固有のハッシュ値が設定されており，このハッシュ値に対して検索が行われる．ハッシュ値とは，ある入力を特定の演算手法で変換することによって得られる固定長の擬似乱数で，ハッシュ値を得る演算手法はハッシュ関数と呼ばれる．同じ入力を同じハッシュ関数で変換すれば，常に同じハッシュ値が得られ，少しでも異なる入力を用いると全く異なるハッシュ値が得られる．DHT で使用されるハッシュ関数の出力するハッシュ値は主に 160bit であり，データ間でハッシュ値の衝突が起こらないように十分な大きさに設定されている．このようにして，ハッシュ値によってデータを一意に識別する．

DHT の基本的な考えは，ノードとデータの両方にハッシュ値を割り当て，それぞれを結びつけることにある．ノードのハッシュ値としては，そのノードの IP アドレスから生成されたハッシュ値が利用され，データのハッシュ値としては，データ名あるいはデータ

そのものから生成されたハッシュ値が利用される. そのハッシュ値を用いトポロジの構築, 検索が行われる.

DHT の代表例としては Chord, Pastry, Content-Addressable Network(CAN), Tapestry などがある．各ピアは，あるデータを検索する場合，検索対照となるデータのハッシュ値を求め，ハッシュ空間内の最も近いピアに検索クエリを転送し，データを取得する．各ノードは，このようなハッシュ値を管理し，クエリの転送先を明確に決定することにより，検索を効率化している．

・データの登録

DHT では，データを保有するノードが，データのハッシュ値とデータの組を特定のノードに登録することから動作が始まる．また一般的に，ハッシュ値とデータの組は，データのハッシュ値に最も近いハッシュ値を持つノードに格納される．このようにして，全てのデータはいずれかのノードが管理する．

・データの検索

データの検索を行うために，各ノードは他のノードへの経路表を持っている．この経路表は，ハッシュ空間全体の中で各ノードが管理する範囲に属するノードのハッシュ値と，そのノードの IP アドレスの対応表である. データの検索は, 次のような手順で行われる.

1．検索したいデータのハッシュ値を計算する.
2．自身の管理している経路表から，そのデータのハッシュ値とハッシュ空間上で最も距離が近いハッシュ値を持つノードを選択肢，クエリを転送する.
3．転送した先のノードがデータを管理していない場合，さらに近いハッシュ値を持つノードへクエリを転送する.
4．クエリを受信したノードが，要求するデータを管理している場合，データを検索要求元のノードに送信する.

3．のクエリの転送を繰り返すことで，目的データを保有するノードにたどり着く．一般的にクエリの転送ごとに 1/d(d≧2)に検索範囲が狭まる場合，検索ホップ数は $O(\log_d N)$ (N=全ノード数)に収束する．そのため，DHT では検索ホップ数のおおよその上限を推定することができる.

2．3．1　Chord

Chord[5]では円状のハッシュ空間を形成する．スキップリストという概念を使うことによって，非常に高速にオブジェクトの検索を行っている．ノード数が N の場合，検索ホッ

プ数は$O(\log N)$となる．また，ハッシュ空間を 2^{160} 使い，ハッシュ関数として SHA-1 使用．ネットワークに参加するノードは $0 \le NodeID \le 2^{160}$ を満たす一意なノード ID が割り当てられる．キーID はキーをハッシュ化することで供給される．

また，ノードがカバーするハッシュ空間は，まず successor(x)という関数を定義する．この関数はハッシュ値 x が与えられた時，値を増加させる方向で次に存在しているノードのノード ID を返す．具体的には図２．７の場合，successor (2~6)=6，successor (7~13)=13 となる．このようにハッシュ空間とノードのマッピングを行う．

・ルーティング

Chord では各ノードに successor list と finger table と呼ばれるテーブルを持たせている．successor list には，値を増加させる方向で存在しているノードを任意の数保持している．また finger table には自分の ID+2^n した ID を管理するノードを保持している．finger table の例を図２．１２に示す．そしてルーティングの際には finger table を用いて大まかに近づき，successor list を用い目的ノードを見つける．図２．１３に示す．

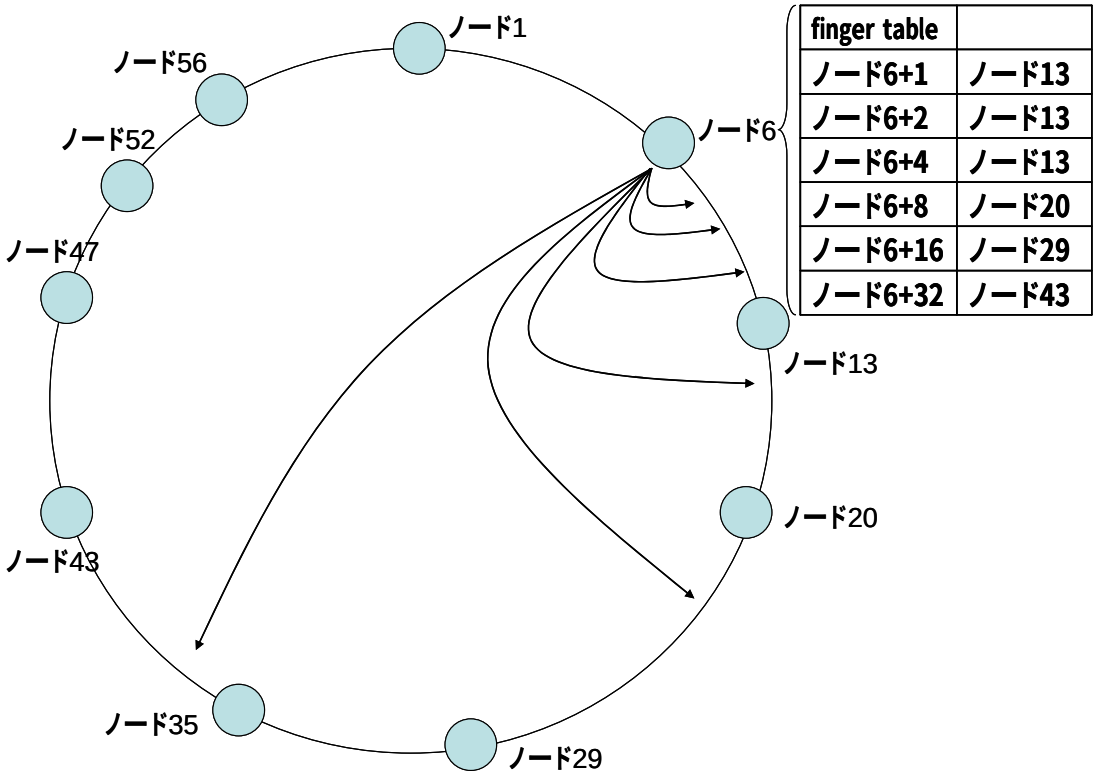


図２．１２　chord

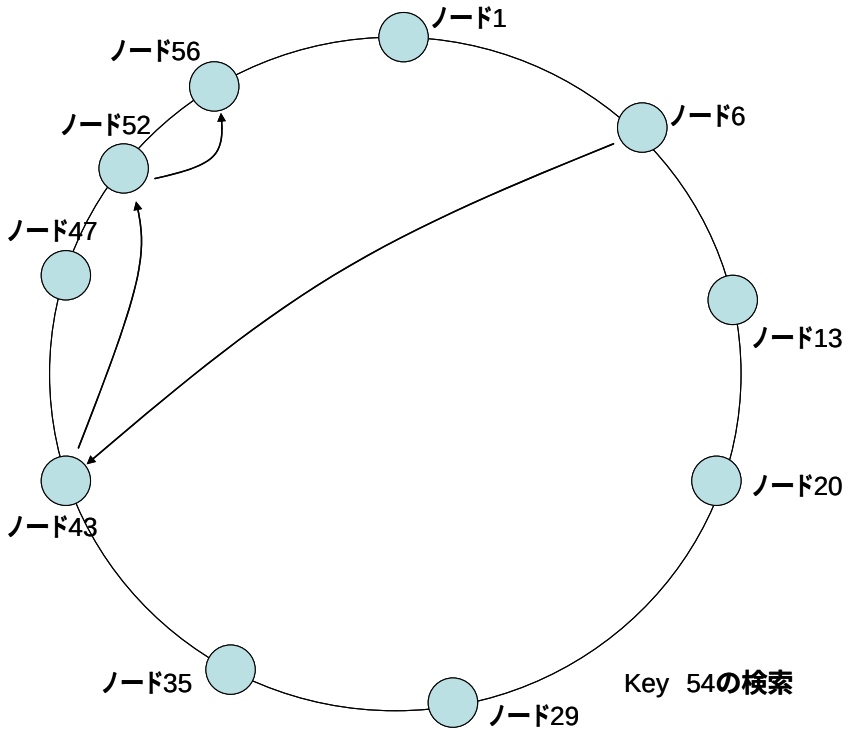


図２．１３　ノード 6 が key54 を検索

・ノードの参加

参加ノードは自ノードのハッシュキーを計算することで，Chord の円上のどこに位置すべきかが分かる．図２．９は，ID が 26 のノードが参加する場合を示している．ノード 21 とノード 32 の間に位置することが分かるので，ノード 26 がしなければいけないことは，優先順に次のようになる．

１．ノード 26 の successor list にノード 32 を追加
２．ノード 32 から，ノード 26 が保有すべきキー(とコンテンツ保有ノードのアドレス)集合を譲り受ける
３．ノード 21 の successor list にノード 26 を追加
４．ノード 26 の finger table を構築
５．ノード 26 を finger table に持つべきノード(理論上，最大 160 個)へ finger table の更新依頼

