第 14 部

WWW キャッシュ技術

第 1 章

はじめに

W4C WG[1]は、近年インターネットにおいてそのトラフィックの大半を占めるといわれる HTTP[121][62] のトラフィック削減および、ユーザ側の視点から見たレスポンス向上に着目し、WIDE インターネット全域にまたがる広域 WWW キャッシュシステムの構築とその効果の検証を目的として設立された。

W4C WG の設立に先立ち、WWW Cache TF(Task Force) と呼ばれる、半ば非公認の研究グループが WIDE 内に存在した。WWW Cache TF には、議論を行うためのメーリングリストも存在し、1996 年 3 月ごろを皮切りに、幾度となく現状のインターネットにおけるトラフィック削減について議論が行われていた。

従って、WIDE における WWW キャッシュ研究の歴史は、現実には 1996 年 3 月ごろからと言ってよいであろう。WWW Cache TF 時代の主な成果物は、Internet Expo '96 の期間中に japan.park.org を用いて行った、日本のインターネットのトップ・キャッシュ・サーバの運営とその評価の研究である。

その後、本格的な活動の必要性が生じ、WWW Cache TF を発展的に解散し、全研究メンバを基本的に包括した形で、新たに本ワーキンググループが発足した。

W4C WG の活動期間の Phase I ('97.3 - '97.9) から、Phase II ('97.10 - '98.3) にかけての全体としての主な成果は、第 5章に述べる WIDE CacheBone 構築・運営作業である。W4C WG では、この作業と並行して、各種 WWW キャッシュ技術についての議論・検討・研究活動を重ねてきたので、本稿で紹介する。

まず、第 2章では、背景として 既存の WWW キャッシュ技術について概説を述べた後、第 3章において、ユーザのレスポンス向上に注視したアクティブキャッシング技術、次に第 4章で、トラフィック削減を前提とした分散キャッシュ技術などの研究について紹介する。

また、第 5章において、W4C の研究用テストベットとしての WIDE CacheBone の構築、第 6章において WIDE 合宿用ネットワークを用いて行った透過型代理サーバの実験について紹介する。

[1] W4C WG: WIDE World-Wide Web Cache Working Group の略。

第 2 章

背景

2.1 WWW キャッシュ技術

2.1.1 ファイアウォールとプロキシサーバ

大学や企業のネットワークにおいてはその組織内のネットワークとインターネットとの間に、セキュリティ上の防火壁であるファイアウォールを設置している場合が多い。ファイアウォールの種類によっては、組織内のネットワークにあるコンピュータからインターネット上の WWW サーバへ直接アクセスできないことがある。この問題を解決するために、WWW サーバへのアクセスを代行するプロキシサーバ (proxy server、代理サーバ) をインターネットとの境界部分に設置する。ブラウザはアクセス要求をプロキシサーバに送り結果を受け取ることにより、結果としてファイアウォールを意識しない透過的なアクセスを実現することができる。

プロキシサーバを使った典型的な構成を図2.1に示す。ブラウザが動作しているコンピュータである WWW クライアントからの要求はプロキシサーバに送られ、プロキシサーバが本来の WWW サーバへのアクセスを代行する。プロキシサーバには複数のクライアントからのアクセス要求が集まること、またアクセスを代行することから、次のような機能を持たせることができる。

1. プロキシサーバで、一度アクセスされた WWW データのキャッシュを行うことにより (キャッシング)、以降に同じデータをアクセスした同じまたは異なるクライアントに対して、キャッシュに保持していたデータを返すことができる。本来の WWW サーバにアクセスする必要がなくなるため、ネットワークのトラフィック及び WWW サーバの負荷の軽減、クライアントから見た応答時間の短縮が期待できる。キャッシュ機能を持たせたプロキシサーバのことを、特にキャッシングプロキシサーバ (caching proxy server) または単にキャッシュサーバと呼ぶ。

2. WWW データがプロキシサーバを経由する際に、データの加工や調査を行うことができる。例えば、日本語の漢字コードを変換することや、アクセスの統計情報を得ることに利用されている。

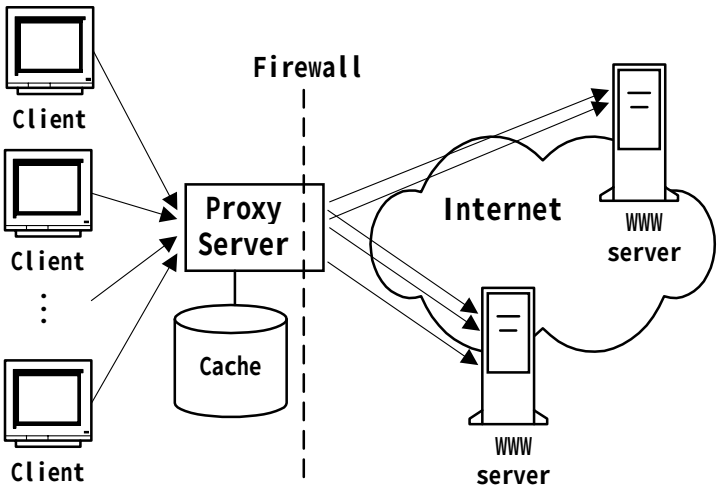


図 2.1: ファイアウォールとプロキシサーバ

3. 本来の WWW サーバに直接アクセスを行うのはプロキシサーバであり、クライアントの IP アドレスに代表される組織内のネットワーク構造などを隠蔽することができる。

2.1.2　プロキシサーバによるキャッシング

WWW サーバ (HTTP サーバ) に、ブラウザが直接アクセスするときと、プロキシサーバ経由でアクセスするときの違いについて説明する。直接アクセスするときブラウザは WWW サーバにファイルパスだけを送る[1]が、プロキシサーバに対してはブラウザは完全な URL をそのまま送る。プロキシサーバは URL を解釈し、URL に記述されたプロトコルでそのサーバに対してアクセスを代行し、得られた結果をブラウザに中継する。

CERN の WWW サーバ [122] は HTTP サーバであると同時に、WWW のプロキシ機能も持っていた。ファイアウォール製品の多くは標準的な機能として WWW のプロキシ機能を持つ。また日本で開発されたプロキシサーバとして、1994 年から電子技術総合研究所のメンバによって開発された DeleGate[123] がある。DeleGate は日本語や中国語の漢字コード変換や NetNews, CU-SeeMe などの WWW 以外のプロキシ機能を実現し独自の発展を遂げてきている。

CERN の WWW サーバは 1995 年頃まではキャッシューバとして広く使われていた。しかし、CERN のサーバは計算機に与える負荷が非常に大きく、WWW のトラフィックが増大するに従って満足な性能が得られなくなってきた。現在最も広く使用されているキャッシュサーバは Squid[124] である。

[1] 実際には多くのブラウザは “Host:アクセスしようとするサーバ名” という部分を要求に付加してくる。また、この Host:は HTTP/1.1 では必須となった。

ところで、プロキシサーバには全ての WWW アクセスのトラフィックが集中するため、その負荷が高くなりがちなことや、故障時には全てのクライアントからのアクセスができなくなってしまうことに注意しなければならない。また、１つのアクセスにつき TCP コネクションをクライアントと WWW サーバの両方に対して保持しなければならないため、多数のコネクションを保持できるオペレーティングシステム (OS) が必要とされ、またさらに後述のキャッシングのためには大容量のメモリや高速な入出力デバイスを持つコンピュータが必要とされる。

運用形態や利用者の構成によっても異なるが、大学や大企業の規模を対象にしたキャッシュサーバにおけるヒット率は、おおよそ 30%から 50%程度になることが実際の運用から得られている。すなわち、キャッシュサーバがないときと比較して、トラフィックを 3 分の 2 から半分程度に抑えることができる。しかしながら運用においては、キャッシュに置かれたデータが古くなり本来の WWW サーバ上のデータとの不整合が生じる可能性があること、キャッシュサーバへの処理の集中による新たなボトルネックが起こる可能性なども念頭におく必要がある。

2.2 分散キャッシュシステム

2.2.1 分散キャッシュ

Squid に代表される WWW キャッシュシステムの大きな特徴は、インターネット上で階層的なキャッシュ構造を実現できる点にある。図 2.2 に示すように、組織内だけでなくある地域や国などを単位にして階層構造を作ることで、キャッシュサーバの負荷の分散、トラフィックの軽減などが期待される。実際にインターネットで全世界的なキャッシュシステムの実験がすすめられている [125]。キャッシュサーバが階層構造を形成するには、サーバ間でのキャッシュ情報問い合わせプロトコルが必要となる。このプロトコルとして squid では ICP(Internet Cache Protocol; RFC2186)[126][127] を使用しており、あるデータが他のキャッシュサーバにあるかどうかの問い合わせメッセージ、またその応答メッセージなどの情報を交換しながら相互に動作する。

なお、Squid は米 National Laboratory for Applied Network Research (NLANR) で開発されているフリーな WWW キャッシュサーバである。コロラド大の Harvest Project [128] の一環として作られた Harvest Cached がベースになっている。Harvest Cached はその後 NetCache Cached として商品化され、現在は Network Appliance 社の製品となっている。

2.2.2 Squid キャッシュ

Squid キャッシュの特徴は以下の通りである。

- CERN httpd (W3C httpd) などの従来のキャッシュサーバと比べて速い。

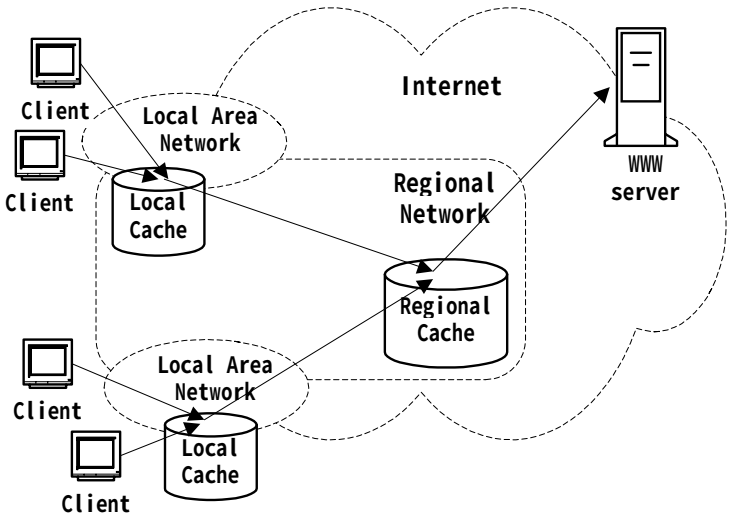


図 2.2: キャッシュサーバの階層構造

- リクエストごとに `fork()` せず、非同期 I/O を使って一つのプロセスで全リクエストを処理する (ただし、DNS 検索や FTP プロキシは別プロセス)。
- 後述する ICP を使い、複数のキャッシュサーバのキャッシュで連係して動作することができる。
- SSL や KeepAlive などの最新のプロトコルに対応している。

しかしながら、Squid は Unix の単一プロセスとして実装されており多数の要求を処理するためには、TCP 接続受入れの accept キューの限界、select 関数によるポーリングの限界、利用可能なファイルディスクリプタの限界などを解決していく必要がある。

2.2.3　キャッシュ連係プロトコル ICP

ICP (Internet Cache Protocol) とは WWW キャッシュサーバ間でのキャッシュデータのやりとりを行うためのプロトコルである。現在は RFC2186 で定義された ICP version 2 が広く利用されている。

ICP は基本的に以下のように動作する:

1. キャッシュサーバから neighbor サーバへ URL を指定して query を出す。
2. neighbor サーバは URL で指定されたオブジェクトが自分のキャッシュの中にあれば hit、なければ miss を response として返す。

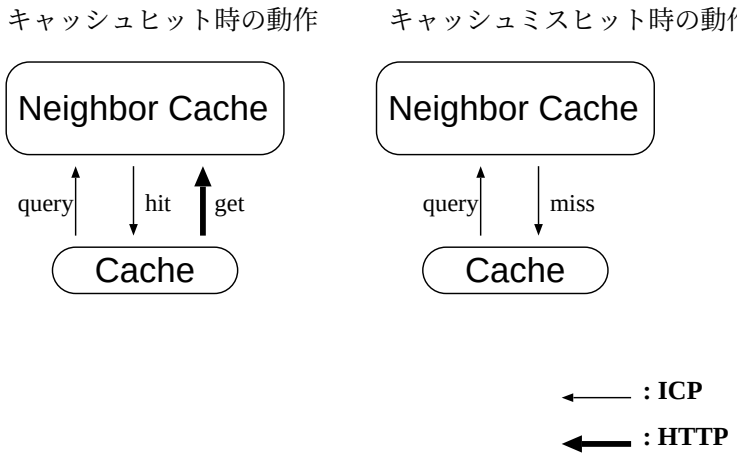


図 2.3: ICP の基本動作

3. キャッシュサーバは hit が返って来た neighbor サーバから HTTP でオブジェクトを転送する。

図 2.3に ICP の基本動作の概念図を示す。

Squid では、設定ファイルで指定した複数の neighbor サーバに対してこの ICP による問い合わせを行い、hit が返って来た neighbor サーバから HTTP でオブジェクトを転送することができる。また、以下のような処理も行っている。

- 複数の neighbor サーバから hit が返ってきた場合は、もっとも早く hit を返した neighbor サーバを利用する。
- neighbor サーバに明示的に重み付けを行なって優先的に使わせることもできる。
- どの neighbor からも hit がなかった場合、静的設定によってオリジンサーバまたは指定された parent サーバに HTTP リクエストを出す。

図 2.4に、Squid による ICP の利用状況を示す。

なお、ICP のような協調動作のためのメッセージを使用せずに、キャッシュ全体のヒット率を高く保つことを意図した分散キャッシュシステムも存在する。クライアントにおいてプロキシサーバ選択のための動作が記述できること [129] を利用して、アクセスしようとするオブジェクトの URL[130] 文字列をもとにキャッシュサーバに決定的にアクセスを行うことで分散処理を行う方式がある [131][132]。これらの方式では、同一の URL で表されるオブジェクトは同じサーバにアクセスされるため、ICP を使用しなくてもキャッシュのヒット率を維持したままでキャッシュサーバの分散が可能である。しかしながら、動作記述を設定できるのはブラウザに限られており、複数のキャッシュサーバから成る階層的なキャッシュシステムを自動的に構成することはできない。また、各キャッシュサーバにおけ

