修 士 論 文

WWW におけるオブジェクト参照予測による応答速度向上に関する研究

指導教官　篠田陽一 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

牧野泰光

1999 年 2 月 15 日

目 次

図 目 次

表 目 次

第 1 章

序論

1.1 本研究の背景と目的

Internet 上では World Wide Web(以下 WWW と示す) による情報提供が一般的なものとなり、WWW 上で情報を高速に入手できることは、非常に重要な課題である。しかし、クライアント–WWW サーバ間のネットワーク距離や WWW サーバ負荷などの要素により、応答速度のばらつきが起こり、ブラウザでの閲覧において快適でないオブジェクトが存在する。

これらの応答速度の遅延を緩和する方法として、キャッシングプロキシなどの一度アクセスした情報を再利用する方法が広く用いられている。既存のキャッシングプロキシは、一度取得した再利用可能オブジェクトをキャッシュに格納する。そして、今後再利用された場合にキャッシュオブジェクトを提供することによって重複するネットワーク負荷を減少する。しかし、再利用可能オブジェクトには、取得にかかるコスト、URL 別のアクセス頻度に差が存在する。それにも関わらず、全てのオブジェクトに対して均一の管理コストがかけられている。そのため、取得コスト、アクセス頻度を考慮した応答速度向上に効果的な一部のオブジェクトに対してさらなる高速化手段を導入する。これにより、ユーザから見た応答速度のばらつきを減少させ、応答速度を向上させることが可能である。

本研究では、応答速度向上手法として高速化対象選択、参照予測、更新予測機構を導入する。そして、これらの機構を組み合わせた提案システムを実現する。このシステムは、常時キャッシングプロキシのアクセスログを監視して独自のデータベースを持ち、ユーザアクセスとは独立して動作する。オブジェクトの挙動から、今後参照されるもの、更新されるものがユーザアクセスに先駆けてキャッシュに取り込まれ、ユーザアクセス時に高速に提供される。また、キャッシュオブジェクトが利用されなくなったことも検出できる

ため、キャッシングプロキシに格納されている利用されなくなったオブジェクトを現在のLRU による方法と比較して、効果的に削除することも可能になる。

この提案システムに対して、キャッシングプロキシのアクセスログを元に動作シミュレーションを行ない、システムの有効性を示す。

1.2 本論文の構成

本論文は、8 章から構成される。各章の内容は以下の通りである。

第 2 章ではキャッシングプロキシの概要、動作について述べる。

第 3 章では WWW アクセスの特徴について述べる。

第 4 章ではキャッシングプロキシの動作、WWW アクセスの特徴を踏まえ、本研究における応答速度向上への着眼点、予想効果について述べる。

第 5 章では前章で提案した応答速度向上アプローチを組み合わせた全体システム構成について述べる。

第 6 章ではシミュレーションによる提案システムの性能評価および考察を行なう。

第 7 章では得られた結果および考慮不足点から今後の課題について述べる。

第 8 章ではむすびとして本研究をまとめる。

第 2 章

キャッシングプロキシ技術

本研究では、キャッシュオブジェクトを利用した WWW の応答速度向上について行なう。そのため、キャッシングプロキシ技術について紹介する必要がある。本章では、キャッシングプロキシ技術の背景と、動作概要について述べる。

2.1 システム概要

Inetrnet の普及に伴い、特に企業などでは不正利用などからサーバの安全性の確保のため、ファイアウォールの構築が一般的になってきた。ネットワーク管理者としては、全てのクライアントから WWW サーバへの HTTP[1][2] アクセスを許可することはセキュリティ上好ましくなく、管理が困難である。とはいえ、WWW が利用できないのは非常に不便である。そのため、HTTP プロキシサーバが利用されるようになってきた。以後、プロキシ (代理) と記述した場合は、HTTP プロキシを指すものとする。

プロキシサーバは、名前の通りクライアントからの HTTP アクセスを中継し、WWW サーバとの通信を代理で行ない、結果をクライアントに返す動作を行なう。

利用方法としては、クライアントをファイアウォールの内部に配置し、プロキシサーバをファイアウォールの外部に配置する。これにより、クライアントと WWW サーバが直接通信することを禁止しながら、WWW の利用が可能となる。また、プロキシサーバに HTTP アクセスを集中させることにより、全てのオブジェクトがプロキシサーバを経由することとなり、ロギングなどが容易になる。これを利用して、プロキシサーバにストレージなどを用意してやり、オブジェクトのキャッシングを行なうことが可能になる。

複数人のアクセスが集まり、参照の局所性が存在するとすれば、キャッシュオブジェクトの再利用が行なわれ、再利用された場合には、重複する外部トラフィックの削減および

応答時間の短縮が期待できる。

2.2 キャッシングプロキシの動作

前節にて、キャッシングプロキシのシステム概要について述べたが、これらの動作が具体的にどのように実現されているのかを図 2.1に示す。

特に、ここでは既存のキャッシングプロキシの例として、Squid[3] の動作について示す。

Squid とは、FTP や gopher、HTTP データオブジェクトに対応した、キャッシングプロキシであり、オブジェクトに年齢の概念を持たせることにより、オブジェクトの一貫性を十分に保ちながら、クライアントに高速にオブジェクトを提供できるようになっている。他にも DNS 検索のキャッシュやノンブロッキング DNS 検索をサポートするほか、キャッシングプロキシを階層的に配置し、ICP[4] [5] を用いることによりサーバ間の協調動作を行ない、負荷分散およびネットワーク資源の有効利用を行なうことが可能である。

これらのキャッシングプロキシの動作的な特徴としては、以下のようなものがある。

- 図 2.1に示されるように、動作はユーザからのリクエストを元に受動的に動作する
- 一度取得したオブジェクトは、再利用可能であれば全てストレージに格納する
- ストレージ格納後にオブジェクトが WWW サーバで更新されることがあるので、一貫性を確保するための機構を持っている
- オブジェクトに年齢の概念を持たせ、新鮮である場合には WWW サーバに問い合わせることなく、直接クライアントに応答することにより高速に動作する

これらを踏まえた上で、内部動作の大まかな流れを図 2.1に示すとともに、各分岐を踏まえた動作概要を示す。

キャッシュミスの場合

1. ユーザからのリクエストを受け、ローカルのストレージ内にオブジェクトが存在するのかどうかを探索する
2. ストレージ内部にオブジェクトが存在しなければ、WWW サーバから取得し、再利用可能なオブジェクトであればローカルのストレージに格納する。
3. ストレージに格納すると同時に、代理で受けとったオブジェクトをクライアントに応答する

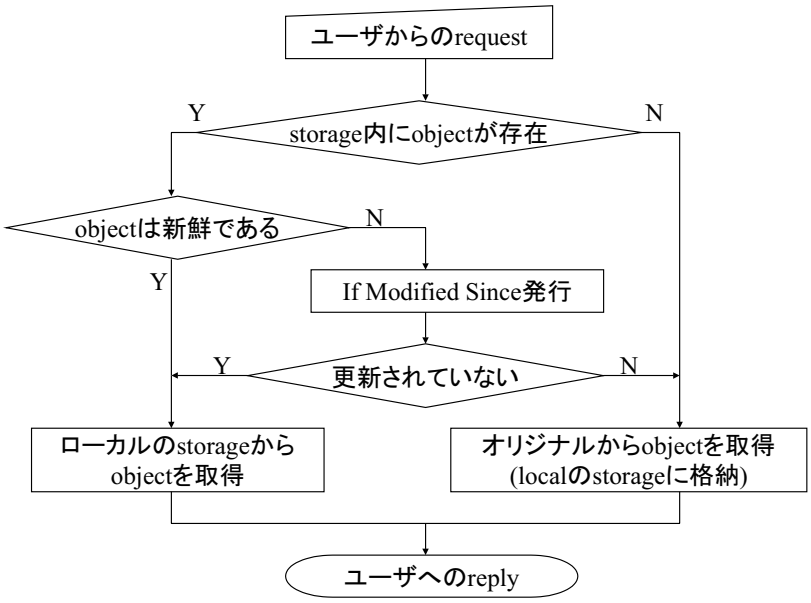

図 2.1: キャッシングプロキシの動作概要

キャッシュヒットの場合

1. ユーザからのリクエストを受け、ローカルのストレージ内にオブジェクトが存在するのかどうかを探索する

2. ストレージ内部にオブジェクトが存在すれば、オブジェクトに設定されている年齢を参照し、新鮮 (キャッシュオブジェクトとオリジナルが同一であると予想される場合)、ローカルのストレージからオブジェクトを取得し、クライアントに応答する

3. 新鮮でないと判断された場合、If Modified Since(以後 IMS と示す) を WWW サーバに発行し、最新であるかどうかの確認を行なう

4. IMS 応答を待ち、キャッシュオブジェクトが最新であった場合、ローカルのストレージからオブジェクトを取得し、クライアントに応答する

5. 最新でなく更新されていた場合、IMS 応答にオブジェクトが含まれて返されるので、ストレージに格納すると同時に、代理で受けとったオブジェクトをクライアントに応答する

第 3 章

WWW アクセスの特徴

本研究では、WWW オブジェクトの挙動から傾向を見つけ出し、応答速度向上に有効な手法を考える。本章では、キャッシングプロキシに記録されているアクセスログから調査を行ない、WWW アクセスの特徴について述べる。

3.1 参照オブジェクトの偏り

WWW には、人気サイトと言われるような非常に多く参照されるものから、ほとんど参照されないものまで、大きな偏りが存在する。

今回、北陸先端科学技術大学院大学のキャッシングプロキシである、proxy.jaist.ac.jp のアクセスログを元に、参照オブジェクトの偏りを調査した。

proxy.jaist.ac.jp のシステム構成を表 3.1に示す。

マシン名	proxy.jaist.ac.jp
ハードウェア	Sun Enterprise 3000
OS	SunOS 5.5.1 (Solaris 2.5.1)
CPU	UltraSPARC II 250MHz × 2
Memory	512 MB
使用キャッシュ容量 (Disk) ログ領域	80 GB (RAID) 50 GB (最大)
使用ソフトウエア	Squid-1.1.20

表 3.1: proxy.jaist.ac.jp のシステム構成

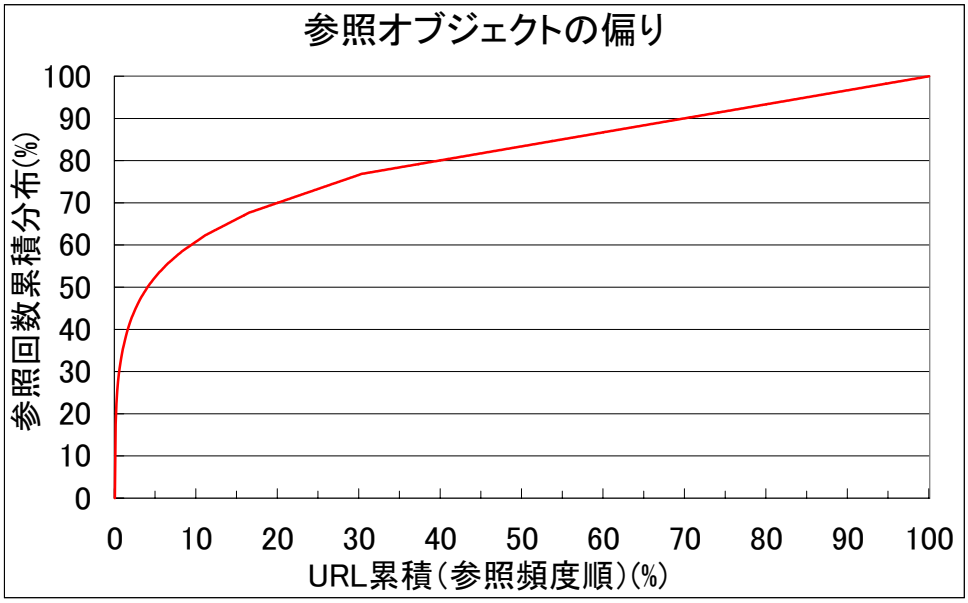

図 3.1: 参照オブジェクトの偏り

このシステムでは、キャッシングプロキシとして Squid を動作させる以外には、特に大きなサービスは行なっていない。また、後述の取得コスト比較システムで、オブジェクト取得時間にシステム負荷に起因する時間のばらつきが存在しないことから、上記のシステム構成は HTTP 中継機能としては十分な性能を持つものと考え、キャッシングプロキシへの負荷集中によるシステム性能低下は十分小さく無視することができる。

