# 第 15 部

# WWW キャッシュ技術

# 第 1 章

# はじめに

W4C WG[1] は、インターネットにおける WWW トラフィック (HTTP[53, 22]) の軽減技術の研究開発を目標としたグループとして、WIDE インターネットを網羅する大規模分散 WWW キャッシュシステムの構築と運営を軸に、1997 年 3 月に設立された。初期の目的の一つであった WIDE インターネット内の大規模分散 WWW キャッシュシステムは、WIDE CacheBone と名づけられ、WG 立ち上げ当初から本稿執筆時点までで約 2 年弱の期間、運営が継続されている。

WWW キャッシュシステムに焦点を当てた研究を継続するにつれ、必ずしも肯定的とも言い切れない状況が明らかになりつつあることも事実である。我々は、こうした事実をも真摯に受け止め、ここに紹介する。

また、WG 立ち上げ時点から本稿執筆時点にいたるまで、インターネット全体の環境の大きな変化もあり、WWW キャッシュシステムのあり方も大きくその立場を考え直さなければならなくなってきている。

W4C WG の２年目の活動報告にあたる本稿では、第 2章において、WIDE CacheBone の現状報告を行った後、第 3章・第 4章で、アクセス解析手法についての議論を紹介し、第 5章において、これまで効果があるとされてきた分散キャッシュシステムについて評価を行う。最後に第 6章で、WWW キャッシュシステムの効果を改めて考察する。

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

# 第 2 章

# WIDE CacheBone

W4C WG の主な目的は、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である。我々は、広域ネットワーク上で、HTTP リクエストデータの共有を行うことにより、全体の利益を計るシステムの構築を目指し、分散キャッシュシステムの検証を行うためのテストベッドとして WIDE CacheBone の構築・運営を行った。

ここでは、本稿執筆時点で運営２年目にあたる、WIDE CacheBone の現状の紹介を行う。

## 2.1 WIDE CacheBone の設計

WIDE CacheBone の設計開始は、1997 年 4 月頃である。当時の分散キャッシュ技術としては、Squid[3] による ICP(Internet Cache Protocol)[126] を用いた分散キャッシュシステムが唯一実用的なものであった。そこで、この Squid を利用して WIDE CacheBone を構築することとなった。

### 2.1.1 Squid

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 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。

Squid 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
- 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 (DNS 検索や FTP プロキシは別プロセス)
- ICP (後述) に対応
- SSL に対応

### 2.1.2 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なうためのプロトコルである。現在は RFC2186[126] 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。
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。

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 neighbour からも hit がなかった場合、オリジンサーバまたは静的設定によって指定された parent サーバに HTTP リクエストを出す。

### 2.1.3 設計方針

W4C WG 設立当初の WWW キャッシュシステムに対する要求は WWW トラフィックの削減を行うことであった。当時はまだ、広域キャッシュシステムに対する指針等がなかったため、設置と並行して以下のような設計を行った。

- WIDE インターネットの中継点である NOC を通過する ICP は極力少なくする。そのために、できる限り NOC 上にキャッシュサーバを設置する。
- NOC でない組織のキャッシュサーバは、一番近い NOC のキャッシュサーバのみと sibling 接続する。
- NOC のキャッシュサーバは、2 ホップまでの NOC のキャッシュサーバと sibling 接続する。

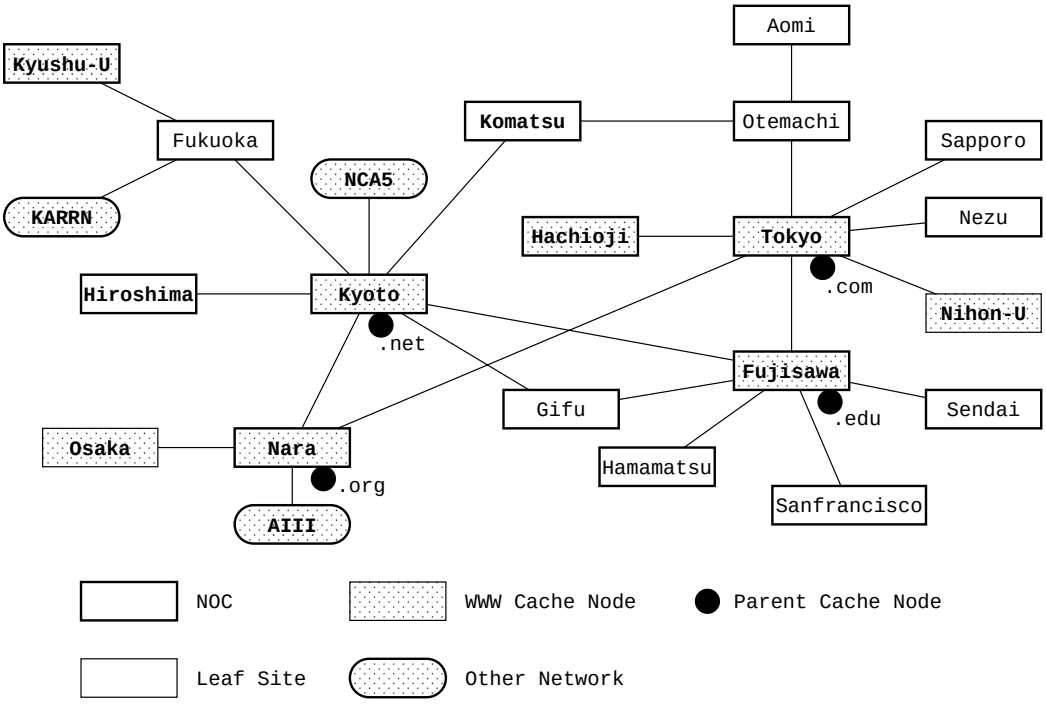



図 2.1: WIDE CacheBone

- 負荷分散のため、回線的、容量的に余裕のある NOC のキャッシュサーバに、ドメイン別の parent となってもらう。

また、本年度の WG の研究により、ICP による過度の sibling 接続は通信量の割に効果が薄く、かえって WWW トラフィックを増加させてしまうことがわかった。そのため現在では、全てのキャッシュサーバで ICP による sibling 接続を停止して、ドメイン別の parent による設定のみによる運用を行っている。ただ、ドメイン別の parent 指定による方法は適切な負荷分散の手法とは到底いいがたいため、より良い手法を模索している段階である。

## 2.2 WIDE CacheBone の構成

現在の WIDE CacheBone のトポロジーを図 2.2に示す。11 の WWW キャッシュサーバのうち、東京 NOC、藤沢 NOC、京都 NOC、奈良 NOC に設置されたサーバが parent キャッシュサーバとして他のキャッシュサーバから参照されている。

## 2.3　課題

WIDE インターネットは常に変化しているため、WIDE CacheBone もそれにあわせて変わっていく必要がある。W4C WG 設立当初から現在まで、WIDE インターネットは徐々に広帯域化しているが、それに伴い WIDE CacheBone に対する要求が WWW トラフィックの削減から利用者への応答時間の改善へと変化している。この要求の変化によって、WIDE CacheBone は再設計する時期に来ていると考えられる。具体的には、高速な WIDE インターネット域内回線を利用して利用者への応答時間を改善しつつ、比較的低速な WIDE インターネット域外回線では WWW トラフィックを削減できるような WWW キャッシュシステムを設計する必要がある。

また、負荷分散については、parent キャッシュサーバを静的にトップレベルドメインで決定する手法は、ドメイン数の偏りが大きすぎて適切な負荷分散手法とはいえない。そのため、動的に複数の parent キャッシュサーバを選択する手法が必要である。

# 第 3 章

# ページ単位の解析手法の提案

ここ数年で急速に普及した WWW では、性能改善や性能評価に関する研究が多く行われているが、そのほとんどの研究はオブジェクトを最小単位として扱っている。しかし、WWW のアクセスはページ単位で行われるのが普通である。1 つのページは複数のオブジェクトから構成される。WWW アーキテクチャ全体をユーザの視点に立って改善するには、ページ単位の解析を行い、ユーザの挙動を知ることが重要である。

本研究では、ページ単位の解析によりユーザの挙動を観測することを目的とし、WWW 代理サーバのアクセスログからページ構成を抽出する手法を提案する。提案した手法を用いて明らかにしたページ単位のアクセス傾向をもとに、ユーザの体感速度を改善するために必要なことについて考察する。

## 3.1 アプローチの検討

ユーザによるページアクセス傾向の解析を行うアプローチとして、1) パケットモニタリングによる方法、2) ログ解析による方法の 2 つが考えられる。

パケットモニタリングによる解析は、詳細な解析が行える可能性がある。しかし、ページ単位の解析を行うには、ヘッダ情報だけでなく、ペイロードの内容もモニタする必要があるため、通常のパケットモニタリングツールでは処理が複雑になる。また、ミクロな視点からの観測であるために不要な情報が多い、パラメータを変えて計測し直すことが困難などの理由により、本研究では採用しなかった。

ログ解析による手法には、サーバでの解析、ブラウザでの解析、代理サーバでの解析が考えられる。サーバでの解析の場合、解析結果は提供者側の視点による観測統計となるため、本研究の目的には合致しない。

ブラウザにおいて解析する場合、ユーザの挙動を正確に分析できる可能性がある。しかし、ブラウザにはログ出力機能がないために、観測対象となる全てのブラウザを改変する必要があり、大変な労力を必要とする。また、ブラウザが分散しているために、解析結果の収集が困難であるという欠点がある。

代理サーバで解析する場合、代理サーバには、複数のブラウザからのアクセスを中継するために、多くのユーザのアクセスの一括収集が容易、機能追加が容易であるという利点

を持つ。そのため、本研究では、代理サーバのログを解析する手法を取った。

## 3.2　解析項目

今回、WWW 代理サーバのアクセスログから以下にあげる項目を解析した。

- ページ取得時間
- ページに含まれるオブジェクト数
- ページサイズ
- ページヒット率
- ページリクエスト間隔（ページ閲覧時間）
- ページ遷移時間

## 3.3　解析手法

今回の解析では、WWW 代理サーバとして広く普及している Squid を対象に解析を行った。Squid のアクセスログには、ブラウザからのリクエストヘッダの内容も表示されるように設定し、リクエストヘッダ中の Referer 情報を参照できるようにした。Referer 情報は、ブラウザからのリクエストがどの URL に基づいているかを示す情報であり、埋め込みオブジェクトの場合や、リンクを辿ったリクエストの場合は、リクエストヘッダに Referer 情報が付加される。WWW ブラウザとして代表的な Netscape や Internet Explorer では、Referer 情報が付加されるようになっている。

### 3.3.1　ページ構成の抽出方法

ページは、ベースとなるテキストページ（ベースオブジェクトと呼ぶ）とページに含まれる埋め込みオブジェクトから構成される。ベースオブジェクト、埋め込みオブジェクトは、Content-type、Referer 情報により判断した。まず、全オブジェクトの内、Content-type が text/html または text/plain をベースオブジェクトとみなした。次に、それ以外のオブジェクトについて、Referer 情報がベースオブジェクトを参照している場合、該当するベースオブジェクトの埋め込みオブジェクトと判定した。

埋め込みオブジェクトを Referer 情報により判定すると、リンク先のオブジェクトが HTML(あるいは Text) でない場合、埋め込みオブジェクトと区別できないことがある。ユーザがページリクエストを出してからページ内のリンクを辿るまでには、ページの表示

や閲覧などにより、タイムラグが生じる。そこで、タイムアウト時間を設定し、ベースオブジェクトのリクエストが出されてから一定時間内にリクエストされた（Referer 情報つきの）オブジェクトだけを埋め込みオブジェクトと判断することにした。

抽出したページ構成について、ページ取得時間、ページサイズ、ページに含まれるオブジェクト数を計算した。

### 3.3.2 ページアクセスの追跡方法

ページアクセスの追跡は、クライアントの IP アドレスと Referer 情報により判断した。まず、アクセスログから、クライアントごとのベースオブジェクトのリクエスト間隔を計算し、これをページリクエスト間隔とした。次に、ベースオブジェクトのリクエストに Referer 情報が含まれている場合、Referer が示す URL（ページ）内のリンクを辿ったアクセスだと判定し、Referer が示すオブジェクトとのアクセス時刻の差を遷移時間として算出した。

