

修士論文

# Chordにおけるchurn状態での耐故障性 の向上手法について



平成 22 年度修了  
三重大学大学院 工学研究科  
博士前期課程 情報工学専攻

岩尾 隆弘

## 目次

はじめに	1
第1章 P2P ネットワーク	2
1.1 オーバーレイネットワーク	2
1.2 P2P ネットワーク	3
1.2.1 ハイブリッド P2P ネットワーク	3
1.2.2 スーパーノード型ハイブリッド P2P ネットワーク	4
1.2.3 ピュア P2P ネットワーク	4
1.3 P2P ネットワークの抱える問題	5
第2章 分散ハッシュテーブル (DHT)	6
2.1 構造型 P2P ネットワーク	6
2.2 ハッシュ表	6
2.3 Consistent Hashing	7
2.4 分散ハッシュテーブル	8
第3章 Chord	9
3.1 Chord の概要	9
3.2 検索手続き	10
3.3 参加・脱退手続き	11
3.4 安定化手続き	12
第4章 関連研究	15
4.1 コンテンツの複製を利用した手法	15
4.2 コンテンツを定期的に再登録する手法 [9]	16
第5章 提案手法	17
5.1 コンテンツの複製を ID 空間上に等間隔に配置する手法	17

5.2 LM(ランドマーク) ノードを用いた経路表の拡張手法 . . . . .	18
第 6 章 評価実験 . . . . .	20
6.1 実験の概要 . . . . .	20
6.2 耐故障性に対する評価実験 . . . . .	20
6.2.1 実験結果 . . . . .	21
6.2.2 考察 . . . . .	25
6.3 検索にかかるホップ数に対する評価実験 . . . . .	25
6.3.1 ID 空間上の距離とホップ数の関係 . . . . .	25
6.3.2 ホップ数に対する評価実験 . . . . .	26
6.3.3 churn 状態でのホップ数に対する評価実験 . . . . .	28
6.4 評価実験のまとめ . . . . .	30
おわりに . . . . .	31
謝辞 . . . . .	32
参考文献 . . . . .	33

## はじめに

近年インターネット上での音楽や動画といった高品質なコンテンツの利用のニーズが高まってきている。その結果、クライアントサーバモデルのネットワークでは、一部のサーバや回線へのアクセスの集中によって、サーバからのレスポンスの遅延等が生じるようになってきた。そこでネットワーク上のコンピュータ（以後ノード）同士が対等な立場でサービスを提供しあう P2P (Peer-to-Peer) ネットワークが注目を浴びている。P2P ネットワークでは一部の高性能なサーバにデータを集中させるのではなく、安価なノードを複数用いて負荷を分散させることができる。その中でも古典的な P2P システムである Napster や Gnutella 等では、大規模になるほどノードやコンテンツの検索速度が低速になったり、またはネットワークにノードやコンテンツの存在の有無が正確に分からないという問題がある。

このような P2P ネットワークの問題を解決するために考えられたのが構造型 P2P ネットワークの一つである分散ハッシュテーブル (DHT) である。従来の P2P システムとは違い、ネットワークに特定のトポロジを持たせて効率の良い探索を行うことで通信量を抑えつつ、全てのノードと通信を可能にすることで上述の問題を解決している。また、ハッシュを用いることで効率的な負荷分散を行うこともできる。構造型 P2P ネットワークでは、ノードが頻繁に参加・脱退を繰り返す churn 状態でも安定してトポロジを維持する必要がある。

本研究では、DHT の一手法である Chord に着目する。Chord は DHT の中で最もアルゴリズムがシンプルであり、代表例として挙げられている。そこで Chord における churn 状態での耐故障性を向上させる手法として

- コンテンツの複製を ID 空間上に等間隔に配置する手法
- ランドマークを用いた経路表の拡張手法

を提案し、シミュレーションによる評価を行った。耐故障性については検索の成功率で表す。

本稿では、第 1 章では P2P ネットワークについて、第 2 章では分散ハッシュテーブルについて、第 3 章では Chord について、第 4 章では関連研究について、第 5 章では提案手法について、第 6 章では提案手法の有効性を示す実験について述べる。

## 第1章

# P2P ネットワーク

本章では、P2P システムにおいて通信を行うためのネットワークの構成について説明する。1.1 でオーバーレイネットワークについて述べる。1.2 で P2P ネットワークの分類について述べる。1.3 で P2P ネットワークの抱える問題とその解決策について述べる。

### 1.1 オーバーレイネットワーク

オーバーレイネットワークとは、図 1.1 のようにネットワークに参加しているホスト (以下ではノード) で、物理的なネットワークである IP ネットワークを意識せずに構築された仮想的なネットワークのことである。IP ネットワークを考慮しないことで、サービス等の目的に応じたネットワークが構成できる。

オーバーレイネットワークの主な利用形態としてはファイル共有、IP 電話、SNS、クラウドコンピューティング等が挙げられる。

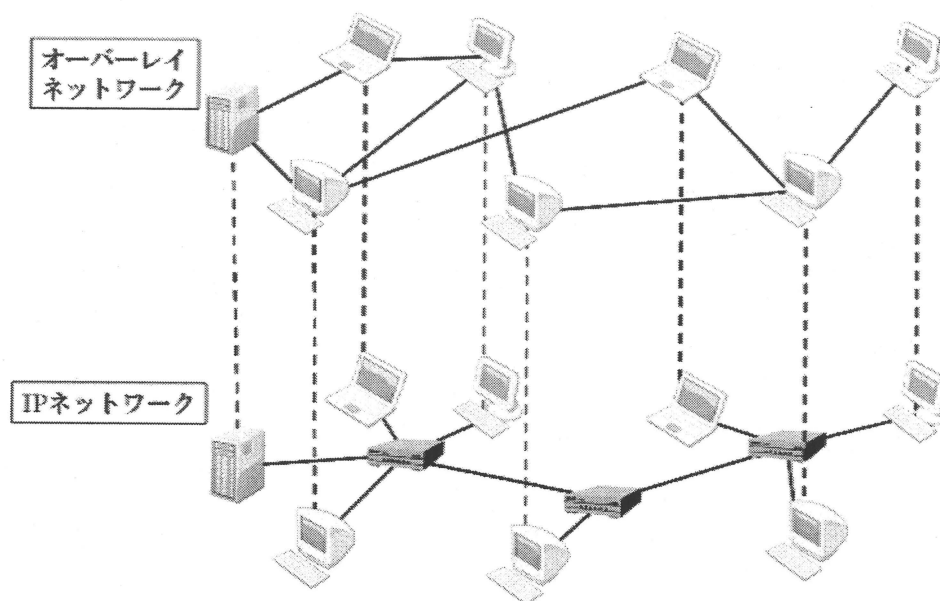


図 1.1 オーバーレイネットワーク

## 1.2 P2P ネットワーク

P2P ネットワークとは、オーバーレイネットワークの一種で、クライアントサーバモデルのような、サービスを提供する側とサービスを要求する側という形での役割分担をせずに、Peer の「対等の」という意味の通り、全ノードが対等な立場で、各ノードが時にはクライアントとして働き、時にはサーバとしても働くことで通信を行うネットワークである。

P2P ネットワークの利点として、一部のサーバや回線にアクセスが集中することでレスポンスの低下等を避けることができる。また、高価なサーバを用いずにネットワークを構築するので、コストを抑えることができる。しかし、P2P ネットワークではネットワーク全体の状況を把握することが難しいため、クライアントの情報を一元管理したり、操作を抑制するといったことができない。また、認証を必要とするサービスも難しくなる。

P2P ネットワークは、大きく分けてハイブリッド P2P ネットワーク、スーパーノード型ハイブリッド P2P ネットワーク、ピュア P2P ネットワークの 3 種類のネットワークの形式に分類される。

### 1.2.1 ハイブリッド P2P ネットワーク

ハイブリッド P2P ネットワークとは、図 1.2 に示すようにインデックスサーバと呼ばれるサーバを用いて構成される P2P ネットワークである。インデックスサーバにはユーザが登録したコンテンツのメタデータ (コンテンツの所在や名前といった情報) を持たせておくことで、コンテンツ検索の際に無駄なトラフィックを無くし、コンテンツの通信は各ノードが行うのでネットワークへの負荷が小さくなる。ただし、ハイブリッド P2P ネットワークではクライアントサーバモデルと同様にサーバが故障等でネットワークから脱退するとサービスが利用できなくなる問題がある。

このネットワークが採用された P2P アプリケーションとしてファイル交換ソフトの WinMX がある。

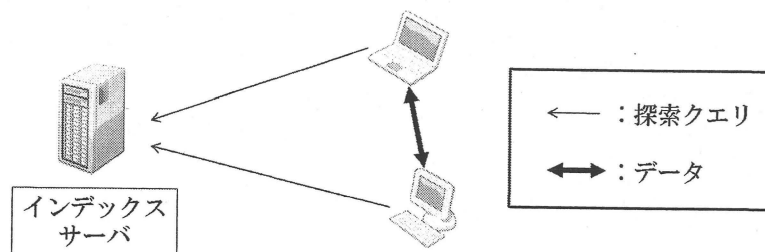


図 1.2 ハイブリッド P2P ネットワーク

### 1.2.2 スーパーノード型ハイブリッド P2P ネットワーク

スーパーノード型ハイブリッド P2P ネットワークとは、名前の通りハイブリッド P2P ネットワークの一種である。ただし、図 1.3 に示すようにハイブリッド P2P ネットワークにおけるインデックスサーバの役割を、複数の高性能なノード群 (スーパーノード群) が果たす点が大きな違いである。このネットワークでは、スーパーノードの数を動的に変化させることによって一つのノード (サーバ等) の故障によってサービスが利用できなくなることを防いでいる。