このとき、連続 45 日間 (約 200 万件) に参照された URL 別の参照頻度の分布について、図 3.1に示す。

この累積分布より、参照されるオブジェクトには大きな偏りが存在することが確認できた。例えば、アクセス数上位 10%の URL で、全アクセス数の 60%を占め、逆に、下位 25%の URL は、一度しか参照されていないことがわかる。

現在のキャッシングプロキシでは、これらのキャッシュオブジェクトが非常に利用される場合、全く利用されない場合について差を設けていない。つまり、参照の偏りを考慮せず、均一に管理コストがかかっている。

3.2 オブジェクト取得コストの差

クライアントがオブジェクトを取得する際にかかる時間は、主に以下の要素から決定される。

1. 入力された URL から、DNS を用いて名前を解決するための時間
2. WWW サーバの処理時間 (応答生成に必要な時間)
3. オリジナルの WWW サーバから、キャッシングプロキシまでのデータ転送時間
4. クライアントのオブジェクト取得能力

1. の名前解決の時間は、DNS キャッシュにキャッシュされている場合は非常に高速に行なわれ、ほぼ無視できるが、キャッシュされていない場合、WWW サーバの所在によっては 1 秒〜5 秒程度かかってしまう場合があり、応答を悪くしている [6]。

2. の WWW サーバの処理時間 (応答生成に必要な時間) であるが、使われているハードウエア、ソフトウエアに対しておよび負荷 (同時接続要求数) から決定する。

3. の WWW サーバからキャッシングプロキシまでのデータ転送時間は、実際のオブジェクトがネットワークを経由して、キャッシングプロキシが受けとるのに必要な時間であり、実効帯域幅、ファイルサイズなどの要素により決定する。

4. のクライアントのオブジェクト取得能力は、ネットワーク経由で受けとった情報を展開し、必要な形でブラウザ上で表現するために必要な時間である。

キャッシングプロキシの利用により、WWW サーバに問い合わせない場合 2. と 3. の合計時間を、WWW サーバに問い合わせを行なう場合 3. の時間を LAN 速度にすることが可能である。つまり、全体の応答速度としては、1. や 4. の時間に対して、2. や 3. の時間が大きければ、キャッシュ利用による大きな応答速度向上が得られる。そのため、2. と 3. に相当する時間がどれだけを占めているのかを調べるため、図 3.2に示すシステムを構成する。

このシステムにより、常時アクセスログを監視し、キャッシュヒットしたオブジェクトが発生した場合、リクエスト用のクライアントモジュールに送信する。

クライアントモジュールではリクエストに対して、最大 8 個まで並列にオブジェクトを取得する。

オブジェクト取得プロセスにより、別に用意したキャッシュ無しのキャッシングプロキシににリクエストし、直接取得時の取得時間を調べる。この時、参照する DNS はオリジナルのキャッシングプロキシと同一とすることにより、DNS キャッシュにヒットし、クライアントモジュールの取得処理は十分速いことから、2. と 3. の合計時間を計測できる。

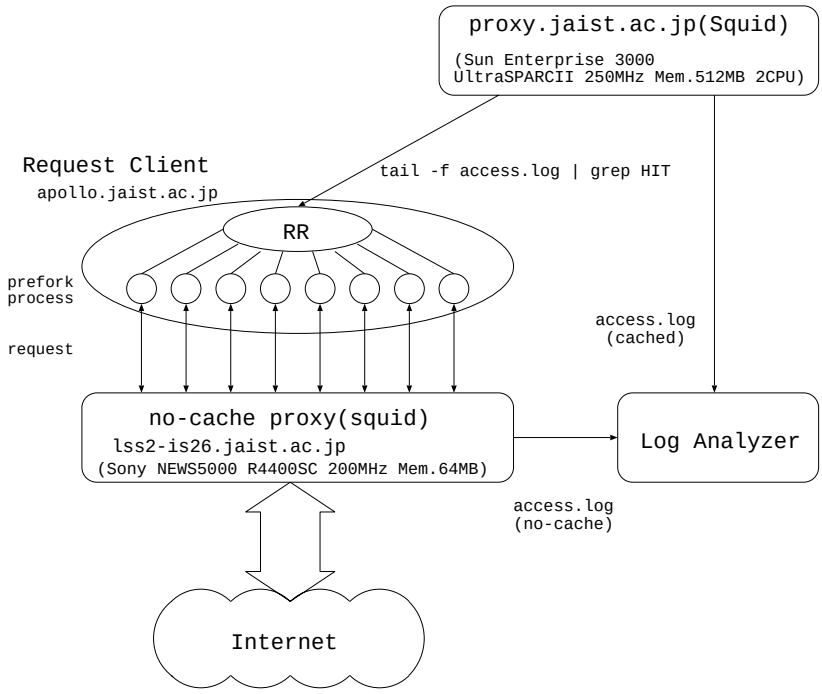

図 3.2: 取得コスト比較システム構成

ネットワーク的に近く WWW サーバに余裕がある場合

ここでは、proxy.jaist.ac.jp からネットワーク的に近く、WWW サーバに余裕があり高速であると思われる例を 2 つ選び出し、図 3.3、図 3.4に示す。

これらを見ると、直接オブジェクトを取得した場合であっても、WWW サーバから、キャッシングプロキシまでのデータ転送にかかる時間は 100ms にファイルサイズを加味した程度であることがわかる。しかし、実際にクライアントが取得に必要な時間は、クライアントのオブジェクト取得能力から大きくばらついているのが読み取れる。また、直接取得時の速度にほとんどばらつきがないことからも、時間帯、曜日などに関わらず、十分な帯域幅と WWW サーバ負荷への余裕があることが伺える。

これらのものをキャッシュすることは、クライアントから見た全体の応答時間に対してデータ転送時間が占める割合が低いことからも、応答速度的にはさほど効果がない。つまり、キャッシュしなくても十分高速に取得することが可能である。

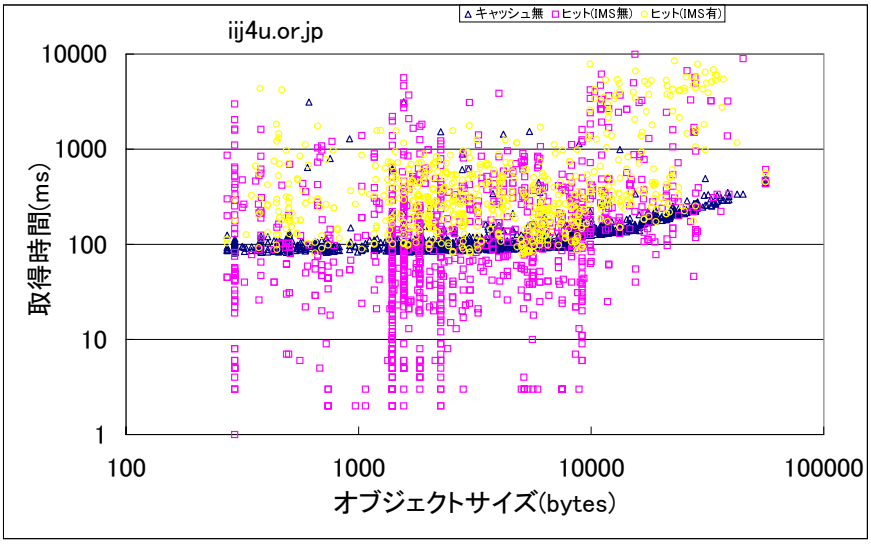

図 3.3: ネットワーク的に近い例 1(iij4u.or.jp)

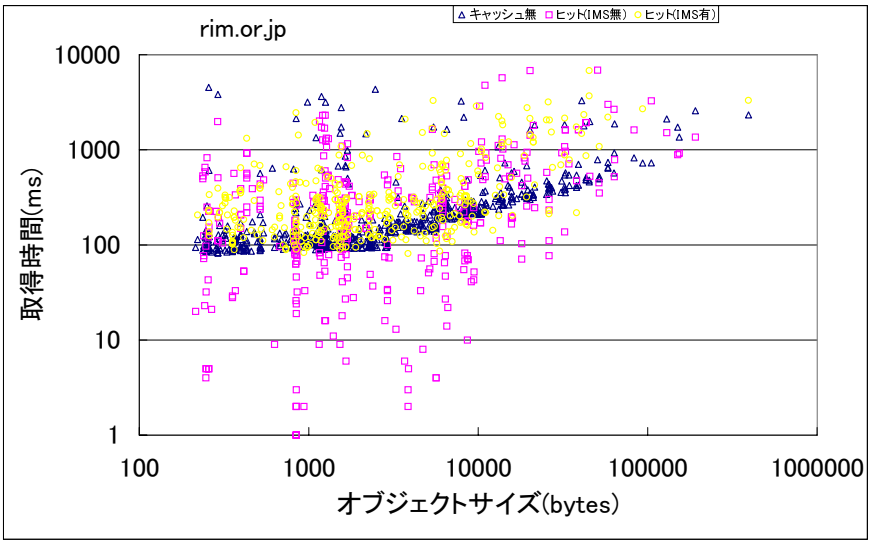

図 3.4: ネットワーク的に近い例 2(rim.or.jp)

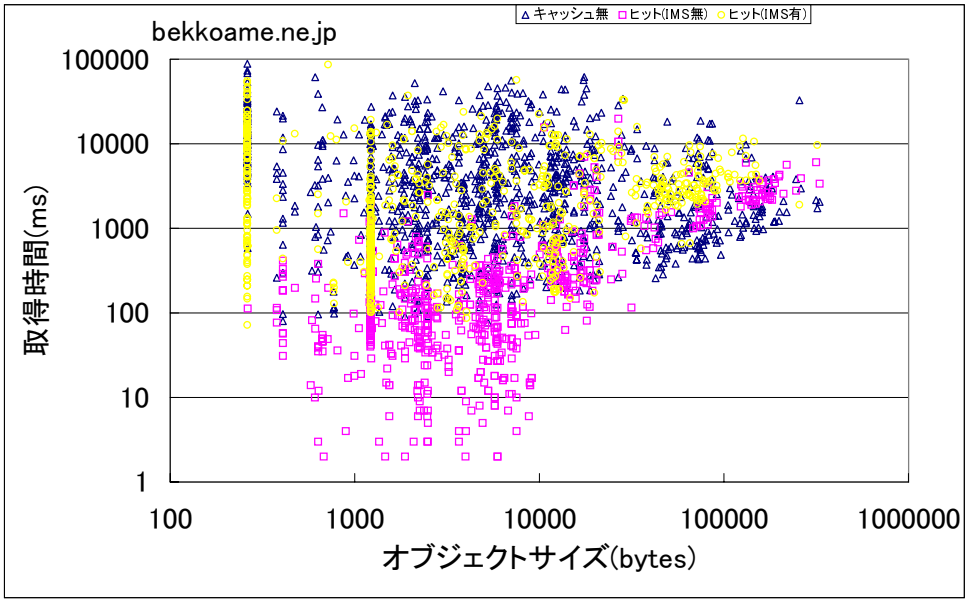

図 3.5: サーバ負荷が高い例 (bekkoame.ne.jp)

ネットワーク的に近いが WWW サーバの負荷が高い場合

同様に、proxy.jaist.ac.jp からネットワーク的には近いが、WWW サーバの負荷が高いため十分な応答速度を実現できていない例を、図 3.5 に示す。