## 3.4 解析結果

奈良先端科学技術大学院大学で運用されている代理サーバ（Squid 2.0）の 1998 年 12 月の 1ヶ月間のアクセスログを解析した結果について述べる。アクセスログ中の GET メソッドによるリクエストだけを対象とし、総オブジェクト数は約 430 万件、ページ数は約 100 万件だった。ページ構成の抽出では、タイムアウト時間の設定により、解析結果が異なる。最適なタイムアウト時間を探るために、タイムアウト時間を 1 分、5 分、30 分に設定して、3 通りの解析を行った。

ページ取得時間の累積分布を図 3.1に示す。80%のページが 10 秒以内に収まっている。また、タイムアウト時間による、ページ取得時間の違いはほとんどないことが分かる。

1 つのページに含まれるオブジェクト数の累積分布を図 3.2に示す。ベースオブジェクトもページに含まれるオブジェクトとして数えている。80%のページには、10 個以内のオブジェクトしか含まれていないことが分かる。

ページサイズとオブジェクトサイズの累積分布を図 3.3に示す。ページサイズはオブジェクトサイズに比べて大きいが、似たような分布を示しているといえる。

ページ単位のキャッシュヒット率を表 3.1に示す。ページ内の全オブジェクトがキャッシュヒットした場合をページのキャッシュヒットと定義した。オブジェクトのキャッシュヒット率 39.6%に比べて、ページのヒット率が 4.5% とかなり低くなっていることが分かる。

ページのリクエスト間隔とページ遷移時間の累積分布を図 3.4に示す。ページのリクエスト間隔、ページ遷移時間とも、10 秒〜 1 分に分布しているといえる。リクエスト間隔とページ間遷移時間に差が出るのは、ユーザがアクセス履歴を戻った後で、別のページに遷移する場合である。リクエスト間隔とページ間遷移時間の分布に差がないことから、ユーザの挙動として、アクセス履歴をさかのぼることはほとんどないといえる。

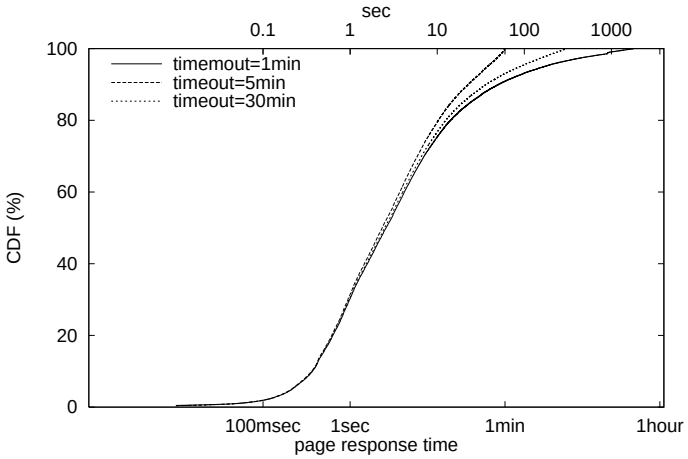



図 3.1: ページ取得時間

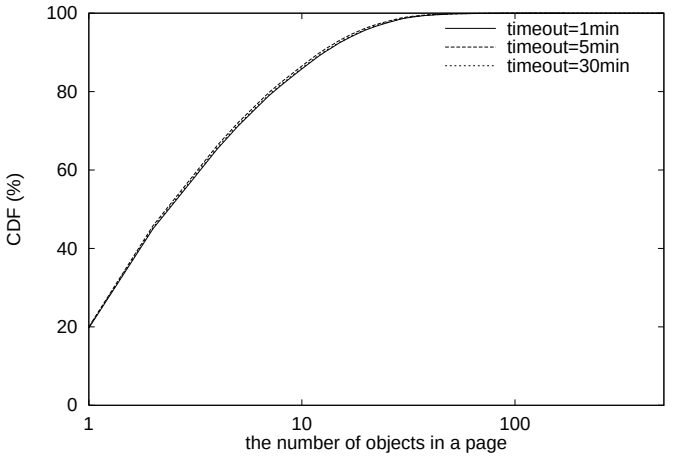



図 3.2: 1 ページの総オブジェクト数

表 3.1: ページヒット率

| タイムアウト時間 | 1 分 | 5 分 | 30 分 |
|---|---|---|---|
| ページのヒット率 | 4.6 % | 4.4% | 4.6% |
| オブジェクトのヒット率 | 39.6% | | |

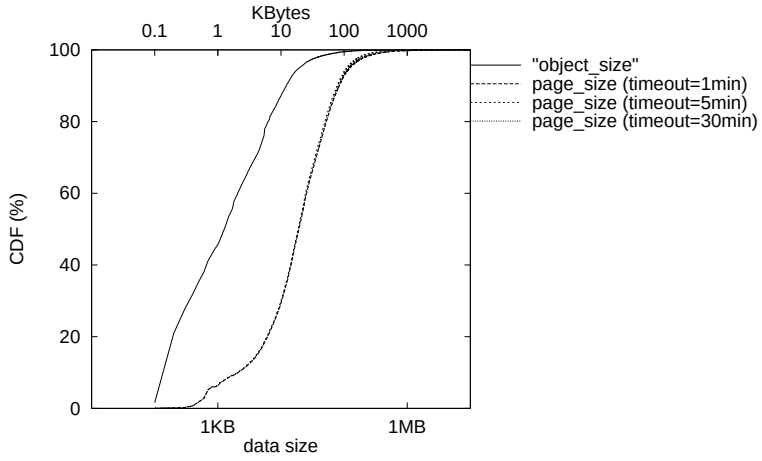



図 3.3: ページ/オブジェクトのデータサイズ

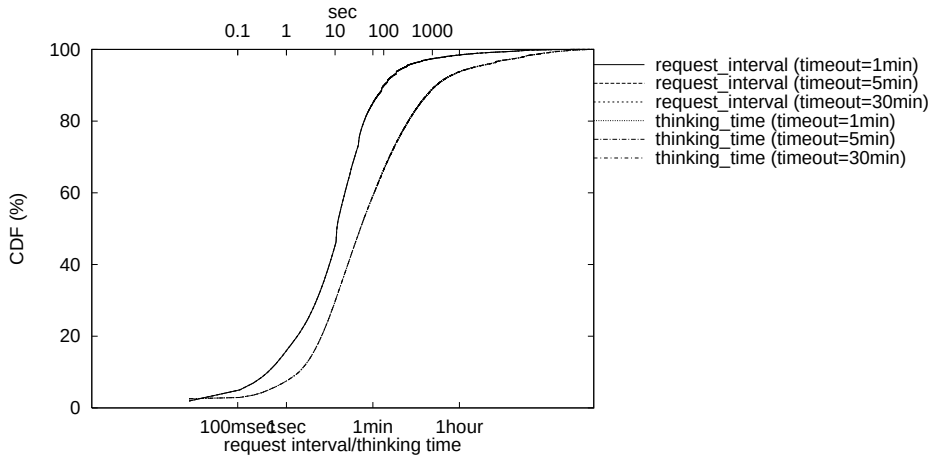



図 3.4: ページリクエスト間隔

## 3.5　考察

タイムアウト時間を 1 分、5 分、30 分と変化させたが、ページの構成にほとんど変化が見られなかった。理由として、タイムアウト時間の設定が 1 分でも長いことが考えられる。最適なタイムアウト時間の検証は今後の課題である。

HTTP はオブジェクトを単位としたプロトコルである。1 つのページに含まれるオブジェクトが 10 個程度あることから、HTTP 通信におけるコネクション確立のオーバヘッドは無視できないといえる。1 つのコネクションで複数のオブジェクトを転送する方法として、PHTTP が提案されているが、PHTTP でも、1 リクエスト＝ 1 オブジェクトという HTTP の通信モデルは変わらない。1 つの HTTP リクエストに対して、サーバがページ内に含まれている全てのオブジェクトを返すように、WWW アーキテクチャを改善することで、取得時間の短縮が図れると考えられる。また、ブラウザによっては、埋め込みオブジェクトを全て取得するまで、ページの描画を始めないブラウザもある。そのようなブラウザでは、オブジェクトを取得するたびに、レイアウトを再計算するように改良することで、ユーザの体感速度を上げることができる。

ページ単位のキャッシュヒット率がかなり低いことから、ページ単位のキャッシュアルゴリズムが必要だと考えられる。ページサイズの分布とオブジェクトサイズの分布では、サイズは異なるが、分布の傾向は似ていることから、キャッシュの粒度をあげることでページのキャッシュヒットが向上する可能性がある。また、ページに含まれるバナー広告などキャッシュできないオブジェクトの存在もキャッシュヒット率を下げている要因として考えられる。ページのデザインも含めて、バナー広告のあり方を考える必要がある。

## 3.6　まとめ

本研究では、WWW アーキテクチャの性能改善手法を確立するために、代理サーバのアクセスログからページ構成を抽出し、ページ単位のアクセス傾向を解析した。その結果、ページを単位とした、WWW サーバ、HTTP、ブラウザ、代理サーバページのチューニングが必要であることを明らかになった。また、解析結果をもとに、チューニング項目について提案し、提案した解析手法が WWW のチューニングパラメータの推測に利用できることを示した。

# 第 4 章

# パケットモニタによる WWW サーバ挙動観測

従来用いられている WWW サーバシステムの評価方法の 1 つは、SPEC web[6] に代表されるベンチマークプログラムである。また、WWW サーバプログラムのログ解析による性能評価方法も存在する。SPEC web はベンチマークシステムなので、運用中のシステムには用いることができない。WWW サーバプログラムは、アプリケーション層で動作しているので、サーバプログラムのログ解析によるシステム全体の評価を行うことは不可能である。

ここでは WWW サーバシステムの挙動観測を行うために、WWW サーバシステムに対するパケットモニタリングを行うことを提案する。WWW サーバシステムをブラックボックスに見立て、通過するパケットを解析することで WWW サーバシステムの挙動観測を行う。これによりシステム全体の挙動を観測することが可能となる。

本章では WWW サーバシステムの挙動観測手法としてのログ解析の限界を述べ、パケットモニタ手法でどのような項目が測定可能かを示す。パケットモニタ手法の有効性を示すために、実際に稼働している WWW サーバシステムの観測例を述べる。観測の結果、本システムでは正確なコネクション継続時間および WWW サーバシステムが同時に保持しているコネクション数の測定が可能であることを明らかにした。ここで、正確なコネクション継続時間はクライアントに情報を提供する時間である。また、WWW サーバシステムが同時に保持しているコネクション数は WWW サーバシステムがどれぐらいの数の要求を同時に処理できるかの指標である。

## 4.1 TCP コネクションと WWW サーバシステムの挙動

WWW のデータ転送は HTTP によって行われる。HTTP は TCP 上で動作するプロトコルであり、クライアントからのリクエストによってサーバが応答するプロトコルとなっている。

TCP コネクションを監視することによって、HTTP でどのようなデータが転送されているかが明らかとなる。以下では、HTTP に必要な TCP コネクションの確立と切断について説明し、WWW サーバシステムの挙動との関係を述べる。

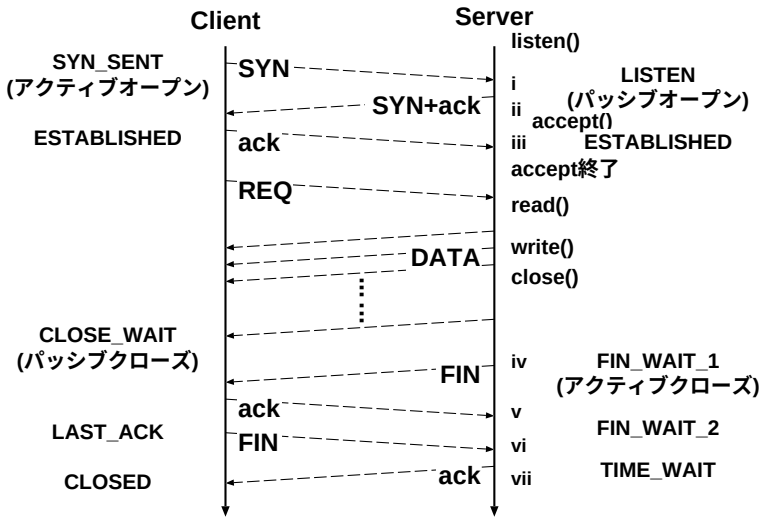



図 4.1: TCP コネクションの状態の流れ

### 4.1.1　TCP コネクションの確立と切断

TCP コネクションを確立するステップは以下の通りである [118]。

1. クライアントは、接続先のサーバのポート番号とクライアントの初期シーケンス番号 (ISN) を指定した、SYN パケットを送る (図 4.1の i)。

2. サーバは、サーバ側の初期シーケンス番号を含む SYN パケットで応答する。また、サーバはクライアントの ISN+1 の ack を送信することでクライアントの SYN に確認応答する (図 4.1の ii)。