ノード 32 の探索は，ハッシュキー26 に対するキー保有ノードの探索そのものであり，successor list の更新は，近傍ノードの間の通信で済むのでコストは低い．

finger table の構築は理論的には 160 回の探索を行う必要がある．しかし，ノード 32 の finger table を参照することで，探索の多くを省略できる(全て存在するわけではないので)．

他ノードの finger table を更新することは更に面倒で, 2 の i 乗ずつ半時計まわりに遠いノードへ更新依頼を行う必要がある．更新依頼を受けたノードは，必要なら finger table を更新する．更新依頼は，半時計まわりに伝播を続ける必要がある．

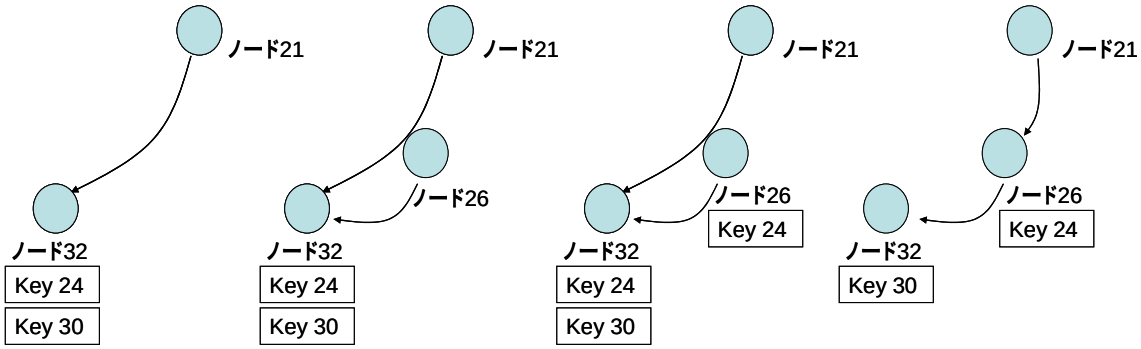


図２．１４　ノードの参加

２．３．２　Pastry

Pastry[6]ではノードに 128 ビットのノード ID が割り当てられる．ノード ID は join した時にランダムに割り当てられる(ノードの公開鍵や IP アドレスなどから暗号ハッシュを計算することにより．また，N 個のノードが存在する場合，$\log_{2^b} N$ ステップ以下で与えられたカギに数的に最も近いノードへ送ることができる．

ノード ID のキーは 2^b をベースとする数字の列である．メッセージは与えられたキーと数的に最も近いノードに送る．具体的にはそれぞれのルーティングステップで，メッセージを少なくとも 1 つの数字(b ビット)長いプレフィックスであるキーを共有するノード ID を持つノードに転送する．そのようなキーがない場合は，メッセージは現在のノードと同じ長さのキーのプレフィックスを共有し，数的には現在のノード ID のキーより近いノード ID を持つノードに転送される．

ノードは Routing table，Neighborhood set，Leaf set を維持する．まず Routing table R は, それぞれが 2^b-1 のエントリーを持つ $\log_{2^b} N$ 個の行を持つ. Routing table の 2^b-1 個のエントリーを持つ行 n は，それぞれ現在のノード ID の最初の n 桁を共有するノードを持つが，n+1 番目の桁は現在のノードの ID をのぞいて，2^b-1 通りの数字を持つノードを参照する．つまり n 行目は上から n 桁目までが同じであり，Routing table の下に行くほど, 数的に近くなる. また b の選択は Routing table の集中する部分のサイズ(約 $\lfloor \log_{2^b} N \rfloor \times (2_b - 1)$エントリー)と人為のペアのノード間を送るのに必要なホップ数の最

大の数$\left(\log_{2^b} N\right)$との間でのトレードオフを含んでいる．b 推奨値は 4 となっている．

neighborhood set M は，$|M|$ノードでローカルに最も近いノード ID と IP アドレスを含んでいる．実際には IP アドレスが近いノードが保持されている．

leaf set L は，現在のノードの ID と比較して，数的に大きく最も近いノード ID を持つ$|L|/2$のノードセットと，数的に小さく近いノード$|L|/2$のノードセットである．

図２．１５はノード ID が 12301230(base4)でシステムが 16 ビットのノード ID と b=2 を使用した場合のノードの状態を表している．

NodeID 12301230

Leaf set

SMALLER		LARGER	
12301220	12301031	12301310	12301301
12301211	12301211	12301233	12301332

Routing table

-0-2213212	1	-2-1320314	-3-1232123
1-0-112320	1-1-302231	2	1-3-321021
12-0-22310	12-1-03133	12-2-12321	3
0	123-1-2313	123-2-1102	123-3-0322
1230-0-322	1	1230-2-312	1230-3-213
12301-0-11		2	12301-3-21
123012-0-0			3
0			

Neighborhood set

13021022	21302130	03213200	13102132
22312132	21032110	11232102	21302121

図２．１５　テーブル情報

・ルーティング

最初にメッセージを渡されたノードはキーが，leaf set でカバーされているノード ID かどうかを調べる．カバーされていれば，メッセージは直接宛先のノードに渡される．そうでなければ routing table が使用される．その時，メッセージは，キーと数的に最も近いノ

ードに送られ，プレフィックスがより長く一致しているノードに転送される．

具体的に書くと，ノード A がメッセージをキーD と共に受け取った場合，leaf set を見て, キーD が leaf set の範囲内にあるかどうか判定する. なかったら routing table を見る. キーD とノード A の共通のプレフィックスの桁数を l とした場合，routing table の l 行, キーD の l 桁目の数字の列を見る. 空でなければ, そのエントリーに転送する. 空ならば, 共通のプレフィックスが少なくとも l 以上で, D と比べ自分(ノード A)よりも数字的に近いノード ID を持つノード(ノード B)へ転送を行う．このとき $|T-D|<|A-D|$ である．

・ノードの参加

新規ノードが join する時，テーブルを更新，他のノードへの通知をする必要がある．新規ノードは近く(メトリックに依存)にいるノード A について最初に知っているとした場合, 自動的に位置を決められる．そのときには IP マルチキャストや外部チャンネルを通じて得る．

新規ノードのノード ID を X とした場合，X と等しいキーと join メッセージを送るようにノード A に依頼する．ノード X と数字的に最も等しいノードをノード Z とする．

join メッセージを受け取った，ノード A,Z と A~Z の間にあるノードはノード X にテーブルを送信する．新規ノード X は送信されたテーブルを元にテーブルを以下の方法で初期化する．

ノード A は新規ノード X に近い(IP アドレス的に)と仮定しているので，ノード X の neighbor set はノード A のものを使用する．そして，Z は X に近いノード ID を持っているので，leaf set はノード Z ものを使用する．次に，ルーティングテーブルを考慮する．0 行目から始め，A と X かまったく同じプレフィックスを持っていないという，最も一般的な状況を考えている．ここで A_i を A の routing table のレベル i 行目のことを意味するとする. routing table の 0 行目のエントリーはノードのノード ID に対して孤立であるため, A_0 は X_0 のための，適切な値を含んでいる．A と X の ID は共通のプレフィックスを共有しないので，A のルーティングテーブルの別のレベルは X に使われない．しかし，A から Z までの間で最初に出会ったノード B の B_1 からは，X_1 にとって適切な値を得ることができる．次にノード C から X_2 を得ていき，Z まで繰り返すことで，routing table を作成する．

最後にノード X は作成し終えたテーブルを neighbor set，leaf set，routing table の作成に用いたノードにコピーを送る．受け取ったノードはその情報を元に自分のテーブルを更新する．

・ノードの離脱

定期的に生存メッセージをテーブル内のノードに送信する．そして leaf set のノードの

離脱の場合には leaf set 内のノードに，neighbor set のノードの場合は neighbor set のノードに，routing table のノードの場合には routing table 問い合わせ埋め合わせをする．また routing table の場合には，l 行目 d 要素のノードが離脱した場合，l 行目の他のノードに問い合わせる．そのノードの l 行目 d 要素が埋め合わせになるノードである．l 行目 d 要素が空の場合は l+1 行目のノードに問い合わせ，そのノードの l 行目 d 要素のノードで埋め合わせる．

