

平成13年度学士論文

Peer-to-Peer システムにおける
動的な木構造の生成による
検索の高速化

東京工業大学 理学部 情報科学科
学籍番号 98-0701-7

柿原 聡

指導教官
千葉 滋 講師

平成14年2月7日

概要

Gnutella のような Peer-to-Peer(P2P) システムでは、他のホストに対するファイルの検索の際にキャッシュを行わないため、検索に時間がかかる。また、検索する度にルーティングを行うため、ネットワーク全体で無駄なメッセージが多くネットワーク資源をより多く消費する。そこで、本研究では、動的に情報を集めて木構造になるルーティングを行い、その上でキャッシュを行う完全分散型の P2P システムを提案、Linux 上で実装した。このシステムは各ホストの状態がなるべく忠実に反映されるよう実装され、従来型の P2P システムと比較した場合、検索の時間を短くし、ネットワークのメッセージ数を節約することが実験により示された。

謝辞

本研究を進めるにあたり，研究の方向付けや論文の組立て方についての助言をして頂いた指導教官の千葉先生に感謝致します。

東京大学の光来健一氏にはお忙しい中論文のほぼすべてを見ていただきました。感謝致します。

そして筑波大学の横田大輔氏には論文の組立て方やプログラムの設計について見ていただきました。感謝致します

筑波大学の立堀道昭氏には，さまざまな助言を頂き感謝致します。

並びに同室で共に研究をしてきた皆様にも励ましていただきました。重ねてお礼を申し上げます。

目次

第 1 章	はじめに	6
第 2 章	関連研究	8
2.1	階層型集中型	8
2.1.1	Domain Name System(DNS)	9
2.1.2	階層的な Web Caching System	11
2.2	Peer-to-Peer 型	11
2.2.1	Napster	13
2.2.2	Gnutella	14
2.2.3	Freenet	15
第 3 章	p2p-cache システムの提案と実装	17
3.1	p2p-cache の提案	17
3.2	木の生成法	17
3.3	検索の Protokol	19
3.3.1	Query, QueryHit のフォーマット	19
3.3.2	検索の手順	21
3.4	実装の詳細	22
3.4.1	初期化	22
3.4.2	ノード間通信	23
第 4 章	実験	25
4.1	実験で生成したネットワーク	26
4.2	実験結果	26
4.2.1	検索時間	26
4.2.2	メッセージ数についての測定結果	28
4.3	実験結果のまとめ	34
第 5 章	まとめ	38

目 次

2.1	DNS のしくみ	10
2.2	一般的な Web Caching System	12
2.3	Napster における検索	14
2.4	Gnutella における検索	15
3.1	Query のフォーマット	19
3.2	検索	21
4.1	p2p-cache での木 A	27
4.2	p2p-cache での木 B	27
4.3	p2p-cache での木 C	28
4.4	従来型のネットワーク	29
4.5	木 A での検索の応答時間	30
4.6	木 B での検索の応答時間	30
4.7	木 C での検索の応答時間	31
4.8	従来型のネットワークでの検索の応答時間	31
4.9	木 A での検索の 100 回平均の応答時間	32
4.10	木 B での検索の 100 回平均の応答時間	32
4.11	木 C での検索の 100 回平均の応答時間	33
4.12	従来型のネットワークでの検索の 100 回平均の応答時間	33
4.13	木 A でのメッセージ数	34
4.14	木 B でのメッセージ数	35
4.15	木 C でのメッセージ数	35
4.16	従来型のネットワークでのメッセージ数	36
4.17	木 A でのメッセージ数の 100 回平均	36
4.18	従来型のネットワークでのメッセージ数の 100 回平均	37

表 目 次

4.1 Query の流量の平均の比較	37
-------------------------------	----

第1章 はじめに

我々が普段利用しているインターネットでは、主に2種類検索方法がある。一つは、googleのようなWebの検索エンジンに代表される集中型の検索方法である。集中型の検索では、特定のホストにインデックスが集中し、そこでのみ検索を行う。もう一つは、分散型の検索方法である。分散型では、インデックスがネットワーク上に分散し、検索はネットワーク全体で行われる。

分散型には、さらに、人間によって管理されたネットワークと管理されないネットワークがある。管理されたネットワークとは、Domain Name Systemのようにノード情報、階層構造を人間の手によって管理するネットワーク形態である。管理されたネットワークでは、検索経路が管理されているので、動的に経路を決定する際の無駄なQueryが減らせる長所がある。さらに、効率的なキャッシュを行いやすい長所も持っている。

一方、管理されないネットワークとはPeer-to-Peer(P2P)型のネットワークを指す。Gnutellaに代表されるP2P型のネットワークでは完全分散型のネットワークを形成している。完全分散型のネットワークでは各ノードが互いに対等であり、ノード情報は人間に管理されない。一つのノードに情報が集中しないため、フォールトトレラントである長所をもつ。すなわち、一つのノードが停止してもネットワーク全体に大きな影響を与えにくく、全体的に安定している。一方、このような完全分散型では、検索時に検索の経路を決定するためその分のオーバーヘッドが生じネットワーク全体のメッセージ数が増加する欠点をもつ。これは、すでにQueryの到達しているノードへQueryを出すことがメッセージ数増加の原因になっている。さらに、検索の経路は実行時によって異なるため、効率的なキャッシュを行うのが難しい。以前にネットワーク上のノードが行った検索項目がキャッシュに載っていたとしても、それを効率よく利用できない。

そこで、このような欠点に対処するため、本研究では完全分散型のP2Pシステムに集中型、階層型のシステムの長所である木構造を動的に取り

入れ、さらに、キャッシュの実装を行い、検索の高速化を目指した。本実装は Peer-to-Peer cache の略で p2p-cache と呼ぶ。P2P ネットワークにノードが追加されるたびに木構造の検索経路を動的に決定し、検索時にその経路を使うようにした。さらに、その経路上で効率の良くなるようにキャッシュを自動的におくようにした。

本研究では、p2p-cache を実際に Linux 上に実装し、実験で検索速度を測定した。実験から、本実装でネットワークの流量を減らし、検索が高速になることが示された。これは、集中型、階層型のシステムの特徴である木構造を取り入れることで、検索の度に経路を決定するオーバーヘッドを取り除き、ネットワークの流量を減らすことができることが理由の一つに入る。さらに、経路をあらかじめ定めることによって、効率良くキャッシュができ、検索が高速になったことも理由として挙げられる。木構造を導入することでネットワーク全体のノード間の最大距離は増大したが、それは、検索の度に経路を決定するオーバーヘッドの除去で十分相殺できた。

以下ではまず次章で、関連研究の説明とその長所短所の指摘を行い、3章で p2p-cache の実装を述べる。4章では従来型の完全分散型 P2P システムと p2p-cache の比較実験を行い、5章で本稿をまとめる。

第2章 関連研究

この章では、階層型集中型のネットワークシステムと P2P 型のネットワークシステムについて説明し、両者の長所短所の説明、比較を行う。

階層型集中型のネットワークは、閉路を含むネットワーク上に木構造を導入し階層構造を形成したネットワークである。各階層ごとに、効率よく検索ができるようにしたネットワークである。このようなネットワークでは検索は早い、ネットワークの帯域が少なく済む、安定性安全性に優れるという長所を持つ。しかし、サーバーに対する負担が大きくなり、コストがかかるという欠点がある。

一方、P2P 型のネットワークはネットワーク上の各ノードが対等の立場で、サーバーが不要でありサーバー管理コストが削減できる長所をもつが、一方でネットワークの帯域を階層型集中型より消費する、検索が遅いという欠点をもつ。

P2P 型のネットワークシステムは、新しい技術ではない。実際、数年前までは、サーバー・クライアント方式がネットワークシステムにおける検索方法の主流であったが、近年のネットワーク回線の高速化、コンピュータの高速化に伴って、再び P2P 型のネットワークが脚光をあびるようになって来ている。理由は、万人にとって貴重なコンテンツやリソースをネットワーク上に分散しながらも、所有者の管理下におくことができるネットワーク形態だからである。

以下に、階層型集中型ネットワークと P2P 型ネットワークの具体例を挙げながら、その長所短所について説明、比較を行う。

2.1 階層型集中型

階層型集中型のネットワークは、閉路を含むネットワーク上に木構造を導入し階層構造を形成したネットワークである。このようにすることで、効率よく検索ができるようにしたネットワークである。このような

ネットワークでは検索は早く、ネットワークの帯域も少なく済む、安定性安全性に優れるという長所を持つ。一方で、サーバーに対する負担が大きくなりコストがかかると言う欠点がある。

P2P 型のネットワークが脚光を浴びるようになった現在でも、インターネットにおいて階層型集中型のネットワークが主流を占めている。その理由としては大規模なネットワークを管理するためには、階層型集中型のネットワークの方が検索速度、ネットワークの帯域幅の消費に関して有利であるからである。

階層型集中型のネットワークシステムの代表的なものとして、Domain Name System が挙げられる。Domain Name System はインターネット上、IP アドレスとホスト名をマッピングして相互解決するための仕組みである。インターネット上で、最も一般的で大規模な検索システムである。

さらに、Web Caching System についても調べた。Web caching system はインターネット上の World Wide Web でキャッシュを行うシステムである。インターネットのバックボーンの容量は年に 60 パーセント以上の割合で増加しているが、インターネットに対する需要は年々増加している。実際、近年の一月あたり約 15 パーセントの割合で静的な web コンテンツが増加している。このような背景のなか Web でのパフォーマンスを増加させ、ネットワークのトラフィックを減少させるための方法の一つに Web Caching System がある。

基本的には、プロキシサーバーでキャッシュを行い、Web ページの取得を高速化し、ネットワークの混雑を緩和させる。欠点としては、一つのプロキシサーバーに Query が集中するとプロキシサーバーが処理に対処できず遅くなることである。この節では階層型の Web Caching System について研究について紹介する。

2.1.1 Domain Name System(DNS)

DNS は、インターネット上、IP アドレスとホスト名をマッピングして相互解決するための仕組みである。この仕組みは、名前解決と呼ばれる。DNS は図 2.1 のような階層構造をなしている。

普段使用されるドメイン名と呼ばれるホスト名は、”.” でいくつかの階層に区切られる。例として、”forina.csg.is.titech.ac.jp” のように表現さ