この場合、直接 WWW サーバから同じファイルサイズのものを取得しているにも関わらず、時間帯などの要素から変化する同時利用者の大小により WWW サーバ負荷が異なる。WWW サーバ負荷が大きくなると、キャッシングプロキシからの要求を処理するのに大きな時間を必要とするため、転送時間に大きなばらつきが出ているのが読みとれる。

しかしファイルサイズと転送時間の関係を見ると、あまりファイルサイズの増加による速度の変化が見られないことから、帯域幅不足による応答速度減少はあまり感じられない。

これらの場合、全体の応答速度に対して、WWW サーバの処理時間 (応答生成に必要な時間) の占める割合が非常に大きい。そのため、キャッシュオブジェクトを利用することによって常に最新の情報に保っておき、WWW サーバへの問い合わせを少なくすることができれば、応答速度向上が期待できる。

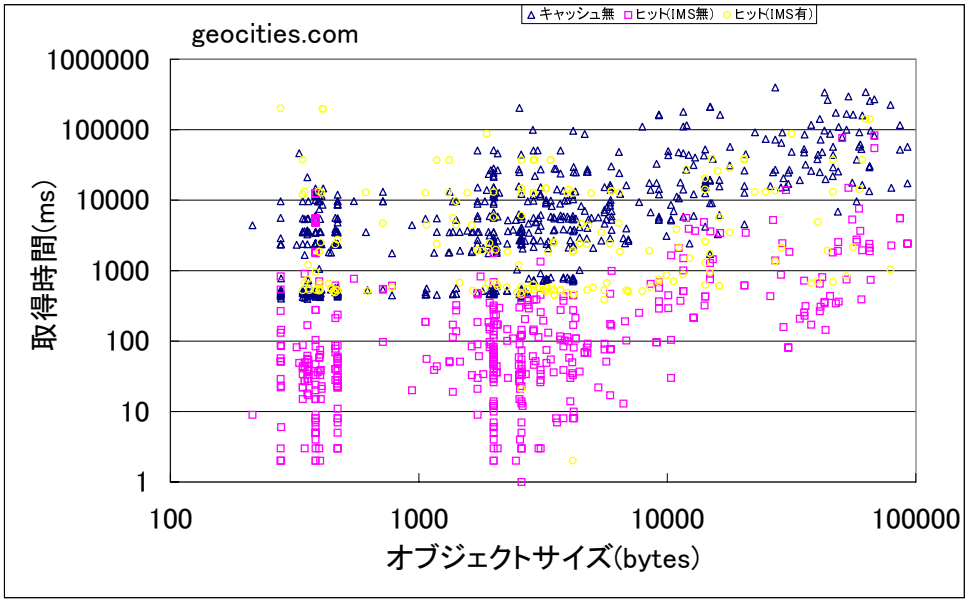

図 3.6: ネットワーク的に遠い例 (geocities.com)

ネットワーク的に遠く、低速な場合

今度は逆に `proxy.jaist.ac.jp` からネットワーク的に遠く、低速であると思われる例を、図 3.6に示す。

この場合、全体の応答速度に対して、WWW サーバからキャッシングプロキシまでのデータ転送にかかる時間が数秒から数十秒と非常に大きい。さらに、帯域幅が狭いためファイルサイズの変化に伴い、転送速度が遅くなっていることがわかる。

これらのものを直接取得するためには、大きな応答時間を必要とし、快適でない場合が多いと思われる。そのため、他の WWW サーバのオブジェクトと比較して優先的にキャッシュしておき、さらに最新情報に保っておくことが可能であれば、応答速度を LAN 速度にすることが可能であり、応答速度を大きく向上させることが期待される。

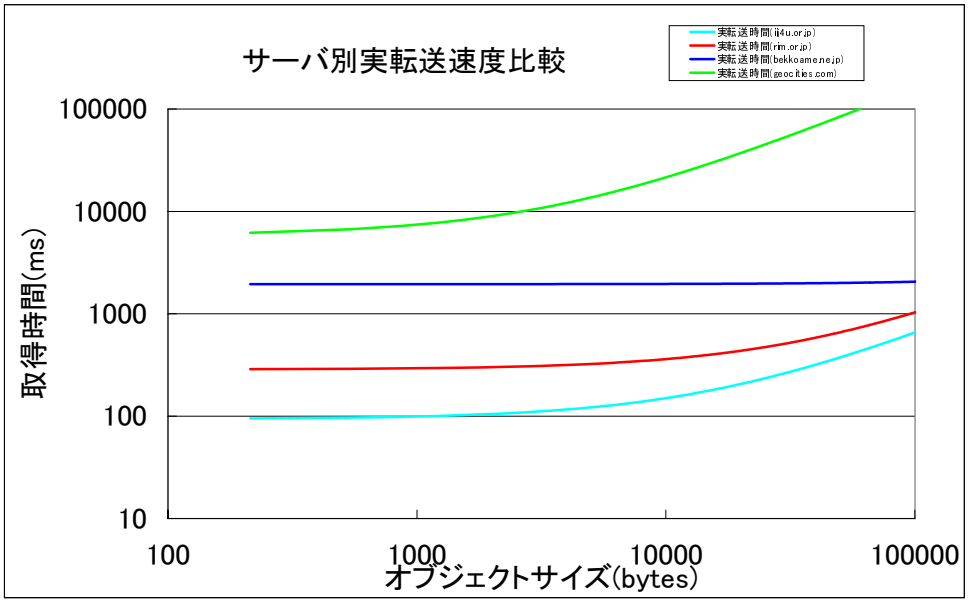

図 3.7: WWW サーバ別転送速度比較

取得コストの差のまとめ

いままでの調査から、キャッシングプロキシがオブジェクトを取得するためのコストはWWW サーバ別に求めた場合、以下の要素で構成されている。

- WWW サーバとキャッシングプロキシまでの実効帯域幅 (ネットワーク距離)
- オブジェクトのファイルサイズ
- WWW サーバ負荷

このとき、各 WWW サーバ間で、どのくらいの取得時間差が存在するかを示すため、今回例としてあげた 4 つのサイトに対して直接取得した場合のオブジェクト取得時間を、図 3.7に示す。

WWW サーバによっては、大きく値のばらつくものがあるが、見やすくするために近似曲線を用いて表している。これらの値は、前述の図 3.2によって得られた直接取得時の

取得時間であり、DNS の検索時間およびクライアント側の取得能力によるばらつきはないものとする。

このときの取得時間は、WWW サーバ別に大きく異なった値を示し、この例だけでも 80 倍以上の取得時間差が存在していることが読み取れる。

オブジェクトが最新状態に保たれているキャッシュ利用時には、キャッシングプロキシからオブジェクトを更新確認なしに取得できるため、WWW サーバに関わらず、ほぼ同じ応答時間を示す。これより、WWW サーバ別取得時間の大きいものに対してキャッシュを利用することによって、大きな応答速度向上が可能になる。

第 4 章

応答速度向上手法

WWW の閲覧を快適にするため、オブジェクトを高速に入手することは重要であるが、実際の帯域幅を越える速度でオブジェクトを取得することは不可能である。本章では、応答速度向上について様々な角度から検討し、本研究での応答速度向上手法を決定する。

4.1 応答速度向上への着眼点

WWW 閲覧時の応答速度を改善するためには、以下のような方法が考えられる。

1. WWW サーバ–キャッシングプロキシ間の帯域幅を増やす
2. 近隣にミラーサーバを用意し、内容を同一にする
3. キャッシュオブジェクトを利用し、見かけ上の応答速度を向上させる

1. の方法は有効であるが、閲覧する全ての WWW サーバまでの経路全体に対して帯域幅を増やさなければならなく、一般的には不可能である。

2. の方法は、比較的コストが少ない割には、大きな効果が得られる。しかしミラーの所在を伝えることは困難であり、例えば DNS のラウンドロビンなどで利用するとしても、近いものが選択される保証がない。

3. の方法は、一番コストが低く大きな効果が得られる可能性がある。基本的に 1. や 2. は、WWW サーバ側の意向がない限り実現できないが、3. は利用者側に近い場所で実現可能である。

これらの理由から、本研究では 3. のキャッシュオブジェクトを利用した応答速度向上方法について検討を行なう。

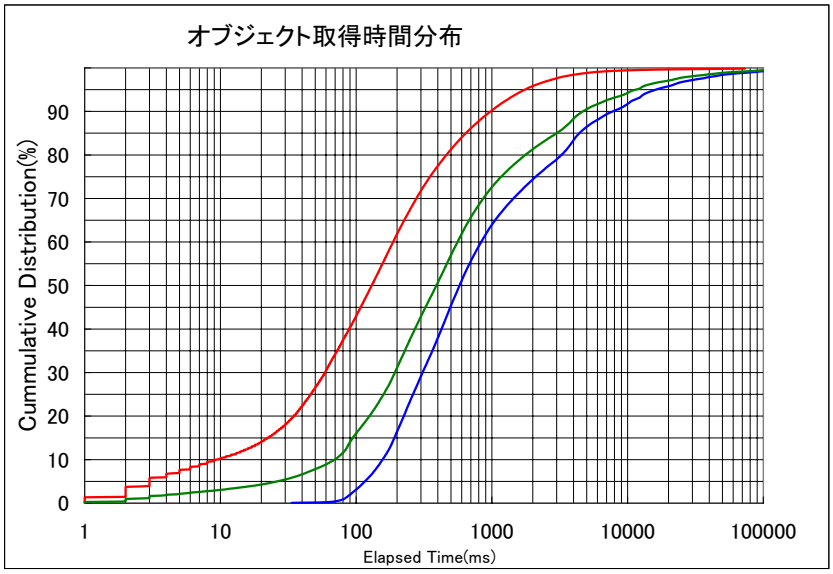

図 4.1: 累積分布によるキャッシュ効果予測

キャッシュオブジェクトの利用

WWW 参照時にキャッシュオブジェクトを利用することにより、見かけ上の応答速度を向上させることが可能である。しかし、キャッシュのヒット率は 30%〜40%程度と決して高い訳ではなく、キャッシュにヒットしないものについては速度を改善することはできない。また、WWW サーバ側で動的に作成されるオブジェクトなど、再利用が不可能なオブジェクトも存在する。そして、WWW サーバのオブジェクトは更新されるため、キャッシングプロキシにキャッシュされているオブジェクトとの一貫性を保つ必要がある。例えば更新されているかどうかの問い合わせ動作を行なう。

実際に、キャッシュオブジェクトを利用した場合に、どれだけ応答速度の向上が可能であるかを、キャッシングプロキシのアクセスログから考察する。

キャッシュ効果予測結果を図 4.1に示す。

このうち、一番低速 (右) のものは、キャッシュミスによる取得時間の累積分布であり、キャッシングプロキシのオーバーヘッドが無視できると考えると、直接取得時の応答時間とみなすことができる。

そして、中間にあるものが全体の累積分布であり、キャッシングプロキシ利用時の応答

時間である。

そして、一番高速 (左) のものは、オブジェクトが最新状態であるキャッシュヒットによる取得時間の累積分布であり、これは参照されたオブジェクトが全てキャッシュに最新状態で入っていた時の応答時間とみなすことができる。

これらより、全てのオブジェクトを持っていて最新状態を保つことができれば、直接取得時と比較しておよそ 80 倍の速度でオブジェクトを取得することができ、全体の 90%のオブジェクトが 1 秒以内で取得可能である。

そこで、キャッシュオブジェクトを利用して応答速度をあげるためには、例えば以下のような方法が考えられる。

- 最新であるなしに関わらず、とりあえずオブジェクトが存在すれば応答し、どうしても最新が欲しければクライアントがリロードする
- WWW サーバで動的に作成されるオブジェクトはキャッシュできないため、特に重要でないものは表示しない、もしくは動的作成でないものを応答する。
- 常に最新であるように、WWW サーバ側でオブジェクトが更新されたら、世界中に存在するキャッシングプロキシに対して更新を通知する
- WWW サーバで参照回数を数えて、人気の高いものを世界中に存在するキャッシングプロキシに通知して、ユーザが使う前にキャッシュに取り込む