２．３．３　CAN

CAN[7]分散ハッシュテーブルの空間として N 次トーラスが用いられる．図２．１６はノード数が 5 で，[0,1]×[0,1]の 2 次トーラスを示している．

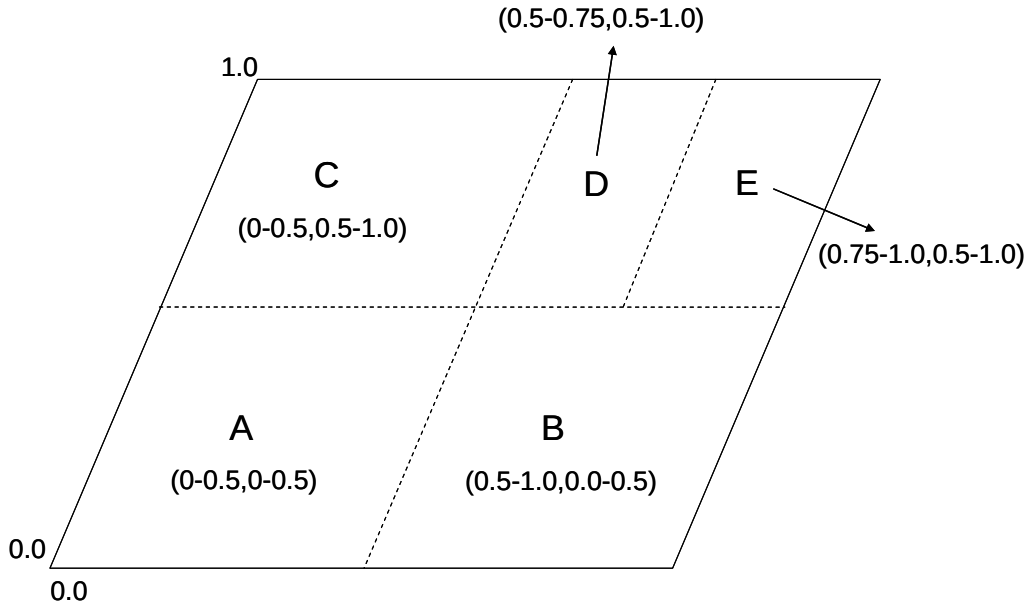


図２．１６　2 次トーラス，ノード数 5 の例

・ノードの参加

ネットワークに新たに参加するノードはランダムに(key,value)を発生し，そのエリアをカバーしているノードを探し，エリアを半分もらうように要求する．

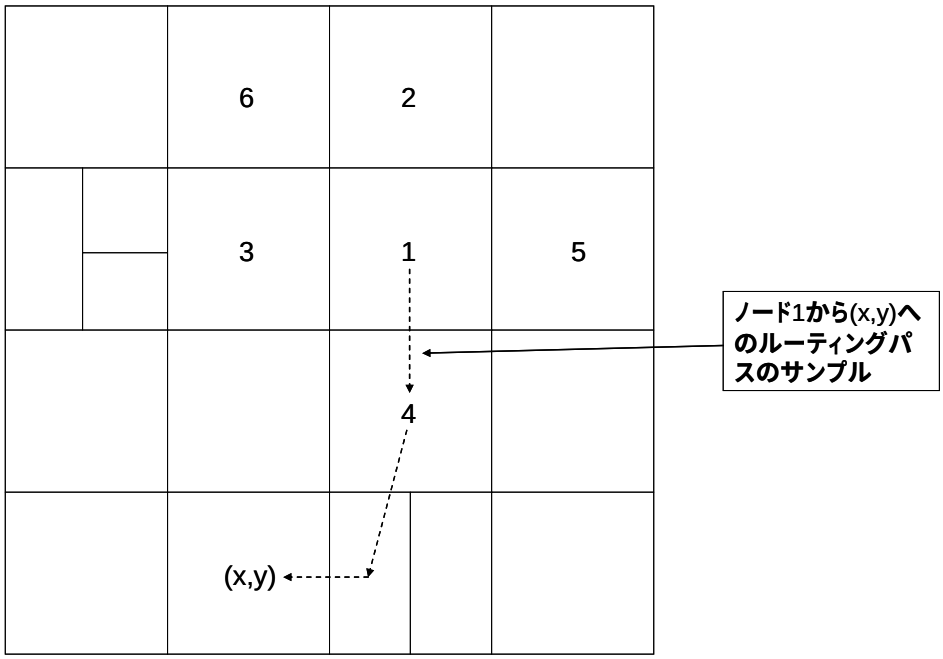
・ルーティング

隣接しているノードの IP アドレスをもとにルーティングを行っている．

・join 手順.

CAN に参加しているノードを 1 つは知っている必要がある．CAN のルーティングアルゴリズムを用い，エリアをカバーしているノードを探し，エリアを半分もらう．

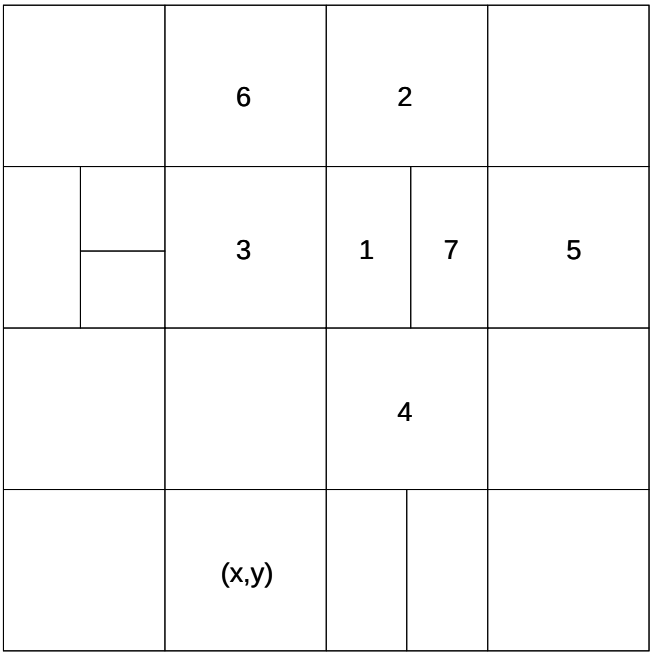
自分と隣接しているノードに参加したことを知らせる．図 2．１７では 7 が join する前，図２．１８では 7 が join した後となっている．



1 の neighbor set = {2,3,4,5}

7 の neighbor set = {}

図２．１７　ノード 7 が参加前



1 の neighbor set = {2,3,4,7}

7 の neighbor set = {1,2,4,5}

図２．１８　ノード 7 が参加後

・Leave

隣接しているノードがエリアをカバーできればそのノードがエリアをカバーする．もし出来なければ, 隣接ノードのうちカバーしているエリアが一番小さいノードがカバーする．

・Failure

隣接ノードにエリアの状況や隣接ノードのリストを送っている．長期に渡り隣接ノードからアップデートがなければ，離脱したとみなし，隣接ノードかそれぞれエリアをカバーするようにしている．またカバーしたノードは得た離脱したノードの隣接ノードに知らせる．

・ハッシュ関数

次に，ハッシュ関数を導入するには，ハッシュ関数を 2 つ用意する．それをここでは hash_x(),hash_y()とする．まず，コンテンツの名前と IP アドレスの登録は,「あ」という名前のコンテンツがある場合，(key,value)=(hash_x(あ),hash_y(あ))を含むエリアをカバーしているノードにハッシュ値と IP アドレスを渡す.

検索方法を図２．１９を用い示す．ノード A が「あ」というコンテンツを持っているノ

ードを検索したい場合, まず, hash_x(あ),hash_y(あ)を計算する. ここでは, (x,y)=(0.9,0.9)とする．そこで A は(x,y)=(0.9,0.9)をカバーするノードを探す．この時，自分は隣接ノードの情報のみ知っているので，より近いノードに検索を依頼する．この場合 D に依頼し，F に渡している．そして，F は自分のエリアに(key,value)=(0.9,0.9)があるので，{0.9,0.9,address}というテーブルから「あ」という IP アドレス(ノード B)を D に返し，A に渡す．そして，A はノード B と P2P 通信を行い,「あ」を得る.

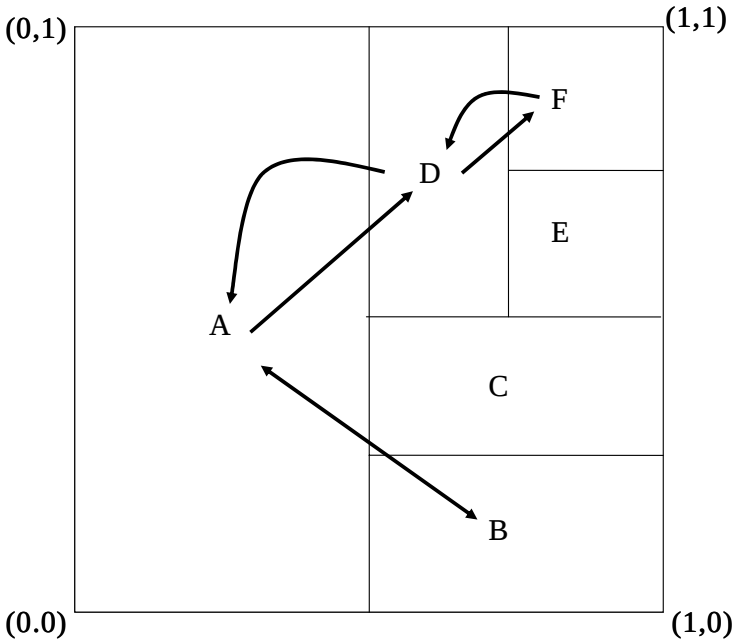


図2．１９　ルーティング

第３章　従来手法

３．１　複製配置手法

Unstructured P2P において，データの複製を配置することで検索のヒット率が上がり，効率よく検索を行うことができる．また，Structured P2P に用いることで，アクセスの分散を行うことができる．

３．１．１　Owner Replication

Owner Replication[11]は Gnutella などに用いられており，検索がヒットした時に，Requester にだけ複製を配置する方式である．検索 1 回あたりに配置される複製が高々1 個であるため，単純で，かつネットワーク負荷などのコストが最小ではあるが，複製がネットワーク内に広まるまでに十分な時間を要する．