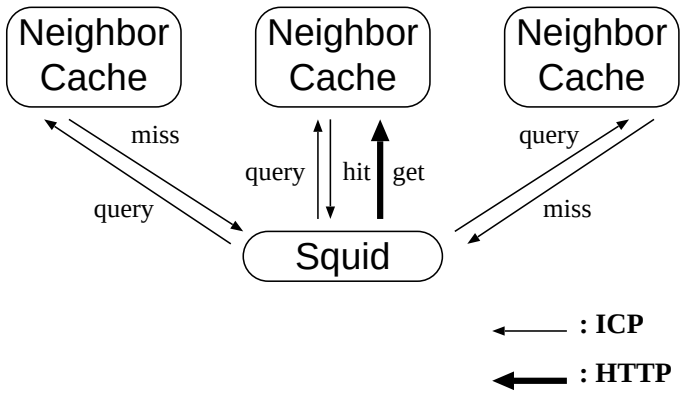


図 2.4: Squid での ICP の利用

る過去のアクセス記録をもとにして、キャッシュサーバでどのオブジェクトをキャッシュするかを事前に決定し、その情報を共有することでキャッシュ全体のヒット率を向上させる方式も提案されている [133]。

2.3　商用キャッシュサーバの動向 (CISCO Cache Engine)

キャッシュ技術が広く使われるとともに、いくつかのベンダーから商用のキャッシュサーバ製品が発売されるようになってきた。これらの製品の多くは既存の UNIX などの汎用オペレーティングシステムを用いたものだったが、最近は専用ハードウェアと組み合わせた製品もいくつか登場している。

ここでは、プロダクトとして Cisco 社の Cache Engine(CCE) を取り上げ、その特長について述べる。

2.3.1　Cache Engine の特長

CCE は大きく 2 つの部分から構成される。一つはエンジン本体となる専用ハードウェア部分である。製品としての CCE はこれを指すことになる。もう一つはこれをサポートするルータ側に搭載された WCCP (Web Cache Control Protocol) 機能である。WCCP は Cisco IOS に標準搭載されており、ハードウェアの導入と同時に利用可能になっている。

CCE 本体は、現在の製品は IBM-PC 互換機を元にしたものである。キャッシュ用の DISK として 32GB(8GB x 4) を搭載している。OS はリアルタイム OS をもとに構築されているが、詳細は不明である。

WCCP は、Cache Engine とルータ間の通信を定義するもので、これを設定するとルータは WWW サーバへの要求だけを目的のサーバではなく Cache Engine へ送信することに

図 2.5: 設定例

```
ip wccp
!
interface XXXX
ip web-cache redirect
!
end
```

なる。Cache Engine 自体をどのようにコントロールするかはルータ側で管理されることになる。

WCCP の設定自体も非常に単純でルータに図 2.5のような設定を追加するだけで運用することが可能である。

CCE を複数台おいてキャッシュファームを構築することが可能である、ルータは自動的にこれらを負荷分散しながら使用する。ネットワークトポロジーにそってキャッシュファームの階層的配置も可能である。

2.3.2 CCE の利点

利用者から見た場合、ルータのレベルで WWW サーバへのアクセス要求を CCE にリダイレクトするため、クライアント側にプロキシサーバの設定など特別に追加しなくても利用することができる。ユーザは CCE の存在を特別意識することなく利用可能になっている。

CCE 本体の運用が通常の UNIX などの運用と比較すると楽になっている。例えば shutdown の手順も電源を切るだけであり、手間がかからない。システムの再起動も、大変高速に行なわれる。

CCE のコントロール自体はルータ側で行うので、故障などによる CCE の切り離しや複数台の CCE への分散を自動的に行う事が可能であり、耐故障性や負荷分散を容易に実現することが可能である。

2.3.3 CCE の問題点

これは WWW キャッシュ全般に当てはまる問題であるが、アクセスされるサーバ側から見た場合 Source IP Address が CCE のものになってしまう。これは例えばクライアントの IP アドレスを元にした認証などが正常に働かなくなる可能性がある。特に CCE の場合ユーザがプロキシを利用するかどうかを選択出来ない為、問題を回避できない場合がある。また CCE のキャッシュポリシーは Squid のそれと比べ柔軟性に乏しく (現状全くないといっていい) 問題が発生する可能性が高い。

また、現状ではリダイレクトされた WWW アクセスの詳細な記録が取れないため、問題が発生した場合などに、記録を参照することができない。ISP などで顧客サポートが発生するような場合、記録が参照できないの問題が発生する可能性が高い。

さらに IOS の WCCP 関係の設定では、ルータの OutBound のインターフェースにしか Redirect の設定をすることができない。そのため CCE 設定の柔軟性がなく、意図しない接続に関しても CCE にリダイレクトされてしまうケースが存在する。これに関しては、InBound のインターフェース毎に Redirect するかしないかが選べるような改善が必要だと考えられる。

2.3.4　まとめと提案

2.3.5　CCE の利用に関する問題点

CCE はルータとの組合せで劇的なキャッシュ効果が期待できる反面、ISP としてこれを導入するには幾つかの問題点があると考えている。

プロキシ／キャッシュサーバとしてみた場合、細かいキャッシュポリシーを記述できない為、本来キャッシュしては困るような情報についてもキャッシュしてしまうケースがある。また HTTP アクセスが透過的に見えず、全て Cache Engine を介した接続となる。そして明示的にこれを外す手段がユーザに提供できない。

これらの影響は深刻で、例えば複数の接続ドメインの組織が同じ線を共有した場合、全てのアクセスは Cache Engine に設定した IP アドレスを仲介した形になる。そのためホスト単位で認証を書けているような WWW サイトへのアクセスが正常にできなくなる。

上記のような問題がある限り例えば ISP のバックボーンネットワークに対して適用することが不可能だと考えられる。また ISP のサービスの枠組の中の非常に限定されたケースで利用する事ができる可能性があるがこの場合、設備投資に見合う効果が期待できるかどうか分らない。

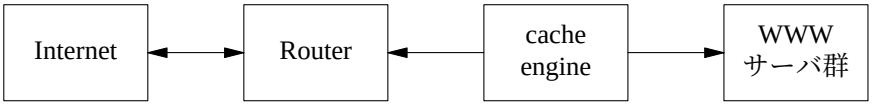
2.3.6　CCE に関する提案

一方キャッシュエンジンで導入されたルータとの組合せは、これまでできなかった WWW サーバとの連係などを実現する可能性があると考えている。 ここでは一つの提案として、キャッシュエンジンを WWW サーバのアクセレーションとして使用する事を提案したいと思う。

2.3.7　WWW サーバのアクセレーションとして利用

現在非常にアクセスの多い WWW サーバの負荷分散方法として、Local Director や Distributed Director などがあるが、Cache Engine にこれらを適応することが可能だと考えられる。すなわちアウトソースされた WWW サーバ群のあるセグメントに対して、キャッ

図 2.6: アクセレーションとしての配置例



シュエンジンを適用することで、対象となる WWW サーバ群全体に負荷分散機構を働かせることが可能になる。

複数のキャッシュエンジンをキャッシュファームとして構築し、WWW サーバ群に対するアクセスの負荷分散を図ることが可能だろう (図 2.6)。Local Director が個々の WWW サーバに対して負荷分散を図るのに対してキャッシュエンジンならセグメント全体に対して負荷分散することが可能になる。

しかし現在のキャッシュファームの実装は、対象となるホストの IP アドレス単位でキャッシュエンジンに個別に割り当てる実装になっている。この場合、キャッシュファーム全体で対象となる IP アドレスの負荷を分散させることができないので、IOS の設定にキャッシュファームへの割り当てポリシーを設定できるような拡張が必要になると考えている。

第 3 章

アクティブキャッシング技術 (先読み代理サーバ)

World Wide Web (WWW) におけるキャッシングに関する従来の研究は、トラフィックの軽減を目的とした、受動的なシステムの設計・開発が主体であった。近年、それ以外の高速な情報取得や新鮮な情報の収集等を目的とした、キャッシングを利用した能動的なシステムが研究されつつある。

本章では、これらのキャッシングを利用した能動的なシステムの研究の一つとして、W4C WG が取り組んでいる先読み代理サーバの設計と実装について述べる。先読み代理サーバは情報取得の高速化を目的とし、WWW 資源の先読みによって利用者からみた情報取得の高速化を実現する。

3.1 まえがき

コンピュータネットワークの普及にともない、高速な情報サービスが求められている。高速な情報サービスの実現には高速な情報取得が不可欠である。従来の高速化手法は高速ネットワークの実現によるサービスの高速化を目指していたが、その限界が認識されつつある。また、従来のサービスでは、情報が必要になった時点で情報を取得することを前提としていたが、大規模なコンピュータネットワークでは、中継装置の性能や信号の伝搬遅延が高速な通信の障害となる。特に インターネットでは、通信の多くが遠距離の通信であり、高速な通信の大きな障害となる。本研究は必要となる情報をあらかじめ取得することで通信を回避し、中継装置の性能や信号の伝搬速度に影響されない情報取得を実現し、利用者からみた通信の高速化を図る。

インターネット上の多くの情報サービスはクライアント・サーバのモデルに基づいて設計・実装されている [134]。そして、情報が必要な際にクライアントがサーバへ情報を要求し、サーバから情報を得る。本論文ではこの情報の取得モデルを「オンデマンド情報取得モデル」と呼ぶ。

オンデマンド情報取得モデルが広く利用されている原因は、クライアント・サーバモデルとの相性の良さとその実装の容易さによるもの、との認識が一般的である。オンデマンド情報取得モデルでは情報を必要とする際に情報の取得を行う。したがって、取得に要する時間は帯域と遅延に影響される。前述のように伝搬速度に依存した高速化手法は限界に

達しつつあり、オンデマンド情報取得モデルに基づいた情報サービスの高速化は非常に困難である。よって、情報サービスの高速化にはオンデマンド情報取得以外のモデルの導入が必要である。

本研究では、利用者の要求を待たずにあらかじめ情報を取得するモデルの導入を検討した。情報をあらかじめ取得することで利用ネットワークの帯域の差や遅延を吸収し、高速なサービスを実現する。本論文ではこのあらかじめ情報を取得するモデルを「事前情報取得モデル」と呼ぶ。本研究では、事前情報取得モデルに基づいて後述する先読み技術を情報サービスに導入し、情報サービスを高速化した。

3.2 WWW 先読みシステムの設計

本研究では、事前情報取得モデルに基づき、情報サービスにおいて利用者の要求を待たずにあらかじめ情報を取得する技術として、「先読み (prefetching)」を導入した。

先読みは従来よりメモリ (memory) やディスク (disk あるいは disc) 等の記憶装置における高速化技術として用いられている。記憶装置における先読みは、情報の参照の局所性に基づいて、情報をあらかじめ取得して、利用時の取得時間を短縮する技術である。情報サービスにおける情報の利用にも局所性があることから、この技術を情報サービスに導入することにより、情報サービスの高速化が大きく期待される。

一方、事前情報取得モデルは帯域や遅延に影響なく高速なサービスを提供できるが、以下の二つの問題が存在する。第一に内容が頻繁に変化するサービスでは、あらかじめ取得した情報が必要な時点では内容が変化している可能性がある。第二に提供する情報が多数あるいは大量なサービスでは、取得の際に大量のトラフィックが発生する。

本研究では、これらの問題を回避するために、サービスで提供される全ての情報を取得するのではなく、利用者の利用に応じて、近い将来利用されると思われる情報のみを取得する。これによって、取得した情報が取得後に変化する可能性を抑制し、大量の情報の取得を回避する。

WWW のサービスはページと呼ばれる単位で提供されている (図 3.1)。そして、ページは一つのネットワーク資源 (以下、資源)、あるいはハイパーテキストである。ハイパーテキストには、画像や音声等の資源の包含や、他のページへの参照が HTML によって記述される。また、資源は URL[135] によって識別される (図 3.2)。HTML を走査して URL を取り出すことによって、参照ページやページに含まれる資源に関する記述の抽出が可能である。利用者によるアクセスの多くは URL で指定された参照に沿って他のページを要求することであり、参照ページを先読みすることが応答時間の短縮となる。また、参照ページの応答時間を短縮するには HTML によるページの記述だけでなく、参照ページに含まれる資源も先読みする必要がある。参照ページに含まれる資源の先読みは、特にインラインイメージと呼ばれる画像資源がページに含まれている際に有効である。

ここで、ユーザが利用する (クライアントの要求する) ページを基ページ (base page) と

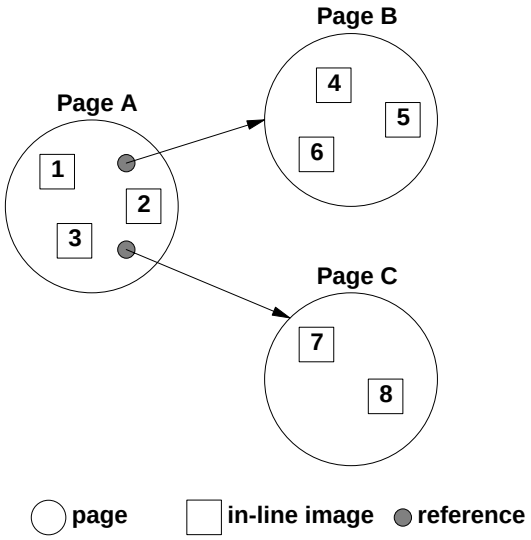


図 3.1: ページでみる資源の提供形態

呼ぶ。先読みシステムは基ページに関するクライアント要求処理を終了すると、基ページから参照されるページを取得し、各参照ページに含まれる資源を取得する。以上のアルゴリズムを疑似コードで示すと以下のようになる。

```
begin
  refs := SeekReferences(BasePage);
  foreach R in refs do
  begin
    Get(R);
    includs := SeekIncluded(R);
    foreach I in includs do
    begin
      Get(I);
    end;
  end;
end.
```

3.2.1　HTML 走査による先読み対象の推測

前述のように WWW のページは HTML で記述されている。HTML はタグ (tag) と呼ばれる形式で、ハイパーテキストの各種の指示を行っている。これらのうち先読みに関連するのは、参照ページを指示する A タグ、ページに含まれる資源を指示する IMG, FIG, BODY, FRAME, EMBED, APPLET タグである。これらのタグは以下のような形式で記述されている。

```
<A HREF="page" > ...
```

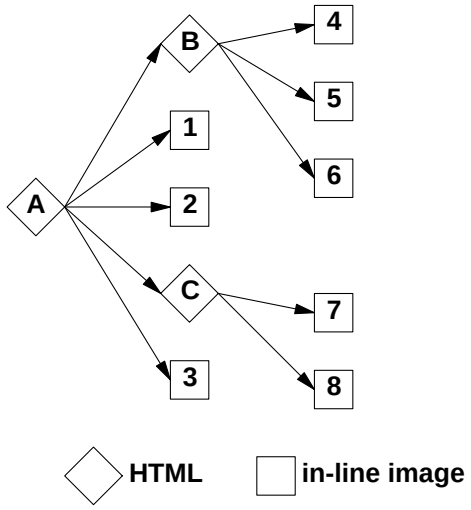


図 3.2: 取得する資源の視点でみる情報の構造

```
<IMG SRC="image" > ...
<FIG SRC="abc"> ...
<BODY BACKGROUND="image" > ...
<FRAME SRC="url">
<EMBED SRC="url">
<APPLET CODE="url">
```