これらの方法の共通するところは、キャッシュオブジェクトをそのまま利用するのが、非常に速いことを利用しているからである。しかし、これらの方法はどれも非常に大きな欠点を持っていて、そのままでは利用することができない。そのため、ある制限を設けることによって、問題点を現実的なところに押え込むことにより、応答速度の向上を行なっている。

例えば Squid の場合、オブジェクトに年齢の概念を導入し、新鮮である場合は更新されていないものとしてそのまま応答するといった制限を設けることにより、新鮮時のオブジェクトの一貫性確保動作に相当する応答速度の向上を行なっている。この場合、古いオブジェクトが返されてしまうことがあるが、その可能性は十分小さいことから、現実的なものとなっている。

提案キャッシュシステム

今回の研究では、WWW における応答速度を向上させることが目的であるが、実用上の問題点が生じないように、以下の条件を満たしてくれるものが優秀なシステムであると考える。

1. 応答速度が優れていること
2. 内容は最新に保たれている必要がある
3. トラフィックに対する負荷は、現実的な量に抑えたい

これを実現するため、以下のような制限を加えた。

- キャッシュの内容は常に最新に保たれているため、クライアントからの参照時は最新であるかの問い合わせなしにそのまま応答できる
- キャッシュの内容が常に最新であるように、定期的に更新確認を行なう
- キャッシュ内容は完全にオリジナルと同一にはできないが、ほぼ最新の状態が保たれている
- キャッシュしておいても意味のないものや効果の少ないものはこれらの高速化対象から外し、資源の浪費を抑える

この制限をとりいれることにより、現実的なトラフィック増加ながら、より一層の応答速度向上が期待できる。
そして、この制限を行なうために、今回は以下のような手法を採用した。

- キャッシュ効果の高いものを選択する高速化対象選択導入により、高速化対象を限定する
- 過去の参照回数の変動から、参照の可能性を予測を行なう参照予測を行ない、キャッシュ効果の高いものの中からさらに参照されるものだけを対象にして、対象を現実的な数に抑える。
- 過去の更新間隔、分散から更新の予測を行なう更新予測を行ない、更新の頻度を現実的な数に抑える

以下の節では、それぞれの導入、効果ついて考察する。

4.2 高速化対象選択

クライアントが WWW サーバから直接取得する場合と、キャッシュオブジェクトを利用する場合の取得時間の差が、ほとんど存在しないものから、大きな差があるものまで存在することが明らかになった。そのため、取得時間の差が大きいものを優先的に高速化してやることにより、より一層の応答速度向上が期待できる。しかし、優先的に高速化を行なうためには、取得コストの高いオブジェクトを決定するための指標が必要である。

本研究では、オブジェクト取得コストを WWW サーバ別応答速度の平均とし、高速化対象の決定を行なう。

4.2.1 高速化対象選択の導入

キャッシングプロキシのアクセスログには、オブジェクト単位で転送にかかった時間などを記録している。特に Squid の場合、access.log の Elapsed Time フィールドには、クライアントソケットの accept() から close() までの時間が記録されている。これを元に、各 WWW サーバ別に応答時間の平均を求め、これをオブジェクト取得コストとして高速化対象であるかないかの判断を行なう。

4.2.2 高速化対象選択の効果

WWW サーバ別の応答時間には、DNS による名前解決時間、クライアントのオブジェクト取得能力が含まれているため正確さに欠けるが、名前解決時間やクライアント能力はさほど変動しないことから、実転送時間が長くなるにつれて全体を占める割合は無視できるほど小さくなり、指標として有効である。

4.3 参照予測

高速化対象選択機構により、キャッシュした時の効果の大きさを決定でき、キャッシュ効果のあるオブジェクトが選択可能になった。しかし、いくらキャッシュ効果が大きくても全く参照されないオブジェクトであればキャッシュする意味がない。

そこで、今後の参照オブジェクトの予測が可能であれば、クライアントからの要求よりも前に取得することが可能になり、応答速度の向上が可能である。

特に今回は、過去参照回数から参照を予測する手法について考える。

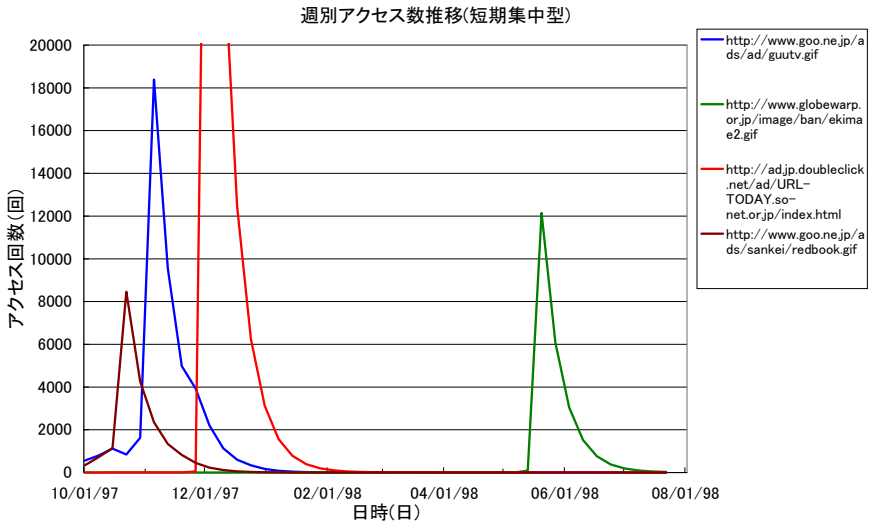

図 4.2: 週別アクセス数推移 (短期集中型)

4.3.1 参照予測の導入

参照予測の実現であるが、参照されるオブジェクトには何らかの特徴があるものと仮定して、URL 別にオブジェクトの挙動を監視した。そのため、過去のアクセスログをもとに、WWW オブジェクトの参照回数ついてどのような特徴が存在するのかを調べてみた。

まず、時間軸を固定した場合における参照間隔の特徴を、図 4.2、図 4.3、図 4.4に示す。

3 本の異なったグラフが示すように、WWW オブジェクトの利用期間には大きな差が存在し、数日程度で急激な参照回数を数えた後全く参照されなくなってしまうものから、ある程度の時間をかけて生成から消滅するもの、定期的に同程度の参照回数を示すものなどがある。

今回は、全ての WWW オブジェクトは利用期間を引き延ばして揃えて、十分に平滑化してやることにより、図 4.5に示すような挙動を示すものと仮定し、この挙動グラフのどこに存在するのかを調べることにより、今後の参照が予測できるものと考えた。

これより、図 4.2は間隔が非常に短いもの、図 4.3は SR が非常に長いもの、図 4.4はその中間に位置するものと考えられる。

参照予測の実現方法は、ある単位時間で参照回数を区切った上で、前回との変化量から

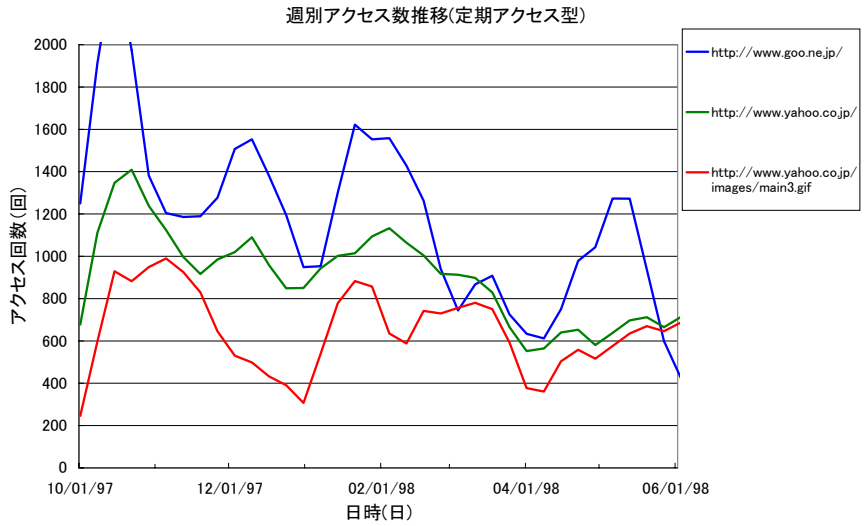

図 4.3: 週別アクセス数推移 (定期アクセス型)

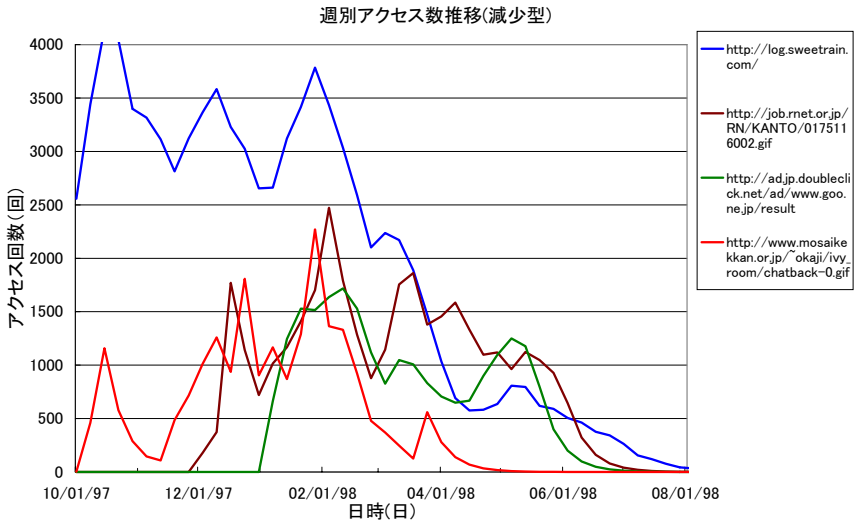

図 4.4: 週別アクセス数推移 (減少型)

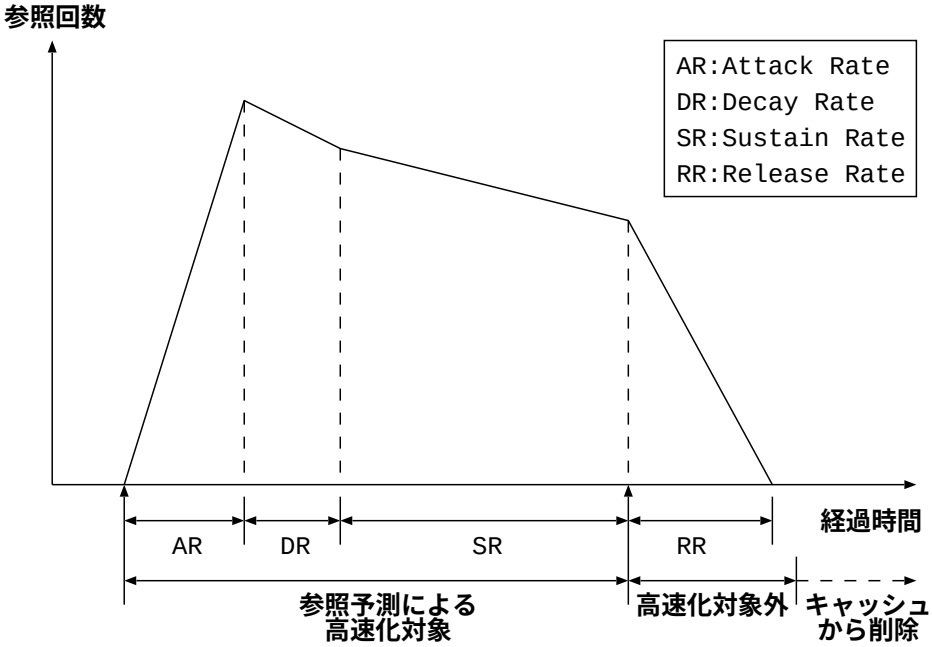

図 4.5: オブジェクトの参照モデル

今後の参照頻度を予測する。

4.3.2 参照予測の効果

参照されるオブジェクトがユーザ要求前に予測できることより、ユーザ要求前に取得動作を行なうことが可能であり、実際にユーザ要求時にはキャッシュ内容を高速に提供することが可能になり、応答速度の向上が可能になる。