このネットワークが採用された P2P アプリケーションとして P2PIP 電話ソフトの Skype がある。

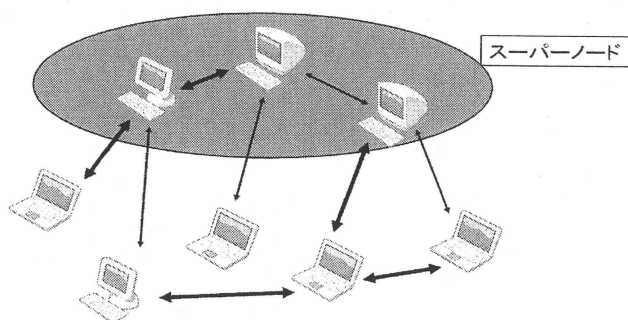


図 1.3 スーパーノード型ハイブリッド P2P ネットワーク

### 1.2.3 ピュア P2P ネットワーク

ピュア P2P ネットワークとは、図 1.4 に示すようにインデックスサーバやスーパーノード等は存在せず、全てのノードが完全に対等な立場で通信を行うネットワークである。このネットワークでは、特定のノードが故障等によりネットワークを脱退してもサービスを利用できなくなることはない。ただし、コンテンツを検索する場合等のネットワークを探索する際に、その在処がわからないので検索に時間がかかることが多くある。

このネットワークが採用された P2P アプリケーションとしてファイル交換ソフトの Winny がある。

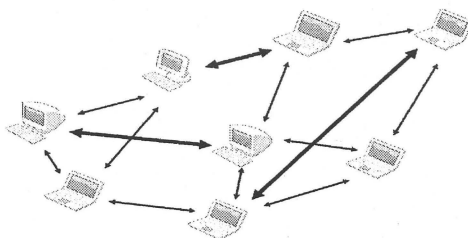


図 1.4 ピュア P2P ネットワーク

### 1.3 P2P ネットワークの抱える問題

前節で述べたような P2P ネットワークでは、探索技術において様々な問題がある。ハイブリッド P2P ネットワークとスーパーノード型 P2P ネットワークのようにサーバの一部の役割を担うノードが存在すると、そのノードが脱退してしまうとネットワーク全体が利用できなくなる。また、ピュア P2P ネットワークのようにサーバの役割を担うノードが全く存在しない場合に、探索に時間がかかってしまう問題がある。ピュア P2P ネットワークの探索技術の一般的な手法としてはフラッディングと呼ばれるバケツリレー方式がある。フラッディングとは、図 1.5 のように探索クエリを自身の全ての隣接ノードに対して送信し、受信したノードも自身宛のクエリでなければ自身の全ての隣接ノード（クエリの送信元を除く）に対して探索クエリを転送する。こうして探索クエリを伝搬させて目的のデータを探す。この探索技術では、ネットワークの規模が大きくなると探索クエリの転送回数が増えすぎてしまうことを防ぐためにクエリの転送回数 (TTL: Time to Live) や生存時間を制限しておくことがある。そのため、探索クエリがネットワーク全体に届かないことがある。例として図 1.5 ではノード A を開始ノードとしてフラッディングを行い、TTL を 3 とした場合である。もしノード B を目的ノードとした場合に、TTL によって途中でクエリが破棄され、クエリがノード B に到達していない。そしてノード A はクエリがネットワーク全体に行き渡ったわけではないので、ノード B がネットワーク上に存在するのかが正確にはわからない。

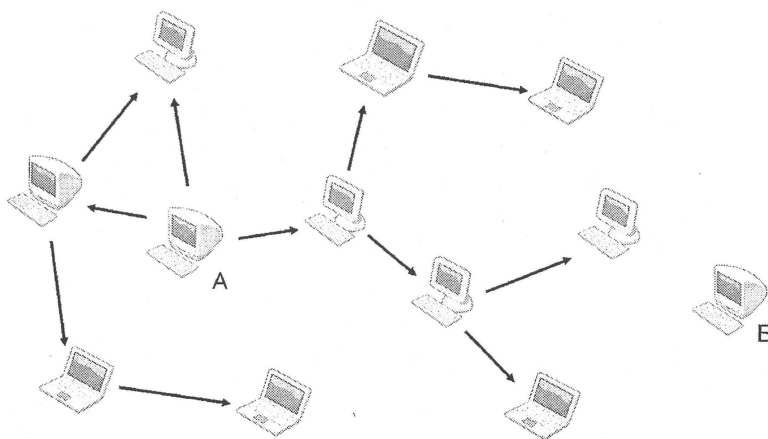


図 1.5 フラッディング

上記の問題を解決するために、ネットワークに特定のトポロジーを持たせた構造型 P2P ネットワークの研究が行われている。



## 第2章

# 分散ハッシュテーブル (DHT)

本章では構造型 P2P ネットワークの 1 つである分散ハッシュテーブルについて述べる。2.1 で構造型 P2P ネットワークについて述べる。2.2 でハッシュ表について述べる。2.3 で Consistent Hashing について述べる。2.4 で分散ハッシュテーブルについて述べる。

### 2.1 構造型 P2P ネットワーク

構造型 P2P ネットワークとは、あらかじめコンテンツを管理するノードを明確に決定しておくことや各ノードが持つ経路表 (他ノードへのリンク) が特定のトポロジを構成しているため、検索にかかるクエリのホップ数 (クエリが経由したノード数) を小さくすることができる。つまり検索するコンテンツのキーに応じてクエリを送信するノードを経路表から選ぶことで、クエリを転送する毎に検索範囲を縮めている。また、データを管理するノードが一意に決定されることから、ネットワークに検索するデータが存在しているのかが明確にわかる点も構造型 P2P ネットワークの利点である。また、このネットワークの欠点はネットワークにトポロジを与えたことにより、その維持や実装が困難になることである。

構造型 P2P ネットワークの主な例として、ハッシュ関数を利用した分散ハッシュテーブル (以下では、DHT) がある。

### 2.2 ハッシュ表

DHT の説明の前にハッシュ表について説明する。

ハッシュ表とは、表 2.1 に示すようにあるコンテンツから得られる key (名前等) と key をハッシュ関数に与えて得られる value (ハッシュ値) を格納した表である。ハッシュ関数とは、ある key を一定長のデータに変換するための関数を指す。探索において、このハッシュ表を用いることで高速に目的のキーの情報が得られる。

ハッシュ表の作成については、図 2.1 のように key をハッシュ関数に与えて得られる {key, value} のペアを表に格納する。ただし、value の取れる値が小さい時、異なる 2 つの key から同じ value を得てしまうことがある。これをハッシュの衝突という。衝突した際には、value を一つずらす等の方法が成される。

探索アルゴリズムを表 2.1 を使って説明する。プログラムでは、1 次元配列に hashtable[1]=リ

表 2.1 ハッシュ表

key	value
リンゴ	1
バナナ	2
スイカ	5

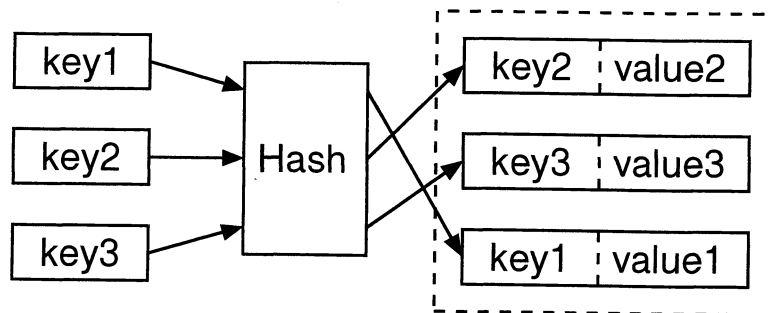


図 2.1 ハッシュ表の作成

リンゴ, hashtable[2]=バナナ, hashtable[5]=スイカとなるように格納する。ここで「スイカ」を探索する場合に、ハッシュ表が作られた時に用いたハッシュ関数と同じハッシュ関数に key「スイカ」を与え、value として 5 を得る。このハッシュ関数から得られた value を配列の添字として hashtable[5] を参照する。逐次探索である線形探索の  $O(N)$  に対して、ハッシュ探索では  $O(1)$  で済むため、高速な探索が可能となります。

## 2.3 Consistent Hashing

DHT で用いられている Constant Hashing と呼ばれる手法を説明する。この手法は、分散データベースなどで使われる技術で、コンテンツをどのノードが管理するのかを決める方法である。