本研究では HTML の走査によってこれらのタグを抽出し、先読み対象を推測した。なお、今後の WWW 関連技術の動向によって、新たなタグの追加が適宜必要である。

3.2.2 先読み対象の選択

各利用者が利用できる時間やコンピュータの能力、そしてネットワークの帯域は有限であるため、推測された先読み対象が大量に存在する場合には全ての対象を先読みするのではなく、選択して先読みする必要がある。本節では、WWW における先読み対象となる資源の効果的な選択方法を検討する。

まず、参照される資源の属性から選択する方法を検討する。応答時間を短縮する点を最優先に考えた場合には資源のサイズが重要であり、新たな資源を得る点を最優先に考えた場合には、資源の更新時刻あるいは破棄時刻が重要である。また、画像やテキストのような資源の種類を知ることができれば、種類に応じた選択方法が設計可能である。しかし、URL からその URL が示す資源のサイズ、更新時刻、破棄時刻を取得することは不可能である。また、一部の URL の末尾の拡張子による種類の類推を除いて、URL から種類の取得も困難である。したがって、本研究では資源の属性による対象の選択は実現できなかった。なお、ほとんどの属性はサーバに問い合わせることによって取得可能だが、大量の先

読み対象を選択する時点で、サーバに全ての資源の属性の問い合わせることは全ての情報を取得すると同等の処理が必要であり、多くの時間を必要とする。このことは、本研究の目的である高速化には適さない。

本研究では WWW のクライアントの挙動から先読み対象の決定方法を検討した。多くのクライアントは HTML の記述を評価し、HTML に記述されている順に資源を要求する。それらの資源のうち、ページの先頭にはページの説明やページを印象づける画像等が多く、利用者がもっとも高速に得たい資源であると考えられる。つまり、ページに含まれる資源は HTML に記述されている位置に応じて重要さが異なる。このことから、大量の推測結果のうち、先頭から数個を先読み対象として決定する。先頭から何個を先読みするかは、実際の運用時に定める。

3.2.3　システムの配置

本節では、先読み技術の WWW サービスへの適用として、先読みを実現したシステムの配置を検討する。先読みシステムの配置は、現状の WWW の情報の流れを妨げず、最小の変更で先読みを適用することが理想である。配置方法としては新たなプログラムの導入、そしてクライアントや代理サーバでの実装が考えられる。

クライアント、代理サーバ以外の新たなプログラムの導入は現状からの変更が大きいことから、本研究では新たな他のプログラムの導入を避け、クライアントあるいは代理サーバで先読みを行うことを検討する。

クライアントにおける先読みは、応答時間の大幅な短縮が望める。しかしながら、現状で利用されている多くのクライアントは実行形式で配布されたプログラムである。したがって、既存のクライアントへ先読み機構を追加することは困難である。そして、既存のクライアントに対して機能追加を行わず先読み機構をもったクライアントを新たに作成した場合、利用者への普及は容易ではない。以下、本論文では先読みを行うクライアントを「先読みクライアント」と呼ぶ。

次に、これらの障害を克服して先読みクライアントを実現した場合、個々のクライアントが独自のスケジュールにしたがって先読みを行うため、広帯域のネットワークが必要となる。そして、先読みを行うために先読みクライアントが稼働するそれぞれのコンピュータへ大きな負荷を与える。

代理サーバにおける先読みは、複数のクライアントからよせられる要求を基に行われることから、一つのスケジュールで先読みが行われ、重複を省くことが可能となり、先読みによって生ずるトラフィックを軽減することが可能である。代理サーバにおいて先読み結果を保存することによって、各クライアントの要求に対する先読み結果の共有も可能である。さらに、先読み結果の共有は従来のキャッシング方式を拡張して実現が可能であり、既存のクライアントを変更せずに先読みを導入することが可能である。本研究では、これらの利点から、先読み機構を付加した代理サーバを介した既存のクライアントのアクセスによって WWW サービスの先読みを実現する。以下、本論文ではキャッシュ機構をもつ代

理サーバを「キャッシング代理サーバ」、先読み機構をもつ代理サーバを「先読み代理サーバ」と呼ぶ。

また、クライアントと代理サーバで先読みを行う別の方法として、先読みクライアントとキャッシング代理サーバの組み合わせが存在する。この方式では、キャッシング代理サーバを用いて先読み結果の共有は可能だが、各々の先読みクライアントが独自のスケジュールで動作するため、多くの要求が発生しクライアントの稼働するコンピュータへの負荷が増加する。そして、キャッシング代理サーバはその膨大な要求を処理するため負荷が増加する。したがって、先読みクライアントとキャッシング代理サーバの組み合わせは、クライアントと代理サーバのトラフィック、そしてクライアントが稼働するコンピュータへ与える負荷の点で、既存のクライアントと先読み代理サーバの組合せに劣ると判断し、本研究では用いなかった。なお、前述のように先読みクライアントの開発と導入は容易でないことから、先読みクライアントとキャッシング代理サーバの組み合わせの実現と普及は容易ではないと思われる。

3.3 実装

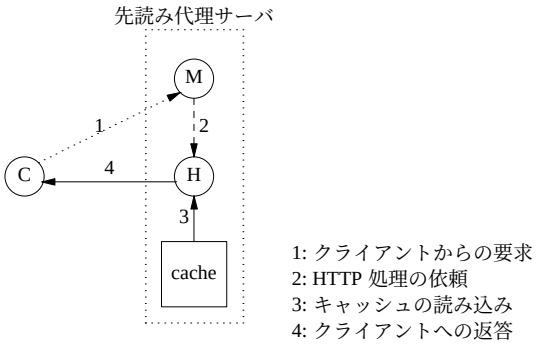
WWW サーバ、および WWW 代理サーバの運用は UNIX 上で行うことが一般的である。そこで、本研究では先読み代理サーバを UNIX 上で C 言語によってデーモンプログラムとして実装した。プログラムの大きさは約 12,000 行であり、POSIX で定められたシステムコールと BSD ソケットライブラリを利用して作成したため、その他の多くの UNIX 系の OS で動作するものと思われる。既に、以下のような UNIX 系の OS 上での動作を確認した。

- SGI IRIX 5.3
- DEC OSF/1 3.2 (Digital Unix)
- SUN SunOS 5.4 (Solaris 2.4), 5.5.1 (Solaris 2.5.1)
- BSDI BSD/OS 2.1
- SONY NEWSOS 6.1.1, 6.1.2

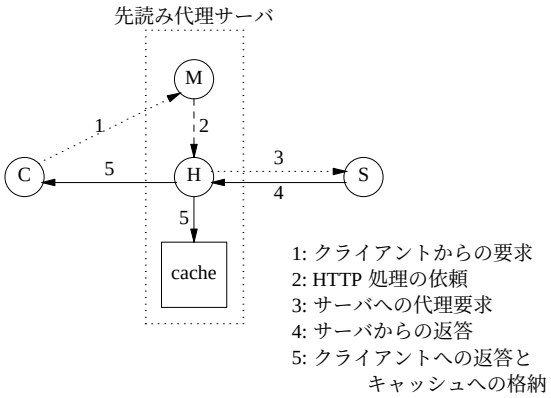
以下、負荷分散のためのモジュール分割、具体的な処理手順とスケジューリングについて述べる。

3.3.1 モジュール分割

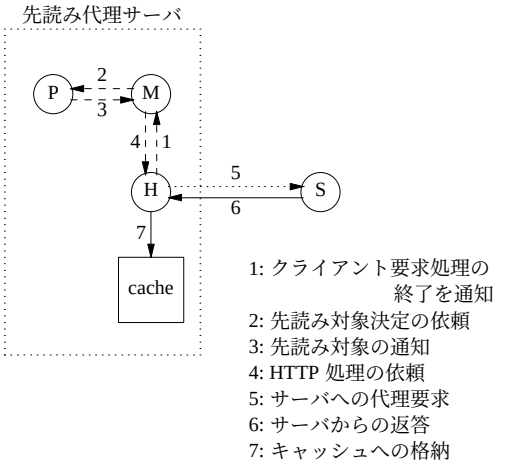
WWW で用いられる HTTP[62] のアクセスはバースト的で、サーバや代理サーバの稼働するコンピュータで大きな負荷が発生することが知られている。また、前述のように先読



(a) クライアント要求処理
キャッシュに情報がある場合



(b) クライアント要求処理
キャッシュに情報がない場合



(c) 先読み処理

Ⓒ クライアントⓈ サーバ

Ⓜ マネージャ部Ⓗ HTTP 処理部Ⓟ 先読み対象決定部

要求　　返答　　処理依頼

図 3.3: 処理手順

み機能は同時に多くの通信が必要であり、従来のキャッシング代理サーバに比べ、より大きな負荷の発生が予想される。サーバが稼働するコンピュータへの負荷の軽減には、”pre-forking” および ”file descriptor passing” と呼ばれる手法が有効であることが知られている [136]。実装した先読み代理サーバでは pre-forking を行うため、先読み代理サーバを処理内容に応じてモジュールに分割した。

分割したモジュールには、要求の受け付けとスケジューリング、各モジュールの管理をする「マネージャ部」、HTTP を処理する「HTTP 処理部」、先読み対象を決定する「先読み対象決定部」、キャッシュを整理する「ごみ集め部」がある。各モジュールは、プログラム起動時に fork システムコールによって分岐し、それぞれ単独のプロセスとして動作する。特に、同時に多数の HTTP の制御するため、HTTP 処理部を多数用意する。各モジュールは sendmsg, recvmsg システムコールを用いて制御情報 (クライアントからの要求等) や file descriptor を交換する。

3.3.2 処理手順

先読み代理サーバにおける処理を大別すると、クライアントからの要求の処理 (クライアント要求処理) と、クライアント要求処理から派生する先読みのための処理 (先読み処理) に分かれる。本節では、各処理の手順について述べる。

クライアント要求処理:

先読み代理サーバは、従来のキャッシング代理サーバと同様に、クライアントからの要求に対して代理アクセスとキャッシュへの格納を行う。

まず、クライアントからの要求をマネージャ部が受取り、HTTP 処理部へ処理を委ねる。HTTP 処理部は、キャッシュに資源が存在した場合には、キャッシュ内容をクライアントへ返答する (図 3.3(a))。キャッシュに資源が存在しない場合には、サーバへ資源の要求とキャッシュへの格納を行い、クライアントへ返答する (図 3.3(b))。

先読み処理:

先読み代理サーバは、クライアント要求処理後に先読み処理を実行する。前述のように先読み処理は、参照ページ先読みとページに含まれる資源の先読みがある。以下、それぞれの先読みについて述べる。

参照ページ先読み処理:

クライアント要求処理時にサーバから取得した資源が HTML によるページの記述であるとき、先読み対象決定部が取得内容から A タグによって記述されている参照先の URL を取り出して先読み対象に決定する。決定した先読み対象それぞれに対して HTTP 処理部がサーバへ資源の要求を行い、資源をキャッシュへ格納する (図 3.3(c))。

ページに含まれる資源の先読み処理:

参照ページ先読み処理時にサーバから取得された資源が HTML によるページの記述であるとき、先読み対象決定部が取得内容から IMG 等のタグで記述されているページに含まれる資源の URL を取り出して先読み対象に決定する。決定した先読み対象それぞれに対して HTTP 処理部がサーバへ資源の要求を行い、資源をキャッシュに格納する。

3.3.3　処理スケジューリング

代理サーバはクライアント要求処理と先読み処理の 2 種類の処理を同時に多数行わなければならないが、同時に処理できる数は限られていることから、待ち行列を設けてこれらの処理を待機させる。そして、処理が可能となった際に HTTP 処理部へ処理を割り当てる。どのように待ち行列から処理を取り出して実行するかは処理の種類によって異なる。以下、各処理における待ち行列からの取り出し方と HTTP 処理部への処理の割り当て方について述べる (図 3.4)。

クライアント要求処理スケジューリング:

クライアント要求処理において、要求は待ち行列に蓄えられ、先頭から一つずつ取り出される。取り出された要求は、待機中の HTTP 処理部へ割り当てられる。また、待機中の HTTP 処理部が存在しない場合には、先読み処理中の HTTP 処理部の処理を中止させ、クライアント要求処理を割り当てる。

先読み処理スケジューリング:

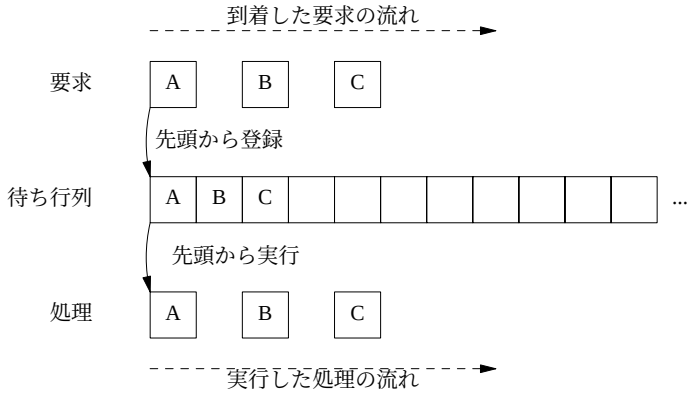
先読み代理サーバの目的は応答時間の短縮である。先読みは利用者から要求される前に実行されると大きな効果をもたらすことから、新たに発生した先読み処理の実行は、それ以前の先読み処理の実行に比べ、先読みの効果が大きい。したがって、複数の先読み処理を実行する際には、新しい先読み要求を優先的に処理することが望ましい。

先読み処理のスケジューリングでは、待ち行列の最後の先読み要求を取り出す。取り出された要求は、待機中の HTTP 処理部へ割り当てられる。また、極端に古い先読み要求は、処理しても先読みの効果が薄いことと、待ち行列の有効利用のために破棄する。

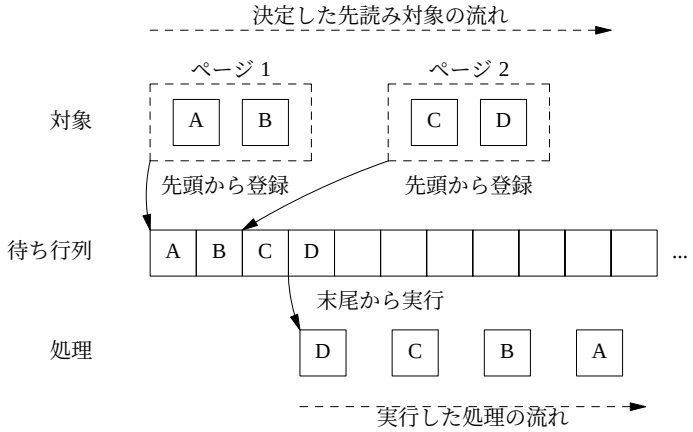
3.4　運用による評価

本研究で実装した先読みサーバを実際に運用して挙動を観察した。

前述のように本研究では、代理サーバで利用されたページの HTML による記述を走査して、先読み対象を決定する。現在のコンピュータとネットワークの性能では、参照される全てのページの先読み、およびページに含まれる資源の全てを先読みすることは困難で