3. クライアントはサーバから送られてきた SYN に対して、サーバの ISN+1 の ack で確認応答する (図 4.1の iii)。

これら 3 つのパケットによって、コネクションの確立が完了する。

コネクションの切断は、サーバ、クライアント、どちらからでも行うことができるので、コネクションの確立よりも少し複雑である。

最初の FIN パケットを送信するホストをアクティブクローズ側と呼び、アクティブクローズの FIN パケットを受け取った後に FIN パケットを送信するホストをパッシブクローズ側と呼ぶ。

アクティブクローズ側は、FIN パケットを送信 (図 4.1 の iv) し、FIN に対する ack(図 4.1の v) を待つ。そして、パッシブクローズ側の FIN パケット (図 4.1の vi) が到着すると、受け取ったシーケンス番号+1 の ack を送り返し (図 4.1の vii)、TIME_WAIT 状態になる。

パッシブクローズ側が FIN パケット (図 4.1の iv) を受け取ると、受け取ったシーケンス番号+1 の ack(図 4.1 の v) を送り返す。そして、アクティブクローズ側に FIN パケットを送信し (図 4.1の vi)、それに対する ack パケット (図 4.1の vii) を待つ。ack パケットが到着すると、CLOSE 状態となる。

これら 4 つのパケットによって、コネクションの切断が完了する。

### 4.1.2 WWW サーバシステムの挙動

本論文では、WWW サーバシステムの挙動として、サーバの応答時間および同時に処理できる要求の数を考える。

WWW サーバシステムがどれぐらいの要求を処理できるかを調べるためには、WWW サーバシステムが同時に処理しているコネクション数の計測が重要である。同時コネクション処理数は、WWW サーバシステムがどれぐらいの要求を同時に処理しているかの指標である。この値を得るためには、まず、WWW サーバシステムが保持しているコネクションの正確な継続時間を計測する必要がある。正確なコネクション継続時間を知るためには、コネクションの開始時刻と終了時刻を知る必要がある。

## 4.2 ログ解析の限界

これまで、WWW サーバシステムの挙動観測は、WWW サーバプログラムのログ解析で行われてきた。ログ解析では、提供したコンテンツの属性、時刻、サーバプログラムの処理時間などを得ることができる。しかし、WWW サーバプログラムは図 4.2でのユーザ側で動作しているので、OS の挙動を観測することはできない。したがって、WWW サーバプログラムのログ解析だけでは、システム全体の挙動観測を行うには至らない。例えば、ログ解析からは、利用者から見た反応時間や転送時間を推測するのに重要なコネクションの継続時間はわからない。すなわち、図 4.1の accept システムコールから、close システムコールまでの時間しか測定することができない。アプリケーション層では、実際に ack が返ってくるまでの時間を測定することができないためである。

## 4.3 パケットモニタによる挙動観測

従来のログ解析の限界を克服するため、本研究では、パケットをモニタし、そのネットワーク上のコネクションを追跡することによって、WWW サーバシステムの挙動を観測するという新たな手法を提案する。本手法では、観測される WWW サーバシステムの系を乱さずに WWW サーバシステムの挙動観測が可能である。また、WWW サーバシステムに負荷を与えないで、挙動観測することも可能である。

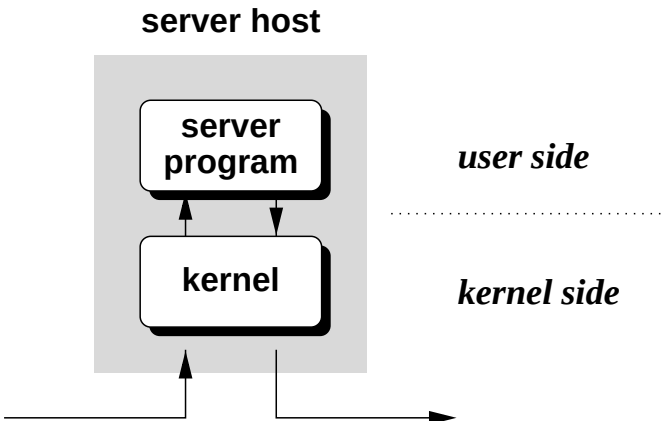



図 4.2: ホスト内のパケットの流れ

従来のログ解析手法では、図 4.1の accept システムコールから close システムコールまでしか測定できなかったが、パケットモニタを行うことによって、i 〜 vii までの時間を測定することが可能となる。これにより、WWW サーバシステムに対するコネクションの正確な継続時間が測定できる。

### 4.3.1　コネクションの追跡

HTTP セッションは TCP コネクションである。WWW サーバシステムの挙動観測を行うため、HTTP の大半を占める 80 番ポートに対する TCP コネクションの監視に焦点を当てることとする。今回の観測では、コネクションの開始時刻と終了時刻を記録した。

### 4.3.2　コネクション開始の検出

コネクションの開始はクライアントから、アクティブオープンにより開始する。そこでまず、クライアントからの SYN パケットを検出する。同時に、コネクションの継続時間の測定のためにパケットの到着時刻を記録する。次に、サーバがクライアントに対して SYN+ack パケットを返したかどうかを検出する。さらに、SYN+ack パケットに対してクライアントが、ack を返したかどうかを検出する。以上の結果により、コネクションが確立されたと判断する。

### 4.3.3　コネクション終了の検出

コネクションの終了は、サーバとクライアントのどちらか一方がアクティブクローズを始めることで行われる。通常、HTTP セッションが正常に終了した場合、サーバがアクティブクローズを行う。しかし、ユーザがブラウザの中止ボタンを押すなど、クライアントが

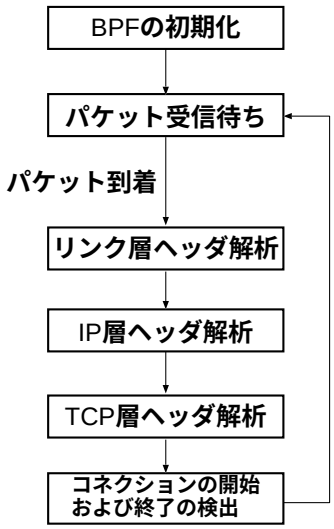



図 4.3: パケットモニタの制御の流れ

アクティブクローズする場合もある。よって、サーバ側からの終了と、クライアント側からの終了について考慮する必要がある。

コネクション終了の検出は、まず、アクティブクローズの FIN を検出し、その FIN に対する ack を検出する。そして、パッシブクローズ側の FIN を検出し、その FIN に対する ack を検出する。パッシブクローズに対する ack を検出した時点で、コネクションが終了したと判断し、その時間を記録する。また、コネクション終了のもう 1 つの可能性として、RST パケットがある。RST パケットが送信されてきた場合、ただちにそのコネクションの終了を行い、終了時刻を記録する。記録された開始時刻と終了時刻によって、正確なコネクションの継続時間を算出する。

## 4.4 実装

### 4.4.1 パケットモニタ

広く用いられているパケットモニタのシステムとして、バークレイパケットフィルタ (BPF)[88] がある。この BPF を用いたライブラリとして、libpcap がある。パケットモニタプログラムは libpcap を用い、C 言語で記述を行った。モニタプログラムは、ユーザレベルで動作するので、カーネルを変更せずに、一般的な UNIX 上での実現が可能となっている。

### 4.4.2 プログラムの処理の流れ

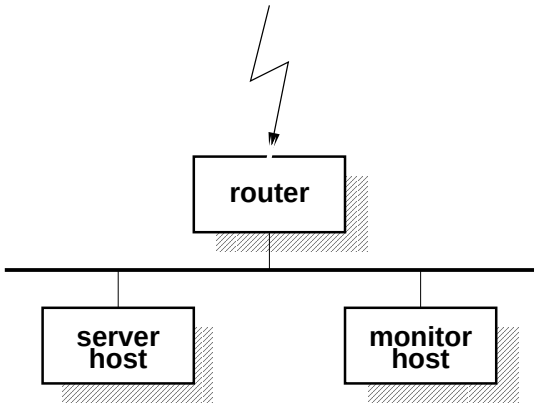



図 4.4: 観測環境

図 4.3に示すように、まず、BPF の初期化を行う。そして、パケット受信待ちに入る。次にパケットが到着すると、リンク層のヘッダ解析を行い、IP 層のヘッダ解析を行う。IP 層のヘッダ解析の際に、クライアントおよびサーバの IP アドレスを記録しておく。次の TCP 層のヘッダ解析の際には、クライアントおよびサーバのポート番号、そして、シーケンス番号および ack 番号を記録する。また、TCP 層のヘッダ解析で、SYN、FIN、ACK、RST などのフラグを確認することにより、コネクションの開始と終了を記録する。コネクションの終了を検出すると、終了したコネクションに関する情報を出力する。

## 4.5　観測例

### 4.5.1　観測対象

本研究では、アクセスが集中的に発生する、第 80 回全国高校野球選手権大会の Internet 中継に用いられた WWW サーバを観測対象とした。

### 4.5.2　マシン構成

モニタホストには、IBM-PC を用いた。スペックは、CPU が PentiumII 300MHz、メモリは 64MBytes、OS は FreeBSD 2.2.6-Release である。ネットワークインタフェースとして、100Mbit Ethernet Interface Card を用いた。WWW サーバホストには、Sun Enterprise 450 を用いた。スペックは、2 個の Ultra SPARC 300MHz、メモリ 512MBytes、OS は SunOS 5.6 である。

### 4.5.3　観測方法

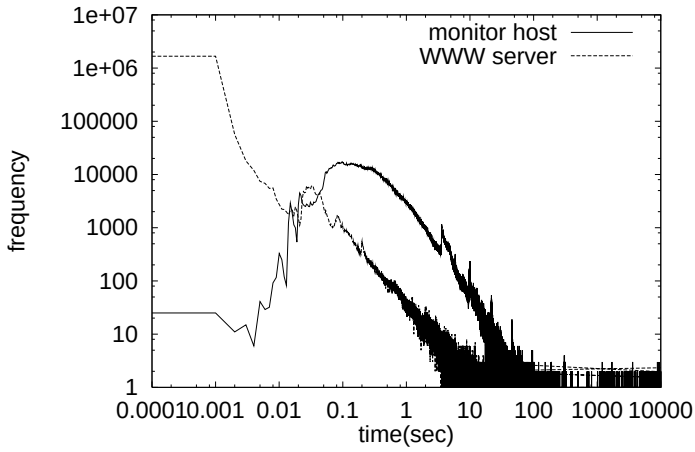



図 4.5: コネクション継続時間頻度分布

図 4.4のように、WWW サーバホストとモニタホストを同一セグメントに置き、モニタホストでパケットモニタプログラムを起動した。サーバシステムの挙動を観測するために、WWW サーバホストとモニタホストはバス共有型のネットワークで接続した。

### 4.5.4 結果および考察

図 4.5にコネクション継続時間の頻度分布を示す。WWW サーバプログラムの記録しているコネクションの継続時間が、パケットを送受信している時間と異なることがわかる。図 4.5の WWW サーバプログラムの出力ログによる出力結果では、0.001 秒が最も多く、次に 0.02 秒付近が多くなっている。モニタプログラムの出力ログでは、0.1 秒付近の継続時間が最も多い。これにより、WWW サーバプログラムの出力結果では、実際の継続時間よりも短く報告されていることが明らかとなった。

