平成１３年度 卒業研究

Web を用いた C/S 型マルチメディアトランザクション

についての研究

～　WebStone 2.5 の解析　および　HTML リンク抽出機能の拡張　～

2002 年 2 月 12 日

島根大学 総合理工学部 数理・情報システム学科

計算機科学講座（田中研究室）

ｓ９８４５３－Ｕ　高木　明

目次

はじめに

マルチメディア化が進んだ今日、CATV、ADSL、光ケーブルなどブロードバンドと呼ばれる高速な通信回線の普及が進み、大容量の転送を行うネットストリーミングなど、それに対応した Web コンテンツも増えてきた。しかしブロードバンドによって速くなるのはプロバイダからクライアントまでの通信速度のことであり、サーバの速度が速くなったわけではない。つまり、サーバのレスポンスや、接続回線の品質が悪ければブロードバンド向けコンテンツの有利性を生かしきれていないといえる。そのため、コンテンツが快適に使用できるかどうか、サーバ自身のソフトウェアおよびハードウェア性能を測定することが求められる。このような性能を測定するベンチマークソフトは多々あるが、本研究ではオープンソースである Mindcraft 社の WebStone を使用することにした。WebStone では実際に Web ブラウザで Web ページを参照した際に行われるリンク抽出機能をシミュレートしないので、追加機能としてリンク抽出する関数を作成した。

第１章　　WebStone 2.5 の導入

１-１　WebStone とは

WebStone は Web サーバの性能を測定するベンチマークソフトである。SGI 社(シリコングラフィックス社)が初版を作成し、その後 Mindcraft 社がその権利を獲得した。Mindcraft 社は信頼性と移植性を向上させた WebStone2.0.1、そしてそれに CGI テスト、API テストを追加した WebStone2.5 を作成した。現在は Mindcraft 社がメンテナンスを行っており、GPL の元に Mindcraft 社のホームページ(http://mindcraft.com/　または　http://www.mindcraft.com/)にて配布されている。

また WebStone には Ataman Software 社の rexec プログラムを含む。

WebStone は総括プログラムである Webmaster プログラムを実行している１台の WebStone コントロールマシンと、その管理下にある WebClient プログラムを実行する複数台の WebStone クライアントマシン、そしてベンチマーク対象である Web サーバから構成される。WebStone ではこれらすべての機能を１台のマシンで行うこともできる。

WebStone2.0.1 および WebStone2.5 の測定結果は CGI および API テスト以外の点では比較可能である。しかしそれ以前のバージョンとの互換性はないため注意が必要である。

１-２　WebStone の動作

WebStone によるベンチマーク測定ではまず、Webmaster と呼ばれるプログラムを実行する。その Webmaster は実際に計測を行う WebClient プログラムと試験内容ファイルを LAN 上のクライアントコンピュータに配布し、これを総括する。WebClient は Web サーバからファイルを読み込むことで、ユーザが Web ブラウザによって Web ページを閲覧することをシミュレートし、サーバに負荷をかける。このときサーバに高い負荷をかけるため WebClient は自身のコピーをホストに作り出す。これによって１台のコンピュータに 100 を超える WebClient を作り出し、実際にあるホストが作り出す負荷よりもはるかに大きな負荷をかけることができるようになる。その高い負荷の下で WebClient はファイル読み込みの一連の動作にかかる時間とその読み込んだデータ量を計測し Webmaster に返す。Webmaster はそれらの結果を集計しユーザにベンチマークの結果を返す。WebStone プログラムのサイズは小さくキャッシュから起動でき、サーバのディスクアクセススピードに影響せずに試験できるため、純粋に CPU、ネットワークスピード、OS をテストすることができる。

以上のような動作は Prel によって自動実行できる。しかしコマンドラインから Webmaster プログラムを直接実行することでも測定が可能である。

１-３　WebStone 2.5 の測定対象

WebStone は３つのアクセス方式の異なったタイプを使い Web サーバをテストする。

・ HTML ：

このテストでは Web サーバそれ自身は Web クライアントに通常 Hyper Text Markup Language で書かれた指定ファイルを返す。このアクセス法はたいていの Web ページを提供するために使われる。

・ CGI ：

このテストでは Web サーバは、リクエストで指定したパラメータをそのプログラムに渡す Common Gateway Interface プロトコルを使い、もう１つのプログラムを動作させる。この方法は一般的にフォームを処理し、そして Web ページの上にユーザ入力に対する返答を提供するために使われる。CGI 作業負荷は、それぞれのバイトのごとに乱数関数を呼び出すことによって、返すデータが戻ったと計算する。この作業負荷は集中的な計算がおこなわれ、Web サーバソフトウェアの CGI インタフェースの効率より CPU スピードをいっそう反映する。WebStone 2.5 では、任意の作業負荷、CGI2.5 作業負荷があり、それは環境からファイル名を受けとり、ファイルでデータを返す。