ハッシュ関数を使う場合に、一般的にはノードにはそれぞれ番号を割り当て、コンテンツはその key に対する value をノード数で割った余りのノード番号のノードに管理させる。この方法でも十分な負荷分散を行うことができるがノード数が変化するに当たって、毎回コンテンツを管理するノードを変える必要がある。これをネットワーク上で行うとトラフィック量が増大するといった問題がある。

そこで Consistent Hashing では、value の値の範囲が大きくなるハッシュ関数を用いて、その value の範囲を ID 空間の大きさとする。そこで、ノードとコンテンツの両方に対して同じハッシュ関数を使ってそれぞれに id を割り当てる。コンテンツはその id から近いノードに管理され

る。例えば図 2.2 のようにノードは ID 空間上での近隣ノードとの間の id を持つコンテンツを担当管理する。

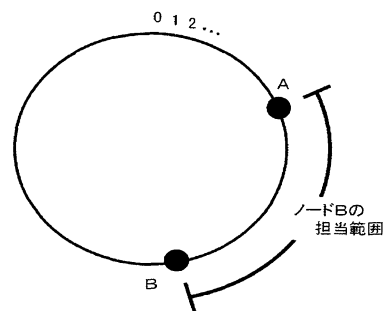


図 2.2 ハッシュ表の作成

この手法におけるノードの参加による id の管理する範囲は参加ノードの id を管理していたノードの一部を管理する。ノードの参加による他ノードへの影響が小さいことが利点である。

Consistent Hashing を用いるために使われるハッシュ関数の代表例は、SHA-1 (得られる value は  $0 \sim 2^{160} - 1$ ) や MD5 (得られる value は  $0 \sim 2^{128} - 1$ ) がある。

## 2.4 分散ハッシュテーブル

DHT とは、表 2.2 のようなハッシュ表に参加する全てのノードで分散して管理する技術である。表 2.2 はノードは IP アドレスとポート番号、コンテンツは名前等を key としてハッシュ関数より得られる value を id とした {key,value} ペアを格納したハッシュ表である。

表 2.2 ハッシュ表

key	value
133.67.35.11:1	5eff889faf52b0bd48bd88cd8469de5930201b96
コンピュータ	84a3dc42a8bc21294bb40506bdd56a716d411acb
133.67.35.11:2	9cf9fb673ee759921325db7876cad4132fa54cca

各ノードはこのような表において自身が管理すべき範囲に含まれるコンテンツとトポロジーに応じた経路表になるようなノードの情報を保持する。DHT のルーティングアルゴリズムには Chord[1], CAN[2], Tapestry[3], Kademlia[4] といった様々な手法が存在する。DHT の利点は、各コンテンツを管理するノードが一意に決定されることで、検索クエリが少ないホップ数で目的のノードに到達することができることである。

本研究では、ある id に対してその id を管理すべきノードを root ノードと呼ぶ。

## 第3章

# Chord

本章では、DHT の一手法である Chord について述べる。3.1 で Chord の概要について述べる。3.2 で検索手続きについて述べる。3.3 で参加・脱退手続きについて述べる。3.4 で安定化手続きについて述べる。

### 3.1 Chord の概要

本研究では、DHT の中で最もアルゴリズムがシンプルで実装が容易であり、DHT の代表例として挙げられる Chord に着目する。

Chord とは図 3.1 のように時計回りに ID が大きくなっていく円状の ID 空間を用意し、そこにノードを配置させて循環リストを構成するネットワークである。以後、図においてノードには N を頭文字とし、その後にそのノードの id を付けて表す。コンテンツには C を頭文字とし、その後にはそのコンテンツの id を付けて表す。

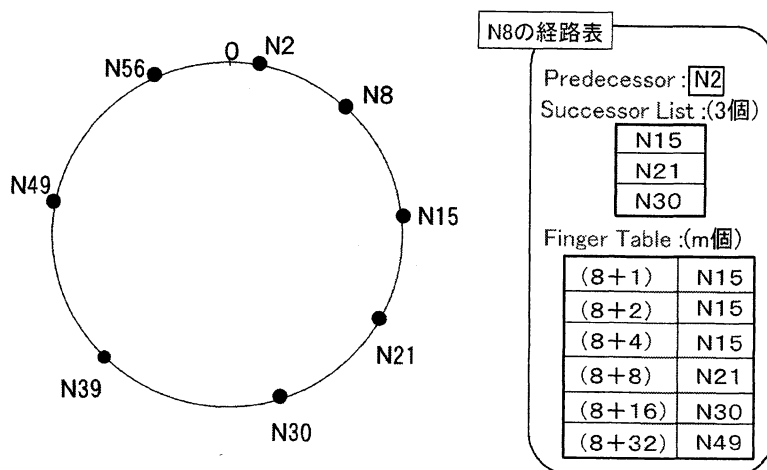


図 3.1 Chord

また、各ノードの持つ経路表では図 3.1 のように時計回りに近い数ノード (Successor List, また Successor List[0] を Successor と呼ぶ) と反時計回りに最も近いノード (Predecessor) と自身の id から  $2^k$  ( $k=0,1,2,\dots$ ) 離れた id の root ノード (Finger Table) の情報を保持する。Finger Table は自身の id から近くに存在するノードの情報を多く持ち、遠くのノードほど少ししか知らない。

探索において Finger Table を用いることで二分木探索に近いアルゴリズムになるので、少ないホップ数で目的のノードに到達できる。つまり参加ノード数  $N$  に対して、探索にかかるホップ数は  $O(\log(N))$  である。

Chord では、ノードは Predecessor から自身までの id を管理するため、コンテンツはその id から時計回りに最も近いノードに管理される。例として図 3.1 において  $id=40$  を持つコンテンツがあれば、そのコンテンツを管理すべき root ノードは  $N_{49}$  となる。

次に Chord における主な手続きである検索手続き、参加・脱退手続き、安定化手続きについて説明する。

### 3.2 検索手続き

Chord における検索では、検索をするノードやコンテンツの key から得られる value を検索する。検索アルゴリズムを以下に記す。

1. コンテンツ検索の場合に、自身が目的のコンテンツを保持していれば、検索元に通知し、終了
2. Successor が root ノードであれば Successor に root ノードであることを通知、その Successor は検索元に通知し、終了
3. 自身の経路表の Finger Table の中で、目的の value から反時計回りに最も近いノードに対してクエリを送信し、受信したノードは手順 1 から行う

手順 1 については、コンテンツ検索の際に自身がコンテンツを保持していることがあるので、その場合に検索元のノードに自身が目的のコンテンツを保持していることを通知する。後述での既存手法 (第 4 章) や提案手法 (第 5 章) でコンテンツの複製を root ノード以外に配置する場合もあるので、クエリが root ノードに到達しなくても検索が終了する場合がある。

手順 2 については、自身から Successor の間に目的の value が存在すれば、その Successor が目的の value の root ノードであるはずなので、Successor に root ノードであることを通知する。その通知を受け取ったノードは検索元に対して、ノード検索であれば自身の情報 (アドレスと id) を、コンテンツ検索であればそのコンテンツの有無を通知する。この手順で主に検索が終了する。

手順 3 については、Finger Table を用いて検索の高速化を行っている。自身から目的の value を越えるノードを Finger Table から選択してしまうと永続的にクエリが転送される可能性がある。そこで、Finger Table の中から目的の id における反時計回りに最も近いノードを選んでいる。アルゴリズムとしては遠いノードから参照し、自身の id から目的の id の間 (時計回り方向) に存在するノードが見付かり次第そのノードを選択する。

このアルゴリズムを用いた検索においてどのようにクエリが目的のノードに到達するのかを図 3.2 に示す。N2 が C53 を検索する場合である。

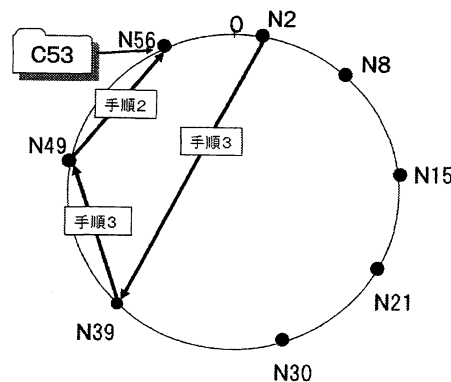


図 3.2 Chord

N2 は手順 1・2 の条件を満たさないで、手順 3 において Finger Table{N8, N15, N21, N39}の中から N39 を選択する。クエリを受信した N39 も同様に手順 1・2 と順に行うが、条件を満たさないで、手順 3 において N49 を選択する。N49 は手順 1 の条件を満たさないが、手順 2 において N49 の Successor の N56 が C53 の root ノードであることがわかるので N56 に N2 が C53 を検索していることを通知し、N56 から N2 へ C53 の存在の有無を通知することで検索を終了する。

### 3.3 参加・脱退手続き

まず参加については、参加ノードは Chord ネットワークに既に参加しているいずれかの既存ノードを知っていることを前提とする。

参加ノードは以下の手順で参加する。

1. 1つの既存ノードに自身の id を検索してもらい、その root ノードを Successor とする
2. その Successor の Predecessor を自身の Predecessor とし、さらに Predecessor に自身が新しい Successor になったことを通知する
3. Successor に自身が新しい Predecessor になったことを通知し、Successor の経路表の情報をコピーする
4. Successor が管理するコンテンツの中で自身が管理すべきコンテンツを受け取る