図 4.5における、WWW サーバプログラムとモニタプログラムでの時間の差について考察する。図 4.6に示すように、WWW サーバプログラムから見ると、SYN+ack に対する ack を受け取った時に、accept システムコールが終了する。その後、コネクションの開始時刻の記録を行う。そして、read システムコールを実行し、HTTP request ヘッダを処理し、その結果を write システムコールを用いて書き込む。8/17 の 90%以上のオブジェクトサイズは図 4.7に示すように、5Kbytes に収まる。今回用いた OS の socket buffer は 8KBytes なので、オブジェクト全てがカーネルの socket buffer に入り write システムコールは終了する。write システムコールが終了すると、close システムコールが呼ばれ、終了時刻が記録される。この流れから、図 4.6に示すように、WWW サーバプログラムのログではほとんどの場合、クライアントからの要求到着直後に終了時刻が記録される。

TCP レベルでのデータ転送を考えると、1 オブジェクトが 4.5KBytes なら、それを構成

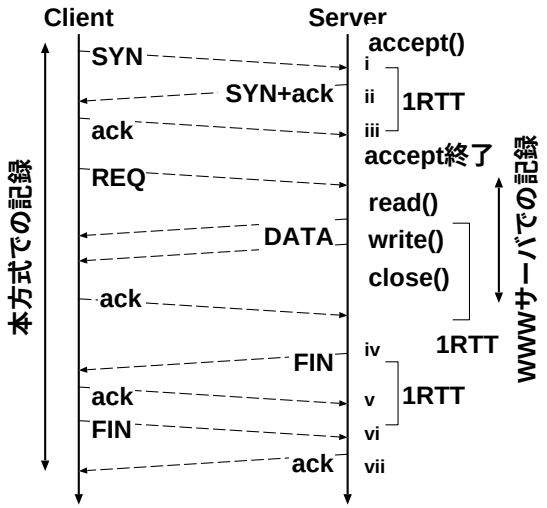



図 4.6: コネクションの状態とシステムコールの関係

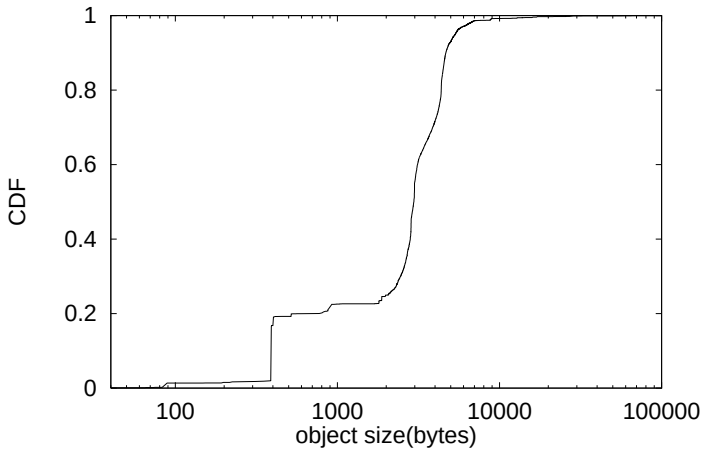



図 4.7: オブジェクトサイズの累積分布

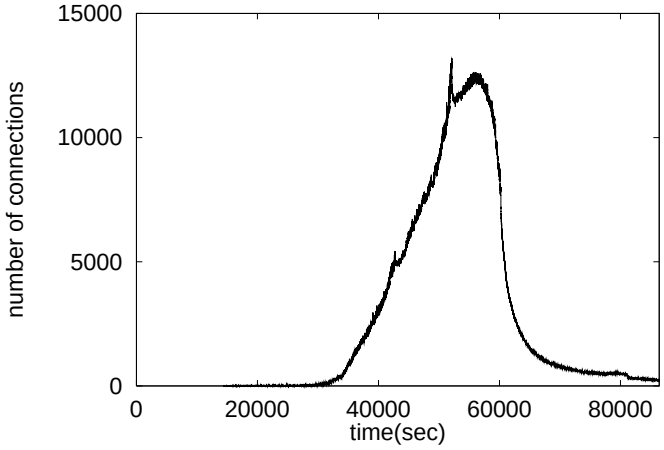



図 4.8: 1 秒間に保持しているコネクションの遷移

するためには少なくとも 4 パケット必要である。4 パケットの送受信には少なくとも 2RTT かかる。そして終了の際に、FIN を送り 1RTT 後に ack が返ってくる。したがって、SYN から数えると、4RTT 以上の時間がかかると考えられる。よって、WWW サーバプログラムの出力ログでは 1RTT に満たない時間であることに対して、IP パケットのレベルでは少なくとも 3RTT かかると考えられる。この差が、パケットモニタによる 0.2 秒と WWW サーバプログラムのログ解析による 0.001 秒という差であると考えられる。

次に、WWW サーバプログラムの出力ログのピークが 0.001 秒であることついて考察する。今回用いた OS のタイマ割り込みが 0.001 秒毎に行われるので、それ以上短い時間は 0.001 秒と計測される。これより、WWW サーバプログラムの出力ログでは、0.001 秒近傍あるいは 0.001 秒以下で処理されたことがわかる。

次に、本システムによる測定の特徴の一つである、WWW サーバシステムが同時に保持しているコネクション数を図 4.8に示す。図 4.8 は、1 秒間に同時に存在するコネクション数の遷移である。この図より、ピークでは、同時に約 13000 コネクションを処理していることがわかる。

本手法との比較のために、図 4.9に同日の WWW サーバプログラムのログ解析による、1 秒間に同時に存在したコネクション数の遷移を示す。図 4.9からは、ピーク時に約 550 コネクションを同時に処理したと算出されている。図 4.8および図 4.9から、本システムを用いることにより、WWW サーバプログラムのログ解析からは分からなかった、WWW サーバシステムが同時に処理しているコネクション数を明らかにすることができた。

モニタホストで十分にパケットを受信できたかを検証するために、単位時間あたりに受けたコネクション数を図 4.10に示す。図 4.10では、WWW サーバプログラムの出力した結果とモニタホストの出力した結果は、ほぼ一致している。したがって、WWW サーバシス

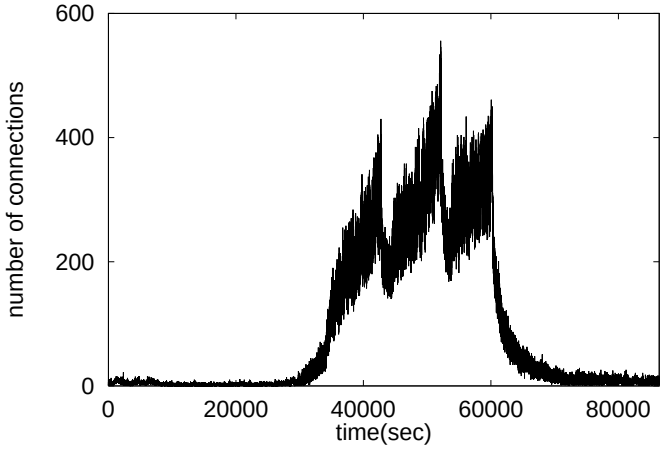



図 4.9: WWW サーバのログ解析によるコネクション保持数の遷移

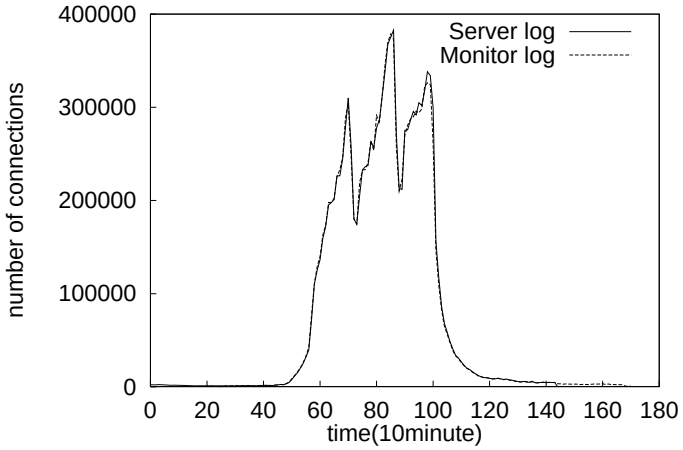



図 4.10: 8/17 日での単位時間あたりに受けたコネクション数

テムに対するパケットを、モニタホストでも正しく受信していることが明らかとなった。

## 4.6 議論

WWW サーバプログラムの出力するログはアプリケーション層での処理であり、OS 側のトランスポート層やネットワーク層での挙動観測は記録することができない。トランスポート層やネットワーク層のログを残す手法も考えられるが、ログ記録のオーバヘッドで OS の時間管理が乱れる可能性がある。

モニタホストは、サーバホストの性能に見合った性能を持つ必要がある。今回の測定では、両者に比較的性能の差があるにもかかわらず、パケットをモニタすることができた。しかし、極端に性能の差があった場合、モニタホストでのパケットの取りこぼしが生ずる可能性がある。

## 4.7 今後の課題

今回の測定にあたって、再送パケットの検出は行わなかった。再送パケットの検出は、TCP ヘッダのシーケンス番号を追跡することによって可能となる。

本方式を用いたシステムの発展として、代理サーバなどの、HTTP 関連システムの観測が考えられる。また、さらに詳細な測定を行うことによって、WWW サーバシステムの評価方法や、WWW サーバシステムのチューニングなどに適用することも可能となる。

## 4.8 まとめ

これまで、運用している WWW サーバシステムの挙動観測の方法は、WWW サーバが出力するログを元に行われてきた。しかし、WWW サーバのログは、アプリケーション層での測定であるので、コネクションの継続時間を正確に測定することはできなかった。

本論文では、パケットモニタによる WWW サーバ挙動観測方法を提案した。本方式により正確なコネクションの継続時間および WWW サーバが同時に保持しているコネクション数を測定することが可能となった。また、本方式を用いたシステムを実際に運用されている WWW サーバシステムに適用することによりその効果を確認した。

# 第 5 章

# 分散型キャッシュシステムの評価

WWW における分散型キャッシュシステムは、複数のキャッシュが連動し、1 つのキャッシュとして機能するシステムである。その目的は、インターネット上のオリジンサーバとの高価で低速な通信を比較的安価で高速なキャッシュ間通信に置き換えることにより、トラヒック量と応答時間を削減することである。

分散型キャッシュシステムに用いられているプロキシサーバには Harvest[32] や Squid[3] などがある。これらのシステムでは、各キャッシングプロキシが sibling や parent と呼ばれる関係を持ち、互いにキャッシュを参照する。各キャッシングプロキシが保持しているオブジェクト情報を交換するのに用いるプロトコルが Internet Cache Protocol (ICP)[126] である。ICP は UDP を用いたシンプルなプロトコルで、実際に発生するトラヒック量が小さいため、効率的であるとされている。

分散型キャッシュシステムにおけるキャッシュヒットには次の 2 通りがある。1 つはローカルヒットであり、HTTP 要求を受けたキャッシングプロキシのローカルキャッシュにヒットした場合である。もう 1 つはリモートヒットであり、sibling や parent へ ICP を用いて問い合わせた結果、リモートのキャッシングプロキシでヒットした場合である。

分散型キャッシュシステムにおいては、リモートヒット率を少しでも上げるために多くの sibling 関係をキャッシュ間で結ぶ傾向がある。その結果、多くの ICP 問い合わせメッセージが小さなパケットとなってバースト的に送られる[1]。

この現象はルータの負荷を増加させる。ルータの負荷は、通過する総トラヒック量だけではなく総パケット数にも依存する。そのため、多くの小さなパケットを交換する ICP は、ルータに対して高い負荷を与える。

これまで、多くの研究でヒット率 [12, 44] やトラヒック量 [21, 146, 167] に関する議論がされているが、パケット数に着目した研究はそれほど行われていない。本章では、ICP を用いた分散型キャッシュシステムをトラヒック量ではなくパケット数に着目してモデル化を行い、ローカルヒット率および設定する sibling の数によるパケット数の変化を明らかにする。さらに、実際に運用されているシステムのログを用いてパケット数の変化を明らかにする。最後に、分散型キャッシュシステム構築の指針を示す。