(a) クライアント処理のスケジューリング



(b) 先読み処理のスケジューリング

図 3.4: 待ち行列の操作

表 3.1: 計測した代理サーバの環境

モデル	Sun Ultra Exterprise 3000
OS	SUNOS 5.5.1 (Solaris 2.5.1)
CPU	Ultra SPARC (168 MHz) x 4
メモリ	256MB
使用キャッシュ領域	500MB (ハードディスク)
ネットワーク	Ethernet (10Mbps) FDDI (100Mbps)

表 3.2: 計測および算出結果

計測期間	連続 12 日間
要求発行ホスト数	250 ホスト
受信要求数	約 33 万件
平均要求数	0.34 件/秒
先読み回数	約 37 万件
ヒット率 (先読みあり)	51.4 %
ヒット率 (先読みなし)	33.6 %
サーバ・代理サーバ間トラフィック	5.79 GB
代理サーバ・クライアント間トラフィック	2.92 GB

ある。そこで、今回は参照ページを 10 個、参照ページに含まれる資源の先頭の 10 個を先読み対象として設定した。

以上のような戦略を用いた先読み代理サーバを、奈良先端科学技術大学院大学のキャンパスネットワークにおいて運用した。先読み代理サーバの環境を表 3.1に、連続 12 日間の計測から各項目を算出した結果を表 3.2に示す。

先読み代理サーバの利点は利用者からみた応答時間の短縮である。現状では、クライアントにおける応答時間を厳密に測定する方法はなく、現在用いられている多くのクライアントを、応答時間が測定するよう変更することは非常に困難である。そこで、本研究では代理サーバにおける測定で応答時間を近似する。取得時間は応答時間の最大値と考えることができることから、代理サーバがクライアントより要求を受け取り、オリジンサーバから受けた資源のクライアントへの送出が終了した時間 (セッション生存時間 (session lifetime) ; 図 3.6) をクライアントにおける応答時間とみなす。

上記のネットワークでは、各クライアントから代理サーバへの経路は FDDI と Ethernet

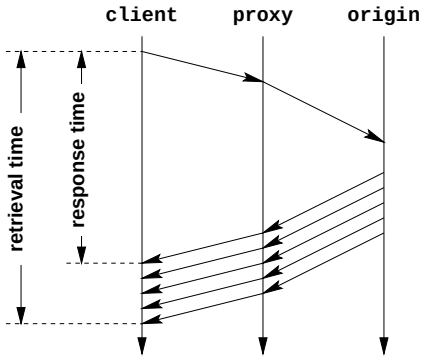


図 3.5: 取得時間と応答時間

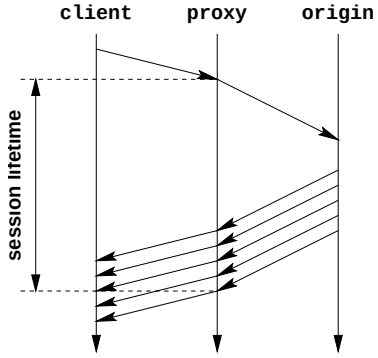


図 3.6: セッション生存時間

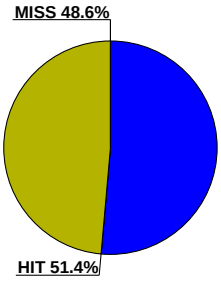


図 3.7: 観測期間全体のヒット率

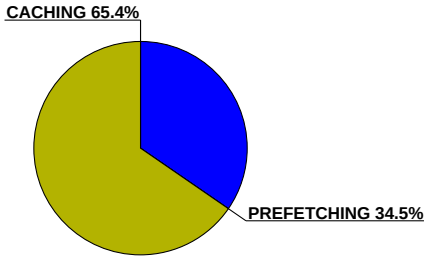


図 3.8: ヒットの内訳

で構成され、帯域とトラフィックには十分に余裕があった。広く用いられている ping プログラムによる RTT (Round Trip Time) の計測では、多くのクライアントから代理サーバへの平均 RTT は 0ms から 2ms 程度であった。したがって、クライアントと代理サーバ間の取得時間は、代理サーバとオリジンサーバとの取得時間に比べ比較的短いと考えて良い。

この結果、ヒット時のセッション生存時間の大部分は 1 秒以内、ミス時のセッション生存時間の大部分は 30 秒以内であった (図 3.9)。そして、先読み代理サーバ全体のヒット率は 51.4% であり (図 3.7)、キャッシング代理サーバに比べ約 1.5 倍のヒット率を得た (図 3.8)。すなわち、先読みは応答時間が著しく短いヒットの割合を増加させ、クライアントからみた全体の応答時間を短縮することが明らかとなった。また、先読み代理サーバの高ヒット率は一時的な現象ではなく、キャッシング代理サーバに対する優位性が持続することも明らかとなった (図 3.10)。

一方、先読みの欠点はトラフィックである。代理サーバとオリジンサーバ間のトラフィックはクライアントとオリジンサーバ間のトラフィックの約 1.9 倍であった。

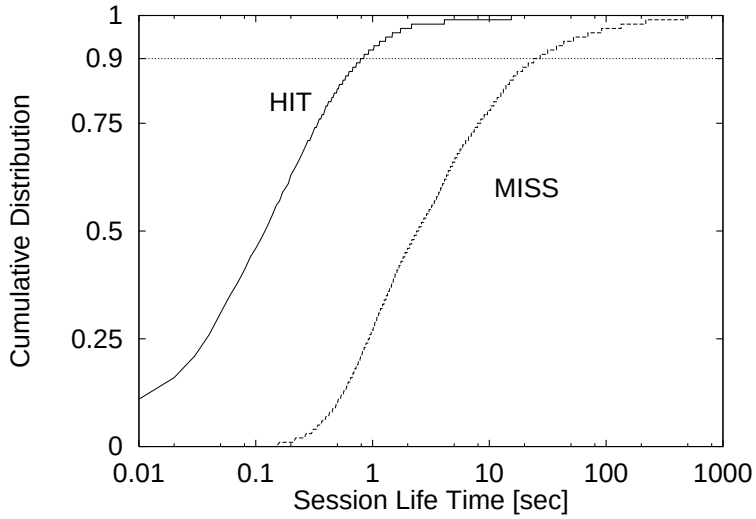


図 3.9: キャッシュヒットとミスのセッション生存時間

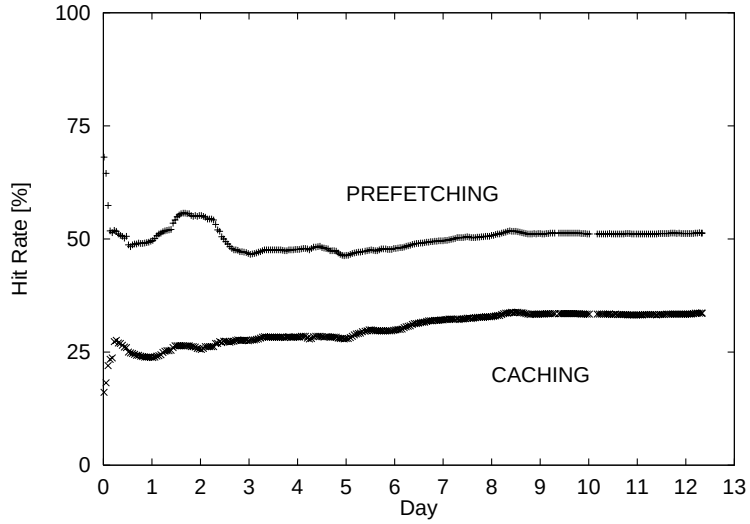


図 3.10: ヒット率の経時変化

3.4.1　考察

先読み代理サーバを実際に運用した結果、先読み代理サーバはキャッシング代理サーバに比べ、クライアントからみた応答時間を改善し、トラフィックが増加した。一般にネットワーク帯域の増加では高速化は期待できないが、先読み技術を導入することによって利用者からみた応答時間が短縮でき、高速なサービスが実現できることが明らかになった。現在では広帯域ネットワークが普及していることから、ネットワーク帯域を必要に応じて拡大することが可能であり、先読みは今後の情報サービスの高速化技術として有望である。

また、運用した先読み代理サーバの戦略は単純であったことから、今後、要求される資源を的確に推測する先読み対象決定戦略を開発することによって、さらに良い性能が期待できる。

3.5　議論

なお、HTTP/1.1[121] では、継続的コネクション (persistent connection) やパイプライニング (pipelining) と呼ばれる手法が開発されている [137]。これらは HTTP のプロトコルオーバヘッドの解決策であり、オンデマンド情報取得モデルの範疇であることから、従来と同様に先読み技術が適用可能である。本研究ではプロトコルの改善は主眼としていないため、本論文ではこれらの方式の詳細については言及しない。

3.6　まとめ

現在、高速な情報サービスが求められている。従来の情報サービスの高速化手法が中継装置の処理速度や伝搬速度に依存しており、高速な情報サービスの実現は困難であった。それらの手法が用いられた原因は、従来の情報サービスがオンデマンド情報取得モデルに基づいて構築されていることによる。そこで、本研究では、中継装置の処理速度や伝搬速度に依存しない情報取得モデルとその取得モデルに基づいた情報サービスの高速化手法を提案し、具体的なシステムを構築した。

本研究で提案した情報取得モデルは事前情報取得モデルである。事前情報取得モデルでは、利用者の情報を要求する前にあらかじめ情報を取得し、利用者の利用に備える。事前情報取得モデルに基づき、本研究では情報サービスに先読みを導入した。そして、具体的な情報サービスとして、インターネット上の代表的な情報サービスである WWW を対象とした。

先読みシステムの導入の容易さと導入後の効果を検討した結果、WWW における先読みシステムは代理サーバでの実現が適切であると判断した。代理サーバにおける先読みはクライアントの変更が必要なく、複数のクライアントのリクエストに対する先読み結果を共有できる。従来の代理サーバに先読みを実現する機構を追加することで、先読み代理サー

バを設計・実装した。先読み代理サーバはクライアントへ送出したネットワーク資源のうち、HTML によるページの記述を走査して先読み対象を決定する。

実装した先読み代理サーバを実際の環境で運用し、以下のような点を明らかにした。

- 事前情報取得モデルに基づく先読みは中継システムの性能や帯域や遅延の影響を受けずに高速なサービスが実現可能。
- オンデマンド情報転送モデルに対して高速なネットワークの導入以外の手段で高速化が可能。
- オンデマンド情報転送モデルに基づいた情報サービスである WWW は事前転送モデルに基づいた先読み技術で高速化可能。
- インタラクティブ先読みはオンデマンド情報転送モデルの拡張であるキャッシング方式に比べ高速なサービスが提供可能。

実装と運用を通して、先読み代理サーバは高速なサービスを実現することが可能であることを明らかにした。

以上のように、本研究では事前情報取得モデルの提案、WWW に対する先読みの導入、先読みを実現する先読み代理サーバの設計と実装を行った。そして、実際の運用を通して代理サーバの評価を行い、先読みの効果を実証した。

第 4 章

分散キャッシュ技術（WebHint システム）

4.1　はじめに

分散キャッシュシステムでは、クライアントから要求を受けたキャッシュサーバは必要なオブジェクトを保存していない場合、他のキャッシュサーバにそのオブジェクトの有無を問合わせる。その応答をもとに、オブジェクトを保持しているキャッシュサーバからオブジェクトを取得しクライアントに転送し、全体としてキャッシュのヒット率を向上する仕組みになっている。現状の分散キャッシュシステムでは各キャッシュサーバの管理者は、ネットワークの構成や他のキャッシュサーバの位置情報をもとに、参照の対象となるキャッシュサーバを手動で静的に指定している。そのため、ネットワークの状態や他のキャッシュサーバの変化に動的に対応できず、管理者に常に負担がかかるだけでなく、適切な設定を保つことが困難であるという問題がある。

これらの問題を解決するために、ネットワークや各キャッシュサーバの状態およびその内容等を把握するヒントサーバを置き、分散キャッシュシステムの設定・管理を自動化するシステムを提案する。クライアントからの要求を受けたキャッシュサーバは、ヒントサーバに対して必要なオブジェクトの所在を尋ねる。ヒントサーバは自身が持つデータベースを検索し必要なオブジェクトを保持しているキャッシュサーバを発見し、要求を出してきたキャッシュサーバにヒントとして与える。ヒントを受け取ったキャッシュサーバはヒントをもとにオブジェクトを取得するサーバを決定する。すなわち、各キャッシュサーバはヒントサーバを知っているだけで分散キャッシュシステムを構成することが可能となり、各組織のキャッシュサーバの設定の自動化が実現できる。また、キャッシュサーバのプログラムは本来の機能である HTTP データの転送処理やキャッシュの読み書きの処理のために負荷が常時高くなる。ヒントサーバではキャッシュ内容の通知および問合せの処理だけを行えばよいため、問合せを受けたときの応答時間がキャッシュサーバに比べて短くかつ安定することも期待できる。

4.2 WWW キャッシュサーバの構成情報の管理

分散キャッシュサーバの構成管理

以下では、階層的に配置されたキャッシュサーバ群において、利用者からみて上位の階層に位置するものを「親キャッシュサーバ」、同じ階層に位置するものを「隣接キャッシュサーバ」、本来の WWW オブジェクトを提供しているサーバを「オリジンサーバ」、利用者が使用する WWW ブラウザを「クライアント」と呼ぶ。

キャッシュ間の協調動作の機構として ICP を使った分散キャッシュシステム（図 4.1）では、あるキャッシュサーバ CS_i はクライアントあるいは別のキャッシュサーバから要求されたオブジェクトが自身のキャッシュにない場合、ICP 問合せメッセージをあらかじめ管理者によって設定された隣接キャッシュサーバに対して送信し、その応答を待つ。CS_i はキャッシュにヒットした旨の応答メッセージを受け取った場合、最初にヒットを返した送信元のキャッシュに対してオブジェクトの取得を行う。また問合せを行った全ての隣接キャッシュサーバにオブジェクトが存在しない旨（キャッシュのミス）のメッセージを受け取るか、あるいは受信のタイムアウト（通常 1 から 2 秒程度）が発生したならば、CS_i はあらかじめ管理者によって記述された規則に従って親キャッシュサーバあるいはオリジンサーバに対してオブジェクトの取得を行う。

