WWW キャッシュサーバの性能評価とチューニング

鍋島 公章
nabe@slab.ntt.co.jp
NTT ソフトウェア研究所

概要： 爆発的な WWW (World Wide Web) 人気の高まりとともに，回線容量やサーバの処理能力不足が深刻になっている．これを解消するために，WWW におけるキャッシュサーバが広く使われはじめている．報告者は，キャッシュサーバの効果的な運用方法を確立することを目的に実験運用を行っている．本稿では，その実験の中間報告として，キャッシュサーバの処理能力と，チューニング方法についての報告を行う．報告者の実験環境では，サーバの処理能力は毎秒 9 リクエスト程度(約 720 Kbps)であることが観測された．また，この実験に使われたサーバは Sun Sparc Station 20-71 という，現状では，旧機種に入るものであったが，システムの最大のボトルネックは Disk I/O 性能であった．そして，この部分のチューニングが有効であり，2 割程度の性能向上を達成した．

WWW Cache Server Performance Analysis and Turning

Masaaki NABESHIMA
NTT Software Laboratories

Abstract: With explosion of WWW (World Wide Web) popularity, the lack of bandwidth and power of server is severe problem. To solve these problems, WWW cache server has begun to be used widely. We are operating a WWW cache serer to establish its effective operation. In this paper, we report an analysis of cache server performance and a result of tunings. In our testbed, the server processed around 9 requests per second (720 Kbps). The server was a Sun Sparc Station 20-71 that is a slow machine nowadays. However, we found the bottleneck of the system was disk I/O performance and disk I/O tuning is effective. This tuning improved the cache server performance until around 11 requests per second.

1. はじめに

WWW (World Wide Web) の人気の高まりとともに，回線や WWW サーバの容量不足が発生し，アクセスが遅いという状況が一般化している．これを解消するために，情報を複製し，適度にリクエストを分散させる方法が必要となっている．このひとつの方法として，プロキシサーバによるキャッシュシステムが，広く使われはじめている．しかし，その効果的な運用方法は，まだ確立されていない．報告者は，これを確立するために，キャッシュサーバの運用実験を行っている[1]．本稿では，この実験の中間報告として，キャッシュサーバの処理能力の解析と，実験で行ったチューニングについて報告する．

すでに，キャッシュシステムの処理限界やチューニングについて，いくつかの報告が行われている[3][4][5]．また，squid-users や proxy-jp などのメーリングリストにおいても議論されている．しかし，それらは，定量的な評価が不十分であったり，考察について不完全な部分がある．本稿では，それらの報告や議論をもとに，より深い考察と定量的な評価を行うことを目標とする．

2. システム

2.1 実験環境

実験に使用したサーバ (cache.imnet.ad.jp) は Sun Sparc Station 20-71 に 512MB のメモリを搭載し，4GB の fast-wide SCSI-2 (7,200 rpm) ディスクを 6 台，キャッシュ用に用意した．これらのディスクは，ひとつの SCSI インターフェイスに接続させた．しかし，

実際には，質の悪いケースを使ったため，6 台全部をキャッシュ領域にすると，SCSI エラーが起こり，最大 5 台しか使用できなかった．システムとログ用のディスクは別の SCSI インターフェイスに接続した．また，OS としては，SunOS 4.1.4 と Solaris 2.5.1 が使用可能であったが，パフォーマンス的に SunOS 4.1.4 の方が優れていると一般にいわれていたので，SunOS の方を選択した．キャッシュサーバソフトウェアは，Squid Internet Object Cache Version 1.1.11 [1] (以降，Squid)を用いた．そして，キャッシュサーバシステムは IMnet (Inter Ministry research NETwork) 東京 NOC の FDDI リングに接続し，一般のユーザに公開して運用実験を行った．

2.2 モニタリング

今回の実験では，5 分間隔で，中継レイテンシ，Squid の稼動状況情報，OS の各種情報をモニタリングした．このモニタリングの結果は，サーバ管理者によるシステム監視を行いやすくするために，1 日単位でグラフ化し，WWW で参照可能にした．