図 3.3 は A と C が既存ノードの一部であり、参加ノード B が参加するときの処理を表している。図 3.3(a) は手順 1 を行い、ノード B の Successor をノード C にしている。図 3.3(b) は手順 2 を

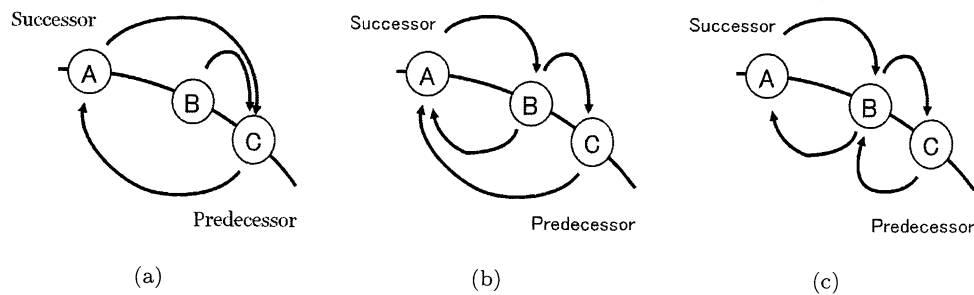


図 3.3 参加手続き

行い、ノード B の Predecessor をノード A とし、ノード A の Successor をノード B にしている。図 3.3(c) は手順 3 を行い、ノード C の Predecessor をノード B にしている。次に、ノード B はノード C から Successor List と Finger Table の情報を受け取る。Successor List は一つずらして (Successor List[0] はノード C にするため) コピーする。Finger Table は Finger Table[0] は絶対に Successor と同じになるので Finger Table[0] を除いてコピーする。最後に手順 4 でノード A からノード B の間 (時計回り方向) の id を持つコンテンツをノード C が管理していれば、そのコンテンツをノード C からノード B に渡す。

脱退については、以下の手順で行う。

1. Predecessor に自身の Successor が新しい Successor であることを通知する
2. Successor に自身の Predecessor が新しい Predecessor であることを通知し、さらに自身が管理しているコンテンツを渡す

脱退手続きでは、図 3.3(c) から脱退手続きでは、図 3.3(a) の状態にするために近隣ノードへの通信を行ってからネットワークから脱退する。

ここで、ノードが故障や悪意のある脱退で、近隣ノードへの通知を行わない場合を通知無脱退と呼ぶ。通知無脱退によって一部のノードの経路表に存在しないノード (すでに脱退してしまったノード) の情報があることでネットワークのトポロジーを維持できない場合がある。

### 3.4 安定化手続き

通知無脱退により不正確な経路表を持つノードが存在してしまうことから安定化手続きを行い経路表を更新する必要がある。例えば図 3.3(c) においてノード B が通知無脱退を行った場合に図 3.4 のように近隣ノードの経路表の一部が空になってしまう。

また、3.2 で述べた検索手続きの手順 2 から Successor へのリンクは重要であるので、Chord では Successor の 1 ノードだけではなく、Successor 候補も知っておくために Successor List を用い

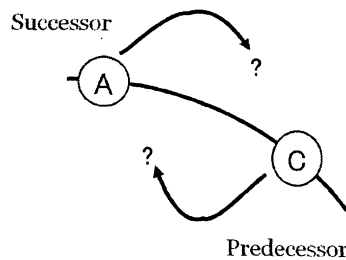


図 3.4 Successor が通知無脱退を行った場合

ている。3.3 で述べた参加手続きにおいて近隣ノードにしか通知が行わないので、ノードの参加でも他のノードへの影響 (Finger Table に参加ノードの情報が入っていない等) があるので安定化手続きが必要である。

安定化手続きは以下の手順で行う。

1. Successor List の近い順からそのノードの Predecessor を用いて確認し、違っていれば更新
2. Finger Table の  $k$  番目の更新として (自身の id) +  $2^k$  を検索し、その検索結果のノードを  $k$  番目に格納

手順 1 では、図 3.5 のノード A を例に説明する。まず Successor List[0] であるノード B の Predecessor を問い合わせる。その結果が自身を指している場合は経路が正確であることがわかる。ただし、次にノード D に問い合わせた結果がノード B なら経路表が正しいがこの場合、ノード C となる。その場合は、図 3.5 のように Successor List にノード C を追加する。ここで、Successor List[0] が変更された場合には、そのノードの Predecessor は自身を指す必要があるので自身を指すように通知する。

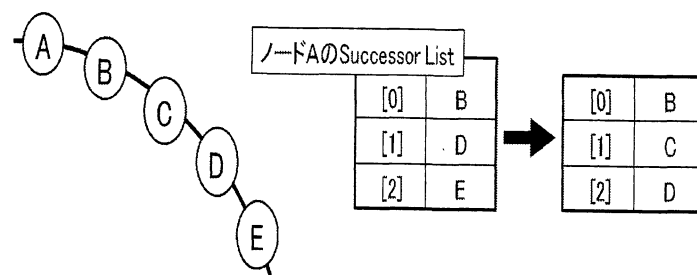


図 3.5 安定化手続きについて

手順 2 では、安定化手続きが行われる度に Finger Table を更新しない。DHT ではトポロジー



を維持するトラフィック量が非構造型 P2P ネットワークと比べて大きいため、Finger Table は検索を高速化することを目的とするだけであることから、更新頻度を抑えることでトラフィック量を抑えている。

## 第4章

# 関連研究

Chord だけではなく P2P ネットワーク全体において, churn 状態 [5] においても安定して検索が成功するための研究が [6][7][8] 等で行われている. その中でも DHT における研究を一部紹介する.

### 4.1 コンテンツの複製を利用した手法

この手法は Chord が提案される際 [1] に同時に提案されている手法である. 第3章において, 1つのコンテンツは1つのノードが管理すると述べたが, この場合に管理するノードが通知無脱退を行った時にそのノードが管理していたコンテンツも同時にネットワークから消失してしまい検索が失敗する問題がある. そこであらかじめ別のノードにコンテンツの複製をしておくことで耐故障性を上げる手法がある.

図 4.1 のようにコンテンツ X とその root ノードであるノード A がコンテンツ X を管理している場合を考える. その場合にノード A の Successor List に含まれるノードの一部 (ノード A から近い順に数ノード) に対してコンテンツの複製を配置する. これは, ノード A が脱退した場合に次に root ノードになると思われるノード (root ノード候補) に対してコンテンツの複製を持たせている.

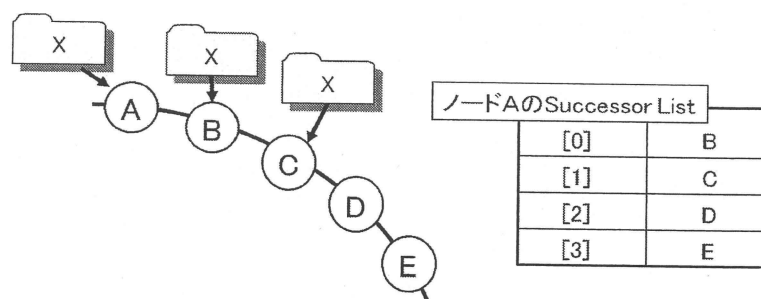


図 4.1 root ノード候補にコンテンツの複製を配置

つまり, ノード A が通知無脱退を行ってしまっても, 安定化手続きによって経路が更新されればクエリはノード B に到達し, ノード B が root ノードとして検索が成功する. これにより耐故障性を上げる手法である.

## 4.2 コンテンツを定期的に再登録する手法 [9]

この手法はノードが脱退することによって登録されたはずのコンテンツが消失してしまうことで検索が失敗することを防ぐために各ノードが自律的にコンテンツを定期的に再登録するものである。

この手法によって一部のノードが通知無脱退を行っても、しばらくすればコンテンツの再登録が行われて root ノードがコンテンツを所持していない場合に検索が失敗することを抑えることを目的としている。もちろん再登録される前にそのコンテンツが検索されれば失敗してしまう。そのために頻繁に再登録が行ってしまうとネットワーク上のトラフィック量が大きくなり、全体の負荷がかかりネットワーク効率が落ちる可能性がある。

## 第5章

# 提案手法

本研究では, churn 状態における検索が失敗する理由であるコンテンツの消失と不正確な経路表を持つノードによるネットワークの切断に注目し, それぞれに対して手法を提案する.

1 つ目は, コンテンツの複製の配置場所を変え, 複製の効果を高めて耐故障性を向上させることを目的とする.

2 つ目は, 各ノードの経路表を拡張し, ネットワーク全体に経路を作ることで耐故障性を向上させることを目的とする.

### 5.1 コンテンツの複製を ID 空間上に等間隔に配置する手法

従来のコンテンツの複製の配置場所では, 参加・脱退が行われることにより, コンテンツの複製の数の増加や root ノード候補以外のノードが複製を持ってしまうことがある. そこで本手法では, 複製に別の id を与えることによって上記の問題の解決を図る.

図 5.1 のようにコンテンツの複製を ID 空間上に等間隔に配置を行う.