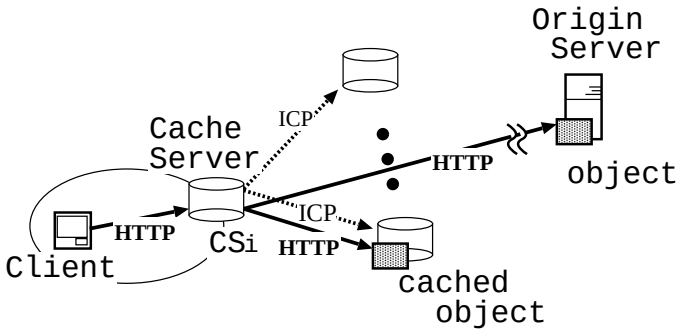


図 4.1: ICP を使った分散キャッシュシステム

現在の分散キャッシュシステムにおいては、サーバの構成情報の管理は各キャッシュサーバの管理者が経験をもとに手動で行っている。すなわち、初期設定時またはキャッシュサーバおよびネットワークの状態や構成が変化したときは、管理者が自身の経験や知識をもとに、キャッシュの動作が適切になるように構成情報を更新している。構成情報とは他のキャッシュサーバを参照するために必要な情報であり、具体的には、サーバのホスト名または IP アドレス、ICP ポート番号および HTTP ポート番号から成る。前述の Squid キャッシュでは、この記述はプログラムの設定ファイル内に書かれており、変更には管理者の介在とキャッシュプログラムの再起動が必要である。

例えば、WIDE を例にみると、バックボーンの再構成のような大規模なネットワークの変更が過去 1 年間に 5 回行われた。また、WIDE 内のキャッシュサーバの変更は多いときで 3 週間に 1 回程度の割合で行われた。ネットワークやキャッシュサーバの構成の変更が起る毎に、各組織のキャッシュサーバ管理者はその情報をもとに構成情報の設定を変更しなければならなかった。さらに局所的なネットワークやキャッシュサーバの状態の変化はより頻繁に発生していたが、キャッシュサーバの管理者は特に不具合がなければそのまま運用を行っていた。

従来の自動管理手法

このような構成情報の管理に必要な多くの時間を削減するために、いくつかの手法が提案されている。Squid キャッシュでは、ICP の問合わせに IP マルチキャスト [138] を使うことができ、その問合わせは同じマルチキャストグループに参加したキャッシュサーバ全てに届く。よって、あるキャッシュサーバ群に加わりたいと思った管理者は、そのグループに対応するマルチキャストアドレスを構成情報として設定するだけでよい。しかしこの方式では、グループに参加するキャッシュサーバの制限を行うことが難しく、さらに ICP の応答メッセージはユニキャストで送信されるため、1 つの問合わせに対してサーバ台数分の応答メッセージの通信が発生するという問題点がある。

また Squid キャッシュでは、キャッシュサーバが他のサーバから参照されるのに必要な情報を定期的にアナウンスする仕組みもある [139]。アナウンスされた情報はあるサーバに収集され、必要に応じてキャッシュサーバの一覧を WHOIS プロトコル [140] を使用して取り出せるようになっている。よって、管理者は WHOIS を使ってサーバの一覧を取り出し、それを構成情報として設定するだけでよく、さらに定期的に WHOIS で情報を得ることでサーバの ICP および HTTP ポート番号の変化には自動的に対応できる。しかし、その一覧からどのサーバを使用するかという判断は管理者に任されており、経験をもとに手動で設定する手間は従来と同じである。

構成情報の自動管理機構の提案

既存のキャッシュシステムにおける以上のような問題は、あるキャッシュサーバのキャッシュ内容およびサーバ自身の設定状態を他のサーバが知らないことから生じている。よって、解決のためにはサーバの状態やキャッシュ内容の変更を他のサーバに通知し、通知されたサーバはその内容を記憶し、適切なサーバを選択するようにすればよい [141]。しかしながら、キャッシュサーバのプログラムは HTTP データの転送処理、キャッシュディスクとの入出力処理などを行い、既にコンピュータの資源を多く必要としていることから [142]、その上に通知された内容を記憶することは負担が大きく実現が困難である。また、各キャッシュサーバで同一の通知内容を記憶することになり冗長であると考えられる。

各キャッシュサーバからのサーバの状態やキャッシュ内容の変更の通知を受け取り、それ

を記憶し、他のサーバからの問合わせに答えるヒントサーバを導入したキャッシュシステムを提案する。ヒントサーバは各キャッシュサーバのキャッシュ内に保持しているオブジェクトを常に把握することにより、キャッシュサーバからの問合わせ要求に対してオブジェクトをどのキャッシュサーバから取得すべきかという情報を応答する。本システムでは、各キャッシュサーバはヒントサーバの存在のみを知っていればよく、他のキャッシュサーバの構成情報の設定を行う必要がなくなり、また管理者がネットワークや他のキャッシュサーバの状態の変化を意識して設定を行う必要がなくなる。すなわち、キャッシュサーバの構成情報の管理を自動化・簡略化することができる。

4.3 WebHint システム

4.3.1 システムモデル

今回提案するキャッシュシステムでは、先に述べた問題点を解決するためにヒントサーバを導入した点が既存のシステムと異なる。システムのモデルは図 4.2のようになり、大きく分けて 4 つの部分すなわち、クライアント、オリジンサーバ、キャッシュサーバ、ヒントサーバから構成される。

ヒントサーバは各キャッシュサーバのキャッシュの内容を把握するために、キャッシュサーバからそのキャッシュの内容が変更された旨の通知メッセージを受け取り、自身の持つデータベースを更新することによって各キャッシュの内容との整合性を保つ。また、キャッシュサーバからの問合わせメッセージを受け取るとデータベースを検索し、キャッシュサーバがクライアントから要求されたオブジェクトをどこから取得すべきかというヒント情報を応答メッセージとして返す。ヒント情報としては、オブジェクトを持っているキャッシュサーバに関する情報、あるいはオリジンサーバから直接取得すべきという情報のいずれかになる。後者の情報が返されるのは、いずれのキャッシュサーバもオブジェクトを持っていないときや、キャッシュサーバおよびオリジンサーバのネットワークの状態などから判断されたときである。問合わせおよび応答には既存の分散キャッシュサーバ間で使用されている ICP と同様のプロトコルを利用すればよいと考えられるが、現在の ICP のメッセージにはこのようなヒント情報を格納することができないため拡張が必要となる。

さらに、ヒントサーバは複数台組み合わせることで、冗長性を持つ構成をとることが可能となる。あるキャッシュサーバが参照するヒントサーバを複数にすれば冗長構成となり、例えば 1 台のヒントサーバからの応答が無くなった場合でも問題なく動作する。また、ヒントサーバからの応答が全く無くなった場合でも、キャッシュサーバはデフォルトの動作設定に基づいて動作することになり、クライアントからみたアクセスへの影響は小さい。

キャッシュサーバはヒントサーバに問合わせを行いその応答を待つことから、通信時における応答時間は十分短い必要がある。既存のキャッシュサーバにおける ICP 問合わせ時の応答時間は通信時間を除いて数 msec から数十 msec 程度である。また、現状のキャッシュサーバの運用形態をみると、同じポリシーで運用されている組織間だけで連携してキャッ

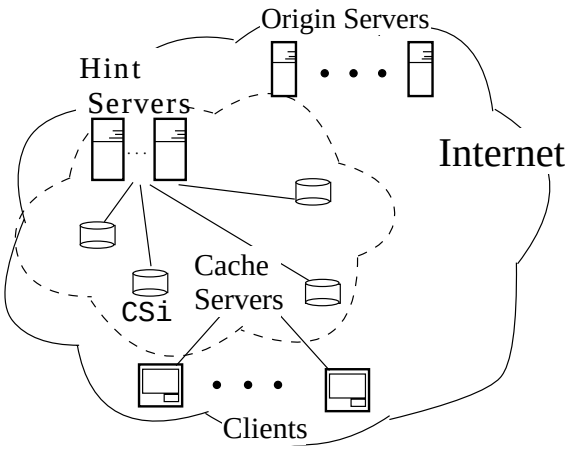


図 4.2: システムモデル

シュ群を構成していることから、ヒントサーバはある AS（Autonomous System）内の組織間や企業などの組織内のネットワークでサービスを行うことが望ましいと思われる。なお、クライアントが直接アクセスを行うキャッシュサーバは、従来どおり高速かつ広帯域のローカルネットワークでクライアントと接続されているものとする。

ヒントサーバからの応答がない場合は、キャッシュサーバは適当な待ち時間を設定し応答がタイムアウトするまで待つ。その場合、キャッシュサーバは自身のデフォルト時の動作設定（一般にはオリジンサーバから直接オブジェクトを取得）に基づいてオブジェクトの取得を行う。このような応答のタイムアウトは従来の分散キャッシュシステムでも同様に存在する。しかし、複数の異なるネットワーク上のキャッシュサーバからの応答を待つ必要があるため、応答時間にばらつきがあり、またタイムアウトが発生する確率も高いことが筆者らのキャッシュサーバの運用から経験的にわかっている。本方式のようにヒントサーバからの応答のみを待つ場合ならば、応答時間のばらつきが減少することが期待できる。よって、過去のヒントサーバからの応答時間の履歴をもとに、タイムアウトまでの待ち時間をキャッシュサーバで予測し適切な値を設定することも可能になると思われる。

4.3.2　プロトコル

ヒントサーバとキャッシュサーバ間のメッセージとして、既存の ICP を上位互換性を保ちながら拡張して利用する。新しい ICP のメッセージ種別としてヒント通知メッセージ（HNOTIFY）、ヒント問合わせメッセージ（HQUERY）、ヒント応答メッセージ（HREPLY）の 3 つを追加する。キャッシュサーバは HNOTIFY を使用して、キャッシュの内容が追加あるいは削除されたことをヒントサーバに通知する。キャッシュサーバは自身のキャッシュ

に要求されたオブジェクトが無い場合、HQUERY を使用してヒントサーバに問合わせを行う。HQUERY を受け取ったヒントサーバは自身の持つデータベースから検索を行い、その結果を HREPLY で返す。HREPLY を受け取ったキャッシュサーバはそのヒント情報をもとにオブジェクトを取得し、クライアントにオブジェクトを転送すると同時に自身のキャッシュにも格納する（図 4.3）。また、通知メッセージの一部が失われても、キャッシュをミスと判断するだけでクライアントへの影響はほとんどないと考えられるため、HNOTIFY は一方向のメッセージで紛失時の再送などは行わない。HQUERY および HREPLY の紛失時も、前述のようにデフォルトの動作設定となりクライアントへの影響は小さいため、メッセージの再送処理は行わないものとする。

具体的な処理手順は以下のようになる。

1. クライアントはあるオブジェクトの要求をキャッシュサーバに送る。要求を受けたキャッシュサーバは自身のキャッシュにオブジェクトが存在すれば、それをクライアントに転送する。

2. オブジェクトが存在しなければ、HQUERY をヒントサーバに送り、HREPLY の応答を待つ。

3. HQUERY を受け取ったヒントサーバは、自身の持つキャッシュ情報データベースをオブジェクトの URL をキーとして検索する。一致するものが見つかれば、ヒントサーバは HREPLY にヒント情報を格納しキャッシュサーバに返送する。見つからなければ、キャッシュに見つからない旨のヒント情報を返送する。

4. HREPLY を受け取ったキャッシュサーバは、そのヒント情報に基づいてオブジェクトの取得を行いクライアントに転送し、同時に自身のキャッシュに格納する。

5. HREPLY の受信タイムアウトが発生した場合は、キャッシュサーバはデフォルト時の動作設定に基づいてオブジェクトの取得を行いクライアントに転送し、同時に自身のキャッシュに格納する。

6. キャッシュサーバは、キャッシュにオブジェクトが追加されたならば、その情報を HNOTIFY でヒントサーバに通知する。

7. キャッシュからオブジェクトが破棄された時には、同様にキャッシュサーバはその情報を HNOTIFY でヒントサーバに通知する。

8. HNOTIFY を受け取ったヒントサーバは、その情報をもとに自身のデータベースを更新する。

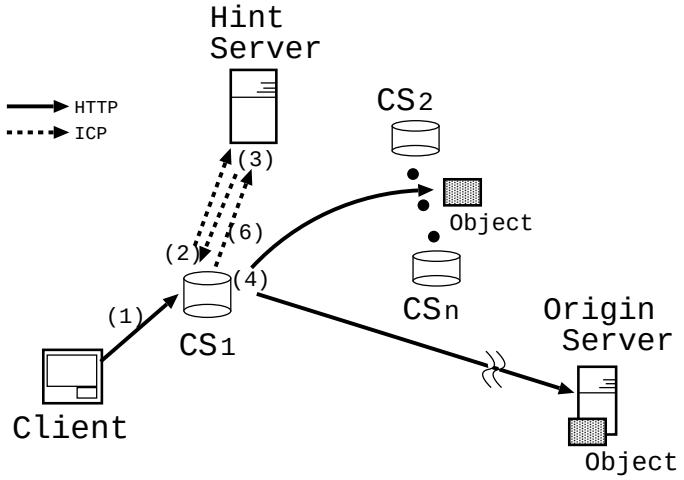


図 4.3: ヒントサーバが存在する場合のアクセス手順

4.4　実装と評価

4.4.1　実装

提案した WebHint システムのプロトタイプを実装して評価を行った。今回実装および評価を行ったのは、キャッシュサーバ群の自動設定動作に関係する部分であり、ヒントサーバによるネットワークの状態の把握とそれをヒント情報に反映させる処理およびヒントサーバ同士の協調動作に関する処理は未実装である。ヒントサーバは C 言語を用いて開発し、一般的な Unix プラットフォームの上で動作する。現在動作を確認している OS は BSDI 社の BSD/OS2.1、SGI 社の IRIX6.3、Sun 社の Solaris2.5 である。キャッシュサーバは、現在インターネットで広く利用されている Squid キャッシュをベースに改造を行った。具体的には、ヒントサーバへの問合わせの処理の追加、ヒントサーバからのヒント情報を解釈しオブジェクトを取得する処理の追加、ヒントサーバにキャッシュの内容の変更を通知する処理の追加から成る。

キャッシュサーバとヒントサーバの間でやりとりされるメッセージには ICP を使用し、既存の ICP との互換性を維持するために Opcode をいくつか追加定義するようにした。新設した Opcode の ICP のフォーマットは図 4.4のようになり、オブジェクトの持つ属性やヒント情報を格納できるようになっている。RFC2186 Header と書かれた部分は既存の ICP と共通の部分である。新規に定義した Opcode は、`ICP_OP_HNOTIFY`、`ICP_OP_HQUERY`、`ICP_OP_HREPLY` の 3 つである。各 Opcode は Options 部分に格納されるサブコードを持ち、ヒント情報の種別などを表すようになっている。また、トランスポート層のプロトコルとしては、既存の ICP と同様に UDP を使用している。