モニタリングにおいて，OS の情報取得には以下のコマンドを使用した．

- netstat: IP, TCP など各プロトコルの処理状態，mbuf の使用状態，
- vmstat: ページング情報，ディレクトリ名ルックアップキャッシュヒット率，
- pstat: inode キャッシュ使用数，
- iostat: ディスク性能の使用率，
- uptime: システム全体の負荷．

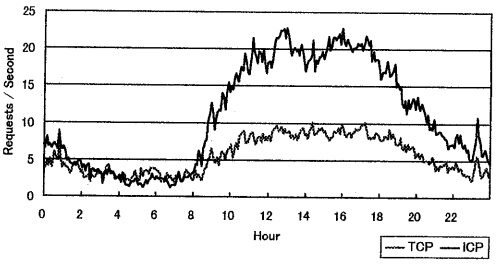
中継レイテンシの計測は，キャッシュサーバ上で動いている WWW サーバ上にある 10KB [1] のファイルを，キャッシュサーバ経由でクライアントが取るのにかかった時間を time コマンドで計測した．実際には 5 分間隔でアクセスしているため，ほとんどのアクセスはキャッシュ上でヒットしており，オブジェクトの中継ではなく，取出しにかかった時間となっている．

Squid の稼動状態のモニタリングには，Squid 付属の client プログラムを使った．

3. サーバの処理限界

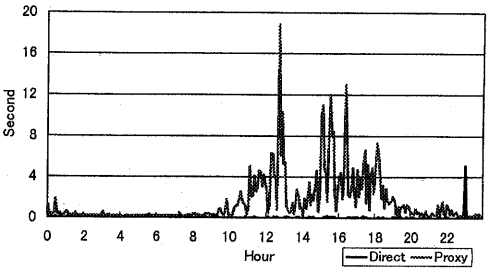
ここで示すデータは 97 年 9 月 5 日のモニタリングの結果である．この日までに，次章で述べる「オンメモリキャッシュ量を増やす」までのチューニングがシステムに対して行われている．

グラフ1：処理リクエスト数



処理リクエスト数の変移をグラフ 1 に示す．約 10 リクエスト/秒が，このサーバの処理限界である．これは，約 800 Kbps に相当する．また，この日のリクエスト総数は，WWW[2] 46 万，ICP[3] 95 万であり，ヒット率は，WWW: 33%, ICP: 18%であった．

グラフ2：レイテンシ



中継レイテンシの変移をグラフ 2 に示す．直接アクセスの方は 0.1 秒前後で安定していた(グラフでは X 軸と，ほとんど重なっている)のに対して，中継レイテンシは，ピーク時にになると 10 秒以上かかっている．したがって，ピーク時には，このサーバがボトルネックとなり，ユーザのアクセス速度が低下していたと推測できる．

中継レイテンシとリクエスト数の相関をグラフ 3 に示す．かなりのばらつきがあるが，毎秒 7 リクエスト程度までは，中継レイテンシは落ち着いている．しかし，これを超えると，かなり大きくなっているのが観測できる．

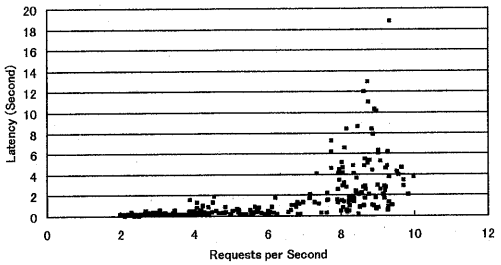
[1] このキャッシュサーバで中継されるオブジェクトの平均サイズは約 10KB である．

[2] キャッシュサーバが中継するリクエスト．

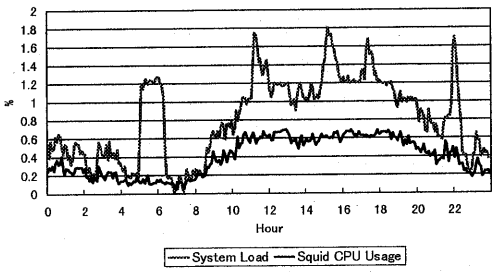
[3] Inter Cache Protocol，キャッシュサーバ間の問合わせ用リクエスト．