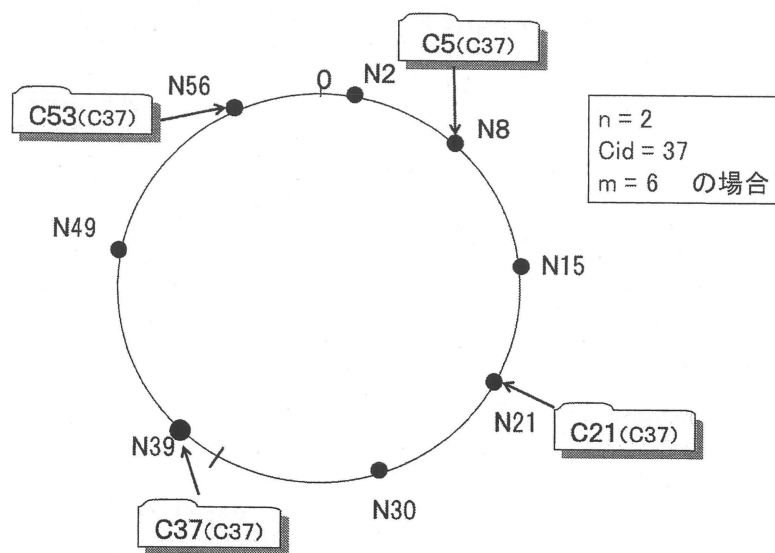


図 5.1 コンテンツの複製の配置場所について

これはコンテンツの本体と複製の合計を  $2^n$  個になるように複製を用意する. 以後説明の簡略の

ために複製の数に本体の数も含める．コンテンツの id を  $Cid$ ，複製の id を  $id_i (i=0,1,2,\dots,2^n-1)$ ，ID 空間の大きさを  $2^m$  とした時に，複製に対して以下の式から求められる  $id_i$  を与える．

$$id_i = (Cid + 2^{m-n} * i) \bmod 2^m \quad (1)$$

そして各複製をそれぞれの root ノードに管理させる．Consistent Hashing では通常ハッシュ値 (value) の衝突は滅多に無いと考えられているが，ハッシュ関数を使わずに複製に id を与えていることで衝突する可能性が増すことに加えて，複製がどの key を基に作成されたのかをわかるようにするために図 5.1 のコンテンツの括弧のように本体の id を持たせておく．

コンテンツ検索において検索ノードは，目的の key から得られる検索 id を式 (1) に代入して複製の id も求め，自身から時計回りに最も小さいものから検索を行うことによって検索にかかるホップ数を抑えられると考える．

## 5.2 LM(ランドマーク) ノードを用いた経路表の拡張手法

従来の Chord では，経路表が不正確なノードが存在することによって耐故障性が低下してしまう場合がある．各ノードが ID 空間上における時計回りに遠いノードをあまり知らないことから遠くの id を検索する際に同じノードを経由しやすいため，そのノードの持つ経路表が不正確であれば一部の id の検索が失敗してしまう．そこで図 5.2 のように LM を等間隔に配置し，その LM の root ノードを LM ノードとし，経路表に追加する．これにより，各ノードはネットワーク全体に経路を持つことができる．

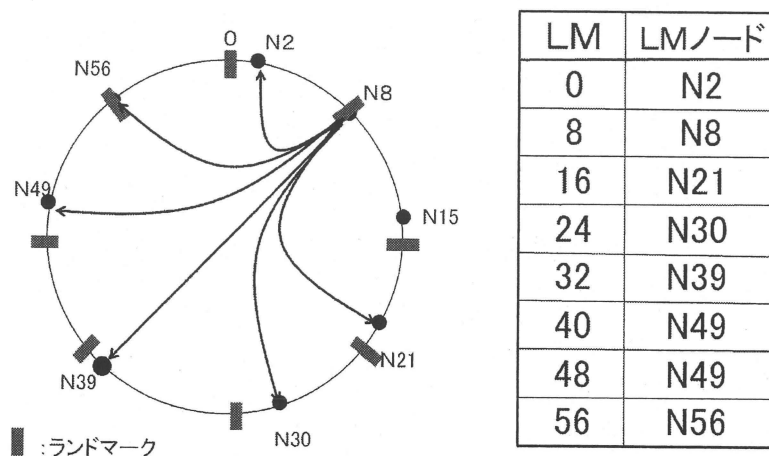


図 5.2 LM ノードを用いた経路表の拡張について

従来では，時計回りに id が  $2^{m-1}$  以上離れていた id の検索の場合 Finger Table の最も遠いノードに全てクエリを送信していたのに対し，本手法ではそれぞれ最も近い LM ノードに送信するの

で、1 ノードからのクエリがある 1 ノードに集中することを避けることができる上に、クエリが 1 ホップで ID 空間上で近いノードに送信できることから検索にかかるホップ数も軽減できる。

LM が絶対的な値で、全ノードが同じ表を持つようになっている。これにより、コンテンツを登録する際にキャッシングを行うことでホップ数を大きく軽減できるからである [11]。DHT の一手法である Tapestry においてコンテンツの登録時にキャッシングを行うことで高速な検索を可能にしている。そこで Chord においてもキャッシングを行う場合のために LM を絶対的な値にして実装している。

## 第 6 章

# 評価実験

本章では、提案手法の有効性を示すための実験について述べる。シミュレータ実験として、churn 状態での耐故障性と検索にかかるホップ数の変化を調査する。

### 6.1 実験の概要

提案手法の有効性を示すためにシミュレーションによる比較実験を行う。実験で用いる評価値は、耐故障性 (検索の成功率) と検索にかかるホップ数とする。

実験に用いるシミュレータには、以下の 7 つの手法を実装している。

- A. 複製を用いない Chord
- A+. コンテンツを定期的に再登録する手法
- B. Successor List の一部にコンテンツの複製を配置する手法
- B+. 手法 B においてコンテンツを定期的に再登録する手法
- C. コンテンツの複製を ID 空間上に等間隔に配置する手法
- D. LM ノードを用いた経路表の拡張手法
- D+. 手法 D においてコンテンツを定期的に再登録する手法

手法 C については、コンテンツの登録する際に他の手法と比べて探索クエリの数が大きいため、コンテンツを再登録する手法の実装を行わない。

また、churn 状態は安定化手続きを行う頻度より参加・脱退を行う頻度が大きいのとし、実験では安定化手続きの頻度に対して 2 倍の頻度で参加・脱退が行われる。

### 6.2 耐故障性に対する評価実験

耐故障性について評価する実験を以下の手順で行う。

1. 10000 ノードで Chord ネットワークを構築
2. 以下の 3 つの手順を実行
  - 2 ノードが参加・脱退のいずれかを実行
  - 1 つノードが 1 つのコンテンツを検索

- 全ノードが安定化手続きを実行

### 3. 検索を 1000 回行う毎に検索の成功率を求める

手順 1 については、まず初期ノードとしての 10000 ノードと後に参加する新規ノードとしての 5000 ノードを生成する。各ノードは 10 個のコンテンツを所持し、参加手続きを行った後に所持する 10 個のコンテンツを Chord ネットワークに登録する。

手順 2 については、参加・脱退の選択、参加の場合に新規ノード・再参加ノード (一度脱退し、後に再び参加するノード) の選択、脱退の場合に通知無脱退・通知有脱退の選択をそれぞれ乱数を用いて行う。ここで脱退における通知無脱退を行うノードの割合をパラメータとする。検索については、検索ノードと目的コンテンツをそれぞれ乱数を用いて決定する。安定化手続きは 30 回につき Finger Table の 1 要素、手法 D については LM ノードを 1 要素の更新を行う。

手順 3 については、最初の検索から累計した検索の成功率と 1000 回毎に分けた検索の成功率を出力する。

#### 6.2.1 実験結果

最初の検索から累計した耐故障性のグラフと、検索回数を 1000 回毎に分けた耐故障性のグラフを示す。

また、パラメータとして脱退ノードの中で通知無脱退を行うノードの割合を 2, 5, 10 割としてそれぞれを示す。実験結果のグラフは、x 軸が検索を千回単位で表している。つまり、x=1 では実験手順 2 を 1000 回行った際の耐故障性を表している。y 軸は耐故障性を表している。



- 通知無脱退の割合が 2 割

累計した耐故障性を図 6.1 に, 1000 回毎の耐故障性を図 6.2 示す.