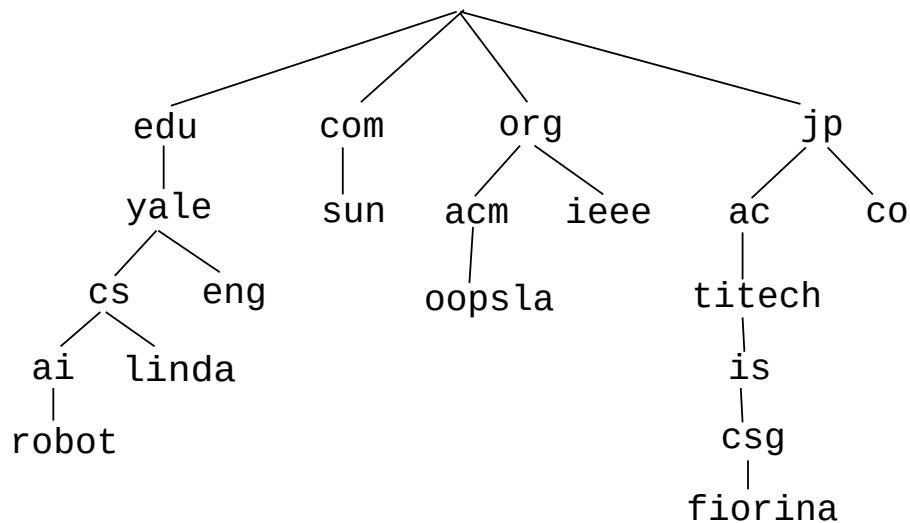


図 2.1: DNS のしくみ

れる。それぞれの階層ごとに、それ以下に含まれる下位ドメイン名やホスト名を管理を行う。

この木の枝分かれの部分に該当するものが、ドメインである。“is.titech.ac.jp”など、上位ドメイン名を付けたものと区別するためにノードとも呼ばれる。また、このようなデータ構造を含む名前を“Name Space”とも呼ぶ。

各ドメインでは、自身の下位ドメイン（サブドメイン）として何があるか、そのサブドメインの情報を持っている DNS サーバは何か、自身に所属するホストの IP アドレスなどの情報などの情報が管理されている。これらを管理しているのが、それぞれのノードに位置して、そのドメインを管理する DNS サーバである。

各ノードに位置する DNS サーバでは、自身の管理するドメイン内情報とサブドメインの DNS サーバ名しか知らない。そこで、DNS への問い合わせ側のクライアントであるリゾルバは、名前解決のとき、この階層を上位から順にたどる。ルートから jp ドメインの DNS サーバへ、jp ドメインの DNS サーバは ac ドメインの DNS サーバを知っているので、次に co ドメインの DNS サーバへ titech.ac.jp ドメインの DNS サーバを尋ねる。そして最終的には、「www.csg.is.titech.ac.jp」の IP アドレスを知る DNS サーバまでたどり着く。この木で図のように名前解決が行われる。

しかし、名前解決の度に全経路をたどるとネットワークへの負荷が増加

する、応答性が悪いと言った問題が生じるので、実際には経路上にあるネームサーバーは以前の名前解決の内容をキャッシュする。ネットワークにおける名前づけの変化に対応するため、キャッシュには Time To Live(TTL) を設けてある。

DNS はインターネットにおける代表的な階層型集中型の検索システムである。したがって、途中のノードが停止するとそのノードを境に名前解決が行えなくなる。例えば、図 2.1 において、titech.ac.jp のネームサーバーが停止すると、その下流のホストからは、その上流にあるホストの名前解決が行えなくなり、さらに上流のホストからも、titech.ac.jp の下流のホストの名前解決が行えなくなる。

DNS は、このような問題を回避し、fault-tolerant なシステムにするために、特にルートに近いホストは複数台のネームサーバーを動かしている。

本システムでは、DNS の効率性を活用した。

2.1.2 階層的な Web Caching System

World Wide Web は巨大な分散ネットワーク環境と考えられる。近年のネットワークトラフィックの増加による情報伝達の遅延を解消する手段の一つとして、Web caching system がある。一般にプロキシサーバーがキャッシュを行い静的な Web コンテンツの伝達の高速化を計る。図?? に典型的な Web Caching System を示す。

階層的な Web caching system は、プロキシサーバーを階層的に構成したものである。階層的な Web caching system は、分散的な Web Caching System と比較して connection time は短い、transmission time は長くなる。

2.2 Peer-to-Peer 型

Peer-to-Peer ネットワークとは、コンピューター同士を対等 (Peer) な立場で接続するネットワークの接続形態である。Peer-to-Peer 型ネットワークでは、接続されたコンピューターがクライアントとサーバー（サービスを提供する側とサービスを受ける側）の両方の機能を持ち相互にハードディスクなどを公開することで、データのやり取りを実行できるのが特徴である。

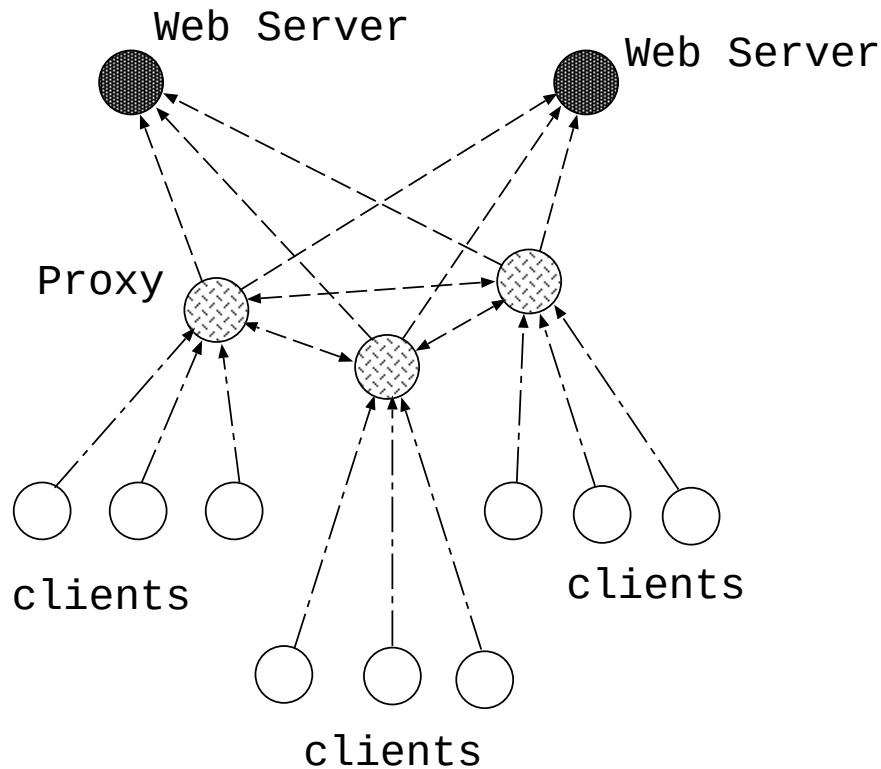


図 2.2: 一般的な Web Caching System

Peer-to-Peer 型の構内情報通信網 (LAN) は 10 台程度の小規模なネットワークに適している。LAN について特別な知識がなくてもパソコン同士を接続し、ソフトウェア面を設定すれば簡単に構築できるサーバー専用のコンピューターが不要なため、ファイルやプリンターの共有ネットワークが低コストで簡単に構築できる。

P2P では数十台の接続もできるが、ネットワークを管理するコンピューターがないので、オペレーティングシステム面でのセキュリティ管理が難しい欠点がある。P2P 型のネットワーク構築は、一般に数台のコンピューターが接続された環境やデータ交換が少ない場合は適しているが、中規模以上の LAN 構築では支障が多いと言える。

Napster は、ファイル交換は P2P で行われるが、ファイルインデックスの管理は中央のサーバーに依存し完全な P2P ネットワークではない。

一方、Gnutella は純粋な P2P ネットワークを形成している。Gnutella

の最大のメリット、特徴としては、サーバが不要なことが挙げられる。従来のサーバー・クライアントのモデルでは、情報はすべてサーバに集中し、サーバの管理コストも増大した。Napster 型では、サーバー・クライアントの役割分担がサーバからクライアントにかなり移行した。つまり、サーバーは、ディレクトリサービスを提供するだけである。実際の情報はすべてクライアント PC に存在していた。このため、サーバの管理コストや、コンテンツの管理コストはかなり軽減された。

2.2.1 Napster

Napster はファイルの転送に関しては P2P で行われるが、ファイルインデックスを管理する中央のサーバが存在し、検索作業はこの中央のサーバを通して行われる。各ノードは、サーバーを通じて Napster ネットワークに加わる。ノード全ての情報はサーバーで管理されている。検索は中央のサーバーに問い合わせることで、行われる。従って、中央のサーバーが停止したときにはネットワーク全体が機能せず、P2P の特徴であるフォルト・トレランスは弱くなっている。

Napster のネットワークは、階層型と P2P 型のネットワークを組み合わせたことからハイブリッド P2P (hybrid P2P) と呼ばれる。

Napster は P2P ネットワークを世に知らしめたが、完全な P2P ネットワークではない Napster では図 2.3 のように、検索を行う。

具体的には次のように検索を行う。

1. クライアントはサーバーにファイルの検索内容を送る。
2. サーバーはクライアントから得たファイルの検索内容を元に自分のデータベースを検索する。検索結果を送信元のクライアントへ送る。
3. クライアントはサーバーから送られてきた検索結果をもとに、直接他のクライアントとファイル交換をする。

ファイル検索に中央のサーバーを用いるので、検索速度は早く、ネットワークの使用帯域も少なく済む長所を持っている。一方、Napster は、検索をすべて中央のサーバーに依存し、そこがボトルネックになっている欠点を持っている。