ヒントサーバのキャッシュ内容情報データベースは URL をキーとしており、ここで扱う

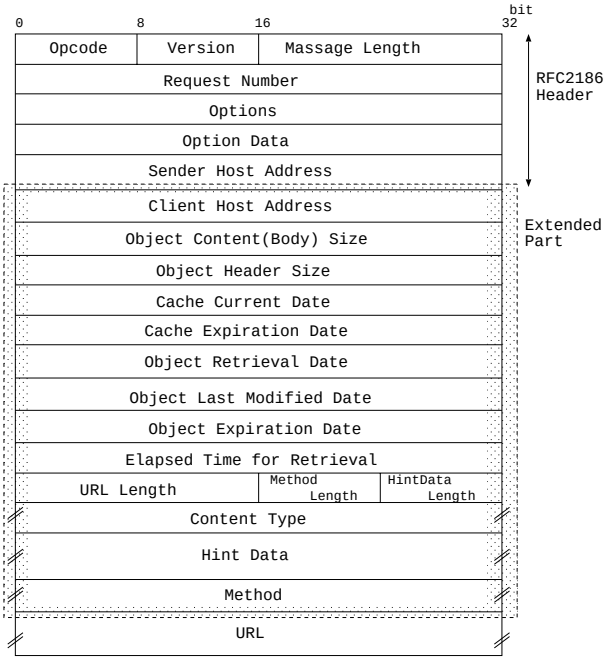


図 4.4: 追加した ICP のフォーマット

必要のあるオブジェクトの数は数万から数十万件になるため、検索を $O(logN)$ で可能な木構造で実装した。データベースに格納される情報は以下のとおりである。

- オブジェクトの URL および属性（ヘッダやコンテンツのサイズ、最終更新時刻、キャッシュにおける有効期限など）。
- そのオブジェクトに関するヒント情報。
- キャッシュサーバの状態（利用可能かどうかなど）。

4.4.2 評価

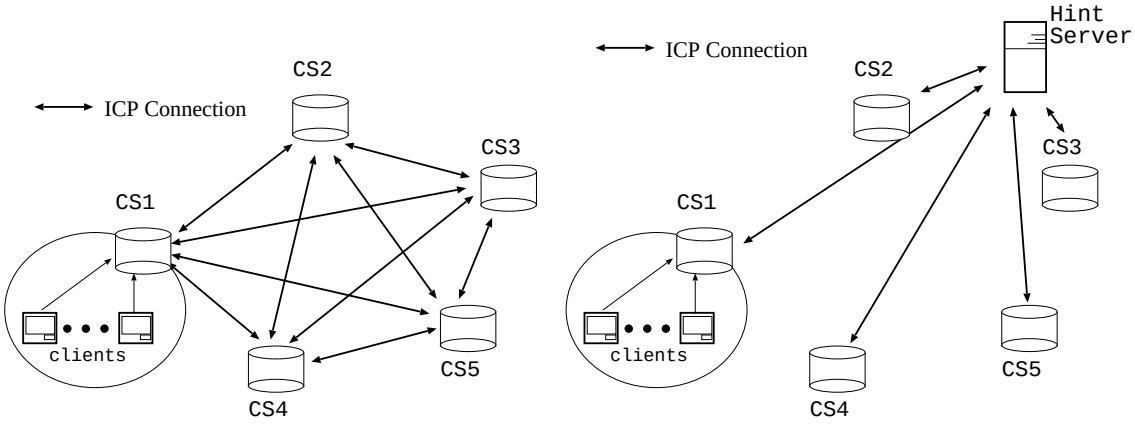
実験内容

今回提案した方式に基づくキャッシュシステムの評価として、従来の分散キャッシュシステムとの比較を行った。実験としては、図 4.5のように両システムの典型的な構成を作成し、クライアントには実際のキャッシュサーバで運用中に記録したアクセスログを入力してシミュレーションを実施した。アクセスログには奈良先端科学技術大学院大学で 1998 年 1 月末の平日 1 週間の間に学内の利用者からアクセスされた内容のうち、キャッシュ可能な HTTP のメソッドである GET リクエストを抽出したものを使用した。

比較した項目はそれぞれのシステムにおける、(1) キャッシュ構造定義の内容、(2) キャッシュサーバの構造に変化があったときに必要な管理者の対応内容、(3) システム全体としてみたときのキャッシュのヒット率、である。(1)、(2) は自動設定機構の動作の確認のためであり、(3) は提案の方式でヒット率が劣化していないことを確認するためである。

今回は実際の運用ネットワークではなくシミュレーションで評価を行った。ネットワークやキャッシュサーバの利用状態が刻々と変化する運用ネットワークでは、ヒット率を比較するのは難しいと判断したためである。但し、このシミュレーションにおいても、オリジンサーバとそこまでのネットワークの状態はそれぞれの実験で同じではないため、実験毎に全く同じ結果が得られるとは限らない点には注意しなければらない。

なお、WIDE プロジェクトで運用中の 11 の組織のキャッシュサーバにおいて、定義されている他のサーバへの参照数を調査したところ 3 から 8 台であったため、今回実験を行ったキャッシュ群におけるサーバの数は 5 台とした。また、一般に余り多くのサーバを参照するとキャッシュミス時における ICP の応答待ちのタイムアウトの確率が増え応答時間の悪化につながるため、数台程度に抑えておくことがインターネットでは推奨されていることも理由に挙げられる。



(a) Conventional Cache System　　(b) Proposed Cache System

図 4.5: 実験システムの構成

実験結果

(1) キャッシュの構造定義

各キャッシュサーバにおける、キャッシュの構造定義の内容（抜粋）は図 4.6のようになる。従来のシステムについては cache2 の設定を例として示したが、実際はそれぞれのキャッシュサーバで異なっている。

従来の分散キャッシュシステムでは、cache1 から cache5 までの各キャッシュサーバにおいて全て設定が異なり、他のキャッシュサーバを参照するための情報を管理者があらかじめ入手し、それを列挙しなければならなかった。提案した方式では、各キャッシュサーバを参照するために事前に情報を入手する必要はなく、また全てのキャッシュサーバにおいて同一の設定で済んだ。

```
#         hostname  type     HTTP/ICP port

cache_host cache1  sibling   3128 3130
cache_host cache3  sibling   8000 8130
cache_host cache4  sibling   3128 3130
cache_host cache5  sibling   8888 3130
```

(a) Conventinal Cache System

```
#         hostname  type     HTTP/ICP port

cache_host hints1 hintserver    0 4649
```

(b) Proposed Cache System

図 4.6: キャッシュ構成情報の比較

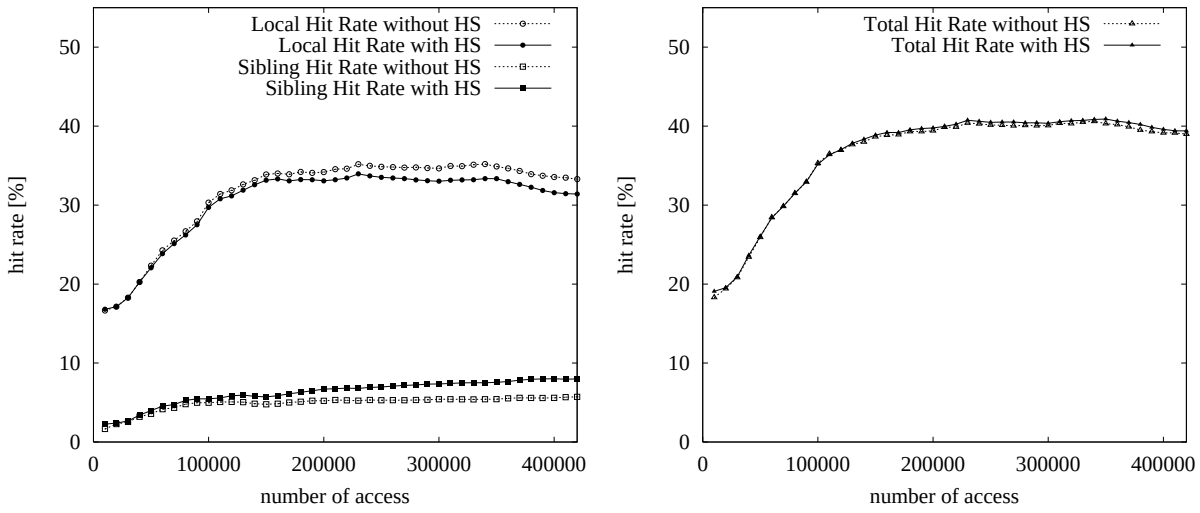
（2）キャッシュサーバの変化時の対応内容

図 4.5の構成において、ある 1 台のキャッシュサーバが停止（そこまでのネットワークが停止した状態を含む）した場合の挙動を調べた。従来のシステム（a）においては、停止したサーバから ICP の応答が送られないため、他のサーバで ICP の問合わせ処理においてタイムアウトが発生した。また、残りの 4 台のサーバから成る構成に変更するためには、各サーバの設定を手動で変更しプログラムを再起動する必要があった。提案システム（b）においては、ヒントサーバが停止したキャッシュサーバを自動的に検出し、ヒントサーバが返す ICP の中のヒント情報から当該キャッシュサーバが除外された。すなわち、残りの 4 台のキャッシュサーバ同士で協調動作する状態へ自動的に移行し、各キャッシュサーバの設定変更は不要であった。

次に、図 4.5の構成にキャッシュサーバを新規に 1 台追加した場合の挙動を調べた。従来システム（a）においては、新設のサーバは既存の 5 台を参照するように手動で設定しなければならず、また全てのサーバすなわち 6 台で協調動作させるためには、各サーバの設定変更とプログラムの再起動が必要であった。提案システム（b）においては、新設のサーバの設定ファイルは既存のキャッシュサーバと共通でよく、また新設のサーバからヒントサーバへキャッシュの状態が通知されると、ヒントサーバはその情報を各キャッシュサーバからの問合わせの応答に含むようになった。すなわち、新設のサーバを含めた 6 台での協調動作を自動的に行うことができた。

（3）キャッシュのヒット率

分散キャッシュシステムにおいては、システム全体のヒット率は各キャッシュサーバのローカルなヒット率と、ICP を使った他のキャッシュサーバあるいはヒントサーバへの問合わせによるヒット率の和で表される。図 4.5の構成で、従来システムと提案システムの各場合におけるローカルなヒット率（Local Hit Rate）と ICP によるヒット率（Sibling Hit Rate）およびそれらの和（Total Hit Rate）を測定したものは図 4.7のようになった。なお、図 4.5における、(a) の従来システムの ICP 参照形態はフルメッシュであり、ICP によるヒット率が最大になる理想的な構成となっている。



(a) Local/Sibling Hit Rate　　(b) Total Hit Rate

図 4.7: キャッシュヒット率の比較

4.4.3　考察

自動設定機構の実現

ヒントサーバを導入した提案システムにおいて、4.4.2節の結果（1）より、分散キャッシュシステムの構成情報の管理の自動化を確認することができた。結果（2）より、各キャッシュサーバはヒントサーバの情報のみを最初に設定することで、サーバの増減に対して自動的に対応できた。また、結果（3）より、システム全体のヒット率は従来の分散キャッシュの理想的な構成の場合と比べて劣化することはなく、提案システムの有効性が確認できた。

ICP のメッセージ数

従来システムと提案システムにおける ICP メッセージ数の比較を行った。システム全体のキャッシュサーバの台数を N とすると、従来システムでは $N-1$ 台のキャッシュサーバに問合わせを行うため、ある 1 台のキャッシュサーバにおける問合わせと応答の ICP メッセージの数は $2(N-1)$ 個となる。提案システムにおいてはキャッシュサーバはヒントサーバのみと通信を行うので、問合わせと応答、キャッシュにオブジェクトを追加および消去した旨の通知、計 4 個の ICP メッセージが必要となる。すなわち、従来システムにおけるシステム全体の ICP のメッセージ数は $O(N^2)$ であり、提案システムでは $O(N)$ となる。提案システムではキャッシュサーバの台数が増加しても、ネットワーク上の ICP 通信量の増加を抑えることができることがわかる。

ヒントサーバの性能

ヒントサーバ自体の性能を評価するために、ヒントサーバの ICP 問合わせに対する応答時間の計測を行った。同時に、性能の低いコンピュータでの応答時間も比較できるように、ワークステーション（SGI O2; CPU R5000 180MHz; Memory 96MB）と PC Unix（BSD/OS; CPU 486DX2 66MHz; Memory 32MB）の 2 台のマシンでヒントサーバを動作させ、キャッシュサーバから同じ ICP メッセージを送って応答時間を計測した。ヒント問合わせメッセージに対する応答時間を図 4.8に示す。ワークステーションにおける応答時間は、1 から 2msec の間で安定していることがわかる。PC Unix も 3msec 以内にほぼ収まっているが、20 万アクセスを超える頃から応答時間が大きくなっている。これはマシンのメインメモリが少く仮想記憶のスワップを起しているためであった。

この結果から、ヒントサーバの応答時間は 2msec 程度であり、これは WWW におけるアクセス時間からみて十分に短い応答時間である。またヒントサーバは各キャッシュの内容を記憶しておくためのメインメモリが十分にあれば性能の低いコンピュータでも十分動作することもわかった。また、各キャッシュの 1 つのオブジェクトの情報を記憶するのに必要なメモリの大きさは平均で 220byte であった。例えば、10 万のオブジェクトを記憶する場合でメモリは約 22Mbyte 必要となる。

ICP のタイムアウト率

実験を行った両システムにおける ICP 問合せ時のタイムアウトの発生割合を図 4.9に示す。なお、従来システムにおける各キャッシュサーバでの ICP メッセージの平均受信数は毎秒 30 個で、提案システムにおけるヒントサーバでの平均受信数は毎秒 90 個であった。ヒントサーバからの応答はほとんどタイムアウトが発生していないが、従来システムではキャッシュサーバ同士の ICP 問合せ時のタイムアウトが高い率で発生していることがわかる。文献 [142] にもあるようにプロキシキャッシュサーバ自体の負荷がかなり高いため、クライアントからのアクセスが集中するとキャッシュサーバにおける ICP の応答処理が遅れ、

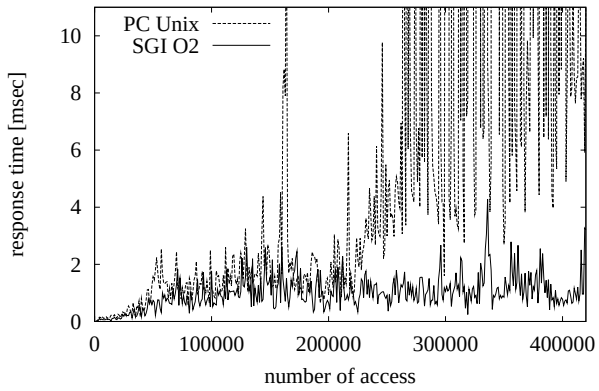


図 4.8: ICP の応答時間

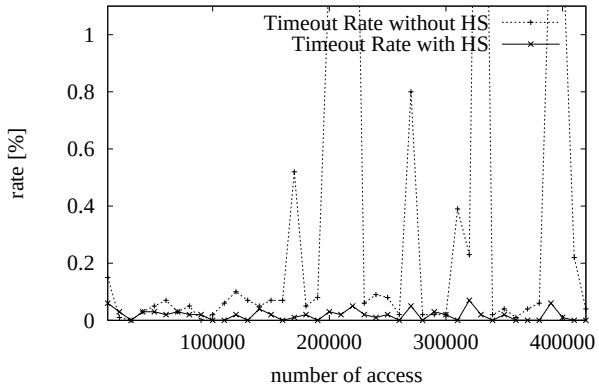


図 4.9: ICP のタイムアウト率

結果としてタイムアウトを発生させているものと考えられる。一方、ヒントサーバは ICP メッセージの処理のみを行えばよいため、受信 ICP メッセージ数が多いにもかかわらず安定した応答時間が得られたと考えられる。