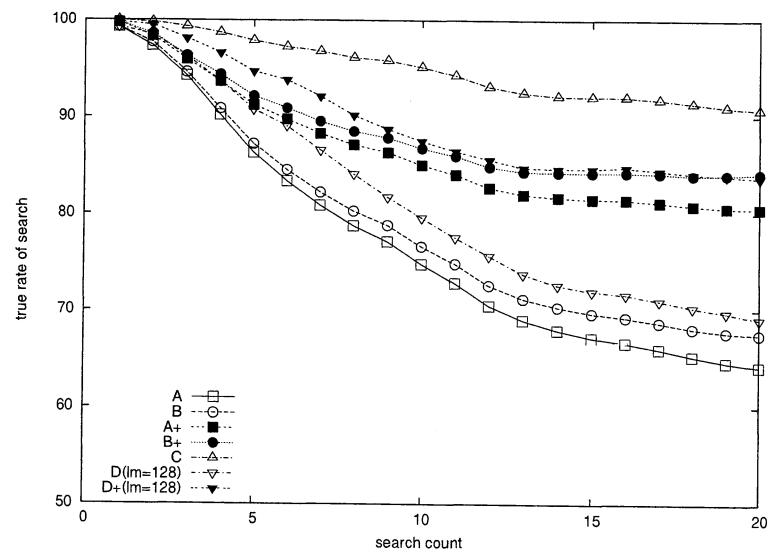


図 6.1 最初の検索から累計した耐故障性 (通知無脱退の割合が 2 割)

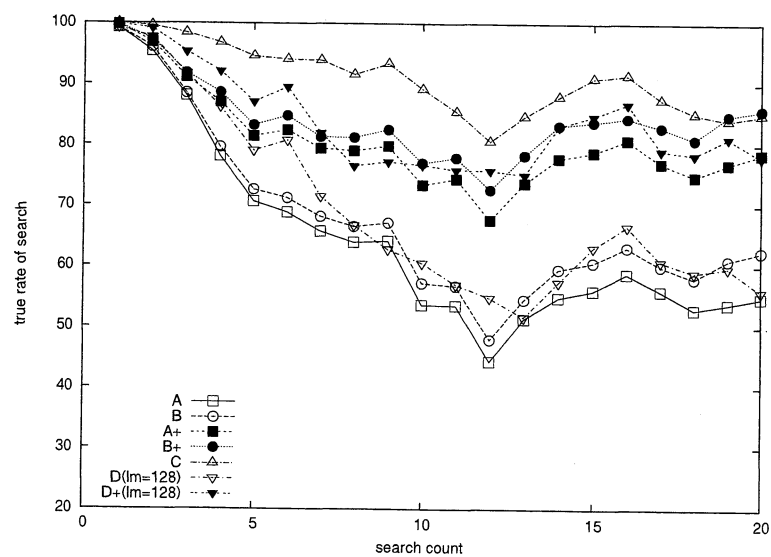


図 6.2 検索回数を 1000 回毎に分けた耐故障性 (通知無脱退の割合が 2 割)

- 通知無脱退の割合が 5 割

累計した耐故障性を図 6.3 に, 1000 回毎の耐故障性を図 6.4 示す.

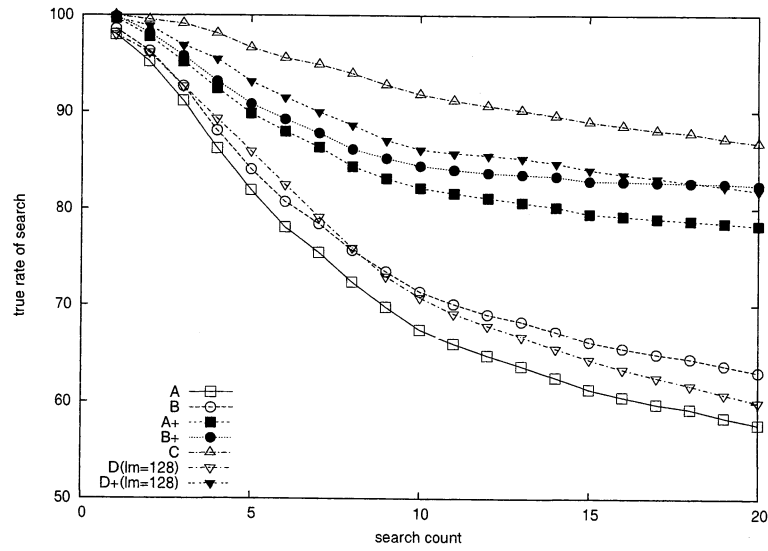


図 6.3 最初の検索から累計した耐故障性 (通知無脱退の割合が 5 割)

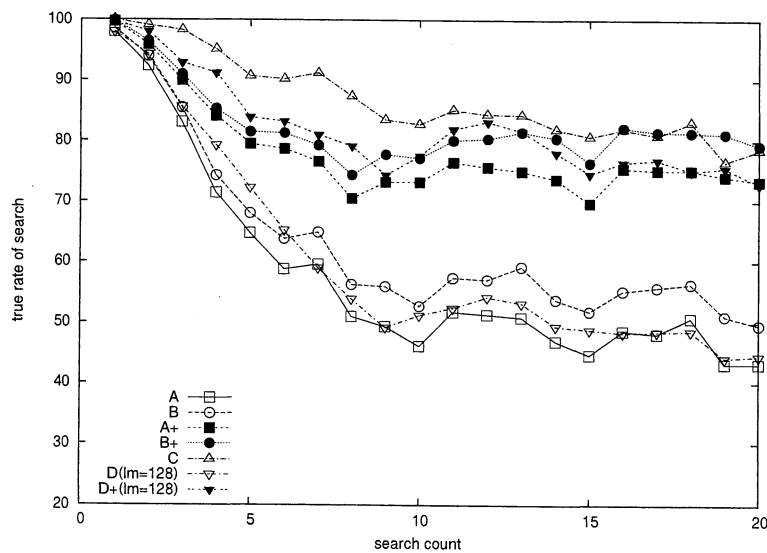


図 6.4 検索回数を 1000 回毎に分けた耐故障性 (通知無脱退の割合が 5 割)

- 通知無脱退の割合が 10 割

累計した耐故障性を図 6.5 に, 1000 回毎の耐故障性を図 6.6 示す.

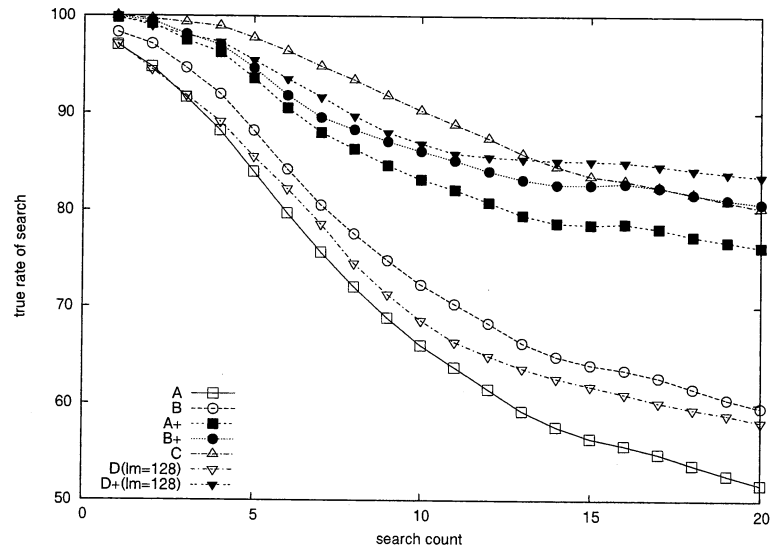


図 6.5 最初の検索から累計した耐故障性 (通知無脱退の割合が 10 割)

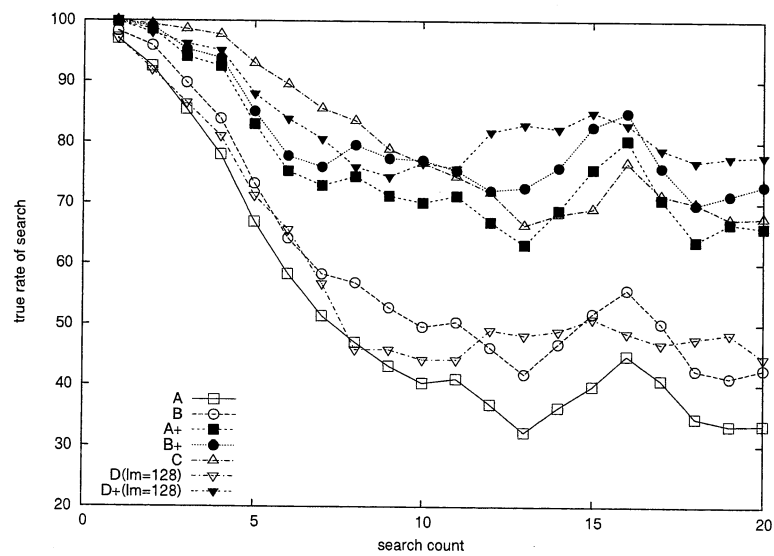


図 6.6 検索回数を 1000 回毎に分けた耐故障性 (通知無脱退の割合が 10 割)

### 6.2.2 考察

手法 C, コンテンツを再登録する手法 (A+, B+, D+), 参加時のみコンテンツ登録を行う手法 (A, B, D) の順に耐故障性が高い結果が得られた. 手法 A, B, D について比較すると, 全体的に提案手法である手法 D が最も高くなっている. この理由としては 5.2 で述べたように, 1 ノードからの検索クエリが一部のノードに集中することを避けることで, クエリが不正確な経路表を持つノードを経由する可能性が減ったからである.