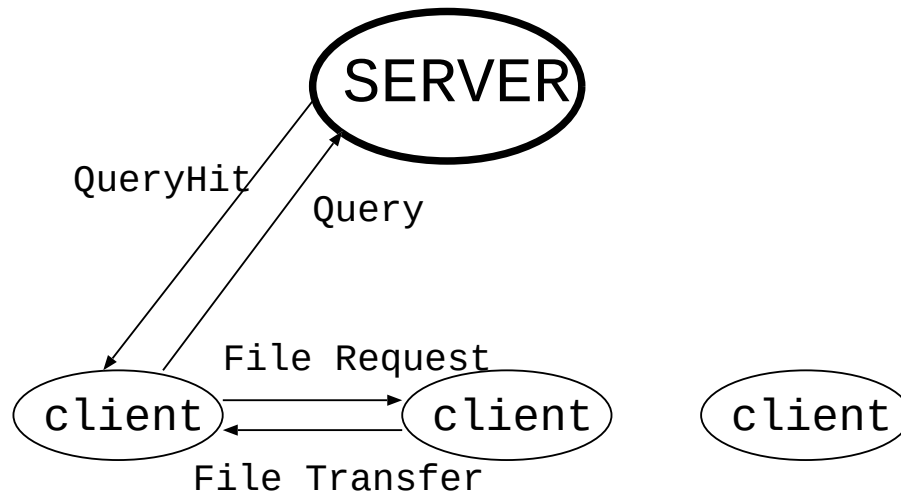


図 2.3: Napster における検索

2.2.2 Gnutella

Hybrid P2P である Napster と異なり、Gnutella は完全分散型の P2P ネットワークをなす。このネットワークでは中心となるサーバーが存在せず、各ノードが対等である。

完全分散型ネットワークは単一のノードに依存しないので、あるノードがシャットダウンしてもネットワーク全体は全く影響を受けず、

Gnutella では図 2.4 のように、検索を行う。

1. あるノードは検索項目をブロードキャストする。このノードは検索時に接続している他のノード全てに Query を出す。
2. Query を受け取ったノードは、まず、その Query が以前に来たものであるか調べる。同一ノードからの同一の Query の重複を避けるために、その Query が以前にきたものであれば、そのノードはそれ以上は何もしない。その Query が初めて来るものであれば、自分がファイルを持っているか調べ、持っていれば、QueryHit で返す。持っていなければ、QueryHit は返さない。

QueryHit は、Query をフォワードしてきたノードに返す。

3. ノードは Query をフォワードしてきたノード以外の接続中のノードに Time-to-Live(TTL) を一つ減らしてからフォワードする。TTL

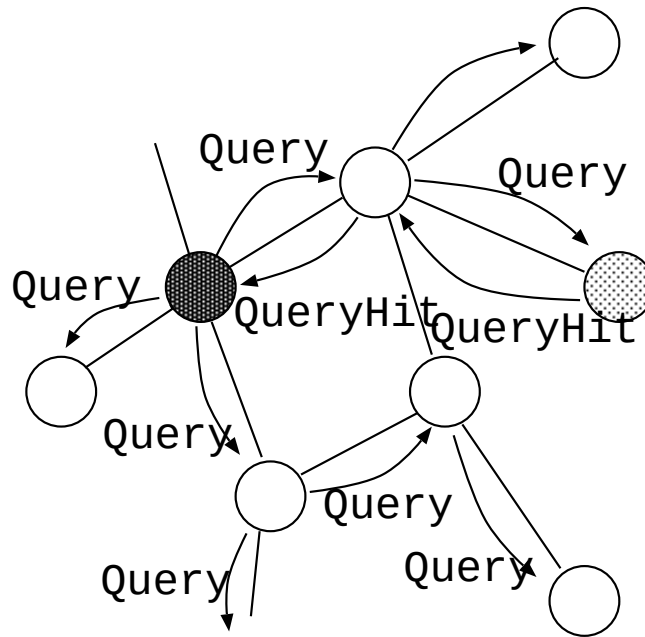


図 2.4: Gnutella における検索

とは、Query のフォワードできる回数である。0 になるとそれ以上フォワードできない。

4. 以下、Query は再帰的にフォワードされていく。

2.2.3 Freenet

Freenet とは、分散型の匿名情報保存、検索システムである。ファイルの名前やデータをハッシュし、そこから得られたハッシュキーを元に検索を行う P2P システムである。ファイル名をハッシュした場合には Gnutella と同じようにファイル名を検索できるが、ファイルのデータをハッシュした場合には、ネットワーク上で匿名性、安全性が維持できる。

次のようにして検索を行う。

1. 検索を行うノードは最初にファイルのハッシュキーを取得する。

2. ハッシュキーと TTL をまず自分のノードの情報に問い合わせる．ハッシュキーと適合するファイルがあれば、検索は終了する．なければ、ノードのルーティングテーブルをルックアップする．
3. ルーティングテーブルでキーの一番近い値を捜し、そのノードへ Query を送る．この検索が究極的に成功すれば、ファイルは経路上のノードを通して送られキャッシュも行われ、ルーティングテーブルにも追加される．

第3章 p2p-cache システムの提案と実装

階層型集中型のネットワークシステム、P2P 型のネットワークシステムでは両者とも 2 章でふれたような長所短所があった。そこで我々はその点を踏まえ両者の長所をしたシステムを提案・開発した。

実装は、オペレーティングシステムは RedHat Linux 7.1 上で、プログラミング言語は C++ を使い、コンパイラは Gnu c++ compiler(g++) を用いた。

3.1 p2p-cache の提案

本章では前章の関連研究での長所短所の比較に基づき、P2P ネットワークに動的に木構造を導入しキャッシュを設けた Peer-To-Peer Cache(略して p2p-cache) を提案・実装する。

3.2 木の生成法

各ノードは P2P ネットワークに、動的に木を生成しながら加わる。この時、実装上の都合から、同時に 2 つ以上のノードが木に加わらない、とした。常に、一つのノードが木に加わるにより、再帰的に木が生成される。各ノードは木にランダムに加わるようにする。なぜなら、再帰的に形成される木の子供のみに加わるようにすると、直線的な木が生成され、検索効率が悪くなるからである。

木を生成するアルゴリズムは実用でのトレードオフを考え、単純なものにした。具体的な仮想コードを次に示す。

```
Join_tree(ノード 自分、 ノード 相手)
  If(自分のディスク容量 > 相手のディスク容量 * 3)
    自分は相手の子供を半分もらう.
    If(相手に親がない)
      自分は相手の親になる.
    Else
      Join_tree(自分、 相手の親);
  Else
    自分は相手の子供になる.
```

次に Join_tree() 関数について詳しく説明する.

- 自分のディスク容量の取得
本実装では、初期化の際にディスクの容量を決め、グローバル変数の int 型の disc に収納する. その値を取得する.
- 相手のディスク容量の情報の取得
TCP コネクションを張り、相手のディスク容量の情報を得る. 自分のノードは相手にディスク容量を要求するコードを送る. listen 状態にある相手のノードは TCP コネクションを張った後に、ディスク容量を要求するコードを受け取り、自分のディスク容量を相手に返す. 本実装では、HTML や XML を参考にして、全てのデータは文字列に変換して送る. 従って、ディスクの容量も文字列で送る.
- 再帰的に木に加わる位置を決定する
実用とのトレードオフを考え、単純なアルゴリズムで比較的バランスの取れた木になるようにするため、自分のディスク容量が相手のノードのディスク容量の3倍より大きいとき、自分は相手の親になれるとした. さらに、バランスを考えて、自分が相手の親になった時には相手の子供の半分をもらうとした. 相手の親になった自分はさらに再帰的に現在の親とディスク容量を比較して、自分の位置を決める. 親がないときには再帰的な位置決めはそこで終わる. 自分のディスク容量が相手のノードのディスク容量の3倍より大きくない時には、自分は親の子供になる.

このアルゴリズムは、非常に素朴で単純であるが、ランダムに木に加わったときには後で実験の章で示すようにバランスの取れた木を形成する.

また、ディスク容量の大きなノードが上位の階層に加わり易いようにしたため、実際に本システムに似た P2P システムを運用したときに、より高い効率が得られると予測できる。

3.3 検索のプロトコル

検索は図にあるようにあるノードが他のノードに Query を出し、ヒットしたノードが Query のフォワードされた経路で QueryHit を返すことで行われる。

キャッシュは、QueryHit がフォワードされた各ノードに対して行われる。

以下に、Query, QueryHit のフォーマット、検索の手順、実装の詳細の説明を行う。

3.3.1 Query, QueryHit のフォーマット

Query, QueryHit のフォーマットを図 3.1 に示す。
各項目は改行コードで分割されている。

Query	ihost	shost	id	ttl	file	num	
QueryHit	shost	id	hhost	rhost	file	ttl	num

図 3.1: Query のフォーマット

- ihost Query をフォワードしたノードの Internet Protocol(IP) アドレスを示す。図の場合、B ノードにフォワードされた Query の ihost は、B ノードの IP アドレスである。
- shost Query を出した元のノードの IP アドレスを示す。図の場合、ノード A の IP アドレスである。
- id 各ノードに固有の Query の識別番号である。最初は、0 から始まり、Query を出す度に 1 ずつ増加する。shost と id の組み合わせで P2P ネットワークにおける一意の識別番号が定まる。

- ttl Time to Live の略であり、一般には Query が破棄されるまでに P2P ネットワーク上で作られる hops の数をあらわす。Query を見たノードは、それぞれその Query の ttl を 1 だけ減らす。TTL が 0 になったとき、そのメッセージは破棄される。本実験の場合、ttl を利用してホップス数を測定した。
- file Query の要求するファイルの名前。アスキーコードで表現される。
- Num 実験を容易にするために特別に付加された Query の識別番号。遠隔のホストから各 P2P のノードに Query の命令を送るときに、順序を明らかにするのに用いられる。

QueryHit のフォーマット

- shost Query を出した元のノードの IP アドレスを示す。図の場合、ノード A の IP アドレスである
- id 各ノードに固有の Query の識別番号である。最初は、0 から始まり、Query を出す度に 1 ずつ増加する。shost と id の組み合わせで P2P ネットワークにおける一意の識別番号が定まる。
- hhost Query によって要求されたファイルが実際にあるノードを示す。
- rhost Query によって要求されたファイルがヒットしたノードを示す。キャッシュの情報がヒットした場合、そのキャッシュを行っていたノードが rhost になる。
- file Query の要求するファイルの名前。アスキーコードで表現される。
- ttl Time to Live の略であり、一般には Query が破棄されるまでに P2P ネットワーク上で作られる hops の数をあらわす。Query を見たノードは、それぞれその Query の ttl を 1 だけ減らします。TTL が 0 になったとき、そのメッセージは破棄される。本実験の場合、TTL を利用して、Query がフォワードされた数を測定する。
- Num 実験を容易にするために特別に付加された Query の識別番号。遠隔のホストから各 P2P のノードに Query の命令を送るときに、順序を明らかにするのに用いられる。