[1] しかも、このリモートヒットとローカルヒットを比較すると、リモートヒットの確率のほうが一般的に低い。

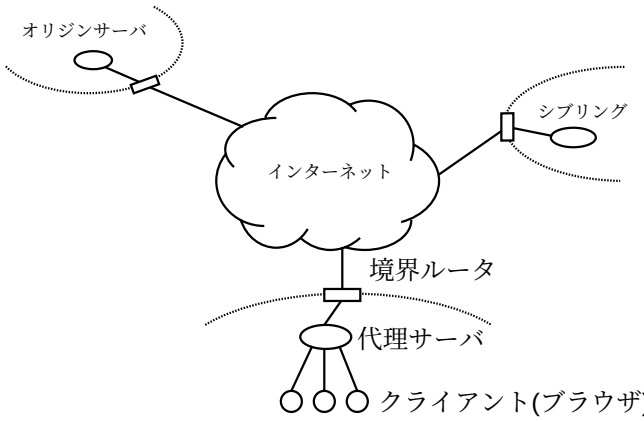



図 5.1: 分散型キャッシュシステムのネットワーク環境

以後、誤解の可能性がない限り、キャッシングプロキシのことを単にキャッシュと記す。

## 5.1 パケット数算出の必要性

ICP は UDP を用いたプロトコルで、URL をペイロードに格納し、20 バイトのヘッダを付加して送る。URL の平均サイズは約 50byte[2]であるため、ICP 問い合わせメッセージの平均データサイズは 100byte 以下となる。WWW オブジェクトの平均サイズは 10kbyte 程度であることから [167]、ICP による通信はトラヒック量的に小さい。

このように ICP はトラヒック量的には問題が少ないと考えられているが、ネットワーク、特にルータに大きな負荷を与えるという問題がある。ルータの性能には主に次の 2 つの制約がある。1 つは、次のホップを決定するためにルーティングテーブルを検索することであり、もう 1 つは到着したパケットを出力先インターフェースへコピーすることである。多くの sibling を参照するように設定すると、ICP 問い合わせメッセージはそれぞれに対してバースト的に送信される。また、リモートヒット率はあまり期待できないこともこの問題をさらに深刻なものにしている。

以上の理由により、ルータの負荷を評価するためにはトラヒック量の分析だけではなくパケット数の分析も必要であり、そのためには HTTP および ICP によるパケットの数を計測する必要がある。

しかし、実際にパケット数を計測しようとすると、多くの障害が生じる。まず、ICP によるパケット数の計測は、送信側と受信側の双方で同時に行う必要がある。なぜなら、ICP

[2]キャッシュに記録された 98 年 5 月のアクセスログ 343 万件の平均値。

は UDP を使用しているために、パケットロスを検出できないからである。送信側と受信側は異なる組織に存在することが多いため、協調して同時に計測することは困難である。

次に、パラメータとパケット数の相関関係を明らかにするために、sibling の参照設定を変更しながら発生するパケット数の変化を計測する必要がある。しかし、実際にユーザに対してサービスを提供しているシステムにおいて、設定パラメータを頻繁に変更して実験することは不可能である。さらに、設定パラメータの変更が可能であっても、HTTP 要求およびキャッシュの状態は変動が激しく、再現性が低い。そのため、計測のたびに結果が変化し、パラメータの変更による影響を判別することが難しい。

本研究では、パケット数の計測が困難であることから、環境をモデル化してパケット数を算出し、各種パラメータのパケット数に対する影響を明らかにする。また、実際に運用されているキャッシュのログにこのモデルを適用してパケット数を算出し、考察を加える。

## 5.2　1 要求あたりのパケット数に関する分析

本章では、HTTP および ICP において送り出される総パケット数の算出に必要となる、1 要求あたりに送り出されるパケット数をそれぞれ導出する。

### 5.2.1　1 回の HTTP 要求あたりに必要なパケット数の評価

HTTP 要求は TCP を用いて行われる。オブジェクトを取得するのに必要なパケットは図 5.2のようにモデル化できる。

まず、HTTP セッションを開始するために 3-way ハンドシェイクが行われる。次に、クライアントからキャッシュもしくはオリジンサーバに対して GET メソッドが送られる。その後、要求したオブジェクトが送られ、最後にセッション終了のための FIN パケットが双方より送られる。

以上のモデルにおいてパケット数を算出する。まず 3-way ハンドシェイクで 3 パケット送られる。次に、GET メソッドを送るのに必要なパケット数は、URL の平均サイズが約 50byte であることから、ここでは 1 とする。オブジェクトデータは、経路 Maximum Transmission Unit(MTU) のサイズのパケットに分割して送られるが、経路 MTU の多くは 1500byte であることが知られている [166]。ここでは、IP および TCP プロトコルヘッダを除いた 1460byte のペイロードに格納するとして、オブジェクトサイズの平均値が約 10kbyte であることから、サーバからクライアントへ 7 パケット送られるものとする。また、この間クライアントから ACK が 3 パケット[3]送られるものとする。そして、サーバーからの FIN パケットとそれに対する FINACK パケット、クライアントからの FIN パケットとそれに対する FINACK パケットで、コネクションの切断に 4 パケット送信される。

以上が典型的な HTTP 要求で、IP パケットが 18 パケット発生する。

[3]delayed ack によって応答するため全データパケットに対して ACK が送られるわけではない

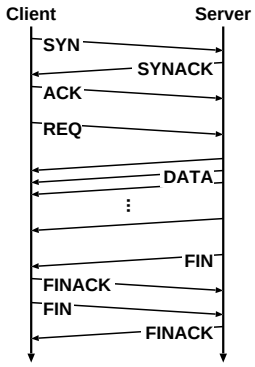



図 5.2: HTTP 要求のモデル (図では DATA に対する ACK を省略)

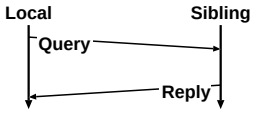



図 5.3: ICP 問い合わせメッセージのモデル

### 5.2.2 1 回の ICP 問い合わせあたりに必要なパケット数の評価

ICP 問い合わせは UDP を用いて行われる。ICP 問い合わせに必要なパケットは、図 5.3 のようにモデル化できる。

まず、キャッシュは sibling に対して ICP 問い合わせメッセージを送る。ICP 問い合わせメッセージを受け取った sibling は、要求されたオブジェクトを検索し、結果をキャッシュに返す。

必要なパケット数は以下のように算出できる。ICP 問い合わせメッセージは、20byte のヘッダに要求オブジェクトの URL がペイロードとして格納される。URL の平均サイズは約 50byte であることから 1 パケットに全て収まり、問い合わせとその返答で 2 パケットとなる。

## 5.3 モデル化

本章では、パケット数の算出に必要な環境のモデル化を行う。

まず、本研究の対象となるネットワーク環境について簡単にモデル化し、次に、要求フローのモデル化を行う。具体的には、それぞれの要求の送信者および受信者をモデル化し、その要求数を算出する。さらに、そのモデルにもとづき、交換されるパケット数を定式化

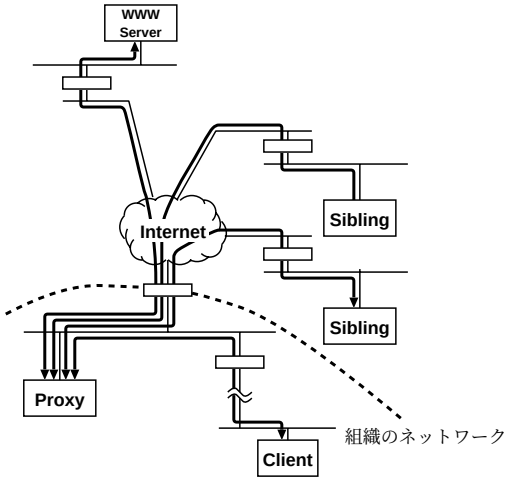



図 5.4: 想定するネットワーク環境

する。

### 5.3.1　ネットワーク環境について

まず、想定しているネットワーク環境について述べる。本モデルでは図 5.4のようなネットワーク環境を想定している。インターネットに接続されている組織において、キャッシュが LAN に接続されている。同じく組織内の LAN 環境に多くのクライアントが接続され、これらはキャッシュを通じてインターネット上の WWW オブジェクトを取得する。また、このキャッシュはインターネット上の複数のキャッシュを sibling として参照し、逆にインターネット上の複数のキャッシュから sibling として参照されうるとする。

### 5.3.2　要求フローのモデル化

次に、HTTP 要求および ICP 問い合わせのフローをモデル化する。

図 5.5のように、クライアントの集合 $C = C_1, ..., C_w$ から送られた HTTP 要求がキャッシュ$P$ へ到着する。この要求は、キャッシュが適切に処理を行い、オブジェクトをクライアントへ送ることで終了する。その HTTP 要求の処理において、$P$ のローカルキャッシュにヒットした場合、トラヒックは外部に向けて発生しない。ミスした場合、$P$ は sibling の集合 $S = S_1, ..., S_u$ に対して ICP 問い合わせメッセージを送る。この ICP 問い合わせは、ヒットもしくはミスの結果を受け取ったら終了する。リモートキャッシュにヒットした場合はその sibling へ、ミスした場合はオリジンサーバ $O$ へ HTTP 要求が送られる。

こうした要求フローと独立に、sibling の集合 $S' = S'_1, ..., S'_v$ から $P$ のキャッシュが参照される。以後、この $S'$ に属する sibling を $P$ の外部 sibling と呼ぶ。外部 sibling は ICP 問

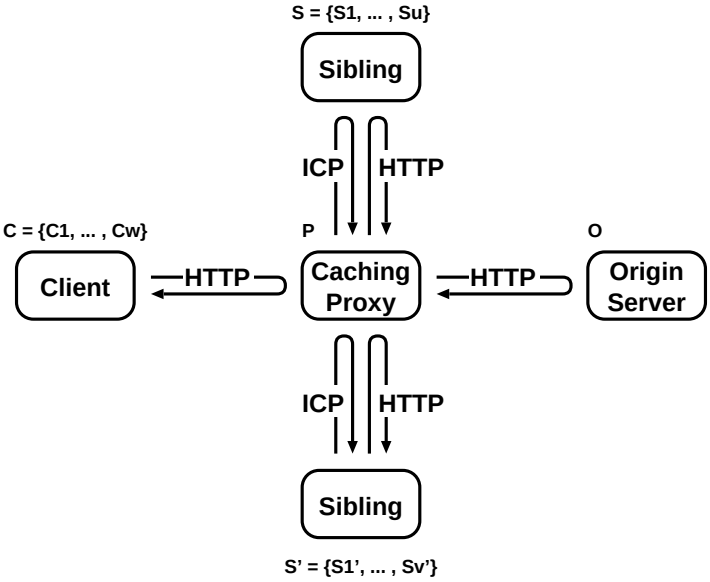



図 5.5: 要求フローのモデル

い合わせメッセージを $P$ へ送り返事を待つ。そして、最初に受け取ったヒットの結果が $P$ からのものであったなら引き続いて HTTP 要求を $P$ へ送る。

以後、このモデルにしたがって分析を行う。

### 5.3.3 必要なパケット数のモデル化

これまでに述べたモデルにもとづき、HTTP 要求および ICP 問い合わせを行う際にネットワークを通過するパケット数を定式化する。

まず、一定時間 $T$(例えば 1 日間) にキャッシュ $P$ が接続されているネットワークを通過する全パケット数を $N$ とする。このうち、クライアントの集合 $C$ の HTTP 要求を処理するために交換されるパケット数を $N_1$ とし、外部 sibling の集合 $S'$ の ICP 問い合わせおよび HTTP を処理するのに交換されるパケット数を $N_2$ とする。

以下、$N_1$ と $N_2$ についてそれぞれ定式化する。

- **クライアントの HTTP 要求を処理するのに必要なパケット数のモデル化**

  先にクライアントが送る HTTP 要求 1 つにつき送信されるパケット数の期待値を求め、その後 $N_1$ を求める。

  ここで、以下のような前提をおく。

  $R$ : $T$ の間に到着するクライアントの HTTP 要求数

表 5.1: クライアントの HTTP 要求あたりのパケット数

| フロー | 確率 | パケット数 |
|---|---|---|
| $C \leftrightarrow P$ (HTTP) | 1 | $n$ |
| $P \leftrightarrow S$ (ICP) | $1-p$ | $um$ |
| $P \leftrightarrow S_i$ (HTTP) | $q$ | $n$ |
| $P \leftrightarrow O$ (HTTP) | $1-p-q$ | $n$ |