4.4 更新予測

参照予測により、今後いつ頃オブジェクトが参照される可能性が高いかを知ることが可能になった。

しかし、参照予測により使われるものであるからといって、全く更新をされていないオブジェクトに対してオリジナルの WWW サーバに問い合わせを行なうことは、トラフィックの浪費を招いてしまう。とはいえ、適切な間隔で更新確認を行なわないとオリジナルとキャッシュオブジェクトの一貫性が保たれない。そのため、適切にオブジェクトの一貫性を保つための機構が必要である。

```
if (CLIENT_MAX_AGE)
    if (AGE > CLIENT_MAX_AGE)
        return STALE
if (AGE <= MIN_AGE)
    return FRESH
if (EXPIRES) {
    if (EXPIRES <= NOW)
        return STALE
    else
        return FRESH
}
if (AGE > MAX_AGE)
    return STALE
if (LM_FACTOR < PERCENT)
    return FRESH
return STALE
```

表 4.1: Squid のキャッシュ更新アルゴリズム

Squid では、年齢の概念を導入し、新鮮であればオブジェクトが更新されていないものとして、WWW サーバに問い合わせることなくクライアントにオブジェクトを提供することによって、応答速度の向上を図っている。

新鮮であれば更新しないというのは、更新されないことを予測していることになるので、一種の更新予測機構であると言える。

今回は、過去の更新間隔から更新を予測する方法について考える。

4.4.1 更新予測の導入

更新予測の実現方法であるが、過去の更新時間を記録しておくことにより、平均更新時間、分散を算出し、今後の更新を予測する。

Squid でも年齢の導入による更新予測が行なわれているため、Squid と同程度の一貫性を保ちながら、応答速度向上の実現を目標とする。

まず、Squid での、更新予測手法を表 4.1に示す。

AGE	検索されてからの経過時間 $AGE = NOW - OBJECT\_DATE$ (OBJECT_DATE: オブジェクトがキャッシュに格納された時間)
LM_AGE	オブジェクトがどれだけ古いのかを示す $LM\_DATE = OBJECT\_DATE - LAST\_MODIFIED\_TIME$ (LAST_MODIFIED_TIME: オブジェクトが最後に書き換えられた時間)
LM_FACTOR	AGE と LM_DATE の比を示す $LM\_FACTOR = AGE/LM\_AGE$
CLIENT_MAX_AGE	HTTP/1.1 の Cache-Control 要求ヘッダから得られるような クライアントが受け取る最大の古さ
EXPIRES	サーバの応答ヘッダからの有効時間

表 4.2: キャッシュ更新用定数、変数

そして、ここで定義されている各定数、変数について表 4.2に示す。

そして、MIN_AGE,MAX_AGE,PERCENT は Squid.conf の設定値が利用され、デフォルトでは、0,4320,20 が使用される。

CLIENT_MAX_AGE および EXPIRES は、オブジェクト単位で明示的にキャッシュ動作に操作を加えるものであり、オブジェクトの挙動から更新を予測する今回のシステムとは関連が薄いため、考慮しないものとする。

LM_FACTOR を参照することにより、最近更新されたものに対しては頻繁に更新される可能性が高いため、速く新鮮でなくなり、逆にほとんど更新されていないものは、今後の更新確率も低いため、長い間新鮮と判断される。

また、MAG_AGE のデフォルト値を利用した場合、あるオブジェクトが新鮮状態に保たれるのは最長で 3 日であり、その後は IMS チェックなどにより再び新鮮状態に戻る動作を繰り返すことがわかる。

Squid による更新予測性能

今回、proxy.jaist.ac.jp 上でのアクセスログを元に、再利用可能オブジェクトの中で、WWW サーバが HEAD 要求に対して Last-Modified: フィールドを正しく応答するオブジェクトの中で、アクセス数上位 200 位の URL に対して調査を行ない、更新予測性能の評価を行なった。

このときの状態別件数を、表 4.3に示す。

状態	件数	状態	件数
TCP_HIT	35,869	TCP_CLIENT_REFRESH	20,547
TCP_IMS_HIT	12,987	TCP_MISS	2,666
TCP_REFRESH_HIT	6,056	TCP_REFRESH_MISS	1,255
その他	11		
合計			79,390

表 4.3: 状態別参照回数

表 4.3の結果から読み取れることは、全体の 61.5%が TCP_HIT もしくは TCP_IMS_HIT と、新鮮な状況に保たれていることがわかる。そのため WWW サーバに問い合わせることなくクライアントに応答しており、高速にオブジェクトの提供が可能であることがわかる。

キャッシュオブジェクトとオリジナルとの一貫性

今回、Last-Modified: 及び If-Modified-Since: フィールドを記録したことにより、実際にオブジェクトが更新された時間を知ることが可能になった。

そのため、年齢が新鮮なものとして提供されたオブジェクトが、本当に最新のオブジェクトであったのかを調べることが可能である。

これより、直接取得した場合とキャッシュオブジェクトを利用した場合の内容の整合性がどれくらい保たれているのかを調査した。[1]

調査結果を表 4.4に示す。

	全参照数	内容が同一	内容が古いもの	更新予測成功
参照数	66,408	65,969	439	35,438
割合 (%)	100.0	99.3	0.7	98.8

表 4.4: Squid におけるオブジェクトの整合性

[1]例えば goo.ne.jp の様に正しく値を返しているが、特殊なサーバのクラスタ化のためであろうか同じオブジェクトだと思われても Last-Modified: フィールドの値が微妙にずれていて、本来はヒットになりそうなものがヒットしないサイトは手動で除去している。

Squid 利用時のオリジナルの内容との整合性は、およそ 99.3%である。実際には、クライアントからのリロード要求などによるリフレッシュ動作もあるためか、かなりの整合性が保たれている。しかし、数十分〜1 時間程度で更新されるものについてはとりこぼしが発生し、ミスが多いことがわかった。

今回提案する更新予測機構では内容の一致よりも速度重視のため、従来のキャッシングプロキシと同程度の更新予測精度を保ちつつ、より高速にオブジェクトを提供できるパラメータを設定する。

オブジェクトの更新間隔

オブジェクトの更新を予測して、適切な間隔でリフレッシュ動作を行ないオリジナルと一致させ、常時キャッシュから情報を提供できれば応答速度を向上させることができることは既に述べた。

ここでは、適切な更新間隔を見つけるため、実際のオブジェクトに対して更新間隔のチェックを行う。

Squid による更新予測性能と同じく、再利用可能オブジェクトの中で、WWW サーバが HEAD 要求に対して Last-Modified: フィールドを正しく応答するもののうち、アクセス数上位 200 位の URL に対して調査を行ない、更新間隔の調査を行なった。

ファイル種別における更新間隔の例を図 4.6、図 4.7に示す。

これらの例にみられるように、更新間隔はファイル種別で大きな特徴があり、さらに同じ URL であればあるばらつきをもった周期的特徴が見受けられた。

そのため、過去の更新間隔から平均更新間隔を求め、分散を求めて更新の予測する。

今回、過去 3 回以上の更新履歴を元に更新間隔の平均値を求め、また平均値からの分散を求めることにより更新間隔のばらつき範囲を求める。

そして、算出された範囲内を 16 分割して、それぞれのポイントで更新の確認を行なうものとし、更新していればキャッシュの内容を最新のものとする。

そして、クライアントからの参照時には、明示的にキャッシュしないものを除き、WWW サーバに問い合わせることなくキャッシュから応答させるものとする。

4.4.2 更新予測の効果

更新の予測により、更新間隔が広いものに対しては粗く更新間隔が決定され、更新間隔が狭いものには、細かく更新間隔が決定する。

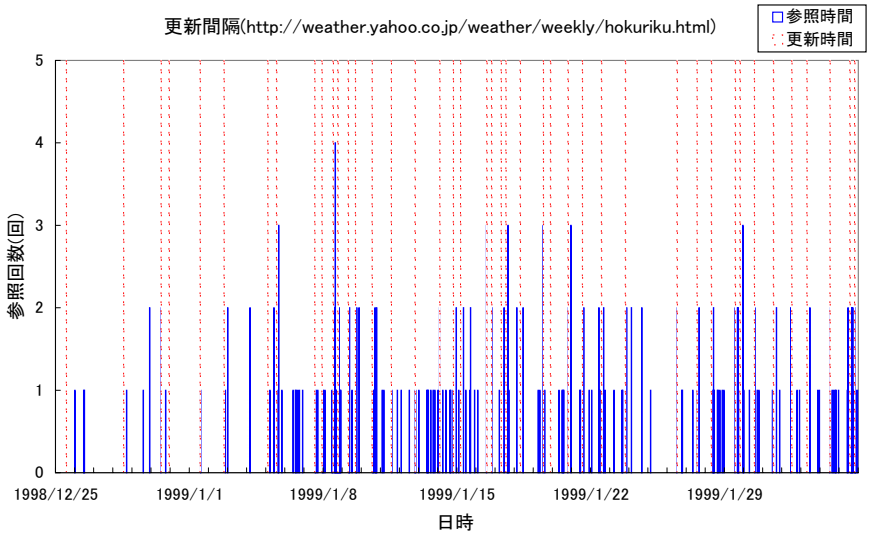

図 4.6: 更新間隔 (http://weather.yahoo.co.jp/weather/weekly/hokuriku.html)

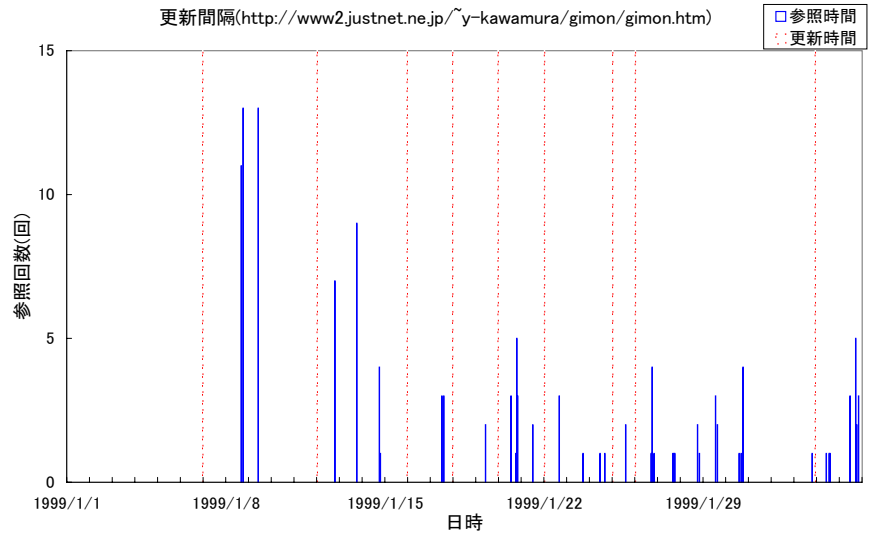

図 4.7: 更新間隔 (http://www2.justnet.ne.jp/ y-kawamura/gimon/gimon.htm)

また更新間隔の分散具合から、定期的な更新間隔のものは狭い範囲で定期的に、ばらつき具合が多いものについては比較的広い範囲で更新の確認が行なわれ、最新のオブジェクト状態に保たれやすい。

これより、更新予測はうまく動作していると言える。

第 5 章

提案システム構成

今までの調査より、ある一部のオブジェクトのみがよく参照され、キャッシュ利用による応答速度向上は WWW サーバ、ファイルサイズなどの要素により構成されるオブジェクト取得コストにより、変化することが明らかになった。