3.3.2 検索の手順

次のような手順で検索が行われる。図 3.2 を参照。

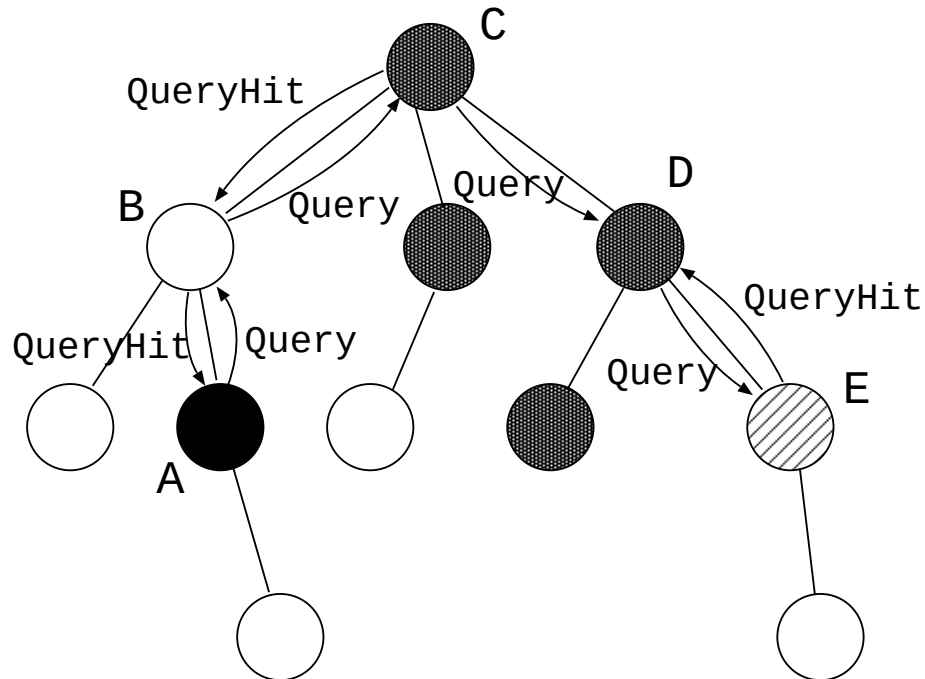


図 3.2: 検索

1. ノード A が Query を P2P ネットワーク上接続しているノードへブロードキャストする。
2. 初期化が完了し、listen 状態にあるノードが、Query メッセージを受け取る。
3. Query メッセージを受け取ったノードは、まず、その Query メッセージが以前に来たものか履歴を調べる。この履歴は STL のハッシュテーブルである set に保存されている。ハッシュのキーはノードの IP アドレスと各ノードに固有の Query の識別番号の組み合わせである。ハッシュには Query のフォワードされたノードの情報も保存されている。Query が以前に来たものであれば、そのノードはそれ以上は何もしない。初めて来た Query メッセージであれば、履歴に登録する。

次に、ノードは Query メッセージの要求するファイルがあるか、調べる。あれば QueryHit を Query をフォワードしたノードへ返す。ノードは Query メッセージの要求するファイルがあるか、キャッシュを調べる。このキャッシュはファイル名をキーとするキーの重複を許すハッシュテーブルである STL の multiset を用いた。理由はファイル名は一つではないからである。キャッシュにはファイルを持っているノード名も保存されている。キャッシュの中に、ファイル名があれば、QueryHit を Query をフォワードしたノードへ返す。

4. Query メッセージを受け取ったノードは、そのメッセージをフォワードしたノード以外の接続中のノードへ Query メッセージをフォワードする。
5. QueryHit を受け取ったノードは、その Query を出したノードが自分でない、または、キャッシュにその情報がのっていない限り、その情報をキャッシュに保存し、Query をフォワードしたノードへその QueryHit を送る。

3.4 実装の詳細

ここでは、上で述べた検索の手順、木の生成法を中心に、実装の詳細を説明する。

3.4.1 初期化

次の初期化を行う。

- ログの初期化
時間の取得 `time.h` に含まれる `time()` 関数を用いて取得する。この情報はプログラム終了時にログファイルに出力する。
ログファイル用のファイルストリームを開く。
Query、QueryHit の送受信、親ノード、子ノードについての情報を出力するファイルストリームを `stdio.h` に含まれる `fopen()` 関数を用いて開く。
- 自分の IP アドレスの取得
`unistd.h` に含まれる `gethostname()` を用いて、自分のインターネッ

トにおける名前を得る。次に、`netdb.h`に含まれる `gethostbyname()` を用いて `hostent` 構造体のポインタを得る。この構造体には、IP アドレスを文字列で表したポインタがあり、それを `arpa/inet.h` に含まれる `inet_addr()` でネットワークバイトオーダーの IP アドレスに変換する。

- 自分のディスク容量を決める
時間を種として乱数を生成し、それを利用してディスクの容量を決める。
- 自分の持っている検索対象となるファイル情報の取得
本実験では、ファイル情報は特定のファイルに収納してある。したがって、そのファイルを開き、内容を効率を考え、ハッシュテーブルに相当する STL の `set` に収納する。
- 通信を受けたときに対応する関数のポインタの配列の初期化
本実装では、`Query`、`QueryHit`、その他の通信は最初の 1 バイトで通信の種類を判別している。最初の 1 バイトを取得後、`int` 型に変換し得られた `int` 型の値を関数へのポインタの配列のインデックスに使用する。この関数へのポインタの配列の各要素は、それぞれの通信に対応した関数を指している。関数のポインタを収容した配列のインデックスにこの値を入れることで、それぞれの通信に応じた処理を行う。この段階ではこの配列にそれぞれの関数のポインタを割り当てる初期化を行う。
- 他のノードからの受信可能状態にする
`sys/socket.h`にある `socket()` システムコールを使用して、ソケットを生成する。ソケットの `family` は IPv4 プロトコル、`type` はストリームソケットを用いる。
次に、先に生成されたソケットにローカルなプロトコルアドレスを `bind()` システムコールを用いて、バインドする。
`listen()` システムコールを用いて、ソケットを `listen` 状態にする。

3.4.2 ノード間通信

ノード間の通信は、実装の簡略化のため通信の度にコネクションを張ることにする。効率良さとメモリの共有、排他制御が容易であり、他のオペ

レーティングシステム上でもソースコードレベルで互換性のある POSIX thread を用いた。

各処理ごとにスレッドを生成すると効率が悪くなるので、あらかじめスレッドを生成しておき、処理ごとに使いまわすようにする。

以下に具体的な手順を示す。

- メインスレッドは `pthread_create()` 関数を用いてスレッドプールを作る。スレッドプールにあるスレッドの一つは、`pthread_mutex_lock()` 関数を用いて `pthread_mutex_lock` を得て、排他制御下で `accept()` を行う。
- `accept()` したスレッドは `pthread_mutex_unlock()` でロックをはずし、他のスレッドがロックを得ることができるようにする。 `accept` したスレッドはまず 1 バイト読み込み、そのデータを `int` 型にキャストしてそれぞれの処理を行う関数のポインタの配列のインデックスに入れる。インデックスのポインタが指す関数を実行することでノード間の通信の処理を行う。
- 通信処理の終わったスレッドは、ソケットストリームを閉じ、スレッドプールに戻り次に `pthread_mutex_lock()` でロックを得ようと待っている。

ソケットストリームの読み込みは 1 バイトずつ `read()` システムコールを用いるとオーバーヘッドが大きくなるので、スレッドセーフな `read_line()` を用いて、ストリームの読み込みを行った。この関数は [9] の P. 617 P.619 を参考にした。

第4章 実験

Pentium 3 の 733MHz の CPU で、メモリが 512MB、ネットワークカードは 100Base-T の計算機を使用した。オペレーティングシステムはカーネルが 2.4.2-2 である RedHat Linux 7.1 を使用した。上記の計算機を 16 ノードで構成したクラスター上で実験を行った。ゲートウェイの計算機は、Celeron の 467MHz の CPU で、メモリは 128MB の PC でオペレーティングシステムは RedHat Linux でカーネルのバージョンは 2.2.14-5.0 である。

従来型の P2P システムと p2p-cache システムとの比較実験を行った。p2p-cache システムでは、各ノードがランダムに木に加わるため、実験の度に異なる形の木が生成される。詳しくは、図 4.1, 図 4.2, 図 4.3, 図 4.4 を参照。しかし、に示されるように、検索時間、流量に大きな変化は見られなかった。

実験は、クラスター上でネットワークの生成に使用されなかった計算機から、TCP コネクションを張って、ランダムにネットワーク上のノードへ検索内容を送り、検索内容を受け取ったノードは検索を実行するようにした。検索内容はあらかじめ perl で下の検索の条件に会うようにランダムに生成しておいた。

検索はすべて次のような条件で実行した。

- 各ノードはそれぞれ 5 つのファイルをもつ。ファイル名はネットワーク上で一意とする。これは、Napster や Gnutella のようなファイル検索システムの実情とは異なるが、一意でないとすると分析がさらに困難になるため本稿では、ファイル名はネットワークにおいて一意とした。
- ネットワーク全体で 8000 回の検索を行う。全部で 16 ノードなので、各ノードはそれぞれ平均 500 回ずつ Query を出すことになる。
- 各ノードは自分のもつファイル名を検索しない。本実験は実験は基

本的に Napster や Gnutella を意識したもので、自分の持っていないファイルを検索すると仮定したからである。

4.1 実験で生成したネットワーク

本実験では p2p-cache システムで作った木を 3 つと従来型の Gnutella モデルで作ったネットワーク一つでの比較を示す。以下、図 4.1, 図 4.2, 図 4.3, 図 4.4 に比較に用いるネットワークを示す。

Gnutella 型のネットワークでは一つのノードが他のノードと接続できる数は 4 までとした。

p2p-cache システムでは、ランダムに各ノードが木に加わるため、実験の度に異なる形の木をつくる。本稿では行われた実験のうち、図 4.1 でバランスのとれた木、図 4.2 で平らな木、図 4.3 でバランスの崩れた偏った木を示す。後で示すように、どの木でも、検索速度、メッセージ数についてほとんど同じ結果が得られた。また、図 4.4 で従来型のネットワークを示す。