$p$ : キャッシュ$P$ のローカルヒット率 (全 HTTP 要求に対する率)

$q$ : キャッシュ$P$ のリモートヒット率 (全 HTTP 要求に対する率)

$u$ : キャッシュ$P$ が参照する sibling の数

$n$ : 1 つの HTTP 要求のためにネットワークを流れる全パケット数

$m$ : 1 つの ICP 問い合わせのためにネットワークを流れる全パケット数

まず、クライアントの集合 $C$ から、HTTP 要求が送られる。この HTTP セッションを開始して終了するまでにネットワークを流れる総パケット数は $n$ である。次に、キャッシュ$P$ におけるローカルヒット率は $p$ なので、$(1-p)$ の確率で $u$ 台の sibling へ ICP 問い合わせメッセージが送られる。すると、確率 $q$ でリモートヒットし、ヒットした sibling へ HTTP 要求が送られる。全くヒットしない確率は $(1-p-q)$ で、HTTP 要求がオリジンサーバ (もしくは parent) へ送られる。以上をまとめると表 5.1 になる。

ゆえに、$N_1$ は以下のようになる。

$$\begin{aligned} N_1 &= R\{n+(1-p)um+qn+(1-p-q)n\} \\ &= R\{(2-p)n+(1-p)um\} \end{aligned}$$

- **sibling からの ICP 問い合わせを処理するのに必要なパケット数のモデル化**

次に、$N_2$ を評価するためにモデル化を行う。ここで、$P$ は parent として参照されることはないとする。これは、parent として参照される場合、ICP 問い合わせがミスした場合でも HTTP 要求が $P$ にフォワードされることがあり、モデルが複雑になるからである。

前述の条件に、さらに以下のような前提をおく。

$R'_i$ : $T$ の間に sibling $S'_i$ に到着するクライアントからの HTTP 要求数 $(i=1,...,v)$

$v$ : キャッシュ$P$ が参照される sibling の数

$p'_i$ : sibling $S'_i$ のローカルヒット率 $(i=1,...,v)$

$q'_i$ : sibling $S'_i$ のリモートヒット率 $(i = 1, ..., v)$

$u'_i$ : sibling $S'_i$ が参照する sibling の数 $(i = 1, ..., v)$

$k_i$ : sibling $S'_i$ のリモートヒットのうち $P$ におけるヒットが占める確率 $(i = 1, ..., v)$

$A$ : $P$ が sibling $S'$ と交換するする ICP 問い合わせのパケット総数

$B$ : $P$ が sibling $S'$ と交換する HTTP 要求のパケット総数

このとき、$S'_i$ に着目すると、まず $R'_i$ の HTTP 要求が到着し、$p'_i$ の確率で $S'_i$ のキャッシュにローカルヒットする。ローカルヒットしなかった HTTP 要求に対しては、ICP 問い合わせメッセージが $u'_i$ の sibling に対して送られ、そのうちの一つが $P$ に到着する。ゆえに $P$ が交換する ICP 問い合わせのパケットの総数 $A$ は次のようになる。

$$A = m \sum_{i=1}^{v} (1 - p'_i) R'_i$$

その後、$S'_i$ から送られた ICP 問い合わせは確率 $q'_i$ でリモートヒットし、$S'_i$ から HTTP 要求が送られる。そのうち $P$ に対して HTTP 要求が送られる確率は $k_i$ であるから、$P$ が交換する HTTP 要求のパケットの総数 $B$ は次のようになる。

$$B = n \sum_{i=1}^{v} (1 - p'_i) R'_i q'_i k_i$$

ゆえに $N_2$ は以下のようになる。

$$\begin{aligned} N_2 &= A + B \\ &= \sum_{i=1}^{v} (1 - p'_i) R'_i (m + n q'_i k_i) \end{aligned}$$

## 5.4 モデルにおけるパケット数の算出

これまでに定義したモデルにおけるパケット数を算出し、パラメータの変動がネットワーク上に送信されるパケット数へ与える影響を考察する。

まず、総パケット数 $N$ は次のようになる。

$$\begin{aligned} N &= N_1 + N_2 \\ &= R\{(2 - p)n + (1 - p)um\} \\ &\quad + \sum_{i=1}^{v} (1 - p'_i) R'_i (m + n q'_i k_i) \end{aligned}$$

ここでは、キャッシュ$P$ の設定によるパケット数の変動をみるために、設定によって変動するパラメータと変動しないパラメータとに分類し、設定から独立しているパラメータについて代表値を定めて固定する。

キャッシュの設定によって変動するパラメータには次のものがある。

表 5.2: 固定変数の値

| 変数 | $R,\ R_i'$ | $n$ | $m$ | $p_i'$ | $q_i'$ | $k_i$ |
|---|---|---|---|---|---|---|
| 値 | $1 \times 10^5$ | 18 | 2 | 0.4 | 0.1 | 0.25 |

$p$ : キャッシュサイズやキャッシュオブジェクト保持時間等のパラメータチューニングによって変化する

$u$ : 参照する sibling の数によって変動する

$v$ : 参照を受け付ける外部 sibling の数によって変動する

その他の変数の値は、以下の理由によりキャッシュ$P$ の設定からは独立している。

$R,\ R_i'$ : それぞれの組織のクライアントの数や性質による

$n$ : クライアントが要求しているオブジェクトの性質による

$m$ : ICP プロトコルによる

$p_i',\ q_i',\ k_i$ : キャッシュ$S_i'$ の設定による

まず、$R$ と $R_i'$ は、それぞれ 10 万とする。これは、経験的な観測値から代表値を定めた。次に、$n$ と $m$ は 5.2.1節と 5.2.2節で述べた値を採用し、それぞれ 18 と 2 とする。$p_i'$ と $q_i'$ については、文献 [127] においてローカルヒット率が 30-50%、リモートヒット率が 10%程度観測されている[4]ことから、ここでは $p_i'$ と $q_i'$ の値はそれぞれ 0.4 と 0.1 とする。また、$k_i$ については、$P$ を参照する sibling のネットワーク的距離や、ヒット率、参照する sibling の数等によって変動するため、モデル化することは困難である。本論文の目的は設定によるパケット数の変化をみることにあるため、ここでは簡単化のために 0.25 とする。以上をまとめると表 5.2となる。

このとき、$N$ は次のようになる。

$$N = 1 \times 10^5 \{36 - 18p + 2(1-p)u + 1.47v\}$$

ここで、以下の 2 通りの場合について総パケット数をそれぞれ評価する。

1 つは、sibling を参照するが外部 sibling からのキャッシュの参照を受け付ける場合で、$v = 0$ となる。もう 1 つは、sibling を参照し、かつ、外部 sibling からのキャッシュの参照も受け付ける場合で、$v > 0$ となる。

### 5.4.1　ケース 1：外部 sibling からの参照を受け付けない $(v = 0)$ 場合

[4]我々が観測している結果よりリモートヒット率が若干高い。

表 5.3: $N$ の値：$v = 0$ の場合 (単位は 100 万)

| $p \setminus u$ | 0 | 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|---|
| 0.3 | 3.06 | 3.34 | 3.62 | 3.90 | 4.18 | 4.46 |
| 0.4 | 2.88 | 3.12 | 3.36 | 3.60 | 3.84 | 4.08 |
| 0.5 | 2.70 | 2.90 | 3.10 | 3.30 | 3.50 | 3.70 |

このケースは、sibling の参照はするが外部 sibling からの参照は受け付けない場合である。外部 sibling からの参照は、その組織のユーザにとって利益はない。特に容量の小さい回線でインターネットに接続している場合、ローカルユーザの通信を優先するため、外部 sibling による参照は受け付けない場合がある。このケースは、そのような運用形態にあてはまる。

表 5.3に結果を示す。参照する sibling の数が増加するのに比例してパケット総数は増加している。その増加率はローカルヒット率が低い場合には高く、ローカルヒット率が高い場合には低い。つまり、ローカルヒット率が低い場合に、sibling 参照を増やすことでリモートヒット率の向上を図ることはルータの負荷を増加させる。

### 5.4.2 ケース 2：外部 sibling からの参照を受け付ける ($v > 0$) 場合

これは、sibling キャッシュを参照し、かつ、外部 sibling からの参照も受け付ける場合である。比較的容量の大きな回線で外部インターネットに接続され、トポロジ的にバックボーン上流に位置する組織などでは、より下流に位置する組織からの参照を受け付ける場合が考えられる。また、広く公開するキャッシュを運営する場合もこのモデルが当てはまる。

ここでは、このモデルの 1 つの例として、さらに参照する sibling の数 $v$ と参照される外部 sibling の数 $u$ が等しいという仮定もおく。これは、sibling 同士でグループを形成し、他のキャッシュを参照する場合はそのキャッシュからの参照を受け付けるというポリシを用いる場合にあてはまる。

表 5.4に $v = u$ の場合の結果を示す。この場合は、$v = 0$ の場合よりもさらに sibling の数の増加にともなうパケット数の増加率が高くなっている。これは、参照する sibling の数が多くなれば自分を参照する外部 sibling の数も増えるためである。特に、ルータの負荷が問題になっている組織の場合、外部 sibling からの参照の受け付けは慎重に行わなければならない。

## 5.5 実際のキャッシュの評価

実際に運用されている複数のキャッシュのログから計測されたパラメータをもとに、5.3 章のモデルを使いパケット数を算出する。データはいずれも 98 年 5 月の 31 日間のデータ

表 5.4: $N$ の値：$v = u$ の場合 (単位は 100 万)

| $p \setminus u$ | 0 | 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|---|
| 0.3 | 3.06 | 3.63 | 4.21 | 4.78 | 5.36 | 5.93 |
| 0.4 | 2.88 | 3.41 | 3.95 | 4.48 | 5.02 | 5.55 |
| 0.5 | 2.70 | 3.19 | 3.69 | 4.18 | 4.68 | 5.17 |

表 5.5: パラメータの値

| キャッシュ | $R$ | $p$ | $u$ | $q$ | $N_1$ | $A$ | $B$ | $N$ |
|---|---|---|---|---|---|---|---|---|
| キャッシュX | 106545 | 0.34 | 8 | 0.065 | 4308680 | 2254 | 1530 | 4312464 |
| キャッシュY | 173750 | 0.53 | 2 | 0.011 | 4924075 | 182920 | 292824 | 5399819 |
| キャッシュZ | 4857 | 0.26 | 5 | 0.020 | 188063 | 136012 | 32904 | 356979 |

から 1 日の平均値を求めている。

### 5.5.1　パラメータの値

ログから、各種パラメータの値を評価した (表 5.5)。$n$ と $m$ の値をそれぞれ 18、2 とし、計測された外部 sibling からの ICP 問い合わせの数と HTTP 要求の数から $A$ と $B$ の値を算出した。また、送られる総パケット数には影響を与えないが、参考のためにリモートヒット率 $q$ の値も計測した。

### 5.5.2　キャッシュの分析

表 5.5の結果にもとづき、それぞれのキャッシュについて考察を行う。

- **キャッシュX**

  このキャッシュは、sibling は参照するが、外部 sibling による参照は受け付けない例[5]と考えることができ、ケース 1 に相当する。ここでは、パラメータチューニングによってローカルヒット率の変化が 10%を越えるのは非現実的であると考え、ローカルヒット率の増加が 0%、5%、10%の場合についてそれぞれ算出する。$p$ と $u$ に関して総パケット数 $N$ の変化を表 5.6に示す。

  結果から、8 つの sibling を参照することにより 6.5%のリモートヒット率を達成しているが、パケット数は sibling を参照しない場合の 319 万から 431 万へ 35%増加して

[5]若干の被参照があるが、非常に小さいのでここでは無視する

表 5.6: キャッシュX(単位は 100 万)

| $p \setminus u$ | 0 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| 0.34 | 3.19 | 3.47 | 3.75 | 4.03 | 4.31 |
| 0.39 | 3.09 | 3.35 | 3.61 | 3.87 | 4.13 |
| 0.44 | 3.00 | 3.23 | 3.47 | 3.71 | 3.95 |