・ API ：

これらのテストではプログラムを起動させる Application Programming Interface が Web サーバ中にリンクしたアクセス要求を渡すべき Web サーバを Web ページのリクエストに返答するために呼ばれる。この手順は一般に CGI 手順よりリクエストに対するプログラムに基づいた返答を実現するいっそう効率的な方法である。WebStone2.0.1 で、API 作業負荷は、それぞれのバイトのために乱数関数を呼び出すことによって、返すデータが戻ったと計算する。 この作業負荷は集中的な計算がおこなわれ、Web サーバソフトウェア CGI インタフェースの効率より CPU スピードにいっそうきめ細やかなものとなる。WebStone2.5 で、任意の作業負荷、API ２.５作業負荷があり、それは求められた URL からファイル名を受けとって、そしてファイルでデータを返す。 WebStone 2.01 はネットスケープの API(NSAPI)に対してのみサポートしていたが、Mindcraft 社はマイクロソフトのインターネット情報サーバ API(ISAPI)のための API テストを移植し、Web サイトでそれらのテストを利用可能にした。WebStone 2.5 は NSAPI とマイクロソフト両方のインターネットサーバーAPI(ISAPI)に対してサポートしている。

１-４　WebStone 2.5 の測定結果

WebStone は主に処理能力(バイト/秒)そして待ち時間(リクエストが完了する時間)を測定する。WebStone は同じく ページ/分、接続レート平均と他の数を報告する。これらのいくつかはユーザがスループット測定の正しさをチェックするのを手助けする。

スループットの２つのタイプ：総計と client 毎 が測られる。両方とも全部のテスト時間とすべてのクライアントベースで平均する。総計のスループットは完全なテスト時間までに分岐したテストを通じて転送されるただの合計バイト(本体＋ヘッダー)である。client 毎スループットは総計のスループットをクライアントの数で割る。

待ち時間の 2 つのタイプ：接続待ち時間とリクエスト待ち時間 が報告する。それぞれの計量のために、平均時間はすべてのデータの標準偏差と同様に、最小・最大時間を加えて提供される。接続待ち時間は接続を確立するのにかかる時間を反映し、リクエスト待ち時間は接続が確立されてから、データ転送を完了するまでの時間を反映する。ユーザによって認知された待ち時間は接続とリクエスト待ち時間の合計に加え、WAN 接続、ルータ、モデムなどのため使われる、どんなネットワーク待ち時間でも含む。

WebStone はまた little's の Ls と呼ばれる計量を報告する。Ls は最小法から得られ、サーバがオーバーヘッドとエラーよりむしろリクエスト処理でどれぐらい時間を費やすかを反映する。 Ls はまた Web サーバがどんな特定の瞬間にでも開くようにする平均接続数の間接の指標である。

この数字はクライアントの数に非常に近い状態でいるべきである。そうでなければいくつかのクライアントは随時サーバにアクセスすることを拒否される。

Web サーバに十分に高い負荷をかけると、結果でエラーが現れてくる。(少なくとも WebStone が関係している限り)それはもっともなことである。それはただサーバに重い負荷がかけられていることを意味し、それらがタイムアウトする前には、いくつかのクライアントはサービスを提供されていない。実際、特定の負荷においてのエラーの数はサーバが非常に重い負荷の下で能力を発揮するであろうふるまいの優秀な指標である。

第２章　　WebStone 2.5 の構成

２-１　WebStone 2.5 のコンパイルおよびインストール

WebStone のソースファイルおよび実行ファイルは Mindcraft 社のホームページ www.mindcraft.com にて入手できる。

２-１-１　UNIX の場合

exec サービスの許可：

WebStone はリモート実行のため exec サービスを利用する。exec サービスはセキュリティ上の問題でデフォルトで使用できない。このため、/etc/securetty ファイルを変更し、アクセスを許可する。また、RPM をインストールし、サービス要求を強化する。

- rsh-server RPM をインストールする。これは CD-ROM またはインターネットでダウンロードできる。
- ntsysv を実行し、rexec を有効化する。
- xinetd を/sbin/service xinetd restart で再起動し、ntsysv の変更を有効化する。
- /etc/securetty に rexec を追加する。

ソースから行う場合：

- Ｃコンパイラがシェルのサーチパスにあることを確かめる。もし「whence cc」または「which cc」で使いたいＣコンパイラの名前が取得されないなら、そのＣコンパイラのフルパスを $CC 環境変数の中に入力する。
- ソースアーカイブを解凍する。ＵＮＩＸとＵＮＩＸ互換のシステムでは、普通はこのようにすればよい：
  gunzip < W25_src.tgz | tar xf -
- カレントディレクトリを WebStone/src へ変更する。
- ディレクトリがきれいであることを確かめる：
  make clobber
- ソースファイルをカスタマイズする：
  ./configure
- 実行ファイルを作り、そしてインストールする：
  make install

make install が使っているシステムの上でエラーなく完了する。

コンパイラが標準Ｃ関数プロトタイプを理解しないなら、WebStone はコンパイルを行わない。もし WebStone が WebStone に知られるライブラリにないライブラリ関数を必要とするなら、このようなエラー見るだろう