4.2 実験結果

4.2.1 検索時間

以下に、検索をしてから、Query Hit が返ってくるまでの時間をそれぞれ図 4.5, 図 4.6, 図 4.7, 図 4.8 に示した。それぞれ、横軸に検索回数、縦軸に Query Hit が返ってくるまでの時間とした。ただし、縦軸の時間の単位はマイクロ秒である。

図 4.9, 図 4.10, 図 4.11, 図 4.12 に検索をしてから、Query Hit が返ってくるまでの時間の 100 回あたりの平均をそれぞれ示す。

検索時間の図からわかるように、実験を行ったクラスター上で 1 ノード間のキャッシュを用いない検索には約 1 ミリ秒かかる。図 4.5, 図 4.6, 図 4.7 から、ほぼどのような形の木でも同様のキャッシュの効果が得られることがわかる。

図 4.9, 図 4.10, 図 4.11 から 100 回ごとに検索時間の平均を取っても、同様のキャッシュの効果が得られることがわかる。ネットワーク全体で 2000 回、ノード一つ当たり 125 回を越える近辺からキャッシュの効果が顕著に

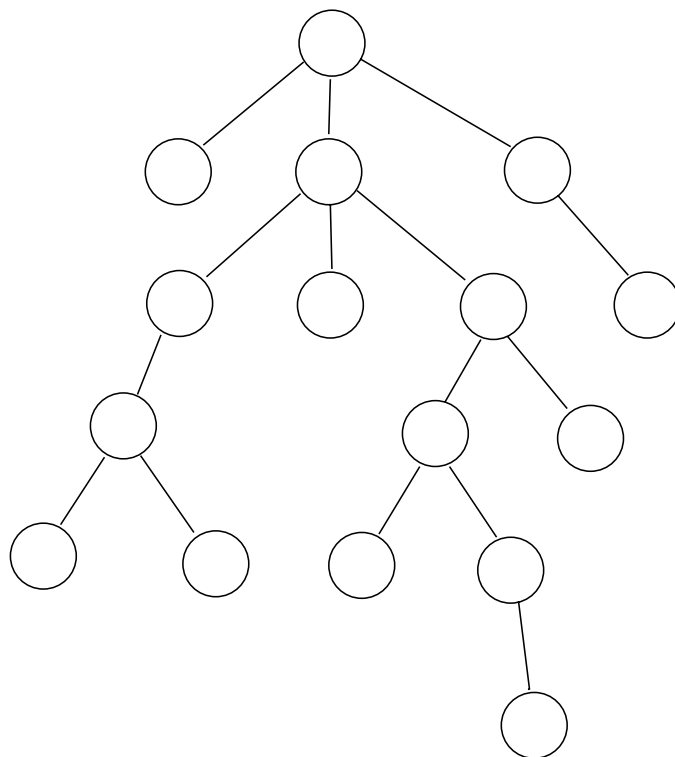


図 4.1: p2p-cache での木 A

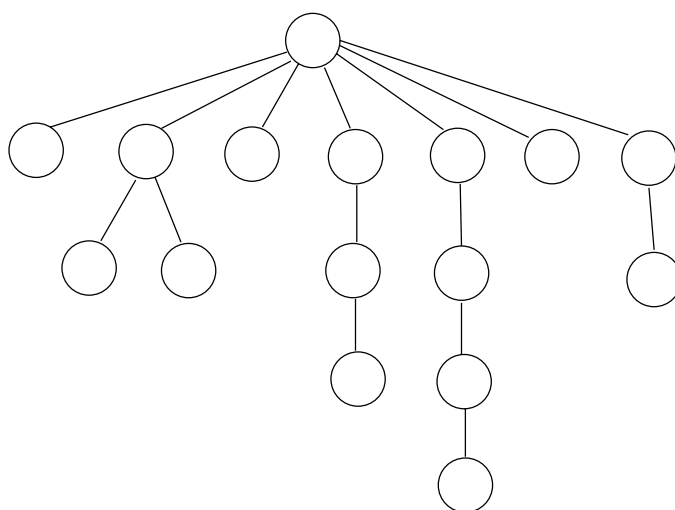


図 4.2: p2p-cache での木 B

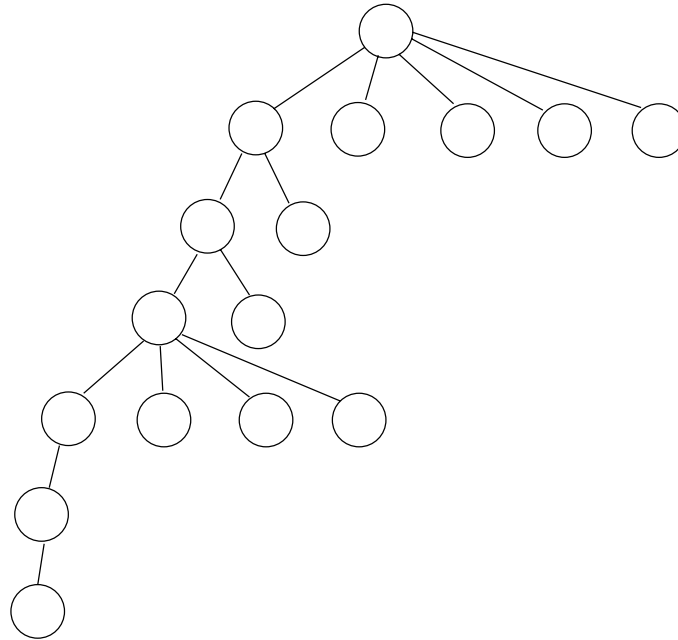


図 4.3: p2p-cache での木 C

効き始め、ほぼローカルなファイルを検索する時間と変わりなくなる。

図 4.8、図 4.12 からわかるように、キャッシュを用いない従来型のネットワークの場合、平均で 2300 ミリ秒かかる。キャッシュをとっていないため、検索回数によらず一定である。

4.2.2 メッセージ数についての測定結果

メッセージ数のグラフを図 4.13、図 4.14、図 4.15、図 4.16 に示した。p2p-cache のメッセージ数のグラフである図 4.13、図 4.14、図 4.15 から判るようにネットワーク全体の検索回数が 2000 回を超えたあたりから、一定になる。従来型の P2P ネットワークでは、流量はほぼ一定で、p2p-cache の場合の約 40 パーセント多く通信をしている。このことから、木構造を導入したことでネットワーク全体のメッセージ数も減らせることがわかった。