4.5　まとめと今後の課題

分散 WWW キャッシュシステムの設定の自動化の支援機構を実現するために、ヒントサーバ付き分散キャッシュシステム WebHint を提案した。ネットワークやキャッシュサーバの状態、また各キャッシュの内容等を把握するヒントサーバを導入することにより、従来の分散 WWW キャッシュシステムでは困難であった分散キャッシュシステムの設定・管理の自動化が実現できた。また、自動化によってシステム全体のヒット率が劣化すること

はないことを確認した。同時に、ICP メッセージ数の削減、応答待ちタイムアウトの発生割合を減少させることが可能となった。

ヒントサーバは、各キャッシュサーバからキャッシュ内容の変更通知を受けとることにより、ヒントサーバのデータベースとキャッシュの内容の整合性を取る。しかしながら、変更通知が届かないなどの障害が起り、この整合性が取れなくなるとヒントサーバはキャッシュサーバに誤った情報を渡してしまうことになるため、この問題がどの程度ヒット率や応答時間に影響を及ぼすかを確認していく必要がある。

今後は、それぞれのキャッシュサーバの間のネットワークの状態やルーティング情報、またオリジンサーバまでのネットワークの状態をもとに、ヒントサーバが適切と思われるキャッシュサーバを選択する機構を設計・実装する予定である。

なお、WebHint システムのソースファイルは以下の URL で公開している。

`http://infonet.aist-nara.ac.jp/products/webhint/src/`

第 5 章

WIDE CacheBone 運用

5.1 はじめに

本格的な規模のインターネットでの、広域分散キャッシュシステムの挙動と有用性についての研究環境を構築するため、W4C では、WIDE インターネット上での広域 WWW キャッシュ環境の構築に着手した。WIDE インターネットの WWW キャッシュのバックボーンという意味合いをこめ、この環境を WIDE CacheBone と名づけた。

WIDE CacheBone 構築は、W4C の発足当初から、その目的のひとつにあげられていたが、Harvest Project[1]に端を発する分散キャッシュ技術の有効性とその限界の確認を行うことが主命題であった。当時、Squid と呼ばれる WWW キャッシュソフトウェアが徐々に知名度をあげてきていた頃であり[2]、世界各地で、公開型の WWW Cache サーバが稼働しはじめていた。

Squid が提案している広域のキャッシュ共有システムについては、当時から効果があるとするものとそうでないというもの、さまざまな意見が交換されていたが、現実に大規模な運営実験に処して、その評価を行う必要があった。

5.2 CacheBone 構築

CacheBone 構築にあたり留意した点は、HTTP リクエストの重複を防ぎ、できる限り無駄なトラフィックが NOC[3]をまたいで往復しないようにすることであった。

当時、Squid を用いた分散 WWW キャッシュの運営の実際として、図 5.1のような形態をよく見かけた。しかしながら、よく考えて見ると、NOC とそれに接続されているサイト同士の通信というのは、すべて NOC を経由する。この図の場合、Cache1 , Cache2 , Cache3 同士の隣接キャッシュ間のデータ共有は、所詮 NOC を経由した通信によって行われていることになる。

[1] http://harvest.transarc.com/

[2] http://squid.nlanr.net/Squid/, 本稿 2.2参照

[3] NOC:Network Operation Center. インターネットへの接続ポイント。

この、NOC を挟んだ隣接ノード間の sibling 設定は、どう考えても無駄であるので、WIDE インターネットの NOC にキャッシュサーバを設置し、NOC に接続されているサイトは単純に NOC 上のキャッシュサーバを parent 指定するという基本運営方針を最初に打ち出した。

これは、CacheBone の効率的な運営[4]のためには、キャッシュマシンを NOC に設置する必要がある事を意味する。WIDE CacheBone では、NOC 上へのキャッシュサーバの導入を優先事項とした。

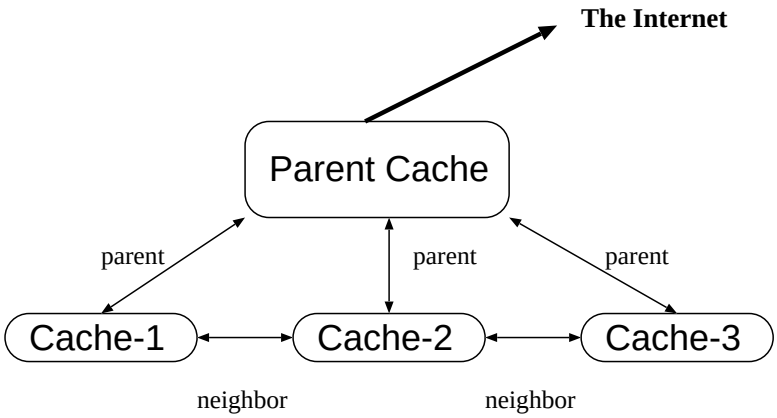


図 5.1: squid の運用例

5.3 隣接キャッシュ

ICP を用いた隣接キャッシュ郡の構成による、WIDE バックボーンに対するトラフィック増加を低く押さえるため、次のような点に留意した。

- 各リーフノードは、一番近い NOC キャッシュのみを参照する
- NOC キャッシュの設定する隣接キャッシュは 2 ポイント以上離れていない NOC とする
- キャッシュの不可分散を測るため、回線的、容量的に余裕のあるサイトにドメイン別 parent の役割を担ってもらう。

最後のドメイン別 parent については、ある程度 NOC キャッシュが稼働してからの作業であり、構築段階においては、近隣の NOC キャッシュへの neighbor 設定のみで運営していたことを付記しておく。

[4] ここで言う効率的な運営とは、回線の利用効率を意識した言葉である。

また、ドメイン別の parent 指定による方法では平等な不可分散の手法とは到底いがたく、議論の分かれるところではあったが、より良い手法が提案されるまでのつなぎという意味合いで、このような形態で運営しているのが実情である。

5.4 CacheBone 稼働

1997 年 3 月の W4C WG 立ち上げの時点から、徐々に NOC キャッシュを稼働させて行き、約半年かかって、図 5.2のサービス形態が整った。現状では、物理的制約条件など様々な理由によって、どうしても NOC に WWW キャッシュマシンを設置できないケースも存在している。本来は NOC のみに設置する形態を目指しつつ、現在は過渡的な形態として、比較的規模の大きいサイトに、キャッシュマシンを設置していただくなどのご協力を願っているポイントがいくつか混在する状況である。

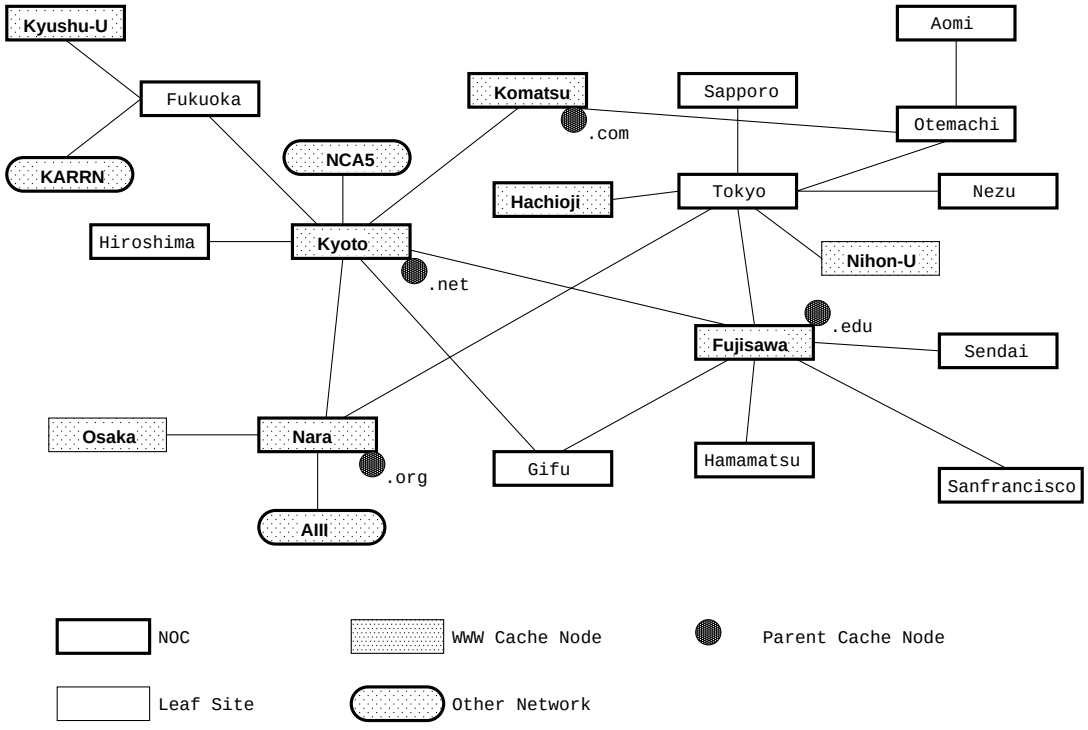


図 5.2: WIDE CacheBone

表 5.1: HTTP リクエスト数と ICP リクエスト数

Server name	HTTP	ICP	エラー	その他	合計
NAIST	10,132,832	1,764,275	125,206	0	12,022,313
Hachioji	6,410,965	6,782,189	71,168	1	13,264,323
Kyushu	12,959,777	2,407,100	385,260	0	15,752,137
Osaka	11,672,354	31,522,226	273,058	19	43,467,657
Nara	719,115	10,910,924	5,444	2	11,635,485
Fujisawa	1,202,714	11,287,082	18,941	1	12,508,738
Kyoto	10,665,469	20,194,864	65,845	329	30,926,507
AIII	6,002,913	7,678,430	134,237	3	13,815,583

5.5 評価

CacheBone の構築が一段落し、比較的安定した時期にあたる 1997 年 8 月から 10 月までの各キャッシュノードのアクセスログをもとに、CacheBoneがどの程度の規模の WWW アクセスを処理しているか、表 5.1に紹介する。

それぞれのサイトが抱えているユーザ数が異なるので、横並びに評価するのは危険であるが、キャッシュサーバ内部のヒット率を

$$\frac{TCP\_HIT\ 数 + TCP\_REFRESH\_HIT\ 数 + TCP\_IMS\_HIT\ 数 + TCP\_REF\_FAIL\_HIT\ 数}{HTTP\ リクエスト数}$$

の式を用いて計算した結果が表 5.2である。

一方、各キャッシュサーバから他のキャッシュサーバへの ICP による問いあわせに対するヒット率を

$$\frac{UDP\_HIT\ 数 + UDP\_HIT\_OBJ\ 数}{ICP\ リクエスト数}$$

の式を用いて計算した結果を表 5.3 に示す。

ざっと全体を見渡してみると、ICP を用いた隣接キャッシュのヒット率は、取るに足らないものであろうという大方の予想通りの結果が見受けられる。しかしながら、例外的にキャッシュサーバ内部のヒット率と拮抗した結果を示すサーバも見受けられる。これらのサーバ間の差異を抽出し、計測結果の数値の裏づけを取る必要があると思われる。

表 5.2: キャッシュサーバ内部のヒット率

Server name	ヒット率
NAIST	0.5045
Hachioji	0.3044
Kyushu	0.5618
Osaka	0.4160
Nara	0.4019
Fujisawa	0.2875
Kyoto	0.1901
AIII	0.3437

表 5.3: ICP による隣接キャッシュのヒット率

Server name	ヒット率
NAIST	0.0209
Hachioji	0.2154
Kyushu	0.1982
Osaka	0.1272
Nara	0.0093
Fujisawa	0.0279
Kyoto	0.1230
AIII	0.3251

5.6 CacheBone の課題

WIDE CacheBone は、その運営を各々少しずつ性質の異なる WIDE 参加各機関それぞれの相互協力で行っているため、足並みを揃えるにも、ある程度の期間を必要とする。現在懸案課題となっているもののうち、一番大きなものは、継続的なログデータの保存手法である。

WWW キャッシュサーバのログは継続的に保存しておくと巨大なサイズに脹れ上がるため、NOC に設置するケースのような本格的な規模のアクセス数を処理するサーバのログは運営上、ある程度の期間を置いて削除するしかない。しかしながら、W4C の研究課題として後日参照可能なログデータを難らかの方法で保存しておきたいという要求がある。

また、不可分散の手法についても、単純にトップレベルドメインだけで区別する以上のより最適な方法が確立する必要があるかと思われる。

現在、こうした要求を満たす良いアイデアを模索中である。

5.6.1 WWW Cacheing 技術に関して

図 5.3, 表 5.4に WIDE CacheBone の構成サーバの一つである、九州大学のサーバを例にとり、別の観点からキャッシュサーバを評価したデータを示す。図 5.3は WWW アクセスのパターンを解析した結果である。

数理言語学の分野では、いわゆる「Zipf の法則」と呼ばれる法則が知られている。この法則は、f を単語の使用度数、r を使用度数の大きい方から振った順位、C, k を定数とした時に

$$fr^k = C$$

がなりたつという経験則である。

興味深い事に、この法則は WWW データのアクセス回数の分布にもあてはまる。図 5.3は、WIDE CacheBone を構成する九州大学のキャッシュサーバ (現状ではもっとも定評のある Squid を使用) にて記録した WWW のアクセスログ (97 年 8 月分約 370 万アクセス) を分析し、データ毎のアクセス回数をアクセス回数順に示したものである。縦軸をデータ毎のアクセス回数 f、横軸をそのデータのアクセス回数にもとづく順位 r とした両対数グラフ上に、データが直線上に集まっているのが読み取れる。

Zipf の法則は URL の分布に関する規則を与えるので、関係式を用いて積分計算を行えば、種々のキャッシュ方式とキャッシュ用記憶容量毎に、その性能 (キャッシュのヒット率) が推定できる。

表 5.4に Squid を用いた九州大学の現状と理論計算の値を比較して示す。

九州大学では現状 20G byte の RAID disk を利用して約 64%のキャッシュヒット率を達成している (表 5.4左)。これは現状ではかなり良く維持されたキャッシュサーバと考えられるが、