最初の検索から累計した耐故障性を示す図 6.1, 図 6.3, 図 6.5 については, churn 状態が続いていくほどに全手法において耐故障性が低下している. そして通知無脱退の割合が増えると耐故障性も大きく低下する. これは, 通知無脱退によって root ノードがコンテンツを管理していない場合の失敗が大きな要因であった. それに加えて, 複製の数が増えすぎることを防ぐために参加手続きの際に新しい Predecessor に対して複製を渡したら, 自身は複製を破棄する仕様になっているため, root ノード候補でないノードが複製を管理してしまっている場合もあったので再登録を行わない手法では耐故障性が低くなっている. それに対して手法 C では, 複製の数の分の検索を行えるのでいずれかが成功すればそのコンテンツの検索は成功としているので耐故障性が高くなっている.

検索回数を 1000 回毎に分けた耐故障性を示す図 6.2, 図 6.4, 図 6.6 についても同様に churn 状態が続いていくほどに全手法において耐故障性が低下している. ただし, 一部耐故障性が右上がりになっているところもある. これは検索ノードと目的コンテンツを乱数を用いて決めていることによるものである. コンテンツを定期的に再登録しない手法においても, 再参加の際にそのノードが再登録を行うこともありどの手法も一定ところで収束していく傾向がある.

## 6.3 検索にかかるホップ数に対する評価実験

ホップ数に対する評価実験において予備実験として脱退を考慮しない場合における提案手法に対する評価実験を 2 つ行い, 次に churn 状態でのホップ数に対する評価実験を行う.

### 6.3.1 ID 空間上の距離とホップ数の関係

Chord では循環リストを用いていることに加えて, Finger Table を用いていることから  $O(\log(N))$  の検索が可能である. そこで 1 ノードから全ノードに対してクエリを送信する場合にかかるホップ数について調査を行った.

実験内容を以下に述べる.

- ノード数を 100 ノードとして Chord ネットワークを構築
- 1 ノードから全てのノードに対してクエリを送信
- そこでかかったホップ数を調査

実験結果を図 6.7 に示す。

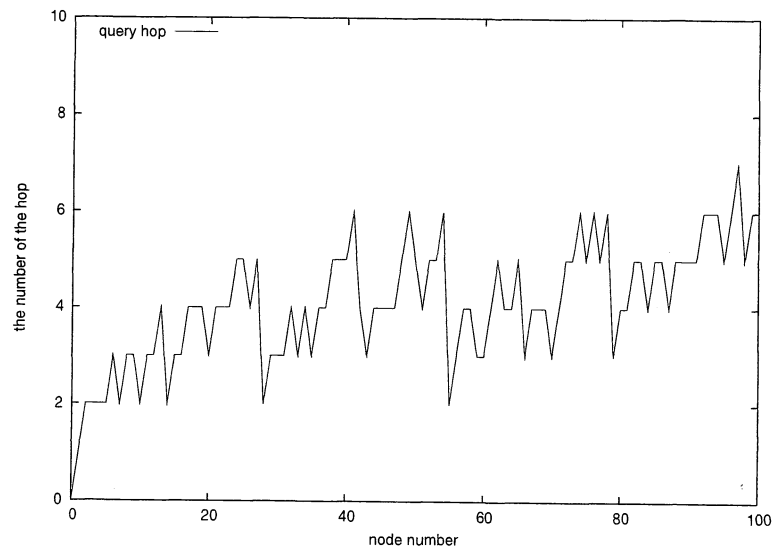


図 6.7 ID 空間上の距離とホップ数の関係

ID 空間上において検索ノードから目的の id が時計回りに離れていくほどホップ数が大きくなる傾向があることが分かる。つまり、検索ノードから目的の id との ID 空間上の距離を大きくしなければホップ数を軽減できる。

また、図 6.7 のようになるのは、各ノードの経路表が近くのノードを密に、遠くのノードを疎になるように経路を持つことで ID 空間上の距離が大きくなるほどホップ数が大きくなるので手法 D において遠くのノード情報を増やすことによってホップ数を軽減することが可能であると考えられる。

そこで手法 C については、コンテンツの複製を等間隔に配置し、検索時に最も近いものを検索していることでホップ数を軽減することが可能であると考えられる。

### 6.3.2 ホップ数に対する評価実験

まず手法 A, C, D において脱退を考慮しない場合におけるホップ数について以下の実験内容で評価を行う [10][11]。

- ノード数を 100, 1000, 10000 ノードとして Chord ネットワークを構築
- 1 万個のコンテンツを登録
- 1 つのノードが全てのコンテンツを検索
- 手法 C は  $n=2,3$  として複製を等間隔に配置

- 手法 D は LM の数を 16,32,64,128 として LM ノードを経路表に追加

1 万個のコンテンツを全て検索した際にかかる平均ホップ数を、図 6.8 で手法 A(既存手法) と手法 C(コンテンツを ID 空間上に等間隔に配置する提案手法) の比較と、図 6.9 で手法 A と手法 D(LM を用いて経路表を拡張する提案手法) の比較を行う。

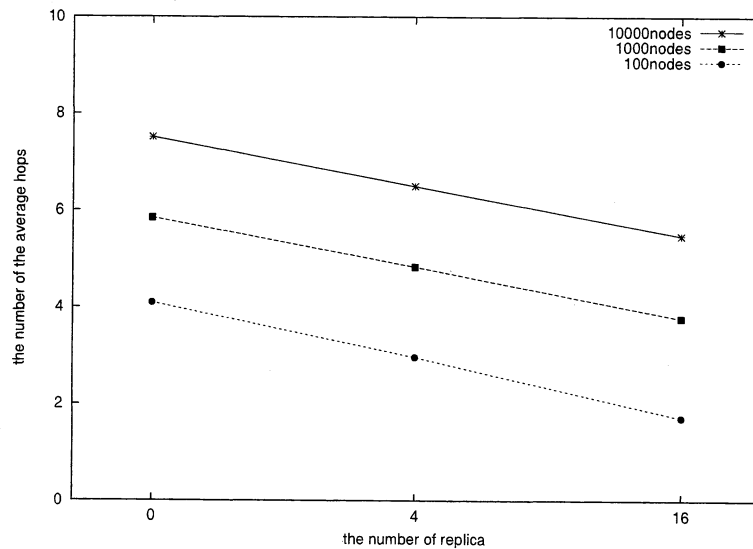


図 6.8 手法 A と手法 C の平均ホップ数の比較

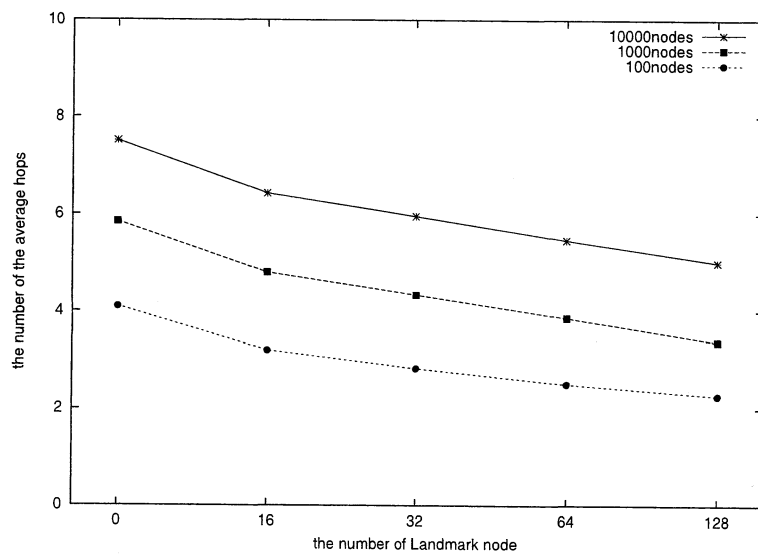


図 6.9 手法 A と手法 D の平均ホップ数の比較

コンテンツの複製の数と LM ノードの数を 2 倍すると平均ホップ数が約 0.5 ホップ小さくなる結果が得られている。コンテンツを ID 空間上に等間隔に配置する手法では、複製の中で検索ノードから最も近いものを検索していることから検索範囲を複製の数の分縮めることができるのでホップ数が軽減されている。LM を用いて経路表を拡張する手法では、LM ノードによって遠くのノードにも小さいホップ数でクエリを送信できるようになったのでホップ数が軽減されている。

つまり手法 C・D においてネットワークを分割した形になっている。

### 6.3.3 churn 状態でのホップ数に対する評価実験

6.2 と同じ実験を行い、検索にかかったホップ数に対する評価を行う。以下のグラフでは、x 軸が検索回数を千回単位で表しており、y 軸は 1000 回ごとにかかった平均ホップ数を表している。