既存のキャッシングプロキシでは、これらのオブジェクト取得コストに関係なく、均一にキャッシュオブジェクト管理コストをかけている。そのため、オブジェクト取得コストの高いオブジェクトに対して大きな管理コストをかける、すなわちさらなる高速化手法をとりいれることが重要である。

前章では、キャッシュ効果の高いオブジェクトに大きな管理コストをかける方法として以下のアプローチを提案し、単体での効果を示した。

- WWW サーバ別の平均応答時間をオブジェクト取得コストと定義した高速化対象選択
- URL 別の参照回数の変動からの参照予測
- URL 別の更新時間間隔、分散からの更新予測

ここでは、従来のキャッシングプロキシに、これら 3 つの機構を組み合わせることによりさらなる応答速度向上システムを提案する。

5.1 全体構成

今回提案するシステムでは、キャッシングプロキシとしての基本動作部分は従来のキャッシングプロキシを利用し、従来どおりの動作を行なう。そして、高速化対象選択機構では

キャッシングプロキシのアクセスログから効果オブジェクトをリストアップし、参照予測機構に渡される。

参照予測機構では、参照が予測されるものについては更新予測機構に渡され、逆に参照が予測されなく (今後使われる可能性が低い) ものは、キャッシングプロキシに対してキャッシュの削除を行なう。

更新予測機構では、参照予測機構の出力結果を元に更新間隔の予測を行ない、更新が予測される場合、プリフェッチャに対して要求を行なう。

ここで言うプリフェッチャとは、HTML 走査による再帰型取得機能をもつもの (例えば Wget や Wcol[9] の HTML 走査による先読み対象の推測部分) を指す。これは、参照の局所性から、ある HTML ファイルからリンクされているものは参照される可能性が高いという近未来の参照予測であると言える。今回、私の提案している参照予測は、比較的長期の統計情報による参照の予測であり、ここでの参照予測とは異なる。プリフェッチャでの先読み個数とヒット率の相関についてなどは先行研究 [10] にて評価が行なわれており、先読み個数の増加に伴い性能の向上が示されている。今回は、プリフェッチャを利用することにより、さらなる応答速度向上が起こるものとして導入し、先読み個数などのパラメータ設定やプリフェッチャの評価には言及しない。

そして、プリフェッチャからの要求は、ユーザアクセスとは独立して発生し、キャッシングプロキシに対して要求を行なう。

提案システム構成を図 5.1に示し、各機構についての機能的特徴を述べる。

5.2 高速化対象選択機構

高速化対象選択機構では、常時キャッシングプロキシのアクセスログを監視し、クライアントからキャッシングプロキシに対して参照が行なわれた場合、結果がアクセスログにに記録され、それに伴うログの新規増分が高速化対象選択機構に送られてくる。送られてきたアクセスログ情報に対して、あらかじめ作成されている WWW サーバ別取得コストテーブルと照らし会わせ、一定のコストを越えるもののみを管理対象とする。

あらかじめ作成してある WWW サーバ別取得コストテーブルであるが、送られてくるアクセスログ情報を元に動的に作成される。

今回、取得コストを各 WWW サーバ別の平均応答時間とし、以下の手順で作成を行なう。

- WWW サーバ別にオブジェクト参照回数、累積データ転送時間を記録し、平均応答時間を算出する。

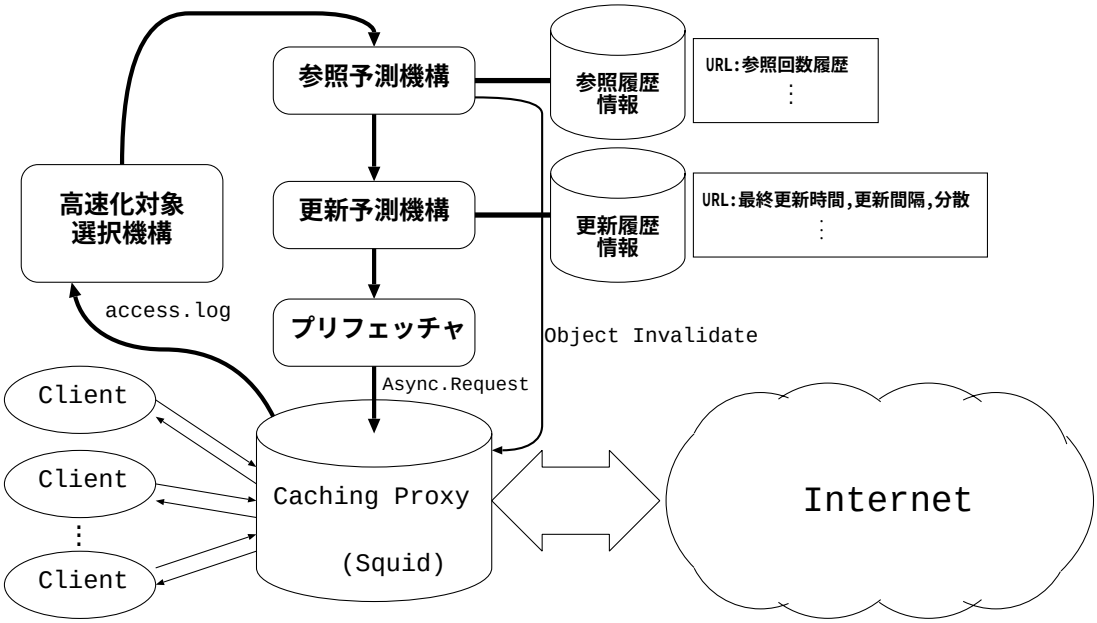

図 5.1: 提案システム構成

- これを定期的に WWW サーバ名:平均応答時間という形にして、WWW サーバ別平均応答時間テーブルに記憶する。

5.3 参照予測機構

参照予測機構では、URL 別に過去の単位時間別参照回数を記録しておき、指数加重移動平均で今後の参照回数を求め、その変化量から、今後の参照確率を求める。

参照確率が一定の数値以上であれば、更新予測機構に対して要求を行なう。

また、参照確率が一定の数値以上にも関わらず、参照がなかったものが連続して 3 回続いたものに対しては、参照が予測されなくなったものと判断して、キャッシングプロキシに対してオブジェクトの削除を行なう。

5.4 更新予測機構

更新予測機構では、URL 別に過去の更新時間を記録しておき、平均更新間隔と分散を求め、算出された範囲内を 16 分割しておく。参照予測機構からの要求が出された時間がそれぞれの分割時間に達していれば、プリフェッチャに対して要求をを行なう。

第 6 章

シミュレーション性能評価

提案システム構成では、限定したオブジェクトを高速に提供することによって、システム全体から見ると応答速度のばらつきを緩和し、性能を向上できるとした。

ここでは、提案システム構成において、proxy.jaist.ac.jp のアクセスログを適用し、シミュレーションを行ない、従来のキャッシングプロキシ (Squid) と比較してどの程度の性能向上がみられたかの評価を行なう。

6.1 シミュレーションモデル定義

参照予測、更新予測動作によるシステム負荷、予測結果からの事前取得によるネットワーク負荷が発生し、キャッシングプロキシ単体と比較して常時の性能低下も考えられるが、ネットワーク帯域幅、記憶装置、CPU 資源には余裕があるものとして今回のシミュレーションでは考慮していない。

また、オブジェクト格納用のストレージには十分な余裕があるものとし、観測期間内にキャッシュオブジェクトの削除は行なわれなかったものとする。

本来、高速化対象選択機構で用いる WWW サーバ別取得コストは、アクセスログを元に時間経過にともない変化していくものであるが、あらかじめ連続 40 日分のアクセスログを元に作成しておき、シミュレーション期間中はこれらの値を変更しないものとする。

また、提案システムで組み込んでいるプリフェッチャに関しては、応答速度の向上に効果があると思われるが、参照、更新予測結果の効果であるのかプリフェッチャ効果なのかの境界が曖昧になるため、シミュレーションでは以下のような構成で行なう。

これより、本研究で行なった手法に対する有効性を評価し、さらにプリフェッチャを加えることによりさらなる性能向上が予想できる。

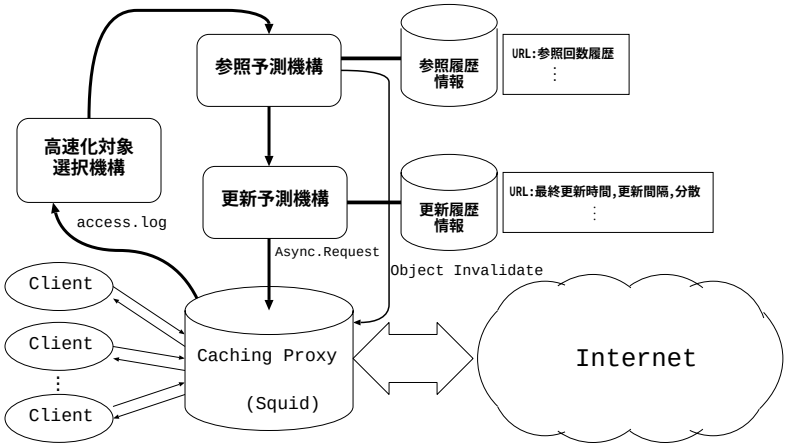

図 6.1: シミュレーションシステム定義

6.2 シミュレーション方法

キャッシングプロキシのアクセスログに記録される取得時間は、WWW サーバとのネットワーク距離、実効帯域幅、クライアント負荷といったようなさまざまな要素により決定する。そのため、例え全ての参照履歴を持っていたとしても、同じ条件、応答速度を再現するのは非常に困難である。

そのため、アクセスログを元に、提案機構導入による高速化部分の数値を置き換えたログを作成し、両者を比較することによって評価を行なう。

今回のシステムでは、限定したオブジェクトに対して、参照が予測される時期にリフレッシュ動作を行ない、クライアントからの要求前に最新にすることを行なっている。ここで言う最新とは、キャッシュの内容がオリジナルの WWW サーバと同一であり、WWW サーバに問い合わせることなく応答できる状態を指す。

そのため、WWW サーバ別取得コストにより限定されたオブジェクトに対しては全てキャッシングプロキシから直接応答できると仮定して、アクセスログに記録されている Elapsed Time を直接応答した場合の時間に置き換える。

しかし、同じ URL であっても、直接応答時の Elapsed Time にはばらつきが起こることがわかっているので、常に同じ値に置き換えるのではなく、ランダムに置き換える (直前に使用された値に置き換える)。

これより、同じ条件下で同じ URL へアクセスした場合の応答速度の違いがシミュレーション可能である。

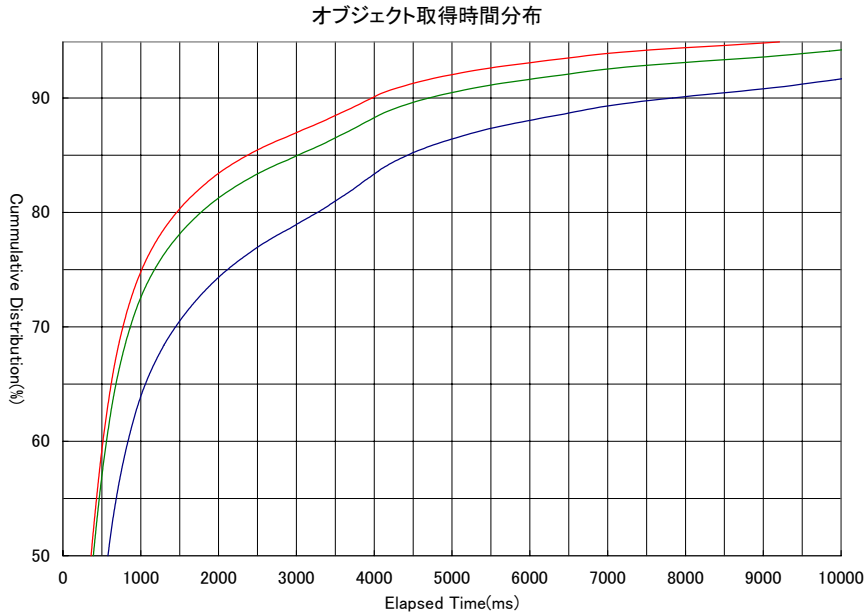

図 6.2: 提案システムの取得時間分布

6.3 シミュレーション結果

Squid のアクセスログ、シミュレーションによって得られたアクセスログから、システム全体の応答速度の累積分布を図 6.2に示す。

このグラフでプロットされたものの一番右の累積分布は、キャッシュヒットしなかったものの累積分布であり、キャッシングプロキシのオーバーヘッドがないものと仮定すれば、キャッシュ無し状態とみなすことができる。