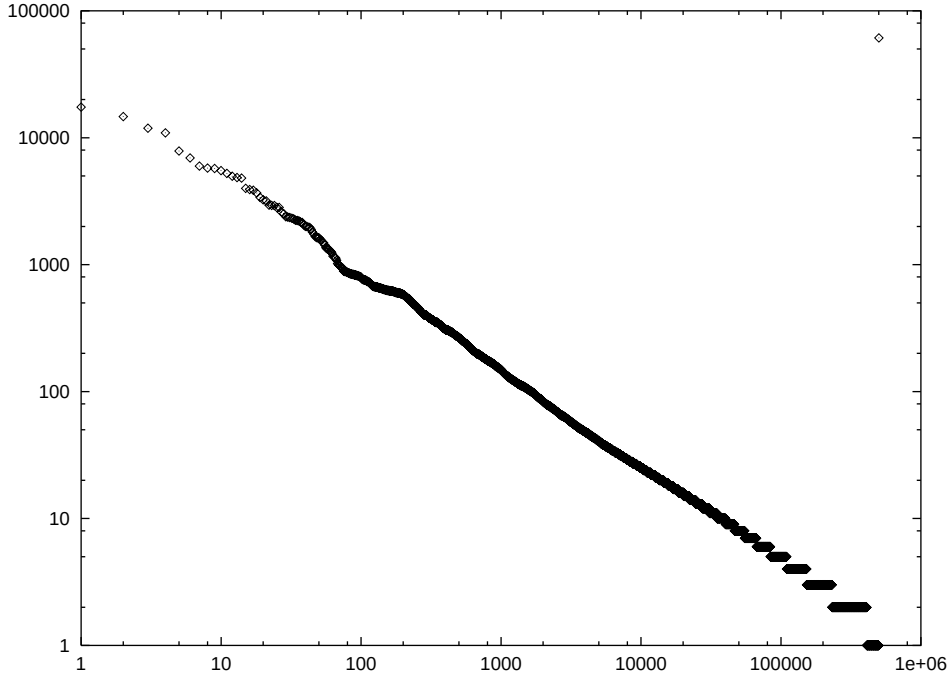


図 5.3: 九州大学の WWW アクセス傾向

	九大の現状	理論限界値	LRU 方式
キャッシュ容量	20G byte	8.43 G byte	1.87 G byte
ヒット率	64.24%	65.25%	60%

表 5.4: キャッシュ性能の比較

1. この期間中 Squid を通過したユニークなデータの総量は 8.43G byte であり、このデータを全てキャッシュする事により 65%のキャッシュヒット率を達成できる (表 5.4中央)

2. 1/10 以下の記憶容量である 1.87G byte を用いた LRU 方式によるキャッシュヒット率は、現状の達成値に近い 60%にのぼる (表 5.4右)

こと等を考えると、効率良いアーキテクチャになっているとは考え難い。

これは現状一般的に使われている TTL ベースのキャッシュ廃棄アルゴリズムが、WWW のアクセスパターンに合致していない事に起因する、現状の WWW キャッシングアーキテクチャの欠点と考えられ、WIDE CacheBone の性能向上には、キャッシングソフトの改良が必要であることを意味している。

第 6 章

パケットリダイレクションによる透過型キャッシング代理サーバ

本章では、WIDE Project 春季研究会 (1998 年 3 月 16 〜 19 日) において行われた、HTTP リクエストパケットの宛先書き換えによる透過型キャッシング代理サーバの実験について報告する。

6.1 はじめに

現在の一般的なキャッシング代理サーバの利用は、利用者の積極的な利用意思に依存している。すなわち、利用者がその意思を明確に持ち、使用するクライアントソフトウェアにその旨を明示的に設定しなければ、代理サーバは利用されず、そのキャッシュの効果を発揮することができない。一般に、末端利用者が代理サーバを利用するためには、そのサイトのネットワーク管理者が代理サーバの利用のための設定情報を利用者に対して広告する必要があるが、現実問題として、これらの情報を周知、徹底することは非常に困難である。また、その設定操作の煩雑さから、それらの情報が利用者に伝達されても、代理サーバの利用をするには至らないケースが大勢を占めている。

このような現状を改善する技術として、クライアントからサーバへの TCP セッションにおけるオブジェクトを透過的にキャッシングする機構、すなわち透過型代理サーバが研究・実装されている。今回は、IP パケットの宛先を通信経路途上で代理サーバ宛に書き換え、TCP セッションの方向を変えることによって常にこれを経由させる方式によって、TCP セッションにおけるオブジェクトの透過的なキャッシングを実現する。本報告では、この方式を「パケットリダイレクションによる透過型キャッシング代理サーバ (Packet Redirection Transparent Proxy ; 以下 PRTP)」と呼ぶ。

なお、資源を透過的にキャッシングするという意味で、TCP セッションを中継するシステムも透過的代理サーバと呼ぶ場合もある。この場合は、クライアントとサーバへの TCP セッションを透過的に扱うことはできない。

本実験では、WIDE 研究会ネットワークにおいて PRTP を配置・運用した場合の挙動を観察した。

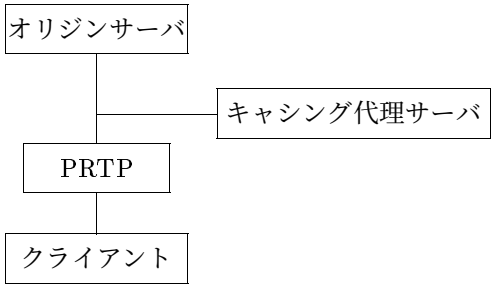


図 6.1: PRTP の配置

6.2　PRTP の動作

クライアント、オリジンサーバ、代理サーバ、PRTP のそれぞれを、図 6.1 のように配置する。PRTP を配置する位置は、そのネットワークの内外の境界となるファイアウォールまたはルータを置く位置とする。

クライアントとオリジンサーバの間で交換されるすべての IP パケットは、PRTP を経由する。PRTP はこれらのパケットのうち、クライアントからオリジンサーバへ向けて送られる HTTP リクエストを含むパケットを取り出し、そのあて先アドレス (サーバのアドレスが記述されている) を代理サーバの IP アドレスに置換する。同時に HTTP リクエストヘッダを、オリジンサーバ向けの形式から代理サーバ向けの形式に変換する (後述)。この一連の処理をリダイレクション (redirection) と呼ぶ。

6.2.1　HTTP リクエストヘッダの変換

HTTP リクエストヘッダは、オリジンサーバと代理サーバに対して 2 種類存在する (図 6.2)。オリジンサーバへのリクエストの第一行目第二フィールドはオブジェクトの URL からホスト部を除いた絶対パスである。代理サーバへのリクエストの第一行目第二フィールドはオブジェクトの URL である。すなわち、オリジンサーバへのリクエストヘッダを代理サーバへのリクエストに変換するためには、オリジンサーバのホストから生成したホスト部の追加が必要となる。今回はリクエストヘッダに含まれる `Host` フィールドを利用して、ホスト部を生成した。なお、IP パケットのあて先 IP アドレスを利用して、ホスト部を生成することも可能である。

```
GET /index.html HTTP/1.1
Host: www.wide.ad.jp:80
```

(a) クライアント サーバ間

```
GET http://www.wide.ad.jp:80/index.html HTTP/1.1
Host: www.wide.ad.jp:80
```

(b) クライアント 代理サーバ間

図 6.2: HTTP リクエストヘッダの変換

6.3 PRTP の特性

PRTP の特性として、動作原理上、次の点が知られている。

1. 代理サーバを経由することを意図しない通信が常に代理サーバ経由になる
   - 特定のポート (通常は 80 番) 向けのすべての TCP セッションを書き換え転送の対象とするので、それを意図していない通信も強制的に代理サーバ経由となる。これは、利用者の意思によって代理サーバの利用を明示的に回避することが不可能であることを意味する。
   - オリジンサーバから見たリクエストの発アドレスは常に代理サーバのものになるので、サーバ側における発アドレスに応じたアクセス制御が、意図通りに機能しない。
   - トラブル発見等の目的で広く行われる、telnet を用いたオリジンサーバの 80 番ポートへの直接通信が不可能
2. PRTP に障害が起きると、そこが経路途上となるすべての通信が途絶する
3. リクエストヘッダに `Host:` フィールドを付加しないクライアントソフトウェアは使用できない。現状ではほとんどのソフトウェアが `Host:` フィールドを付加する[1]が、すべてのソフトウェアが対応しているとは限らないことから、注意が必要である。たとえば、HTTP/0.9 や HTTP/1.0 に準拠したソフトウェアは `Host:` フィールドを付加しない。

[1] 本実験でもこれに該当するソフトウェアは発見していない。

表 6.1: WIDE 研究会ネットワークの規模

利用者数	200–300 名
クライアント IP address 使用数	341

日付	ミス	ヒット	エラー	その他	合計
3/16	10347	4376	587	517	15827
3/17	34133	17729	974	1730	54539
3/18	34106	11417	616	1280	47419

表 6.2: 期間中を通したキャッシングサーバのアクセス数

6.4　評価

PRTP の性能評価として、通常のキャッシング代理サーバと PRTP を用いたキャッシングシステムを比較する。

今回実験を行なった WIDE 研究会ネットワークの規模は、次のとおりである (実験に用いたソフトウェア、ネットワーク構成は 3.10節で述べたので参照されたい)。

両者を比較する方法として、総アクセス数が同程度あると予想される時間帯 (1 時間) を二つ選び、キャッシング代理サーバのアクセス総数および、キャッシング代理サーバを経由しないアクセス総数を計測した。計測時刻は、前日の傾向より判断した。キャッシング代理サーバのアクセス総数は代理サーバのログの解析により、キャッシング代理サーバを経由しないアクセス総数は、研究会ネットワークの対外セグメントに対する tcpdump ユーティリティの出力の解析によって計測した。

なお、本実験では、電子メールを用いて全利用者に対し、事前にキャッシングサーバの利用方法を通達した。

期間中のキャッシングサーバのアクセス数は、表 6.2のようであった。

3/16 の総アクセス数が他の日のそれと比較して極端に少ないのは、研究会初日であり、参加者の多くが夕刻から会場入りしたためである。期間最終日の 3/19 は、当日朝よりネットワーク撤去作業が行なわれたため、データを採っていない。

PRTP を用いた WWW キャッシングシステムと、一般的なキャッシングサーバのみを用いた WWW キャッシングシステムの比較を表 6.3に示す。

一般的なキャッシングサーバのみを用いた構成では、代理サーバを経由するアクセスの合計数が PRTP 使用時と比較し 80% 以上少ない。利用者への情報伝達の方法、頻度にもよるが、これは、キャッシングサーバの利用の如何が末端利用者の判断に委ねられている場合、その使用方法が利用者に伝達されているとしても、全体の 2 割程度の人しかそれを

	代理サーバを経由する通信					
	ミス	ヒット	その他	合計	経由しない通信	総計
PRTP を使用した場合	3488	1346	144	4978	—	4978
一般的なキャッシングサーバ	407	395	59	861	3715	4576

PRTP 使用の場合 3/18 16:00-17:00
一般的なキャッシングサーバ使用の場合 3/18 21:00-22:00

表 6.3: PRTP を用いた構成と一般的なキャッシングサーバのみを用いた構成の比較 (一時間のアクセス数)

	ヒット率
PRTP を使用した場合	0.27
一般的なキャッシングサーバ	0.09

表 6.4: 構成の違いによるヒット率の比較

利用することがないという傾向を示唆している。

ここで、代理サーバを経由しないアクセスはすべてミスヒットとみなして、すべての HTTP トラフィックに対するリクエストのヒット率を次式のように定義する。そして、それぞれの場合についてヒット率を計算したものが 表 6.4 である。

$$\text{ヒット率} = \frac{\text{キャッシュにヒットしたアクセス数}}{\text{代理サーバを経由しないアクセス数} + \text{代理サーバを経由したアクセスの合計数}}$$

PRTP を使用した場合、一般的なキャッシングサーバのみで構成した場合と比較し、ヒット率の面で明らかに有利であることがわかる。

また、研究会ネットワークの対外線が事故で不通となった期間の挙動から、クライアントが目的ホストの DNS 名を解決できない状況では、キャッシング代理サーバ内に目的のオブジェクトが蓄積されていたとしても、それを利用することが不可能であることが分かった。PRTP を用いた構成では、一般的なキャッシングサーバを用いた構成と異なり、クライアントがサーバへの TCP セッションを開始するために、目的となるオリジンサーバの DNS 名の解決を行なう必要がある。もしクライアントが DNS 名の解決を行えない状況では、サーバへの TCP セッションを開始することができないため、PRTP へパケットが到達せず、PRTP が動作しないためにキャッシング代理サーバのキャッシュを利用できない状況に陥った。

6.5　まとめ

本実験では、PRTP を用いて構成した WWW キャッシングシステムを、一般的なキャッシング代理サーバを用いた構成と比較し、そのヒット率の違いについて測定した。この結果、PRTP を用いた構成では、キャッシングの効果が非常に高いことが明らかとなった。

ただし、実用的なネットワークで使用する際には、障害時の通信途絶などの側面を十分に検討し、適切に導入することが重要である。

PRTP の特性に対する検討であるが、その原理上、通信の経路途上となる位置に設けるため、ネットワーク機材としては、ルータ等の基幹機材と同様の性格を併せ持つと考えられる。例えば、障害時の通信途絶に対する検討では、代替経路の準備や二重化などの対策が有効であろう。また、PRTP の動作の安定性をルータや伝送媒体等のそれと同程度まで向上させることも有効であると考えられ、その開発が望まれる。

第 7 章

おわりに

本章では、WWW キャッシュに関する技術の紹介と考察に始まり、W4C WG の一年間の研究活動の報告を行った。インターネットを取り巻く状況は劇的に変化しており、特にインターネットを支える通信回線のデータ転送速度は飛躍的な向上を遂げてきている。

1997 年は、こうした状況の中で、WWW キャッシュの立場が揺らいできた時期であり、W4C WG はそのうねりをダイレクトに被った。W4C WG の活動を開始した頃は、まだまだユーザの要求に対してインターネットのバックボーン回線の太さは十分といえないという実感が、エンドユーザにまで浸透していた頃である。ところが、インターネットの回線速度は日ごとに向上し、WWW サーバのコンテンツを直接取得する場合に比較して、多数のユーザのおもりをする WWW キャッシュサーバのマシン負荷のほうが問題となるようなケースが目立ってきた。増大したバックボーンの容量は、より多く、より早くのデータオブジェクトの処理をキャッシュサーバに要求する。

また、プッシュ配信型のコンテンツの台頭や、オブジェクトのキャッシュ不可を要求するページ埋めこみ型の広告媒体の隆盛など、従来の単純な発想によるキャッシュサーバの設計方針・運営方針は、熟考を要する時代に入ってきている。

こうしたインターネットや WWW を取り巻く様々な情勢を俯瞰しつつ、W4C WG では、1998 年も引き続き WIDE CacheBone の運営を軸とした研究を継続し、WWW キャッシュに関する新たな提案を模索していく。