表 5.7: キャッシュY(単位は 100 万)

| $p \setminus u$ | 0 | 1 | 2 |
|---|---|---|---|
| 0.53 | 5.07 | 5.24 | 5.40 |

いることが分かる。

一方、ローカルヒット率の向上によって、総パケット数を減らすことが可能である。参照する sibling の数によるが、10%のヒット率向上で 6-9%のパケットが削減される。キャッシュのパラメータチューニングが重要であることを示す一例である。

- **キャッシュY**

  このキャッシュは、sibling を参照し、かつ、外部 sibling からの参照も受け付ける例であり、ケース 2 に相当する。キャッシュX と異なり、ローカルヒット率が 53%と高く、パラメータチューニングによってこれ以上大きくローカルヒット率を向上することは難しいと考えられる。そのため、$u$ についてのみ総パケット数の変化を算出した。結果を表 5.7に示す。

  リモートヒット率は 1.1%と小さいため、sibling 参照を停止することでパケット数を削減することが可能である。その場合、パケット数は 540 万から 507 万へ減少し、6.1%の削減となる。

- **キャッシュZ**

  ローカルクライアントは少ないが、外部 sibling からの ICP 問い合わせメッセージは多く到着している例である。

  このサイトに到着した ICP 問い合わせメッセージに対してその 2.7%にあたる数の HTTP 要求がこのキャッシュに対して送られている。しかし、キャッシュZ は parent キャッシュとしても参照されているために、ヒットしていないにもかかわらず HTTP 要求が送られてきている。そのため、実際のヒット率はさらに小さく、多くのパケットが無駄になっていることがわかる。

このようなキャッシュは無駄が多いため、参照しない方がよい。

## 5.6　分散型キャッシュシステム構築の指針

以上の分析にもとづき、分散型キャッシュシステムを構築する上で、ルータに与える負荷という観点から挙げられる指針を示す。

- sibling 設定は必要最小限に抑える

  キャッシュX の例で見たように、参照する sibling を多く設定するとパケット数は大幅に増加する。したがって、sibling を設定する場合はそのヒット率を計測し、ヒットしないキャッシュに対しては sibling 設定から除く必要がある。特に、リモートヒットは ICP 問い合わせに対して最初に到着したヒットの結果が利用されるため、ネットワーク的に距離の離れた sibling の場合はヒットの結果が届くまでの遅延が大きく、ヒットを返しても利用度は低いと考えられる。このような sibling に関しても、ヒット率を考慮しながら設定する必要がある。

- ローカルヒット率向上のためのチューニングを行う

  ローカルヒット率を向上すると、外部サーバへ送られる HTTP 要求、ICP 問い合わせメッセージともに減少するため、多くのパケットが削減される。そのためには、キャッシュに送られる HTTP 要求の傾向を随時計測し、それにあわせて各種パラメータを設定することが必要である。

- 受け付ける ICP 問い合わせは必要最小限に抑える

  キャッシュZ のようにキャッシュサービスを広く公開する目的で設置したのではない場合、ルータの負荷を軽減する方法の 1 つは受け付ける ICP の要求を制限することである。そのためには、外部 sibling からの ICP 問い合わせに対してどれだけヒットを返しているか、また、どれだけヒットが利用されているかを計測し、適切に設定する必要がある。

## 5.7　議論

今回の分析では、さまざまな仮定のもとにモデル化を行った。そのため、いくつかの限界点が存在する。

まず、parent について考慮していないことである。parent としての参照を受け付ける場合、たとえ ICP 問い合わせに対してミスを返しても HTTP 要求がフォワードされる場合がある。そのため、parent を含めたモデル化が必要である。また、今回は局所的な視点でパケット数を算出しているが、分散型キャッシュシステムの性能を評価するためには、バッ

クボーントラヒックを含めた広域ネットワーク上でのモデル化が必要である。さらに、実現が難しいことは 5.1節で述べた通りであるが、実際のパケット数を計測しなければモデルの正当性を証明できない。例えば、HTTP 要求においてはパケットの再送が起こりうるし、UDP パケットは到着は保証されない。そのため、実際のパケット数の計測が必要である。これらは今後の課題としたい。

## 5.8 まとめ

ICP を用いた分散型キャッシュシステムが広く利用されつつある。ICP はシンプルで、トラヒック効率のよいプロトコルであるとされているが、多くの sibling を参照すると ICP 問い合わせメッセージが多数送られるという問題がある。

本研究では、キャッシュが接続されているネットワーク上の通信において、パケット数に着目して環境をモデル化し、分析した。まず、HTTP および ICP について 1 要求当たりのパケット数を算出し、モデルにしたがってネットワークを流れる総パケット数を算出した。その結果、sibling を多く参照すればする程、より多くのパケットが送られることが判明した。また、このパケット数が増加する傾向はローカルヒット率が低い場合により顕著になり、外部 sibling からの参照を受け付けている場合には事態はさらに悪化することも判明した。

さらに、このモデルに従って実際のキャッシュのログを用いて分析し、リモートヒット率に余り貢献しない ICP パケットが多く送られていることを明らかにした。実際のシステムでは、リモートヒット率と ICP 問い合わせによって発生するパケット数のバランスを取ることが重要である。

# 第 6 章

# WebCache の性能限界と対策

多くの組織が WWW の円滑な利用のために、WWW のキャッシュ (cache) を設置している。しかしながら、前章まで紹介してきた数々の研究結果で分かるように、WWW キャッシュも他の例に漏れず万能なシステムではなく、性能の限界が存在する。そして、高い性能を引き出すためにはその性質にあわせた設置が必要である。本章では、キャッシュの性能の限界、そして性能を引き出すための設置の指針について述べる。

## 6.1 とりまく状況の変化

いまや Internet はボランティアベースの黎明期が過ぎ、対価に応じて帯域や reachability が提供される時代に至った。Internet サービスの利用方針や接続方針は、数年前のように帯域に応じて利用を制限するのではなく、利用に応じて帯域を確保する方向に変わりつつある。

このような方針の変化を受けて、WWW キャッシュの設置や運用の方針も変わりつつある。一般的に、WWW キャッシュ設置の目的は帯域の有効利用や応答時間の短縮である。帯域の面で見れば、帯域の拡張以上に安価そして手間をかけずに帯域を有効利用できるのかを考えねばならない。応答時間の面では、キャッシュを介することによって生じる遅延を許容できるほどの応答時間の短縮が実現できるかを考える必要がある。

また、WWW キャッシュは組織に一つ (あるいは一組) 設置されることが一般的である。したがって、キャッシュの設置には組織全体の需要に見合う資源 (ネットワーク、CPU、ディスク、メモリ) 、管理者を投入する予算を用意する必要がある。

## 6.2 WWW キャッシュの限界

### 6.2.1 帯域の軽減はわずか

WWW キャッシュは他のキャッシング技術と同様に、過去に利用された情報が再度利用されることを前提としている。しかしながら、ネットワークサービスでは情報の局所性

(locality) がディスクキャッシュやメモリキャッシュに比べ乏しいことが指摘されている。したがって、キャッシュに蓄えた情報が再利用されるかどうかを慎重に検討する必要がある。

既に多くの運用報告や理論的な研究によって、WWW のアクセスが Zipf の法則 (Zipf's law) に沿っていることが明らかにされている (コラムに Zipf の法則の概要を示す)。その性質から、アクセスされた情報の大部分は、再利用されることはなく 1 回アクセスされるのみである。情報の更新や再読み込み (reload) を無視して、無限のキャッシュ領域を用いるという理想的な状態を仮定しても、キャッシュのヒット率は 60% から 70% 程しか期待できない。そして、現実には再読み込みや更新確認、キャッシュ領域が有限であることから発生するキャッシュの破棄等によって、ヒット率は 20%から 40% 程度が一般的であり、帯域削減も 20% から 40% 程度となる。

キャッシュ領域を拡大することによってヒット率が向上することは知られている。しかしながら、ほとんどアクセスされない情報をヒットするまで蓄え続けるためには非常に大量のキャッシュ領域が必要となり、多少のキャッシュ領域の拡大でヒット率の向上を望むことはできない。

近年では、更新頻度の短いページが増え、キャッシュに蓄えられいる情報が古くなる確率が増していることを指摘する意見も存在する。利用者がキャッシュに蓄えられいる情報が古いと感じた場合には再読み込みの頻度も増え、ヒット率はますます低下することになる。なお、「再読み込みしてください」の旨記述されているページも存在して、再読み込みの増加に拍車をかけている。

以上のように、WWW キャッシュの開発当初想定されていた「WWW アクセスの再利用が可能である」という前提は正しいものの、再利用される確率は低く劇的な効果は望めない。つまり、キャッシュによる大きな帯域軽減は期待できない。

### 6.2.2 応答は早くならない

WWW キャッシュの多くは代理サーバ (proxy server) の付加機能として実現されている。したがって、WWW キャッシュの利用においてクライアントの要求とオリジンサーバの応答だけでなく、代理サーバの要求と応答も発生する。キャッシュヒットの際には代理サーバの要求とオリジンサーバの応答が省かれる。オリジンサーバが Internet 上を介した遠距離にあるのに対して、代理サーバは近距離にあるため、高速な応答が得られる。一般にオリジンサーバの応答は数秒から数十秒、代理サーバの応答は数十ミリ秒から数秒となる。一方、ミスの場合には代理サーバの要求と応答が介在するため、代理サーバを介さない場合に比べ応答が遅れる。

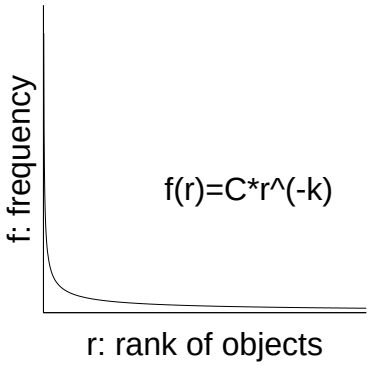
したがって、理論的にはアクセス全体で高速な応答が得られるかどうかは、ヒット率に大きく依存する。オリジンサーバとの通信速度と代理サーバの通信速度に大きな差がある場合、数パーセントのヒット率があればアクセス全体の応答時間は向上する。

コラム: Zipf's law

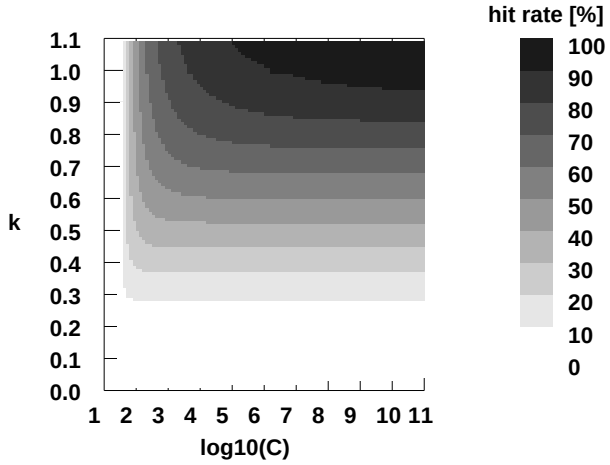
Zipf's law は文章中の単語の出現に関する法則で、「単語の頻度はランクに反比例する」という経験的法則である。この法則は以下のような (6.1) 式で表される。(6.1) 式を (6.2) 式のように一般化すると、ヒット率の上限を求めることが可能となる。

$$f \sim \frac{1}{r} \tag{6.1}$$

$$f(r) = Cr^{-k} \tag{6.2}$$




$C$ と $k$ をパラメータとして、ヒット率の上限は以下のようになる。




ところが、キャッシュには多数のクライアントのアクセスが集中するため、理論的な性能を発揮することは難しい。オリジンサーバは WAN を介した遅い通信が主体であるが、キャッシュは LAN を介した早い通信が主体であるため、オリジンサーバにまして負荷が高い。負荷が高くなるとキャッシュのヒットやミスにかかわらず、キャッシュの応答時間が増加する。