したがって，実際の使用に耐えるという観点では，このサーバの処理限界は毎秒 7 リクエスト程度であるといえる．

グラフ3：レイテンシ相関



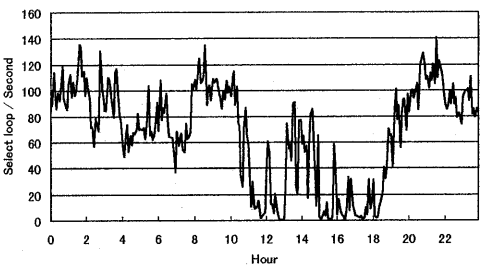
グラフ4：負荷



システム全体の負荷と，Squid の CPU 使用率の変移をグラフ 4 に示す．このグラフのように，ピークにおいても Squid は CPU を全部使っていない．同時に，システム全体の負荷も，ピークでは 1.8 程度まで大きくなっている．しかし，この時に，ps コマンドで確認すると，他に CPU を大量に使うプロセスは動いていなかった．そのため，これは Squid がディスク入出力待ちをしていて，見かけ上，システムの負荷を上げているものだと推測される．また，ピークにおける，ディスク性能の利用率は 30%程度であったが，キャッシュサーバの場合，多数の小さなファイルを扱うために，この程度ですでに限界であるとも考えられる．なお，システム負荷が 5 時から 7 時まで大きくなっているのは，ログの解析と保存処理を行っているためである．

Select システムコールはオブジェクト中継処理の中心を担う，その 1 秒間のループ数の変移をグラフ 5 に示す．ピークにおいては，この監視ループが，ほとんど回っておらず，処理能力が不足していることが観測できる．

グラフ5：Selectループ



4. チューニング

表 1:チューニング結果

チューニング項目	リクエスト処理数/秒
標準	データなし
Socket オープン待ち数を増やす	9.40
ファイルディスクリプタ数を増やす	9.32
高速 malloc の使用	9.65
ディスクキャッシュ量を減らす	9.66
オンメモリキャッシュ量を増やす	10.85
キャッシュ用ディスク数を減らす	10.38
Inode キャッシュ，DNLC 量を増やす	11.42
ファイルディスクリプタ数，TCP バッファを減らす	10.28
TmpFS の使用	16.57

今回行ったチューニングの項目と，その結果を表 1 に示す．これらは，1997 年 7 月から 10 月の間に，基本的に，表の上から順に，前回行ったチューニングを行った状態で，新しいチューニングを次に行った．そして，最低一週間は，その状態で運用した．ただし，“Inodeキャッシュ，DNLC 量を増やす”に関しては，安定したチューニングを行うことができなかった．そのため，チューニングを試行した後，元に戻し，“ファイルディスクリプタ数，TCPバッファを減らす”を次に行った．また，“TmpFS の使用“に関しては他の項目と独立して行った．

処理能力の評価としては，1 秒あたりのリクエスト処理数を指標とした．これは以下のように決めた．1 日の処理能力の指標を，その日の中の高い値 10 個の平

均とし，一週間のうちの高い値を持つ 3 日を選び，その平均を，チューニング項目に対する指標とした．

この章の残りでは，それぞれのチューニング項目について，その説明と，効果に対する考察を行う．なお，チューニングをはじめた状態では，ディスクキャッシュ容量を 18GB，オンメモリキャッシュ(メモリ上のオブジェクトキャッシュ領域)容量を 128MB に設定した．

4.1 Socket オープン待ち数を増やす

Socket 処理ルーチンでは，オープン処理の前に，そのリクエストを，待ち行列に入れる．これの標準長は 5 である．負荷が高くなると，この待ち行列を使い切り，リクエストが拒絶されることが多いといわれている[7]．

そのため，この値を，64 まで大きくした．実際に，高負荷の状態では，6 以上のリクエストが待ち行列に入っていた．たとえば，96年7月9日において，1 日 288 回待ち数を計測したが，そのうちの 24 回は 6 以上の待ち数であった(最大待ち数は 15)．そのため，ある程度の効果はあったものと推測される．しかし，実験計画のミスにより，今回の実験では，チューニング前のデータを取らなかったため，実際の定量的効果の確認には至っていない．