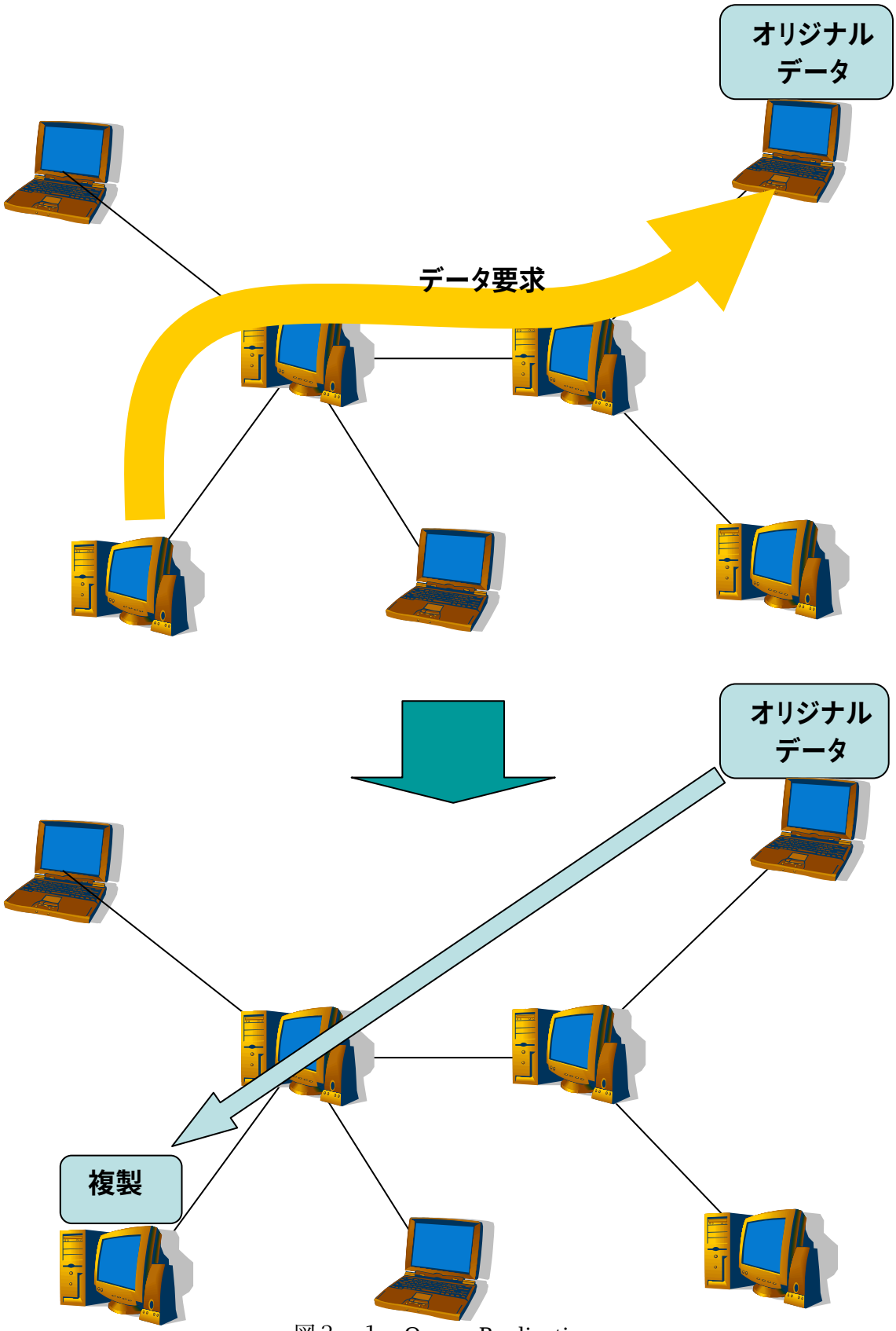


図３．１　Owner Replication

３．１．２　Path Replication

Path Replication[11]は Freenet に用いられている手法であり Requester から Holder に至る検索パス上の全てのノードに複製を配置する方式である．1 度に複数の複製が配置されるため，コンテンツは広まりやすいが，必要なストレージやネットワーク資源が大きくなる．

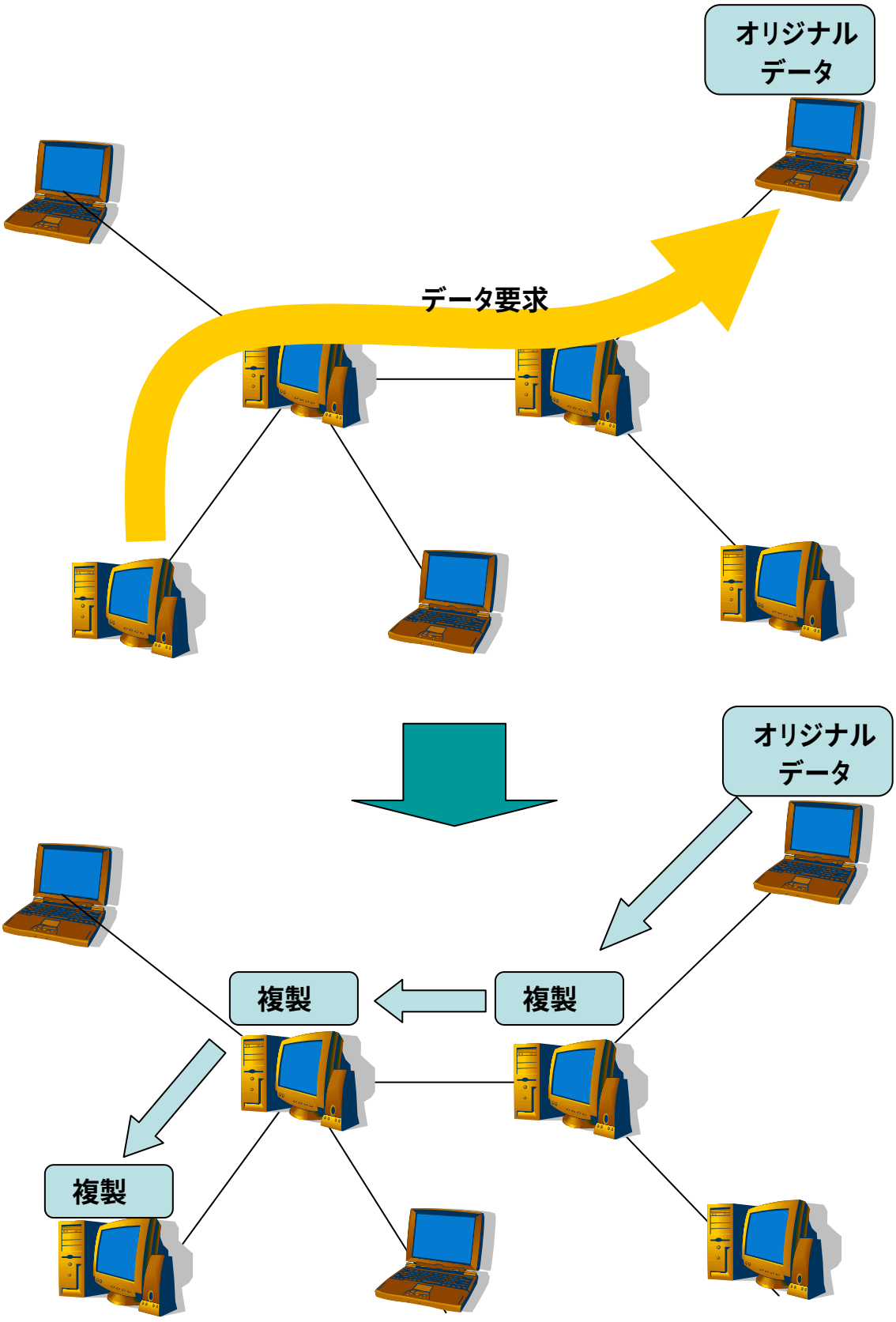


図３．２　Path Replication

３．１．３　Square-root Replication

Cohen らは，文献[9] において，非構造型トポロジを用いた P2P ネットワークを想定し，ネットワーク全体に配置すべき各データの複製数として，ランダム配置モデル，比例配置モデル，平方根配置モデルについて議論している．

・ランダム配置モデル: ネットワーク全体に，等しい数の複製をランダムに配置したモデル.
・比例配置モデル: ネットワーク全体に，データのアクセス頻度に比例する数の複製を配置したモデル.
・平方根配置モデル: ネットワーク全体に，データのアクセス頻度の平方根に比例する数の複製を配置したモデル.

さらに，一度の検索でクエリが伝搬する範囲を平均探索サイズ（ESS）と定義し，ネットワーク全体における各データの複製数の比を，データに対するアクセス確率の平方根の比と等しくする平方根配置モデルにより，ESS が最も小さくなることを証明している．つまり，複製数の比を平方根配置モデルに近づけることで，ネットワークやピアの負荷を軽減することができ，検索効率も向上できるとし，Unstructured P2P における各コンテンツの複製数の理想値を解析から算出している．式(1)，(2)に示す.

$$R = \sum_{i=1}^{D} r_i \qquad (1)$$

$$r_i = R \frac{\sqrt{f_i}}{\sum_{i=1}^{D} \sqrt{f_i}} \qquad (2)$$

$$\begin{cases} D: \text{全データ数} \\ R: \text{全複製数} \\ r_i: \text{データ} i \text{の複製数} \\ f_i: \text{データ} i \text{への検索要求発生率} \end{cases}$$

３．２　負荷分散

３．２．１　保持コンテンツ数の公平性

DHT における負荷分散目的の手法として文献[14]があり, Chord においてノードごとに保有するコンテンツ数の不公平性に着目している.

chord で負荷分散を実現するためにはルーティングテーブル$(O(\log N))$の$(\Omega(\log N))$倍必要である．この増加はネットワークへの負荷の増大に繋がってしまう．P2P において最も制約の厳しい資源はネットワーク帯域であり，トポロジ維持コストはなるべく抑えなくてはならない.

この論文では successor list を使用して負荷分散を実現している. 各ホストは$(O(\log N))$のルーティングエントリを保持すればよく，(1-1/N)の確率で各ホストの負荷は平均負荷 L_m の(1+ε)倍よりも小さくなる(ε:実数). また successor list のサイズを$(\Omega(\log N))$にすることによって，ε は任意に小さな値を取ることができる．また，Chord 以外にも適応可能.

chord において各ホストでは ID 空間内で predecessor ノードと自ノードに挟まれている領域に存在するオブジェクトを管理する．十分に多いオブジェクトがランダムに ID 空間上に配置された場合，各ホストが保持するべきオブジェクト数は円弧の長さに比例する．そして，この分布は幾何学に従う．そのため負荷が最も高いホストの負荷は高い確率で平均負荷 L_m の$\Theta(\log N)$ 倍になる. そのため高負荷のホストでも処理できるようにするためには平均負荷の$\Theta(\log N)$ 倍の CPU 及びストレージ容量を用意する必要があり無駄が多い.