このことから、対外線の帯域が広い、あるいは混雑していないサイト、キャッシュの負荷が高いサイトではキャッシュを介さず、オリジンサーバへ直接通信した方が迅速な応答が

得られる。応答速度を重視する場合には、以上の点を考慮する必要がある。

### 6.2.3 新鮮さに欠ける

キャッシュの本質は過去の通信の再利用であるから、キャッシュは過去の情報を提供するシステムであり、情報の新鮮さに欠けるという欠点が存在する。

過去の傾向から更新を予測する研究が盛んに行なわれているが、決定的な手法は見つかっていない。一方で、情報の有効期限を定める手法も実装されているが、この手法を用いてもいつ更新するかは不明で、古い情報の提供を軽減する効果しかなく、新鮮度を保証するには不十分である。現状で、新鮮度を保証する唯一の方法は蓄えた情報を利用せず、逐一オリジンサーバに更新を確認することである。しかし、この方法は蓄えた情報を利用しないことからキャッシュとは呼べない。また、逐一通信処理を行なえば、クライアントがオリジンサーバへ通信する場合以上に応答時間がかかるため、キャッシュの目的である応答時間の短縮を実現できない。

このように、情報の新鮮度を重視した場合、キャッシュは新鮮な情報の取得を阻害するシステムとなる。なお、この新鮮さに欠けるという欠点が、利用者の再読み込みを誘発していることも見逃せない。オリジンサーバに問い合わせずに新鮮度を保証する技術を確立しない限り、再読み込みはなくならないであろう。

### 6.2.4 頑丈な実装は難しい

前述のように、キャッシュでは理論的には少々のヒット率があれば高速な応答が期待できるが、アクセスが集中しても高い性能を維持しつづける実装が必要である。

WWW で用いられる HyperText Transfer Protocol (HTTP) は、他のプロトコルに比べ比較的単純であり並列処理は容易である。しかし、代理サーバでは同時に数十本から数百本のコネクションを並行処理する必要が生じ、多大な負荷が発生する。そして、これらのコネクションの相手であるオリジンサーバとクライアントは応答速度が異なり、多数のオリジンサーバから遅い読み込みを行いつつ、多数のクライアントへの高速な書き込みを行う等、管理は複雑である。

代理サーバにキャッシュ機能を付加した場合には、キャッシュとなる記憶装置からの情報の入出力、そしてキャッシュ内の古い情報の破棄等、さらに複雑さが増す。記憶装置の入出力が発生するとさらなる負荷の増加を生む。

現状ではプロセスの分岐の軽減、スレッドプログラミング、非同期コネクション、非同期 I/O、Domain Name Service (DNS) のキャッシュといったといった高速化手段が施されている。これらの手法を適用するとコネクション毎に複雑な状態が必要になるため、管理がより一層複雑化している。

このように、キャッシュは複雑なシステムであり、実装は困難である。また、運用した場合には複雑かつ他数の処理を行なうため膨大な負荷が発生し、遅延が発生する。この負

荷の増加による遅延はキャッシュヒットやミスに関係なく発生するため、負荷の高いキャッシュではキャッシュ全体が遅延発生システムとなってしまう危険性がある。アクセスが多いキャッシュでは高性能コンピュータの導入が必要であろう。

### 6.2.5　階層型キャッシュの効果も疑問

近年、Harvest に代表される階層型キャッシュが普及しつつある。階層型キャッシュでは複数のキャッシュのキャッシュ領域を利用することが可能となることから、統計的多重化によってヒット率の向上が望める。しかしながら、前述のようにキャッシュ領域の拡大によるヒット率の向上は容易ではないため、楽観視できない。

加えて、第 5章で詳しく述べたように、使い方によっては膨大なパケットを生成し、かえってトラフィックを悪化させる性質がある。対策として、ホップ数に注意したキャッシュの設置を考慮する必要があるが、ホップ数の近い場合にも、いくつかの注意事項が存在する。階層型の配置では、上位のキャッシュほど大量のキャッシュ領域が必要となる。そして処理が集中することから、上位のキャッシュほど高性能なコンピュータが必要となる。また、階層が深くなるにつれ応答時間が増加する。

このように、階層型キャッシュは単体のキャッシュと同様に、大きなヒット率は望めない。応答時間が劣化するなどの危険性がある。また、組織外のネットワークを介してキャッシュの照会をした場合には、広域にわたって帯域を浪費するため、慎重な設定が必要である。

## 6.3　性能向上の対策

キャッシュの目的は帯域軽減と応答時間短縮であるから、一般にその性能はトラフィックや応答時間で評価される。また、前述のようにトラフィックと応答時間はヒット率が大きな影響を与えるため、ヒット率も評価尺度として用いられることが多い。

これまでの研究によって、キャッシュの性能を変化させるパラメータとして、以下のような項目があげられている。

1) そのキャッシュシステムのキャッシュアルゴリズム

2) Internet への経路の実効帯域やホップ数等のネットワーク的性質

3) CPU やバスなどのスループット

4) メモリやディスク等の記憶容量

本節ではこれらのパラメータを変更する対策について述べる。

### 6.3.1 対外線の帯域の拡大

狭い帯域の対外線を用いた場合には、ルータの負荷が増加して、ネットワーク全体の混雑や応答の遅延を招く。混雑した場合には再送によって、さらなる応答の遅延を招く。円滑な利用を実現するためには帯域の確保が重要である。すでに述べたようにキャッシュの帯域軽減は効果があるものの、劇的な効果をもたらすものではない。したがって、帯域の確保には帯域の拡大がもっとも重要な策である。

多くの組織では対外線の帯域拡大には予算等の非技術的な障害が存在することと思われる。しかし、対外線の帯域拡大は、「電話使用量が増えたら外線を増設する」「交通量が増えれば車線を増やす」というパターンのごくありふれた対策であり、有効性の効果について議論の余地はない。劇的な性能向上には帯域拡大を検討すべきである。

後述するような高性能コンピュータを多数購入して維持する費用を考えた場合、帯域拡大が直接的かつもっとも有効な場合が多いことも指摘しておく。さらに、帯域拡大が容易な場合には、帯域を拡大して管理コストのかかるキャッシュを撤廃する選択肢も考慮すべきであろう。

### 6.3.2 高性能コンピュータの投入

高いスループットを得るため、アクセスの多いオリジンサーバの運用では、スーパコンピュータ級の高性能コンピュータを投入することが一般的である。キャッシュはオリジンサーバに比べ処理が複雑、かつ高速な LAN を対象としたシステムであるため、強力な処理能力が要求される。したがって、アクセスの多いキャッシュでも高性能コンピュータを投入すべきである。高性能コンピュータを用意できない場合には、複数のコンピュータを設置して、全体で一つのキャッシュとして運用することも可能である。

なお、アクセスの少ない場合でも、NetNews や anonymous FTP 等の他のサービスとコンピュータを共有すると大きな性能を期待できない。他のサービスとコンピュータを分離して運用することも重要である。

### 6.3.3 記憶装置の容量拡大

コンピュータ自身の性能とともに、キャッシュ領域としての記憶装置の容量の確保も重要である。アクセスの傾向は Zipf's law に沿うため、キャッシュ領域がある程度の容量まで達するとヒット率が向上しない。したがって、極端に大きな記憶装置を用意する必要はない。たとえば、10 ギガバイトのディスクを 100 ギガバイトにしても、飛躍的な性能の向上は望めない。

必要なキャッシュ領域の容量はそのキャッシュの扱うアクセスの傾向と規模による。また、日々の変動も存在するため、最大の性能が得られるキャッシュ容量を算出することは困難である。現状では現在のアクセス状況から類推し、将来のアクセス増加を含んで設定せざ

るを得ない。一般的なキャッシュでは、64 メガバイトから 256 メガバイトのメモリ、数ギガバイトから数十ギガバイトのディスクが確保されている。

一方で、極端にヒット率が低い場合 (数パーセントから 20 パーセント程度) には、記憶装置の容量不足、あるいは設定ミスのおそれがあるので注意を要する。

### 6.3.4　その他

キャッシュアルゴリズムについては既に多くの研究がなされているが、前述の Zipf の法則により劇的なアルゴリズムが発明される可能性は非常に小さい。また、キャッシュシステムは複雑なため、管理者がアルゴリズムを改良したり、新たに実装するのは困難であろう。したがって、管理者は良いアルゴリズムが実装されたキャッシュシステムのリリースや、効率の良いキャッシュパラメータの設定方法等の情報収集が必要であろう。

## 6.4　最近の動向

現在利用されているシステム群は、理論的に求められた性能に達しているとは言いがたい。この理論と現実の性能のギャップを埋めるために、新たな実装技術の開発が行なわれている。WWW に特化したファイルシステム、トランスポートプロトコル、高負荷に耐えるキャッシュの実装技術の設計・開発が行なわれている。

キャッシュは受動的なシステムであるから、過去にアクセスされた情報にのみ応答時間の短縮が行なわれる。応答時間が短縮される対象を増やすため、先読み (prefetching) や複製 (replication) のような能動的なアプローチが研究されている。また、新鮮さを保証するために、オリジンサーバから更新を通知する機構等の研究が行なわれている。これらの技術の実用化の焦点は、その処理に伴う負荷やトラフィックの増加等のデメリットをいかに軽減するかにかかっていると見られている。

一方、WWW に代わるサービスが登場する可能性も忘れてはならないだろう。WWW 以前から存在するハイパーテキストや分散システムの研究成果を Internet に拡張する研究も盛んに行なわれている。

## 6.5　まとめ

本章では WWW キャッシュの限界について述べた。そして、キャッシュを設置しない場合に比べて利用が滞る可能性や、ネットワーク上のトラフィックを増加させる可能性もあることを指摘した。キャッシュは万能でなく、本来の性能を得るためには慎重な設計と観察が必要である。

また、性能を向上させるための技術的な対策も述べたが、性能を劇的に向上させる決定的な策はない。WWW は利用者の最大の関心であり、利用者のサービスへの評価は WWW

で決まる。利用者の円滑な利用を最優先に設計して、コストを十分にかけることが重要である。

# 第 7 章

# おわりに

2 年間の活動を通じて、WWW キャッシュ技術に特化した研究を行ってきた。この間の成果として、広域 WWW キャッシュシステムとしての WIDE CacheBone の構築や、さまざまな WWW キャッシュシステムの稼働実験、透過型キャッシュシステムなどの新しい形態の WWW proxy システムの実験などがある [139]。また、より効果的なキャッシュシステムについての議論も活発に行ってきた。

その一方、第6章でも述べたような、WWW キャッシュシステムの存在意義自体を問う必要があるような状況も生じてきていることは否めない。実際のところ、本 WG の発足とともに構築が開始された、広域分散環境における大規模 WWW キャッシュの WIDE CacheBone が、WIDE インターネット全体のトラフィックを軽減させる効果を生んだかどうかは疑問である。

特に 1998 年度の活動中は、W4C WG 内の議論においても、ICP によるトラフィックの増加への懸念を含めて、広域 WWW キャッシュの効果を疑問視する声が多く聞かれた。技術者としては、客観的に見て効果のないものは切り捨てていかなければならないのは当然である。しかしながら、W4C WG がその名称内に WWW Cache の名前を冠していることが、その作業を躊躇させる、言わば “足かせ” のようなものとなってしまっていた。

それでも、より広範囲の WWW クライアントアクセスデータの収集源としての WIDE CacheBone は、今でも存在価値を光らせているし、WWW キャッシュシステムの採用によって、快適なアクセス環境を得ることができたという例がすべて妄想であったわけではない。つまり、やみくもにキャッシュさえ導入すれば、トラフィック削減の効果があるという幻想に気づいたというのが、現状を現すに的確な表現であろうと思われる。

このように、現時点では “すべての成し得るべき努力” を果たしたとは言いがたく、“WWW キャッシュ完全不要” を主張するには早計である。ただ、そのような結論に到達する可能性も考慮に入れながら、つまり、Web キャッシュ自体も含めた、有用なシステムの取捨選択を行う必要がある。

そのため、本 W4C WG は、1998 年度をもってその活動を終了し、次なるステップとして、WWW システムのプロトコル自体の改良や、キャッシュ以外の proxy システムをも含めて、 WWW アーキテクチャ全体を視野に収めた研究に向かいたいと考える。