さらに、表 4.1 から、木構造を導入することで平均でメッセージ数が減少することがわかる。木構造が定まっていることで、検索の度に経路を

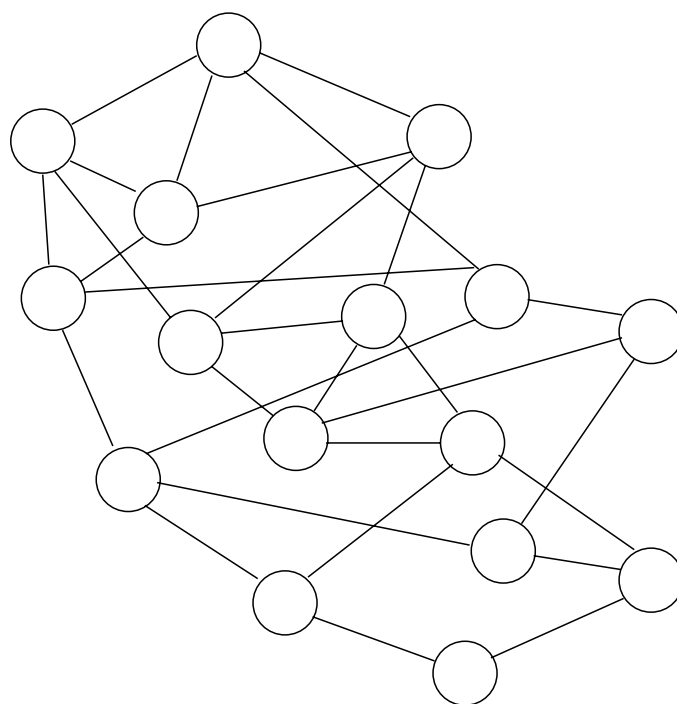


図 4.4: 従来型のネットワーク

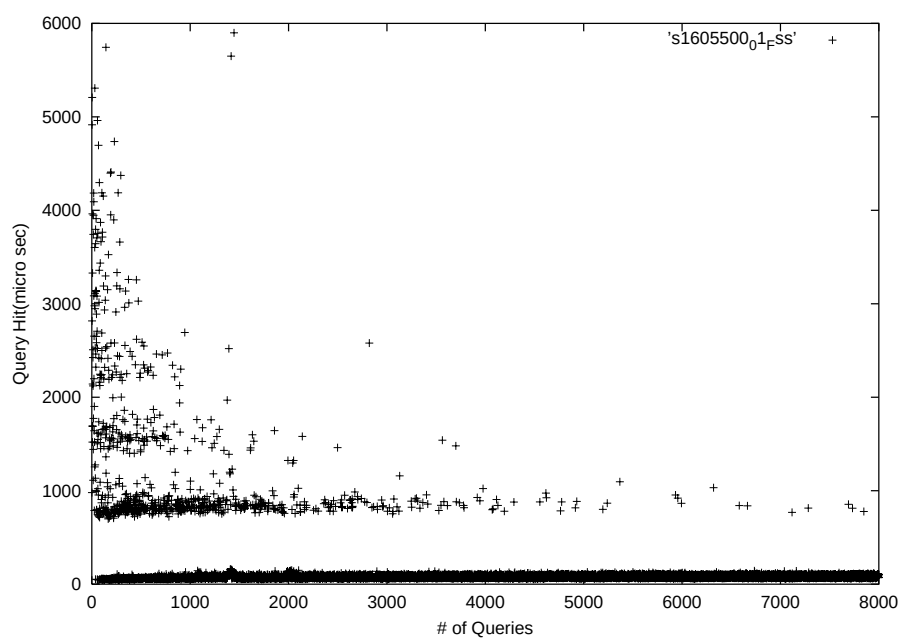


図 4.5: 木 A での検索の応答時間

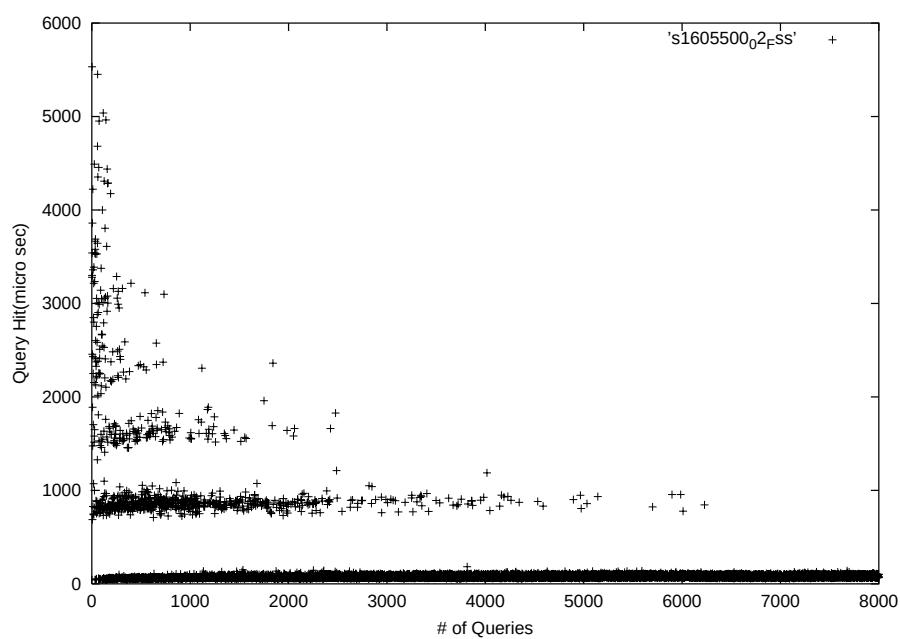


図 4.6: 木 B での検索の応答時間

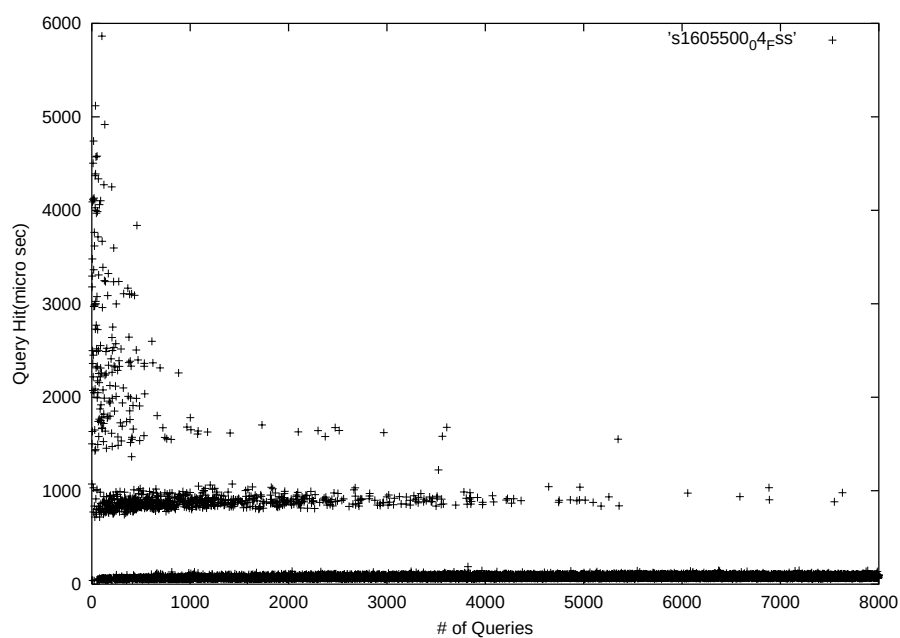


図 4.7: 木 C での検索の応答時間

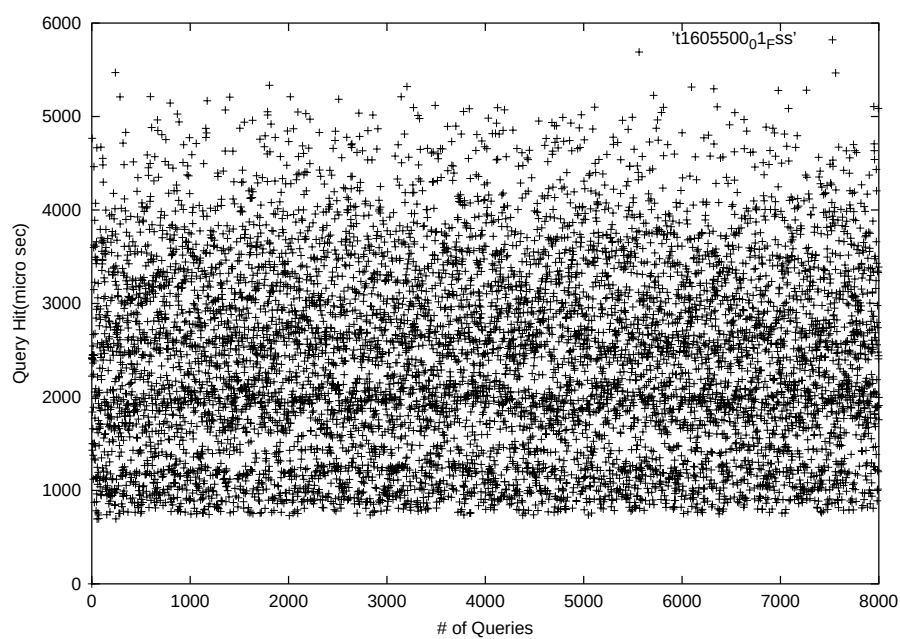


図 4.8: 従来型のネットワークでの検索の応答時間

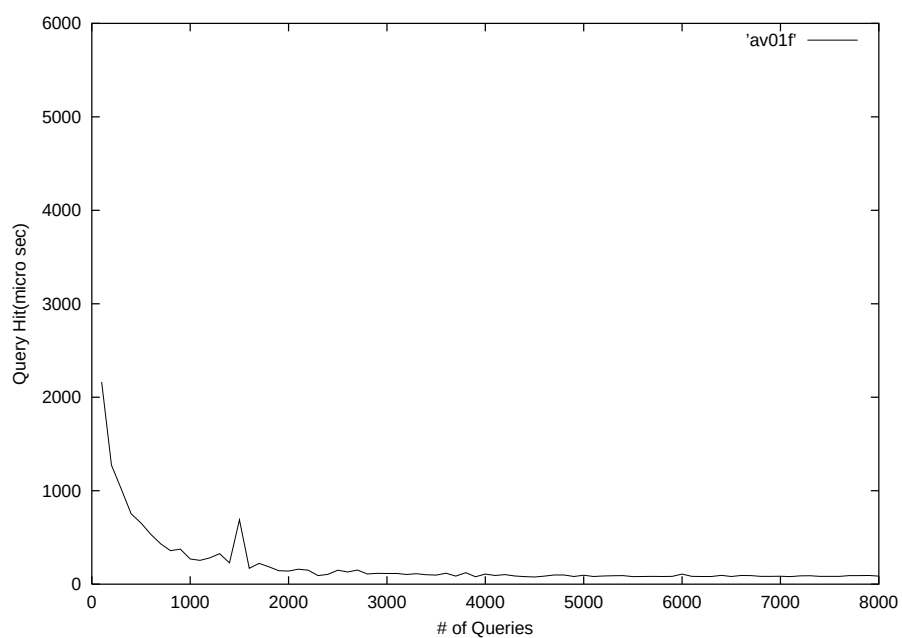


図 4.9: 木 A での検索の 100 回平均の応答時間

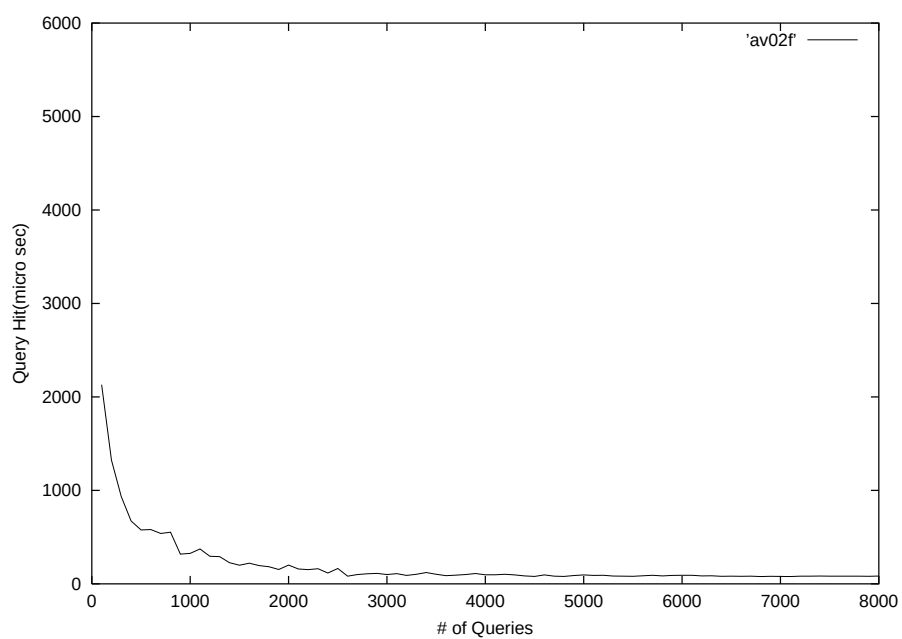


図 4.10: 木 B での検索の 100 回平均の応答時間