そこで virtual servers を用意することでこの問題を解決することができる. ホストを ID 空間上の一転にマッピングするのではなく，ホスト内に$(\Omega(\log N))$個の virtual servers を生成し，各 virtual server 毎に ID 空間上のランダムな一点をマッピングする．そうすることで，ホストから見ると各 virtual server 同士で分布のばらつきを相殺し，結果的にほぼ均一な数のオブジェクトが割り当てられる.

しかしこの場合，各 virtual server 毎にトポロジ維持が必要なために，$(\Omega(\log N))$倍のトポロジ維持コストが必要になる.

そこで，virtual server を使わずに負荷分散を実現する提案をしている．提案ではノードの管理する空間(円弧)を一つのノードで管理するのではなく s 個のノードで管理することで負荷分散を図っている．また successor list 内のノード同士で保有するコンテンツ数を公平にしている．図３．３に示す．しかしこの手法ではアクセス数の不公平性は解決していなく，ノードごとに負荷が異なっている.

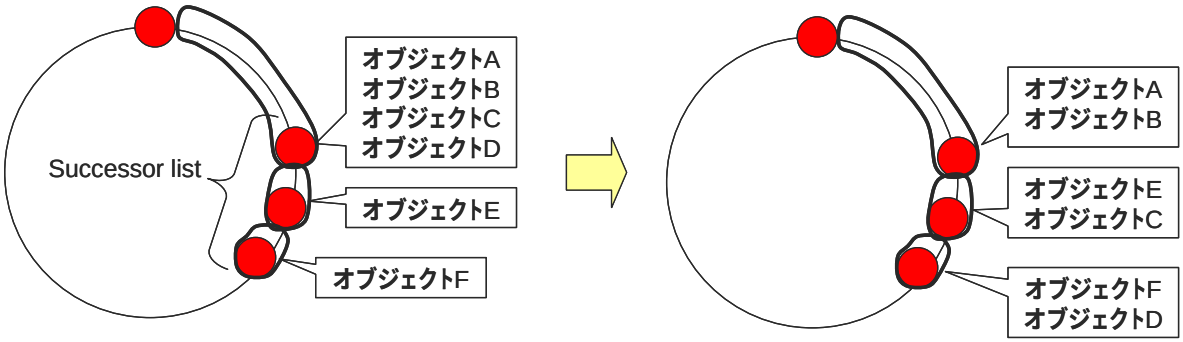


図３．３　コンテンツ分散方法

３．２．２　クラスタ化

以前に我々は地理的配置を考慮したクラスタ化を行い，各クラスタに複製を配置した[15].

・クラスタ化

クラスタ化にはメトリックに地理的な近さを用い，図に示すようにクラスタ ID を上位ビットに負荷する．図３．４に示す．そうすることで，同じクラスタ内のノードとの遅延を減らすことができる．

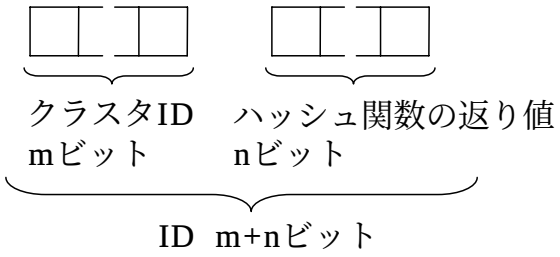


図３．４　ID 空間

・複製配置

複製からクラスタ ID を削った部分，つまり上位 m ビット削った部分に該当するノード全てに配置する. そうすることで, 複製を全てのクラスタに 1 つずつ作成できる. そして, コンテンツ取得の際には自分のクラスタ内のコンテンツを捜索することで，自分のクラスタ内のみでルーティングが行われる可能性が高くなる．もし見つけることができなければ他のクラスタを捜索する．図３．５ではクラスタ数を 4 としクラスタ ID を 00，01，10，11 としている.

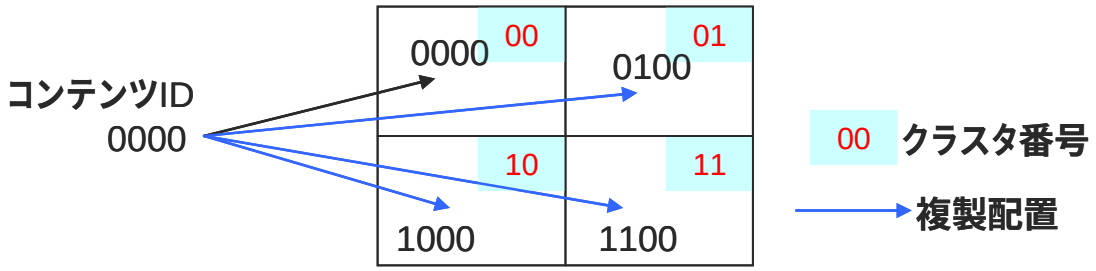


図３．５　複製場所

・評価

Overlay Weaver[16]を用いて実装を行った．比較対象として，Pastry と Chord を取り上げ，その Pastry と Chord それぞれに提案方式を適用し，コンテンツ取得を行った際のホップ数の比較結果を図３．６に示す．自分のクラスタ内のコンテンツを捜索しているために，コンテンツを持っているノードの ID が自分の ID に近くなる可能性が高いため，ホップ数は少なくなっている．

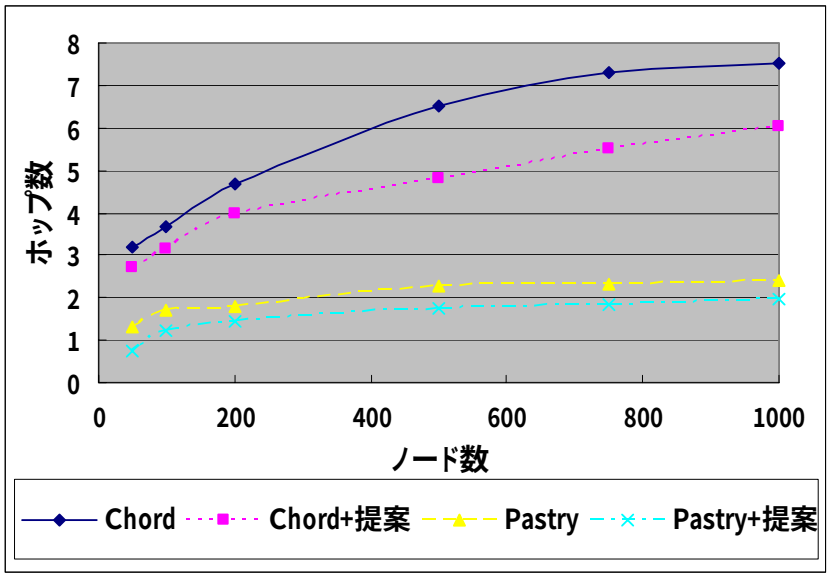


図３．６　ホップ数の比較

次に従来手法としての Pastry，Chord と，それぞれに提案を適用しコンテンツ取得を 1000 回行った時の，アクセス数の分布図を示す．またこの際のアクセスは全てランダムとしたが，従来の方でアクセス数 20 以上のノード数が増えているのは一部のノードにアクセスが集中するようにして行った結果である．提案の方ではその 20 以上アクセスのあったノードの負荷が 15 程度に収まっている．

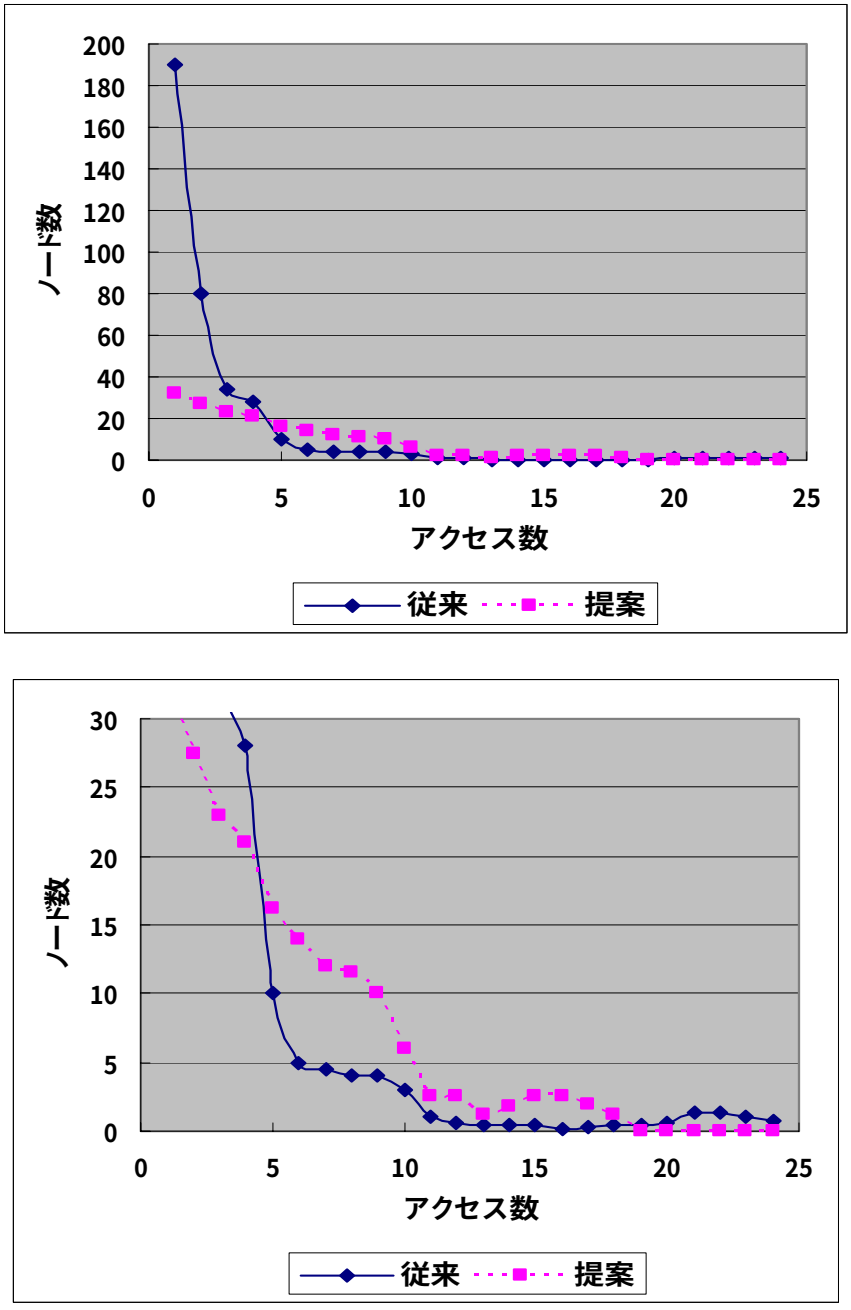


図３．７　アクセス数の分布

しかしこの手法の問題点として，強制的にクラスタ化を行い，複製を配置しているために人気がなくアクセスが来ないコンテンツも複製を配置している．そのため冗長であるといえ，各クライアントのハードディスクを圧迫する原因となっている．また，アクセスを全てランダムであるとして行ったが，アクセスは次節で説明する Zipf の法則に基づくといわれている．