4.2 ファイルディスクリプタ数を増やす

Squid では，同時処理可能なリクエスト数は 1 プロセスが使用可能なファイルディスクリプタ数(標準の SunOS では 256)により制限される．また，プロキシ中継では，ひとつのリクエストに対して，プロキシサーバとクライアント，プロキシサーバと WWW サーバという二つのファイルディスクリプタを用いる．そのため，同時に処理できるリクエストは，ファイルディスクリプタ数の半分である 128 が最大である(ログ管理やシステム用にいくつかのファイルディスクリプタが使われるため，実際の値はこれよりも小さい)．

この処理可能なリクエスト数の制限により，クライアントからのリクエストを，キャッシュサーバが受け付けられない状態が発生した．これを解消するため，SunDBE (Sun DataBase Exelerator) を導入して最大 2048 個のファイルディスクリプタを使用できるようにした．これにより，最大同時処理リクエスト数は 1024 まで拡大された．しかし，サーバの処理能力が，すでに超えているのか，これを増やしても，実際の 1 秒間の処理リクエスト数は増えなかった．

今回の実験では，逆に処理リクエスト数が減っている．ひとつの理由は，このチューニングと同時に，モニタリング頻度を 30 分に 1 回から 5 分に 1 回に増やしたために，システム全体の負荷が大きくなったことだと推測される．

4.3 高速 malloc の使用

Squid のようなキャッシュサーバでは，大量のメモリを使用する．たとえば，18GB のキャッシュオブジェクトを管理するためには，約 100MB の領域が必要である．また，オブジェクトの中継処理にも，最低で数 10MB 必要とする．そして，通常はオンメモリキャッシュに，ある程度の領域を割り当てて運用する(この時点では 128MB を割り当てていた)．実際に，このチューニングを行う前日には，380MB のメモリが Squid プロセスに割り振られていた．したがって，この巨大なメモリ領域を管理している malloc ライブラリを高速なものに代えることにより，性能の向上が期待できる．この実験では標準の malloc ライブラリに代えて，Gnu libc 2.X に使われている malloc ライブラリのベースである malloc-2.6.4 を使用した．これにより，若干の性能向上がみられた．高速化の代償に，メモリ使用量が増えているものと推測されるが，Squid は処理トラフィックによってプロセスサイズが変化するため，malloc ライブラリ変更の影響を計測できなかった．

4.4 ディスクキャッシュ量を減らす

いくつかのサイトで，キャッシュサーバの性能向上を狙って，ディスクキャッシュ容量を縮小させている．これの効果を確かめるために，18GB から 4GB までディスクキャッシュ容量を減らした．しかし，このサーバの場合は，目立った効果は得られなかった．

一方，管理するオブジェクト数が減ったことにより，プロセスサイズが約 75MB 小さくなった．そのため，実メモリ量が不足しているサーバには，スラッシングを防ぎ，性能を向上させる効果があるといえる．

4.5 オンメモリキャッシュ量を増やす

Squid は物理ディスク以外に，メモリ上にもオブジェクトをキャッシュする．そして，オブジェクトがこのオンメモリキャッシュでヒットすれば，物理ディスクへアクセスせず，ディスクへの負荷が軽減される．ここまでは，この大きさを 128MB にして運用していたが，これを 320MB まで大きくした．この結果，1 割程度の性能向上が達成された．

4.6 キャッシュ用ディスク数を減らす

ここまでは，ひとつの SCSI インターフェイスにつながった 5 台のディスクを使用していた．これを，ディスクキャッシュ容量を変化させずに，2 台にまで減少させた．この結果，処理能力が低下した．これは，ディスク