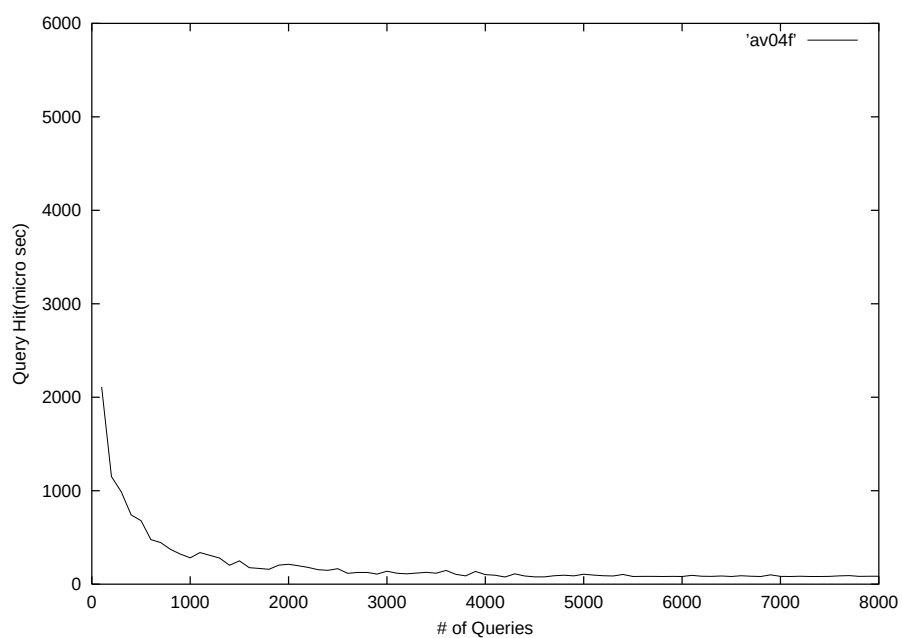


図 4.11: 木 C での検索の 100 回平均の応答時間

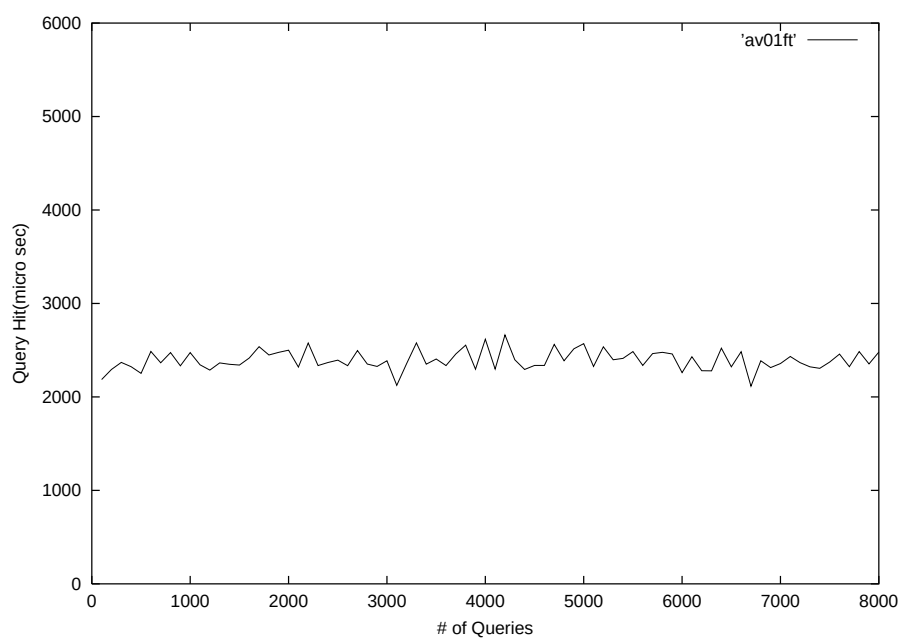


図 4.12: 従来型のネットワークでの検索の 100 回平均の応答時間

決定するオーバーヘッドが取り除かれ Query 数が大幅に少なくなっている．逆に QueryHit については、p2p-cache は大幅に増加している．Query はネットワークの全てのノードに到達するが、Query を受けたノードはその Query のファイル名がキャッシュに載っていれば QueryHit を返すのでその分のオーバーヘッドがかかる．このように返された QueryHit は、途中でそのファイル名がキャッシュに乗っているノードにたどりつけばそこでフォワードが中止される．

QueryHit の数は増加するが、Query の数が大幅に減少するため、合計のメッセージ数では p2p-cache システムの方が従来型の P2P ネットワークより約 40 パーセントメッセージ数が減少している．総合的に見て、ネットワークの帯域幅もより少なく済むことが示された．