３．３　Zipf の法則

一般にデータへの検索要求の発生頻度は Zipf の法則に従うことが知られており, 少数の人気のあるデータを持つノードにアクセスが集中する問題が指摘されている[13].

Zipf の法則[12]とは，サイズが k 番目に大きい要素が全体に占める割合が 1 / k に比例するという経験則で，アクセスには大きな偏りがあることが実環境では起こっていることが，HP へのアクセスランキングや，動画投稿サイトの再生数など様々な事柄に当てはまる法則．P2P を用いたコンテンツへのアクセスにも同様に適応できると考えられている．式(4)に示す．

$$q_j = \frac{j^{-\alpha}}{\sum_{m=1}^{k} m^{-\alpha}} \qquad (3)$$

k は総データ数，α はアクセス確率を決定するための Zipf 係数であり，この値が大きいほど一部のデータが頻繁に要求される．j はランク．

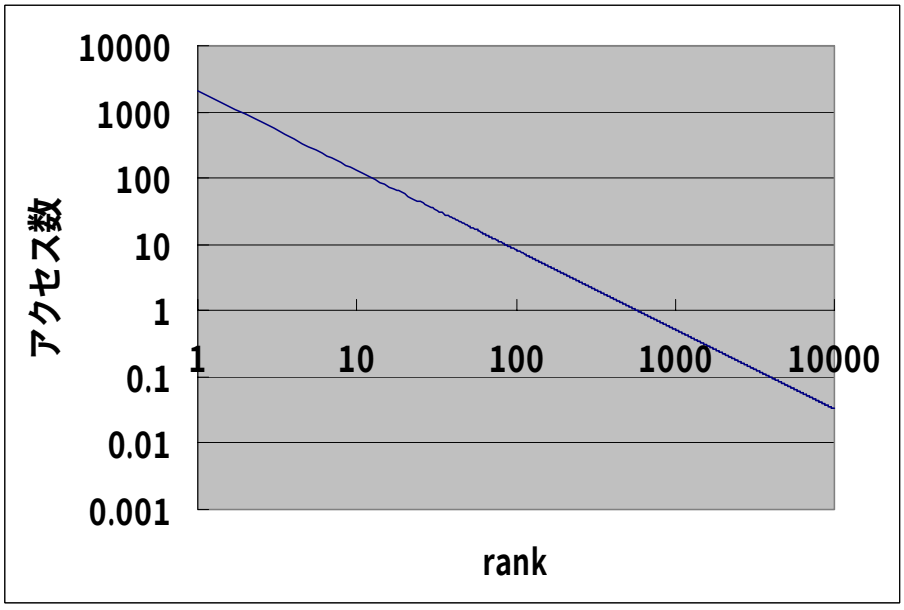


図３．８　Zipf の法則(対数グラフ)

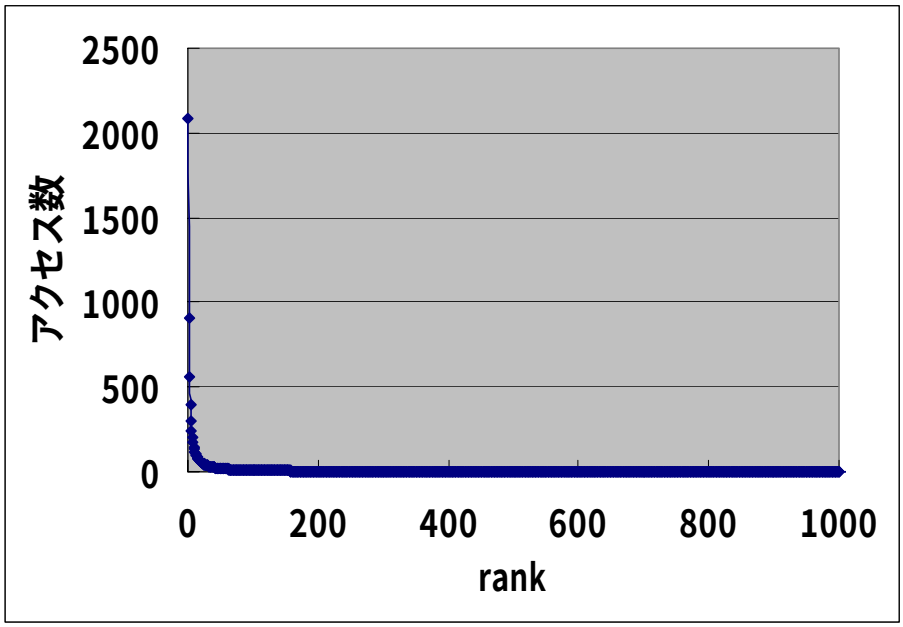


図３．９　Zipf の法則

第４章　提案

４．１　複製配置場所

本提案では finger table を用いて複製を配置していく．配置場所を図 1 に示す．最初の複製を自分の ID+2^{n-1} を管理するノード B に配置し，その次の複製を自分の ID+2^{n-2} のノード C に, その次を先ほど複製を配置したノード B の ID+2^{n-2} を管理するノード D に配置する．このように配置していくことで，複製を ID 空間全体に分布させることができる．また successor list を使用した場合に比べ，アクセスが ID 空間の一部に集中することを防ぐこともできる．

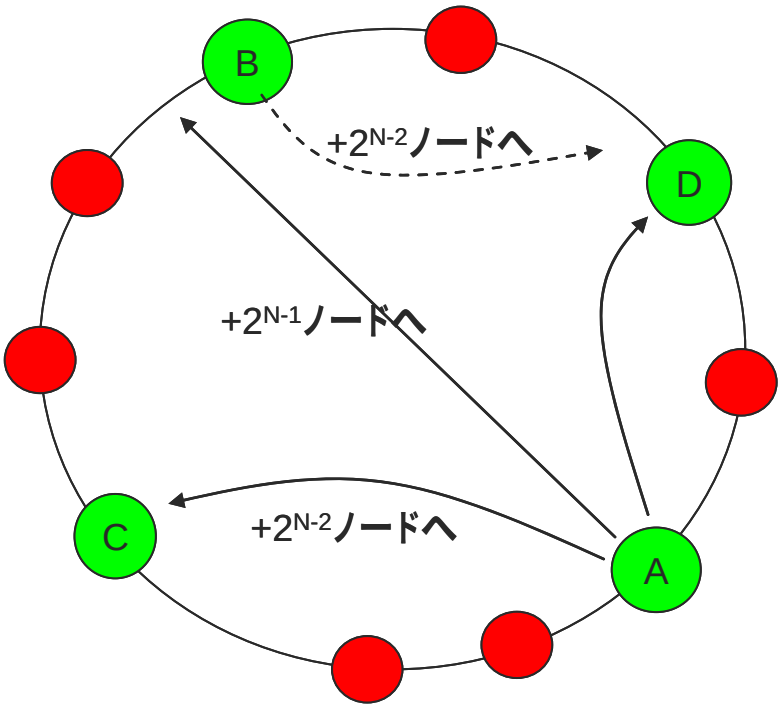


図４．１　複製配置場所

４．２　複製配置条件

オリジナルを管理するノード(図４．１ノード A)においてアクセス数が閾値を超えたら複製を配置する．アクセスの流れを表 1 に示す(閾値＝10 としている)．まず最初にオリジナルを保持するノード A にアクセスが来る．ここでアクセス数が閾値(10)を超えたら，複製

をノード B に配している．そして次にノード A とノード B のアクセス数を比較して少ない方にアクセスがいく．ここではノード B にアクセス数が 10 になるまできている．ノード A とノード B へのアクセス数が同じになったら，ノード A とノード B へのアクセスが交互にきている．そしてノード A へのアクセス数が 20 になった時にノード A がノード C に複製を配置している．このようにすることで，アクセスの公平性の実現と，flash crowd のようなアクセスの集中に対応することができる．

表４．１　アクセス推移

ノード A	ノード B	ノード C
1		
2		
・・・		
10		
ノード A がノード B に複製配置		
10	1	
10	2	
・・・	・・・	
10	9	
10	10	
11	10	
11	11	
12	11	
・・・	・・・	
20	19	
ノード A がノード C に複製配置		
20	19	1
20	19	2

４．３　検索

通常の Chord のアルゴリズムを使用し検索を行う．しかし，検索途中のパス上のノードが目的のコンテンツを保持している場合は検索を終了させる．そのようにすることで，ホップ数を減らすことができる．