の使用率が 5 台の時は，ピークで平均 30%程度であったのが，2 台にしたことにより，60%程度まで上がり，過負荷になったためだと推測される．

4.7 Inode キャッシュ，DNLC 量を増やす

ディスクアクセスを減らす方法として，Inode キャッシュやディレクトリ名ルックアップキャッシュ(DNLC)を大きくすることが有効であるといわれている[6]．これらの領域の大きさはカーネルの設定によって異なるが，今回の実験環境では，それぞれ，5658，3354 であった．そして，DNLC のヒット率を調査すると 70%程度であった．このため，それぞれの領域を 30000，15000 まで大きくした．これにより，DNLC のヒット率が 90%程度になり，1 割程度の性能向上が達成された．

しかし，Inode キャッシュと DNLC 領域はカーネル内の mbuf 領域(SunOS の場合，固定領域)を使い，日中では，数分でこの mbuf 領域を使い尽くしてしまった．この結果，OS に機能障害が発生し，管理者が手動でシステムをリブートさせなければならない状態(以降，フリーズ)に陥った．今回の実験では，一度だけ，2 日間連続して動いた(上記の性能評価は，この時のものである)が，連続運転可能条件の分析には至っていないため，このチューニングは実際には使えず，この結果は参考データにとどまっている．

4.8 TmpFS の使用

ディスクI/O 待ちが，どれぐらいシステム全体の性能に影響を与えているか調べるために，メモリ上ファイルシステム(TmpFS)にディスクキャッシュ領域を置いて運用した．ここでは，メモリ使用量の少ないバージョンの Squid (Squid-NOVM)を使用した．実際に，この Squid のメモリ使用量は数 10MB 程度であった．これにより，スラッシングしないで使える TmpFS の領域を大きく取ることができた(キャッシュ領域は 300MB)．この結果，最大 16.57 リクエストまで性能が大幅に向上した．このことより，逆に，ディスク I/O 待ちのボトルネックが大きかったといえる．

4.9 ファイルディスクリプタ数，TCP バッファを減らす

多くのリクエストを受け付けると，TCP の送受信バッファで mbuf の領域を使い切ってしまい，OS がフリーズすることが多発した．したがって，安定運用を目指すためには，ファイルディスクリプタ数を減少させ，同時処理するリクエスト数に制限をつける必要があった．ファイルディスクリプタ数を 512 に制限し，TCP 送受信バッファを標準の半分である 2048 バイトにすると安定して動くことを確認したが，この安定運用の限界値は，分析中である．

しかし，ファイルディスクリプタ数を制限すると，ピーク時には，いくつかのリクエストがタイムアウトした．そして，その中には Squid の状態モニタリングリクエストも含まれ，安定運用を目指すと，ピーク時のモニタリングが不可能になった．今回の実験では，モニタリングを行うために，ファイルディスクリプタ数を最大で運用したが，実際には，週に 1 度は OS がフリーズしており，そのたびに管理者がコンソールでリブート処理を行っていた．

4.10 ブロックサイズの拡大

ファイルシステムのチューニングとして，ブロックサイズを大きくして 1 回の転送量を多くする方法が一般的である．これの有効性を予測するため，キャッシュ中のオブジェクトサイズを調べた．その結果，平均オブジェクトサイズは 17.6KB であるが，その分布は，1 ブロック(8KB)以下の大きさのオブジェクトが全体の 72.0%，2 ブロック必要とするオブジェクトは 12.9%，同じく 3 ブロックのものは 4.7%でしかなかった．したがって，たとえブロックサイズを標準の倍の 16KB にしても，それほど大きな効果はないと推測されるため，今回，このチューニングは見送った．

4.11 チューニングの考察

今回のチューニングで大きな効果があったのは，「オンメモリキャッシュを増やす」，「Inode キャッシュ，DNLC 量を増やす」，「TmpFS の使用」という，ディスク I/O 処理に関するものである．これは，Squid はシングルスレッドで作られているため，ディスクI/O 待ちがプロセス全体に影響していると，従来いわれてきたことを，裏付けることになった．

また，次のバージョンの Squid においては，ディスク I/O 待ちを避けるために，マルチスレッドが導入される予定である．これにより，大幅な性能向上が達成されると予想される．

5. 運用についての考察

回線に対して十分なサーバの処理能力がある場合は，ボトルネックは回線の太さであり，キャッシュサーバの負荷について特に注意する必要はない．しかし，今回の実験環境のように，回線に対して不十分な処理能力しかキャッシュサーバが持たない場合には，キャッシュサーバがボトルネックとなる．つまり，キャッシュ