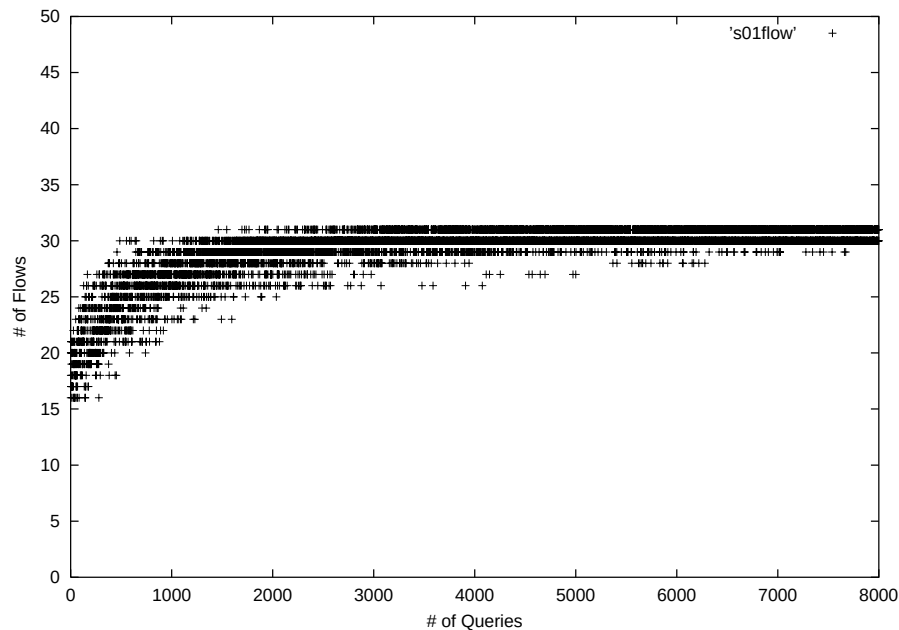


図 4.13: 木 A でのメッセージ数

4.3 実験結果のまとめ

以上の実験から、p2p-cache ネットワークでは検索での流量は約 50 パーセント、従来型の P2P ネットワークより減少した．さらに、検索時間につ

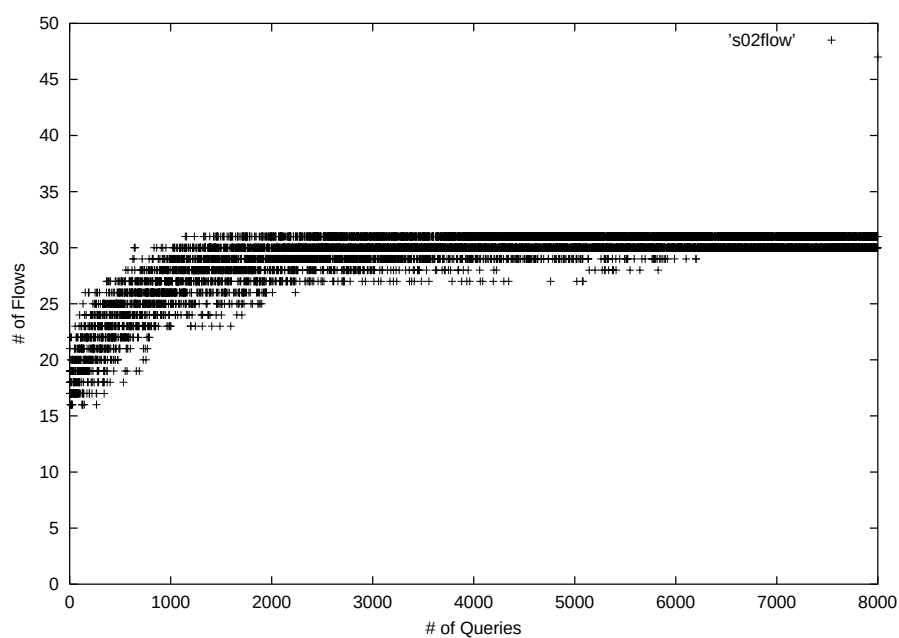


図 4.14: 木 B でのメッセージ数

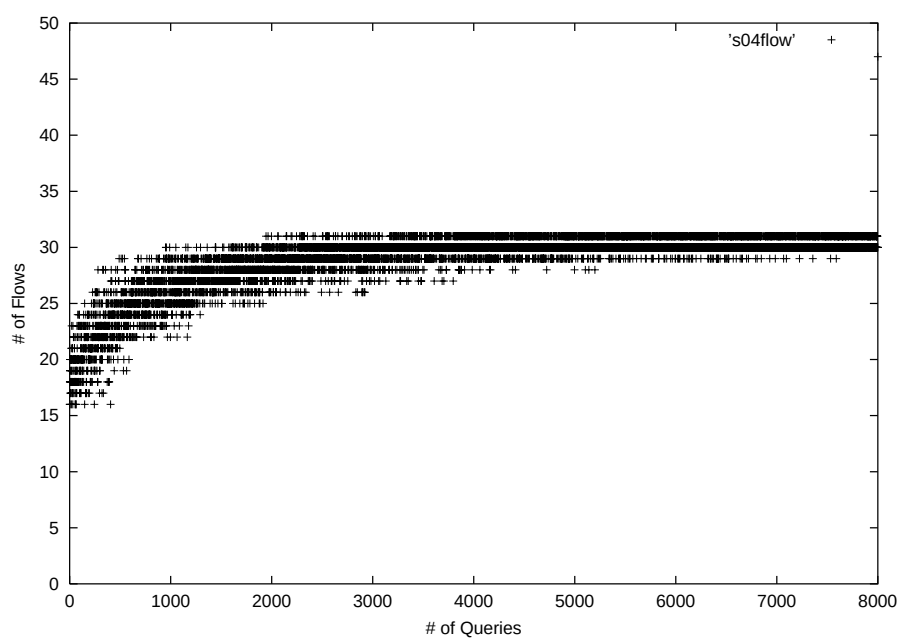


図 4.15: 木 C でのメッセージ数

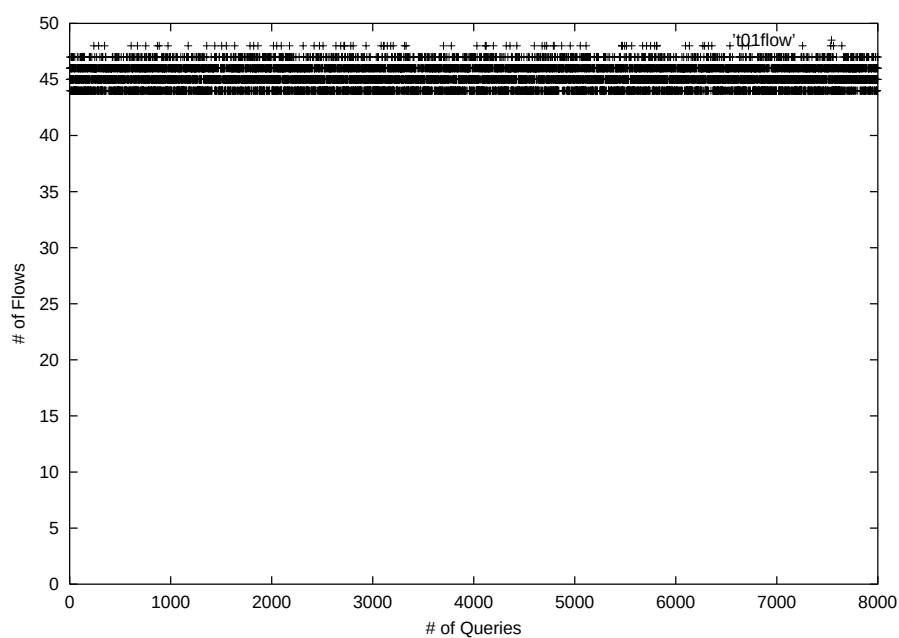


図 4.16: 従来型のネットワークでのメッセージ数

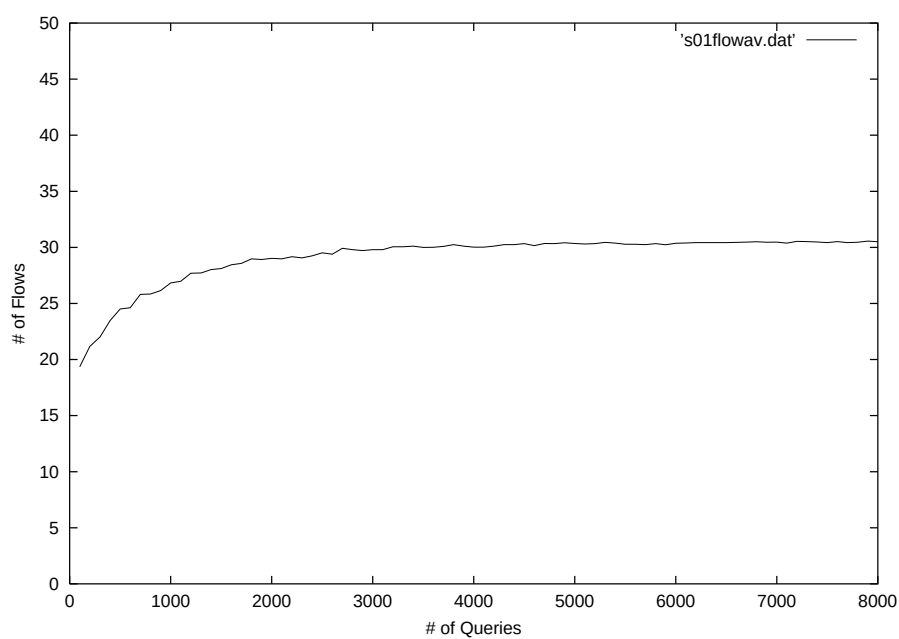


図 4.17: 木 A でのメッセージ数の 100 回平均

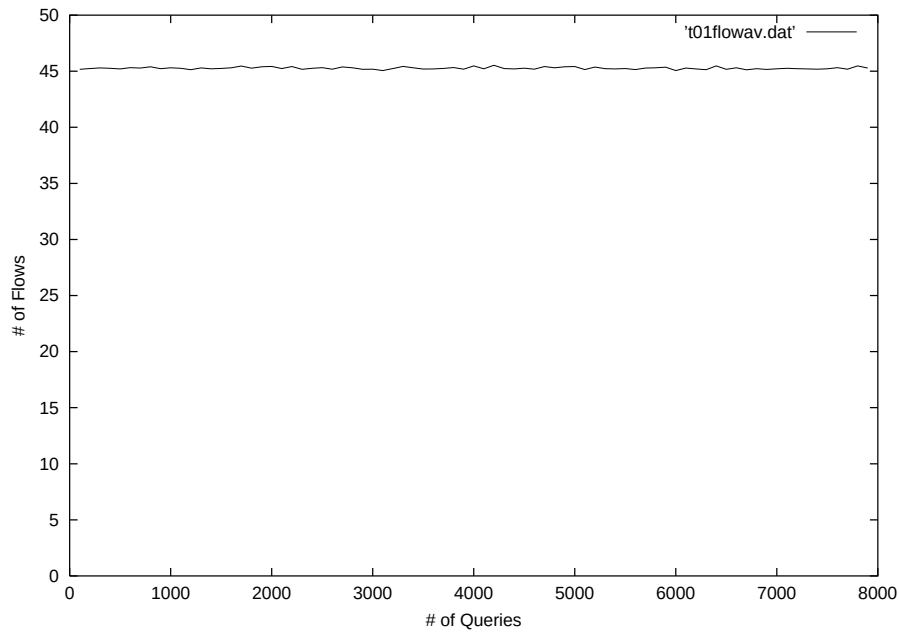


図 4.18: 従来型のネットワークでのメッセージ数の 100 回平均

表 4.1: Query の流量の平均の比較

	Query の流量	QueryHit の流量	合計の流量
p2p-cache	15.002	14.153	29.155
従来型のネットワーク	43.005	2.269	45.271

いても 5 つのファイルを持つ 16 ノードでランダムに検索した結果、2000 回以上すると 1/1000 の時間で済むことが示された。

これは、集中型、階層型のシステムの特徴である木構造を取り入れることで、検索の度に経路を決定するオーバーヘッドを取り除き、ネットワークの流量を減らすことができることが理由の一つに入る。さらに、経路をあらかじめ定めることによって、効率良くキャッシュができ、検索が高速になったことも理由として挙げられる。

第5章 まとめ

Gnutella に代表される Peer-to-Peer(P2P) システムは、完全分散型のネットワークを形成している。完全分散型のネットワークでは各ノードが互いに対等であり、フォールトトレラントである長所をもつ。すなわち、一つのノードが停止してもネットワーク全体に大きな影響を与えにくく、全体的に安定している。一方、このような完全分散型では、検索時に検索の経路を決定するためその分のオーバーヘッドが生じネットワーク全体の流量が増加する欠点をもつ。これは、すでに Query の到達しているノードへ Query を出すことが流量増加の原因になっている。さらに、検索の経路は実行時によって異なるため、効率的なキャッシュを行うのが難しい。以前にネットワーク上のノードが行った検索項目がキャッシュに載っていたとしても、それを効率よく利用できない。

本稿では、Gnutella に代表される Peer-to-Peer(P2P) システムの検索での欠点を指摘し、それに対処するシステム p2p-cache を提案、実装した。完全分散型の P2P ネットワークでは検索時に検索の経路を決定するためその分のオーバーヘッドが生じネットワーク全体の流量が増加する。そこで、本研究では完全分散型の P2P システムに集中型、階層型のシステムの長所である木構造を動的に取り入れ、さらに、キャッシュの実装を行い、検索の高速化を目指した。P2P ネットワークにノードが追加されるたびに木構造の検索経路を動的に決定し、検索時にその経路を使うようにした。さらに、その経路上で効率の良くなるようにキャッシュを自動的におくようにした。

本研究では、p2p-cache を実際に Linux 上に実装し、実験で検索速度を測定した。実験から、本実装でネットワークの流量を減らし、検索が高速になることが示された。p2p-cache ネットワークでは検索での流量は約 40 パーセント、従来型の P2P ネットワークより減少した。検索時間についても 5 つのファイルを持つ 16 ノードでランダムに検索した結果、2000 回以上すると 1/1000 の時間で済むことが示された。これは、集中型、階層型のシステムの特徴である木構造を取り入れることで、検索の度に経

路を決定するオーバーヘッドを取り除き、ネットワークの流量を減らすことができることが理由の一つに入る．さらに、経路をあらかじめ定めることによって、効率良くキャッシュができ、検索が高速になったことも理由として挙げられる．木構造を導入することでネットワーク全体のノード間の最大距離は増大したが、それは、検索の度に経路を決定するオーバーヘッドの除去で十分相殺できた

参考文献

- [1] Abraham Silberschatz, Peter Galvin, A. S.: *Operating System Concepts, 5th Edition*, John Wiley & Sons, Inc (1999).
- [2] Albitz, P., Cricket Liu 著 浅羽登志也 上水流由香監訳: *DNS & BIND*, アスキー出版局 (1995).
- [3] Antony Rowstron, P. D.: Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, *In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01). Chateau Lake Louise, Banff, Canada* (2001).
- [4] Dean Povey, J. H.: A Distributed Internet Cache, *Proceedings of the 20th Australasian Computer Science Conference, Sydney, Australia, February 5-7 1997* (1997).
- [5] Ian Clarke, Oskar Sandberg, B. W. and Hong, T. W.: Freenet: A Distributed Anonymous Information Storage and Retrieval System, *International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, ed. by H. Federrath. Springer: New York* (2001).
- [6] Meyers, S.: *Effective STL*, Addison-Wesley (2001).
- [7] Schildt, H.: *STL Programming*, Osborne/McGraw-Hill.
- [8] Stephen C. Dewhurst, K. T. S.: *Programming in C++*, PRENTICE HALL (1989).
- [9] Stevens, W. R.: *UNIX Networking Programming Volume 1*, PRENTICE HALL (1998).

- [10] Tanenbaum, A. S.: *Computer Networks*, PRENTICE HALL.
- [11] Wang, J.: A Survey of Web Caching Schemes for the Internet, *ACM Special Interest Group on Data Communication* (1999).
- [12] 石田晴久訳: プログラミング言語 C, 共立出版.