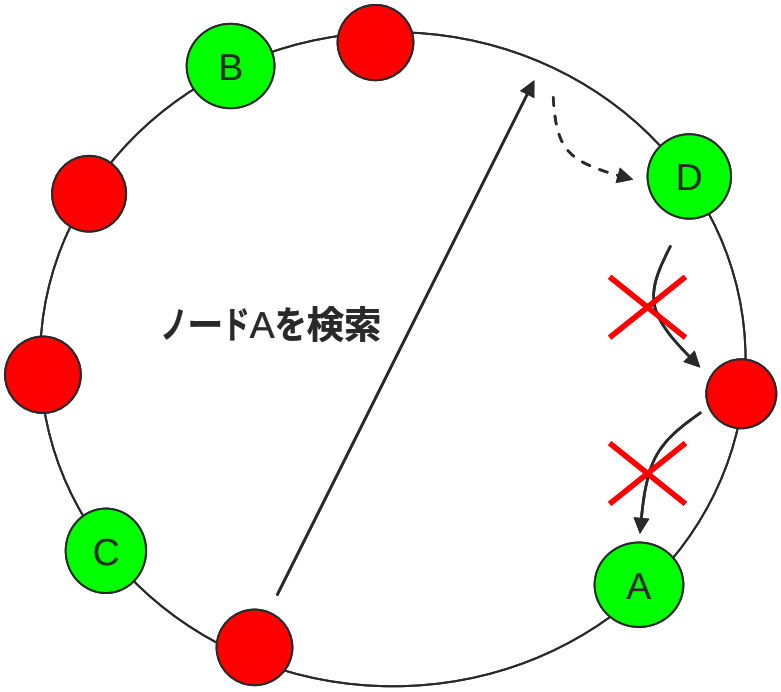


図４．２　検索

第５章　シミュレーション評価

５．１　環境

PlanetSim[2]を用い, ノード数を 1000, コンテンツ数 10000 とし, アルゴリズムに Chord を使用した. そして Zipf の法則(Zipf 係数＝1.2)[5]に基づくようにアクセスを 10000 回行った．また複製配置の際の閾値を 10 とした.

比較手法として，従来手法で述べた複製配置手法として代表的なリクエストしたノードに複製を配置する Owner Replication と, Unstructured P2P の複製数の理想数を求めることができる Square-root Replication を使用した．またこの際の Square-root Replication での各コンテンツの複製数は提案の総複製数を元に算出した．そして，Square-root Replication, Owner Replication 共に最終アクセス先をオリジナル, 複製含め公平とした.

５．２　結果

図５．１に複製数，図５．２に総複製数，図５．３にアクセス数の結果を示す．総複製数は Owner Replication が約 10000 個，提案と Square-root Replication は約 150 個である．複製数は明らかに Owner Replication が多いことが図５．２からわかる．そして複製数が多いために，アクセスが最も分散できている．ただし，ディスク資源を浪費していることは自明であり，特に高精細映像のような大容量コンテンツの蓄積・配信に課題が残る．

一方，Square-root Replication は，複製の総数は提案手法と同じであるが，人気の高いコンテンツの複製数が少ないために，Chord と同様に，特定ノードへのアクセスの偏りが観測される．

これらに対して提案方式では，ディスク資源の節約，アクセスの負荷分散共に良好な特性を実現できていることがわかる．

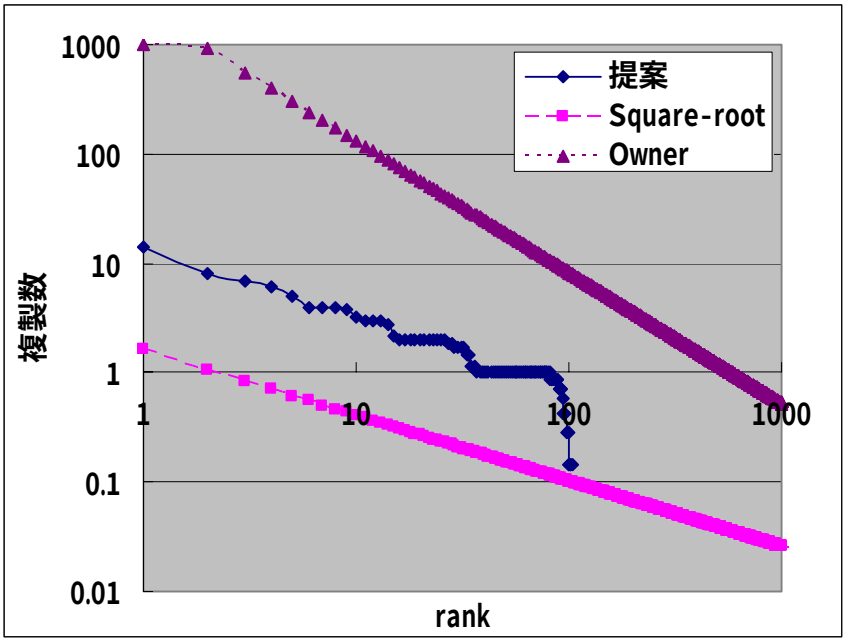


図５．１　複製数

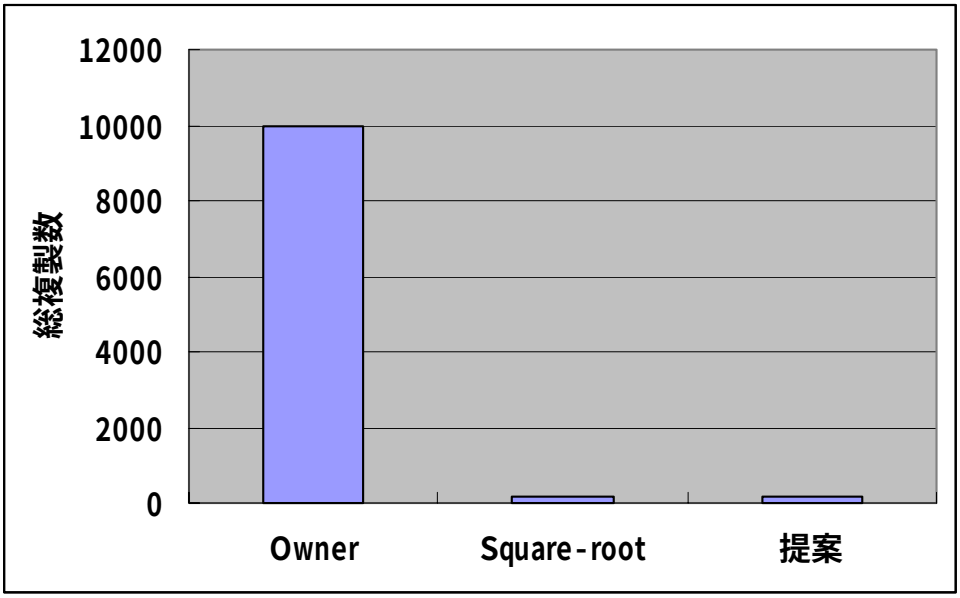


図５．２　総複製数

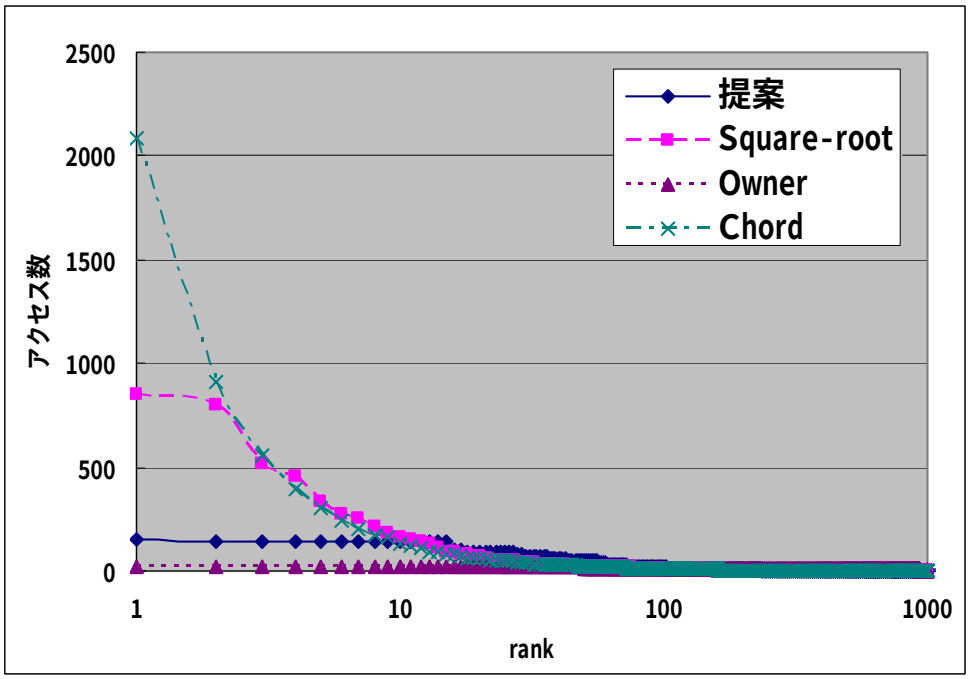


図５．３　アクセス数

次に，図５．４，５．５に複製は提案と同様に行うが，複製場所を successor list にある順番通りとした場合のノード毎のアクセス頻度の比較を示す. successor list を用いた場合，アクセスが ID 空間の一部に集中しているが，finger table を用いた場合，アクセスが ID 空間全体に分散していることがわかる.

もともと DHT では近隣ノードつまり，ID が近いノードとの通信のやり取りが非常に多い．そのため，図５．５のように ID 空間一部分に非常に大きな負荷がかかると，その部分がホットスポットとなり，それ以外のノードにも影響を及ぼし，ネットワーク全体に影響が出る可能性がある.