中間の累積分布は、キャッシングプロキシのアクセスログ全体の累積分布であり、キャッシュ利用時の有効性を示している。

シミュレーション結果によって得られたグラフは、一番左に存在するものであり、これら 2 つのグラフよりも高速な部分の累積分布を増加させており、速いものから遅いものまでの取得時間差が減少していることが読みとれる。

6.4　シミュレーションによる考察

提案システムを利用することにより、キャッシングプロキシを経由して取得したオブジェクトのうち 80%を 1.5 秒以内で、90%のオブジェクトを 4 秒以内で取得することが可能になった。

ネットワーク的に近く WWW サーバに余裕がある場合においても、クライアントの取得能力などの要素から、遅い時には 1 秒〜数秒の取得時間を必要としていることがあることからも (図 3.3参照)、1.5 秒〜4 秒でオブジェクトを取得できれば、不快に感じることはほとんどないだろうと考えられる。

また、WWW 画面は単一の HTML ファイルで構成されていることは少なく、例えば画像が張り込まれていたりする場合がほとんどである。しかも、カウンタやバナーなど、他の WWW サーバに属するオブジェクトが埋め込まれている場合もよく見かけられる。これらの場合、全てのオブジェクトの取得が完了しないことには、完全な画面構成ができなく、もっとも転送速度の遅いものが完了するまで待たされることになってしまう。

今回の手法では、オブジェクト取得が遅いものに対して重点的にキャッシュオブジェクトを利用した高速化手法を採り入れることによって、応答速度のばらつきを減少させている。そのため、WWW ページ全体を構成するという意味では、このグラフに現れる数字以上に効果を生み出している。

第 7 章

今後の課題

7.1 実運用に向けたシステム構成

今回の性能シミュレーションでは、キャッシングプロキシの負荷による応答速度の低下については無視できる程小さいものとして考慮しなかった。しかし、ISP や企業のなどの主要プロキシとして非常に多くの HTTP 要求を処理しなければいけない場合を考えると、負荷による応答速度低下を無視することはできない。今回提案するシステムでは、予測に伴う更新確認、取得動作が行なわれるため、さらなるトラフィック増加が予想され、なお一層の性能低下が見込まれる。

キャッシングプロキシの負荷に起因する問題に対して、主に以下のようなアプローチが行なわれている。

- Inktomi Traffic Server[7] などで使われているクラスタリング技術
- Squid,Hervest[8] などで使われている分散キャッシュ技術

クラスタリングによる高速化は、特に WWW に特化した手法ではなく、それなりの効果が望めるが、非常にハードウエアが高く、対費用効果という意味では、導入できる組織は限られてくる。

分散キャッシュ手法の中でも、特に ICP を用いた分散キャッシュシステムに対しては、ICP 問い合わせメッセージによるパケット数増加から生じる問題について指摘されており [11]、万能ではない。

今回、各 WWW サーバ別の平均応答時間から取得コストという指標を作り、これの小さいものに対してはキャッシュしなくてもそれなりの応答速度がでることから、以下のようなシステムを構成する。

- 親キャッシングプロキシ (1 台のみ)、子キャッシングプロキシ (複数台用意し、各部署に配置する) という構成をとる
- 子キャッシングプロキシは、親を経由することなく直接オブジェクトの取得を可能に設定する
- 取得コストが一定以上のものだけ、parent 接続されている親サーバに要求が渡るようにする
- 親キャッシングプロキシは、子キャッシングプロキシからの接続のみを受け付ける
- 親キャッシングプロキシには、今回の提案システムのように参照予測、更新予測動作を行なわせ、高速化を行なう。

このような構成を採用することにより、親キャッシングプロキシには取得コストの高い、キャッシュ効果の高いオブジェクトのみが集まる。そして、クライアントからの要求のうちコストの低いものに対しては、子キャッシングプロキシで処理され、負荷が分散される。子キャッシングプロキシ同士は協調動作を行なっていないため、隣接キャッシュの内容が利用されることはないが、直接取得してもコストは低いものであるので、総合的には ICP の導入によるオーバーヘッドよりも良好な応答速度を示すのではないかと考える。

ある意味では、squid.conf に対して自分の経験から各ドメイン別に parent として利用する/利用しないを記述しているものと同じであるが、これを自動化してくれるだけでも管理者としてはありがたいものと考えられる。

7.2 取得コストの定義

今回、各 WWW サーバからオブジェクトを取得するのに必要な取得コストを、各 WWW サーバ別の平均応答時間から算出している。これはおおよその指標には十分だと考えるが、以下のような状況を考えると不備な点がいくつか存在する。

- 応答速度のうち、帯域幅による影響と WWW サーバ負荷での影響部分について切り分けができていない
- 時間変動による実効ネットワーク帯域幅の変動に追従できない
- クライアントの取得能力による時間のばらつきや DNS の Lookup 時間の除去をしていない

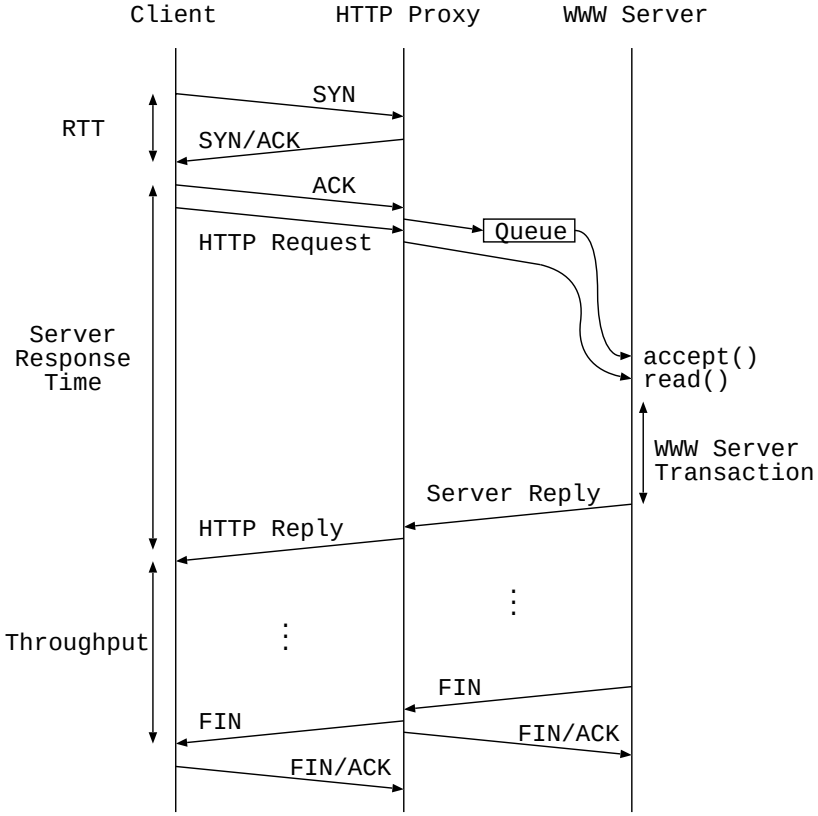

図 7.1: HTTP 処理時間の内訳

これらの問題点については、Squid のアクセスログのみを利用する限り解決できないと思われる。

そのため、Squid のアクセスログからではなく、ネットワーク特性を調査するための専用のトラフィックアナライザを作成するべきだと考える。例えば、内訳を図 7.1のように想定して、RTT, サーバ応答時間, スループットといったような値について解析を行なうツールを作成する。

トラフィックアナライザには、以下の機能を盛り込むことにより影響部分の切り分けおよび時間変動への対応、そして HTTP 応答の傾きからスループットの予測が可能となり、上記の問題を解決できる。

- クライアントソケットの accept() から close() までの時間ではなく、内訳に対しても調査を行なう
- 特性調査のため、各サーバに定期的に問い合わせを行ない、これらの情報を更新する

7.3 ネットワーク負荷への考慮

もし、今回の提案システムが普及して、多くの組織で利用されるようになると、参照、更新予測によるトラフィックが至るところで発生し、Internet 全体的な速度が落ちてしまうのではないかという懸念がある。予測対象を取得コストが高く、参照回数が多いものと絞り込んでいるとはいえ、ただでさえ帯域幅の狭いところにトラフィックを発生させることになるため、逆にその時間帯に本来の目的でネットワークを利用する人にとっては迷惑である。

そのため、予測による取得動作に関しては、優先度を下げるといった対策が必要になる。

組織のトラフィック量は時間帯、曜日 (休日、平日) などにより変動し、国内であればビジネス時間帯やテレホーダイ利用可能時間帯に増加し、早朝、夜間には比較的少ないといったような特徴がみられることがよく知られている。

そのため、例えば MRTG より比較的帯域に余裕がある時間帯に優先して取得したり、取得時の利用帯域幅に制限を加える、混雑時には動作を抑制するなどのネットワーク負荷に対して考慮する必要が考えられる。

第 8 章

まとめ

本研究では、WWW アクセスの特徴を参照オブジェクトの偏り、オブジェクト取得にかかるコストなどといった要素から調査し、ある一部のオブジェクトのみがよく参照され、ある一部のオブジェクト取得に対して大きなコストが発生することを明らかにした。

そして、応答速度向上手法の一つとして既に使われているキャッシングプロキシ技術に対して、特に応答速度向上を主眼においた場合の改善要素を見つけるため、オブジェクト参照間隔の調査、オブジェクト更新間隔の調査、キャッシュオブジェクトとオリジナルの一致確認を行なった。参照間隔、更新間隔の調査の結果、オブジェクト毎にある間隔および変動幅をもった規則的なモデルに一致可能であることが明らかになり、参照および更新間隔の予測を適用することにより、応答速度向上可能性を示した。

そして WWW アクセスの特徴、キャッシングプロキシ動作の特徴から、高速化対象選択、参照予測、更新予測の 3 つを組み合わせたシステムを提案し、シミュレーションによる提案システムに対する評価を行なった。シミュレーションの結果、オブジェクト取得に対してのコスト差を減少させ、全体としての応答速度向上が実現された。

本システムでは、応答速度の向上に主眼を置き、トラフィック量や記憶装置、CPU などの資源利用に関してはほとんど考慮していない。特に、今までのクライアント要求を元にした受動的キャッシュではなく独立して動作する能動的なシステムであるため、既存システムと比較して無駄なトラフィックおよび資源利用は避けられない。これらの改善案として、実運用に向けたシステム構成、専用トラフィックアナライザ導入による精度向上、ネットワーク負荷への考慮といった課題を提示した。

謝辞

本研究を進めるにあたり、終始適切な御指導を賜わりました篠田陽一助教授に心から深く感謝致します。

また、本研究に関して多大な助言を賜わりました田中友英氏に心から感謝致します。

最後に、日頃から貴重な御意見、御助言を賜わりました篠田研究室の学生諸氏に感謝致します。

参考文献

[1] T. Berners-Lee, R. Fielding, H.Frystyk, Hypertext Transfer Protocol – HTTP/1.0, Network Working Group RFC 1945, May 1996. http://ds.internic.net/rfc/rfc1945.txt

[2] R. Fielding, J. Gettys, J. Mogul, H.Frystyk, T.Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, Network Working Group RFC 2068, January 1997. http://ds.internic.net/rfc/rfc2068.txt

[3] D. Wessls, Squid Internet Object Cache, http://squid.nlanr.net/

[4] D.Wessels, K. Claffy, Internet Cache Protocol (ICP), version 2, Network Working Group RFC 2186, September 1997. http://ds.internic.net/rfc/rfc2186.txt

[5] D.Wessels, K. Claffy, Application of Internet Cache Protocol (ICP), Network Working Group RFC 2187, September 1997. http://ds.internic.net/rfc/rfc2187.txt

[6] Michael Nidd, Thomas Kunz, Eryugrul Arik, Prefetching DNS Lookup for Efficient Wiress WWW Browsing, Proceedings of Wireless 97,the Ninth Annual International Conferencd on Wireless Communication Calgary, Alberta. Canada pp.409-414, July 1997.