サーバを使うことにより，逆にアクセス速度が低下するということが発生する．これは兄弟サーバ関係(オブジェクトがキャッシュサーバにある時だけ，アクセスしあう関係)として使用している場合も同様に起こる．適切にリクエスト量を制限し，レイテンシが大きくならない範囲でキャッシュサーバを運用すべきである．

もっとも望ましいのは，処理能力以上のリクエストが来た時に，それをキャッシュサーバ経由でなく，直接アクセスさせる機能をキャッシュサーバが持つことである．しかし，このような動的な機能がない現状では，サーバ側でピーク時間帯は，一部ユーザのアクセスを拒否するか，クライアント側でピーク時間帯はサーバを使用しない，というような運用を行う必要がある．

6. 今後の課題

現在のモニタリングシステムをパッケージ化し，他のプラットフォームでも使用可能にする．そして，現在のものは 1 日単位でしかモニタリング結果をグラフ表示しないが，これを，リアルタイムに表示できるように拡張する．

チューニングに関しては，今回やり残したチューニング (CPU のアップグレード，RAID ディスクの使用，複数 SCSI インターフェイスの有効利用) を行う．さらに，OS を Solaris に変更して，SunOS では行えない DNLC の拡大などの mbuf を大量に使うチューニングを行う．また，Solaris は，SunOS より，多くの項目のモニタリング機能があるため，これらを使った，より詳細なモニタリングを行う．

7. おわりに

本実験では，Sun Sparc Station 20-71 と Squid を用いて WWW キャッシュサーバの処理限界とチューニング方法について調査した．この結果，以下のような結果を得た．

- チューニングを行う前は，毎秒 9 リクエスト程度が処理限界であった．
- 処理限界近くでは，オブジェクトの中継レイテンシが 10 秒以上まで大きくなり，実際の使用に堪えなかった．実用範囲は，このシステムの場合，処理限界の 7 割程度であった．
- ディスク I/O 性能が，全体の性能に大きく関与した．そして，ディスク I/O に関するチューニングが有効であった．
- 最終的なチューニング後は，最大，毎秒 11 リクエスト程度まで性能が向上した．
- メモリ上ファイルシステムにディスクキャッシュ領域を置くことにより，毎秒 17 リクエスト程度まで大幅に性能が向上した．
- SunOS では mbuf の領域が固定されているため，TCPコネクション数が多くなると，mbuf 領域を使い切り，OS に障害が発生した．また，mbuf を大量に消費するチューニングを行った場合も同様の障害が発生した．

謝辞

不安定なキャッシュサーバの面倒をみてくださる IMnet 東京 NOC の皆様に感謝します．この実験をサポートしてくださったソフトウェア研究所の三上リーダをはじめとする IMnet チームの皆様に感謝します．実験の場所を提供してくださった科学技術庁 IMnet 担当の皆様に感謝します．

本研究は，科学技術庁の平成 9 年度科学技術振興調整費による「生活工学アプリケーション研究」の一環として行われている．

参考文献

[1] 鍋島 公章, “基幹系 WWW キャッシュサーバの運用実験について”, 第55回情報処理学会全国大会, 1997 年 9 月

[2] D. Wessels, “Squid Internet Object Cache”, http://squid.nlanr.net/Squid/, 1997

[3] Carlos Maltzahn, Kathy Richardson, Dirk Grunwald, “Performance Issues of Enterprise Level Web Proxies”, 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, June 15-18, 1997

[4] Adrian Cockcroft, “Dissecting proxy Web cache performance”, Sun World, Vol.11 No.7, July, 1997

[5] Oskar Pearson, “Squid Users Guide”, Sep, 1997, http://cache.is.co.za/squid/

[6] Adrian Cockcroft, “Sun Performance and Tuning: SPARC and Solaris”, Prentice Hall, 1994

[7] Mark Morley, “Incleasing SOMAXCONN”, http://www.islandnet.com/~mark/somaxconn.html