しかし, ID 的に分散したとしても, 物理的には分散できずに集中している可能性もある. これを解決する手法としては，３．２．２で示した図５．６のように ID と地理的な位置を関係付けることにより，物理的にも分散することができる.

最後に図５．６にホップ数を示す．途中でルーティングが終了するため，ホップ数が減少している．しかし，ルーティング途中で目的コンテンツを発見することはごく稀であるため，ホップ数の減少はわずかであった.

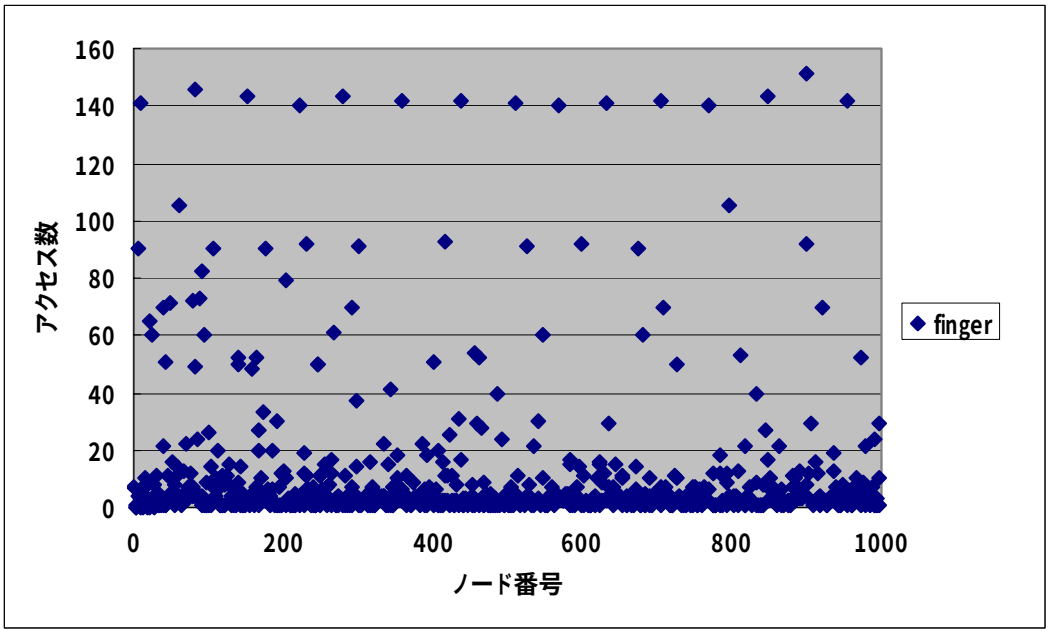


図５．４　successor list 使用時

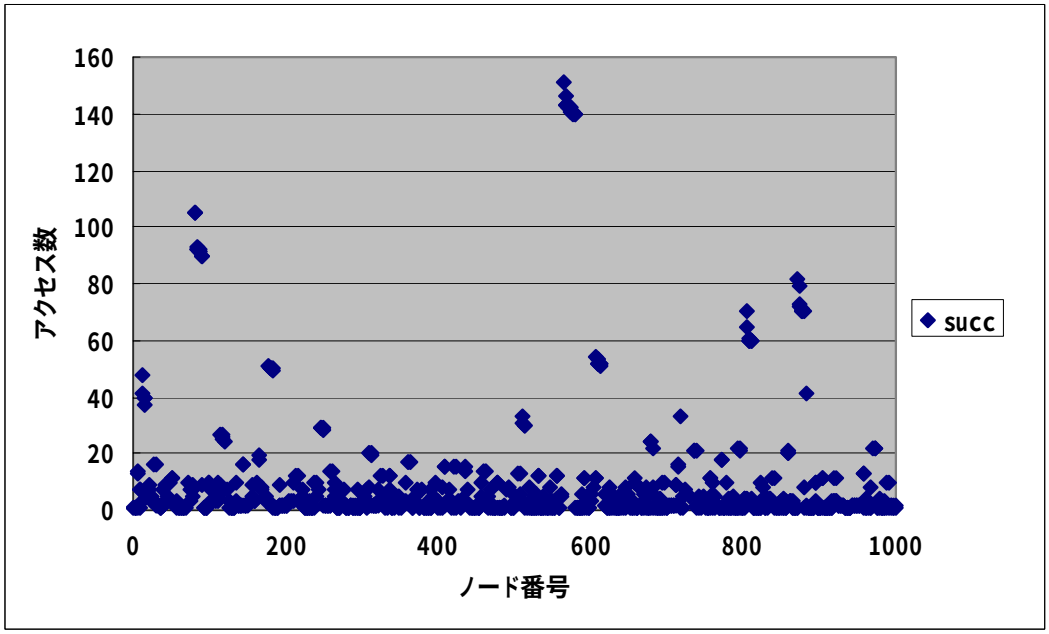


図５．５　finger table 使用時

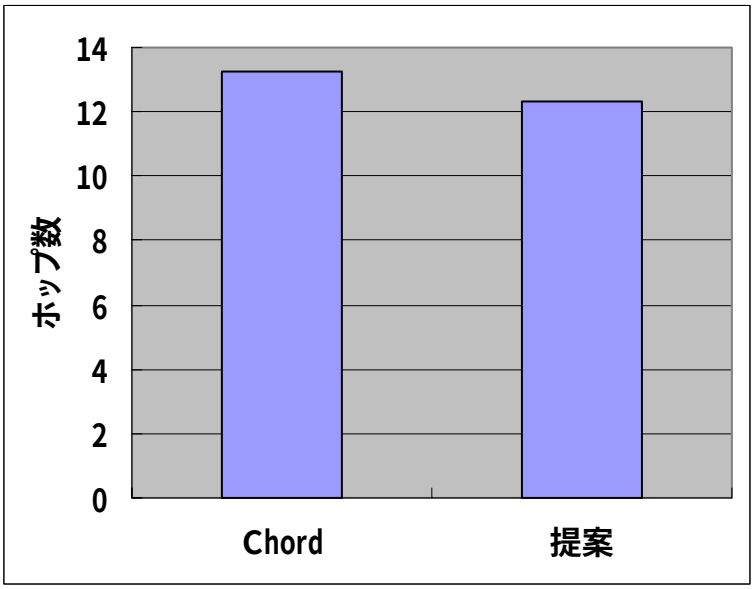


図５．６　ホップ数

第６章　まとめ

６．１　研究総括

- 研究背景

  第 2 章では本研究で取り上げた，P2P 及び DHT について述べた.

- 従来手法

  第 3 章では複製配置手法，アクセス分散の従来手法について述べた.

- 提案手法

  第 4 章では従来手法に対しての提案手法の詳細を述べた.

- 結果と評価

  第 5 章ではシミュレーションによって提案手法の有効性を示した.

- 総括

  本章において本研究のまとめを述べた.

６．２　今後の課題

本稿では DHT のネットワークにおけるアクセスの不公平さに着目し，アクセス数を元に複製を配置することで，アクセスの分散化を行った.

今後は数学的な解析を行い，さらに理想的な複製配置手法の検討や，検索方法の検討などが考えられる.

参考文献

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

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

[3]KaZaA, <URL: http://www.kazaa.com/>.

[4]I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong: “Freenet: A Distributed Anonymous Information Storage and Retrieval System,” in *Proc. ICSI Workshop Design Issues in Anonymity and Unobservability 2000*, pp. 311–320 (July 2000).

[5]Stoica, I., et al. Chord: A scalable peer-to-peer lookup service for Internet applications. In Proceedings of ACM SIGCOMM, San Diego (August 2001); www.pdos.lcs.mit.edu/chord.

[6]A. Rowstron, and P. Druschel, “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems” IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pp.329-350, Nov. 2001.

[7]Ratnasamy, S., Francis, P., Handley, M.,Karp, R., and Shenker, S. A scalable content-addressable network. In Proceedings of ACM SIGCOMM, San Diego,CA (August 2001).

[8]B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph: “Tapestry: An Infrastructure for Wide-area Fault-tolerant Location and Routing,” U. C. Berkeley Technical Report UCB//CSD-01-1141 (Apr. 2000).

[9]E. Cohen and S. Shenker: “Replication Strategies in Unstructured Peer-to-Peer Networks,” in Proc. ACM SIGCOMM 2002, pp. 177–190 (Aug. 2002).

[10]N. Bisnik, A. Abouzeid, “Modeling and Analysis of Random Walk Search Algorithms in P2P Networks,” Proc. of HOT-P2P 2005, July, 2005.

[11]Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker: “Search and Replication in Unstructured Peer-to-Peer Networks,” in Proc. Int’l Conf. on Supercomputing 2002, pp. 84–95 (Mar. 2002).

[12]G. K. Zipf: “Human Behavior and the Principle of Least Effort”, Addison-Wesley (1949).

[13]L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker: “Web Caching and Zipf-like Distributions: Evidence and Implications,” in Proc. IEEE INFOCOM 1999, pp. 126–134 (Mar. 1999).

[14]岡 敏生, 森川 博之, 青山 友紀 "分散ハッシュテーブルの軽量な負荷分散手法の検討” 電子情報通信学会技術研究報告, IN2003-189, February 2004.

[15]高木邦孝, 蘇洲, 甲藤二郎 “DHT における負荷分散を目的とした複製配置手法” 電子通信情報学会 総合大会 March 2007

[16]首藤他, “オーバレイ構築ツールキット Overlay Weaver”, 情報処理学会論文誌: Vol.47, No.SIG12 (ACS 15), pp.358-367, Sep.2006.
[17]PlanetSim “http://planet.urv.es/trac/planetsim/wiki

謝辞

本研究を行うにあたり，日ごろより適切な指導と助言を与えてくださった甲藤二郎教授に心より感謝致します.

そして，研究及び研究室の生活面でいろいろとお世話になった研究室の皆様に，心から感謝致します.

２００８年2月４日

高木　邦孝