[7] Inktomi Corporation, Inktomi Traffic Server, http://www.inktomi.com/products/traffic/

[8] A. Chankhunthod, P.B. Dangiz, C. Neerdaels, M.F. Schwartz, and K.J. Worrell, A Hierarchical Internet Object Cache, Technical Report 95-611, Computer Sceince Department, University of Southern California, March 1995. http://catarina.usc.edu/danzig/cache.ps

[9] 知念賢一, Wcol: WWW Collector Home Page, Auguest 1994. http://shika.aist-nara.ac.jp/products/wcol/wcol.html

[10] 知念賢一, Internet における大規模情報提供と情報取得の高速化に関する研究, 奈良先端科学技術大学院大学 博士論文 NAIST-DT9561026, February 1998

[11] 河合栄治, 知念賢一, 砂原秀樹, 尾家祐二, 分散型キャッシュシステムにおける ICP 問い合わせホスト数とパケット数の解析, インターネットコンファレンス'98 論文集, pp.67-76, December 1998

付 録 A

プログラムの解説

今回 WWW アクセスの特徴調査及び性能シミュレーションのために作成したプログラムについて、概要と使用方法を示す。

A.1 ranking.pl

- **プログラムの概要**
  Squid のログファイル (access.log) を元に、各 URL 別に参照回数の集計を行ない、参照回数,URL の順で標準出力に出力する。

  出力結果を、数値降順でソートすることにより、アクセス数ランキングを作成することが可能である。

- **実行方法**
  ```
  % perl ranking.pl access.log | sort -rn > ranking.txt
  ```

- **引数**
  `access.log` Squid のログファイル (access.log) 名を指定する。
  複数のファイルを指定することも可能である。

A.2 tailf.pl,getwww.pl

- **プログラムの概要**
  Squid のログファイル (access.log) を常時監視して、キャッシュヒットしたもの (タグが TCP_*_HIT に該当するもの) に対して、リクエスト用クライアントに送信する。

今回、access.log 監視モジュールとリクエスト用クライアントの間は、常時一本の TCP コネクション (port 2454 を使用) を張っておき、通信を行なうものとする。

受信したクライアント側では、事前に fork して待機しているプロセスに対して、受信順にラウンドロビンで要求を渡す。

各プロセスは、キャッシュ無し設定になっている Squid に対して要求を行ない、Squid のログ形式で直接オブジェクトを取得した場合の時間が記録される。

- **実行方法**

  Client 側: `% perl tailf.pl /usr/local/squid/logs/access.log &`

  Server 側: `% perl getwww.pl &`

- **引数**

  `access.log` Squid のログファイル (access.log) 名を指定する。

A.3 cmptime.pl

- **プログラムの概要**

  取得コスト比較システム (図 3.2参照) では、access.log を元に、最大 8 並列で no-cache proxy にリクエストを出すため、ネットワーク距離やファイルサイズの関係で、記録される順番が入れ違いになってしまう。

  そのため、両者の access.log から、一致するものを見つけ出し、URL, ファイルサイズ, 直接取得時間, キャッシュ取得時間の順に標準出力に出力する。

  これより、オブジェクトサイズ別取得時間分布の比較が可能になる。(図 3.3〜図 3.6 参照)

- **実行方法**

  `% perl cmptime.pl access-cache.log access.nocache.log > cmptime.txt`

- **引数**

  `access-cache.log` オリジナル側のログファイル (access.log) 名を指定する。

  `access-nocache.log` 直接取得側のログファイル (access.log) 名を指定する。

A.4 retrieval.pl

- **プログラムの概要**

  オブジェクト取得時間 (Elapsed Time) の変化に伴う累積分布を調査し、累積分布 (%),Elapsed Time の書式で標準出力に出力する。

  これより、全体の何%のオブジェクトが、どれだけの時間で取得可能かを調査することが可能になる。(図 4.1参照)

  あらかじめ、Squid の access.log から、Elapsed Time 別に昇順にソートしたファイル (gettimes.txt) を作成しておく必要がある。

- **実行方法**

  ```
  % perl retrieval.pl gettimes.txt > cdf.txt
  ```

- **引数**

  `gettimes.txt` access.log を元に、Elapsed Time で昇順にソートして作成したファイル名を指定する。

A.5 averate.pl

- **プログラムの概要**

  Squid のログファイル (access.log) を元に、各 WWW サーバ別にオブジェクト平均転送速度を求め、サーバ名, 平均転送時間の順で標準出力に出力する。

  高速化対象選択時には、このパラメータを参照して高速化の有無の判断を行う。

- **実行方法**

  ```
  % perl averate.pl access.log > averate.txt
  ```

- **引数**

  `access.log` Squid のログファイル (access.log) 名を指定する。複数のファイルを指定することも可能である。

A.6 dpattern.pl

- **プログラムの概要**

  事前に作成しておいたURL リスト (今回はアクセス数上位 1000 位を記した `topurl.txt`

を使用) に含まれる URL に対して、Squid のログファイル (access.log) を元に、日別アクセス数を調査し、経過日数, アクセス回数の順で、標準出力に出力する。

この出力結果を元に、後述の `dpat2xls.pl` を用いることにより、移動平均による平滑処理を行った日別アクセス数推移グラフ (図 4.2参照) を作成することが可能である。

- **実行方法**

```
% perl dpattern.pl access.log > dpattern.txt
```

- **引数**

`access.log` Squid のログファイル (access.log) 名を指定する。
複数のファイルを指定することも可能である。

A.7 dpat2xls.pl

- **プログラムの概要**

URL 別日別アクセス数ファイル (`dpattern.txt`) に対して、指数的加重移動平均による平滑処理を行ない、Microsoft Excel 上で処理可能な書式に変換し、標準出力に出力する。

これより、日別アクセス数推移グラフ (図 4.2参照) を作成することが可能である。

- **実行方法**

```
% perl dpat2xls.pl dpattern.txt > dxls.txt
```

- **引数**

`dpattern.txt` URL 別日別アクセス数ファイルを指定する。

A.8 dpat2xls2.pl

- **プログラムの概要**

URL 別日別アクセス数ファイル (`dpattern.txt`) に含まれる 1 日辺りのアクセス数の変化量を指数的加重移動平均による平滑処理を行ない、Microsoft Excel 上で処理可能な書式に変換し、標準出力に出力する。

`dpat2xls` との違いは、アクセス数ではなく、アクセス数の変化量に対して調査しているところである。

これより、日別アクセス数推移 (差分) グラフ (図??参照) を作成することが可能である。

- **実行方法**

```
% perl dpat2xls2.pl dpattern.txt > dxls2.txt
```

- **引数**

`dpattern.txt`　URL 別日別アクセス数ファイルを指定する。

A.9　wpattern.pl

- **プログラムの概要**

事前に作成しておいたURL リスト (今回はアクセス数上位 1000 位を記した `topurl.txt` を使用) に含まれる URL に対して、Squid のログファイル (access.log) を元に、週別アクセス数推移を調査し、経過週数, アクセス回数の順で、標準出力に出力する。

この出力結果を元に、後述の `wpat2xls.pl` を用いることにより、移動平均による平滑処理を行った週別アクセス数推移グラフ (図 4.2参照) を作成することが可能である。

- **実行方法**

```
% perl wpattern.pl access.log > wpattern.txt
```

- **引数**

`access.log`　Squid のログファイル (access.log) 名を指定する。
複数のファイルを指定することも可能である。

A.10　wpat2xls.pl

- **プログラムの概要**

URL 別週別アクセス数ファイル (`wpattern.txt`) に対して、指数的加重移動平均による平滑処理を行ない、Microsoft Excel 上で処理可能な書式に変換し、標準出力に出力する。

これより、週別アクセス数推移グラフ (図 4.2参照) を作成することが可能である。

- **実行方法**

```
% perl wpat2xls.pl wpattern.txt > wxls.txt
```

- **引数**

  `wpattern.txt`　URL 別週別アクセス数ファイルを指定する。

A.11　wpat2xls2.pl

- **プログラムの概要**

  URL 別週別アクセス数ファイル (`wpattern.txt`) に含まれる 1 週辺りのアクセス数の変化量を指数的加重移動平均による平滑処理を行ない、Microsoft Excel 上で処理可能な書式に変換し、標準出力に出力する。

  `wpat2xls` との違いは、アクセス数ではなく、アクセス数の変化量に対して調査しているところである。

  これより、週別アクセス数推移 (差分) グラフ (図??参照) を作成することが可能である。

- **実行方法**

```
% perl wpat2xls2.pl wpattern.txt > wxls2.txt
```

- **引数**

  `wpattern.txt`　URL 別週別アクセス数ファイルを指定する。

A.12　renewal.pl

- **プログラムの概要**

  事前に作成しておいた URL リスト (今回は `topurl.txt` を使用) に含まれる URL に対して、Squid のログファイル (access.log) を元に、ある時間での参照回数、そのファイルの最終更新時間を調査し、経過時間数, 参照回数, 最終更新時間での参照回数の順で、標準出力に出力する。

  このプログラムを使用する場合、Squid のログファイルには HTTP 要求時のヘッダが記録されている必要がある。

  このとき、ヘッダ内に含まれる、`Last-Modified:` フィールドもしくは、If-Modified-Since フィールドから、最終更新時間を割り出している。

- **実行方法**

```
% perl renewal.pl access.log > renewal.txt
```

- **引数**

  `access.log`　Squid のログファイル (access.log) 名を指定する。
  複数のファイルを指定することも可能である。

A.13　renew2xls.pl

- **プログラムの概要**

  URL 別最終更新時間ファイル (`renewal.txt`) に対して、Microsoft Excel 上で処理可能な書式に変換し、標準出力に出力する。

  これより、更新間隔グラフ (図 4.6参照) を作成することが可能である。

- **実行方法**

```
% perl renew2xls.pl renewal.txt > renxls.txt
```

- **引数**

  `renewal.txt`　URL 別最終更新時間記録ファイルを指定する。

A.14　coh.pl

- **プログラムの概要**

  事前に作成しておいた URL リスト (今回は `topurl.txt` を使用) に含まれる URL に対して、Squid のログファイル (access.log) を元に、最終更新時間を調査し、記録しておく。

  このとき、ヘッダ内に含まれる、`Last-Modified:` フィールドもしくは、If-Modified-Since フィールドから、最終更新時間を割り出している。

  これより、実際のファイル更新時間がわかる訳だが、Squid の動作では、新鮮である場合、WWW サーバに問い合わせを行なわず応答を行なっていることから、もしかしたらオリジナルでは更新されているかもしれない。

  そのため、実際のオブジェクトの更新時間と、TCP_HIT もしくは TCP_IMS_HIT で応答しているものの最終更新を比較し、オブジェクトの一貫性がどれだけ保たれているかを調査し、同じものを返した数 (キャッシュミスなどによる直接取得も含む), 古いものを返した数, 予測で応答して同じものを返した数の順に、標準出力に出力する。

この出力結果をもとに、表 4.4が作成可能である。

このプログラムを使用する場合、Squid のログファイルには HTTP 要求時のヘッダが記録されている必要がある。

- **実行方法**

```
% perl coh.pl access.log > coh.txt
```

- **引数**

`access.log` Squid のログファイル (access.log) 名を指定する。
複数のファイルを指定することも可能である。