通知無脱退の割合が 2 割の場合を図 6.10, 5 割の場合を図 6.10, 10 割の場合を図 6.10 に示す。

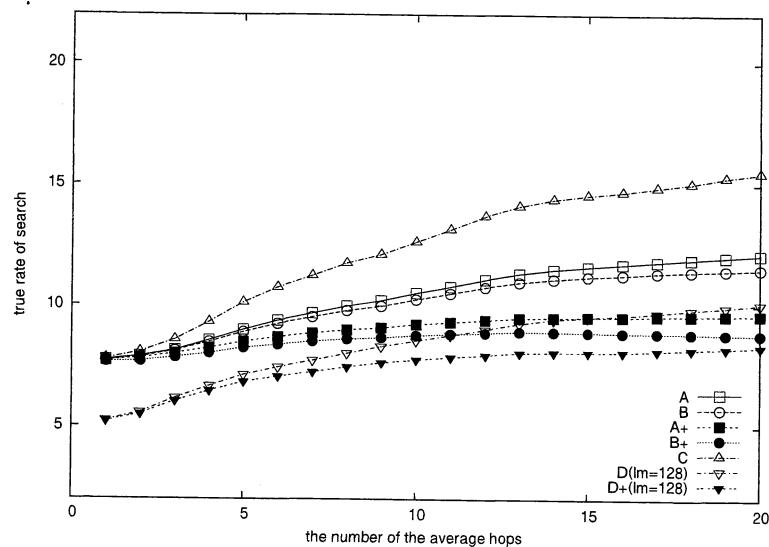


図 6.10 churn 状態での検索にかかるホップ数 (通知無脱退の割合が 2 割)

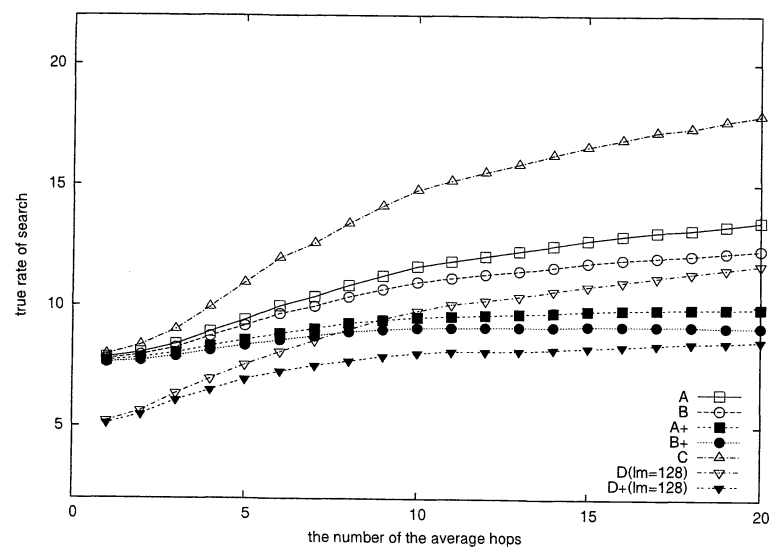


図 6.11 churn 状態での検索にかかるホップ数 (通知無脱退の割合が 5 割)

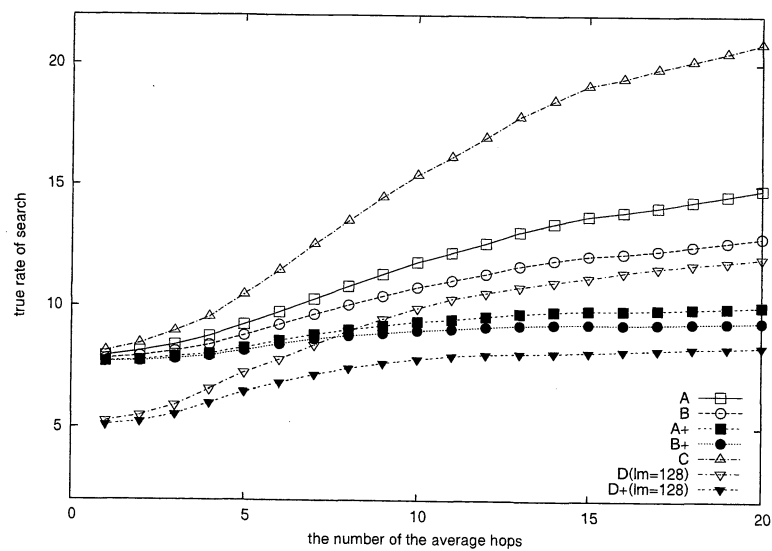


図 6.12 churn 状態での検索にかかるホップ数 (通知無脱退の割合が 10 割)



LM を用いた手法 D が最もホップ数が小さくなっている。この理由は、前述の通り 1 ホップで目的の id に近いノードまでクエリを送信することができるので平均ホップ数が小さくなっている。そしてコンテンツの複製を ID 空間上に等間隔に配置する手法 C は最もホップ数が大きくなってしまっている。この理由は、1 つのコンテンツに対する検索において 1 つの複製に対する検索が失敗しても次に遠い複製を検索していくので、その分検索にかかるホップ数が大きくなってしまふ。

コンテンツを再登録を行うことでホップ数が小さくなっている理由は、コンテンツがネットワーク上に存在することで、検索手続きの手順 1 で検索が終了する場合があるのでホップ数が小さくなっている。

全体的には、churn 状態が続くとノードが持つ経路表の Finger Table の情報が不正確なものになるため、近いノードにクエリを送るのが出来なくなってしまうのでホップ数が大きくなっている。

#### 6.4 評価実験のまとめ

コンテンツを ID 空間上に等間隔に配置する手法は耐故障性が最も高くなった。しかし、検索にかかるクエリのホップ数が大きくなってしまった。ランドマークを用いて経路表を拡張する手法は前者の手法に次いで耐故障性を高くすることができ、さらに検索にかかるクエリのホップ数を既存手法よりも小さく抑えることができた。

## おわりに

本研究では，churn 状態での耐故障性の向上手法として，コンテンツの複製を ID 空間上に等間隔に配置する手法と LM を用いた経路表の拡張手法を提案した．さらに提案手法の有効性を調べるためにシミュレーションによる評価実験を行った．従来手法では通知無脱退によって，コンテンツがネットワークから消失してしまうことと不正確な経路表を持つノードが存在してしまうことで，耐故障性が低下している．そこでコンテンツの複製を ID 空間上に等間隔に配置する手法では，本体の id とは別に複製用の id を持たせていることで複製に対する検索が行えることによって耐故障性を向上させた．LM を用いた経路表の拡張手法では，LM を用いることでネットワーク全体に経路を持たせることができ，不正確な経路表を持つノードにクエリが送信される可能性を減らすことで耐故障性を向上させた．シミュレーション実験の結果から，コンテンツを ID 空間上に等間隔に配置する手法は耐故障性が最も高くなった．しかし，検索にかかるクエリのホップ数が大きくなってしまった．ランドマークを用いて経路表を拡張する手法は前者の手法に次いで耐故障性を高くすることができ，さらに検索にかかるクエリのホップ数を既存手法よりも小さく抑えることができた．

## 謝辞

日ごろから多くの御指導を頂きました太田義勝教授，鈴木秀智准教授，院生の方々に深く感謝いたします。そして，日頃何かとお世話になりました落合美子事務員に感謝いたします。また，本論文作成にあたって特にお世話になりました太田義勝教授に深く感謝いたします。最後に，日頃から熱心に討論して頂いた研究室の諸氏に感謝いたします。

## 参考文献

- [1] I.Stoica, R.Morris, D.Liben-Nowell, D. R.Karger, M.F.Kaashoek, F.Dabed, H.Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications", IEEE/ACM Trans. Networking, 11, 1, pp.17-32 (2003)
- [2] S.Ratnasamy, P.Francis, M. Handley, R.Karp, and S.Shenker: "A Scalable Content-Addressable Network", In Proc. of ACM SIGCOMM 2001, pp.161-172(2001)
- [3] B.Zhou, D.A.Joseph, and j.Kubiatowicz, "Tapestry: a fault tolerant wide area network infrastructure." In Sigcomm, pp. Tech.Report UCB/CSD-01-1141(2001)
- [4] P. Maymounkov and D.Mazieres, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric", In Proc. of IPTPS 2002, pp. 53-65(2002)
- [5] S. Rhea, D. Geels, T.Roscoe and J.Kubiatowicz, "Handling Churn in a DHT, USENIX'04(2004)
- [6] 野口悟, 猪俣敦夫, 藤川和利, 砂原秀樹, 「DHT におけるノードの動作履歴を用いたデータ管理手法の提案と実装」, インターネットコンファレンス 2008 論文集, (2008)
- [7] 洞井晋一, 松浦知史, 藤川和利, 砂原秀樹, 「時間に基づく階層化と Value の集約配置手法による耐 Churn オーバレイネットワーク」, 情報処理学会論文誌, Vol.51, No.4, pp.1142-1151(2010).
- [8] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I.Stoica, "Towards a Common API for Structured Peer-to-Peer Overlays", IPTPS'03, pp.33-44(2003)
- [9] 首藤一幸, 「下位アルゴリズム中立な DHT 実装への耐 churn 手法の実装」, 情報処理学会論文誌コンピューティングシステム, Vol.49, No.2, pp.1-9(2008)
- [10] 岩尾隆弘, 太田義勝, 鈴木秀智, 「Chord におけるコンテンツ複製を利用した検索の高速化手法」, 2009 年度電気関係学会東海支部連合大会 (O-353)(2009)
- [11] 岩尾隆弘, 太田義勝, 鈴木秀智, 「Chord における検索の高速化を目的とした経路表の改善手法」, 2010 年度電気関係学会東海支部連合大会 (E1-3)(2010)