Unresolved external reference：foobar( )

その場合ミスした関数の入っているライブラリを見つけ、WebStone/src/Makefile の「LIBRARY=」定義にそのライブラリを追加する。

実行ファイルのアーカイブから行う場合：

- 実行ファイルのアーカイブを解凍する：
  gunzip < WS_sol251.tgz | tar xf –

2-1-2　Windows NT の場合

・ WinZip でソースアーカイブを解凍する。
・ WebStone¥NT40 ディレクトリを Microsoft Developer's Studio のコンパイルワークスペース定義と nmake を使うコンパイルの Makefile に含める。
・ コンパイルする。

2-2　環境ファイルの設定

コントロールマシン、クライアントマシン、Web サーバそれぞれにせっていが必要である。ここでは、その rcp、rexec といったリモート動作に関する環境設定、および WebStone の設定について作成する手順に沿って述べる。

1. コントロールマシンでのユーザ作成
   WebStone ではコントロールマシンの下、複数台のクライアントマシンをコントロールし、ベンチマークを行う。その際、すべてに全てに一貫したユーザ権限で実行するようにするそのため、ここでコントロールマシンとしてのユーザアカウントを作成する。ユーザアカウント作成は root で作業する。
   ```
   # useradd webstone
   # passwd webstone
   ```

2. クライアントマシンでのユーザ作成
   先に作成した、コントロールマシンでのユーザと 同じ名前でユーザを作成する。パスワードも、少なくとも複数のクライアントマシンを用意する場合、クライアントマシン同士では同じものにする。 これも、root で行う。
   ```
   # useradd webstone
   # passwd webstone
   ```

3. クライアントマシンでの~/.rhost の作成
   コントロールマシンからクライアントマシンに対し、 クライアントプログラムを rcp でコピーする。このため、コントロールマシンからの、rcp 要求を認証し、受け入れる必要がある。具体的には、今作成したユーザの home ディレクトリに、rcp のために~/.rhosts を作成する。

   ファイル形式：
   サーバ名１
   サーバ名２
   　　：
   　　：

   例：
   コントロールサーバの名称が、control.mydomain.com だとすると、 ~/.rhosts ファイルは以下のようになる。
   ```
   control.mydomain.com
   ```

   なお、~/.rhost はパーミッションを所有者以外は、読み書き不能な状態にする必要がある。
   ```
   $ chmod 600 ./rhost
   ```

4. testbed

/usr/share/webstone/conf に WebStone としての設定ファイル testbed がある。これらのうち、特に変更が必要なものは以下のとおりである。

- SERVER

webserver を指定する。IP アドレスでもよい。
例：
SERVER="www.mydomain.com"

- WEBDOCDIR

webserver のドキュメントルート。
コントロールマシン上で webstone -genfiles コマンドを実行すると、 このディレクトリに対し、データが作成、rcp される。
例：
SERVER="www.mydomain.com"

- CLIENTS

クライアントマシンを指定する。IP アドレスでもよい。クライアントマシンが複数台ある場合は、空白で区切る。
例：
CLIENTS="client1.mydomain.com client2.mydomain.com"

- CLIENTPASSWORD

クライアントマシンのパスワードを指定する。クライアントマシンが複数台ある場合でも一つしか 設定できない。従って、前出のクライアントマシンへの アカウント作成時の ID、パスワードも同一のものでないと うまくいかない。
例：
CLIENTPASSWORD="client1.mydomain.com client2.mydomain.com"

- ベンチマークパラメータ

ITERATIONS=”3” テスト全体の実行回数
MINCLIENTS="8" 最小クライアント数
MAXCLIENTS="128" 最大クライアント数
CLIENTINCR="8" クライアント増分
TIMEPERRUN="30" 動作時間
この例の場合、WebClient の個数を８個から１２８個まで８ずつ増加させて動作させる。１回のベンチマークは３０分行われ、テスト全体は３回行われる。

5. filelist

/usr/sahre/webstone/conf にテストする URL を指定したファイル、 filelist がある。WebClient はこれに基づき Web ページをフェッチする。

形式：
URL 取得回数 #コメント
URL 取得回数 #コメント
：

例：
/file500.html 350 #500
/file5k.html 500 #5125
/file50k.html 140 #51250
/file500k.html 9 #512500
/file5m.html 1 #5248000

2-3　WebStone 2.5 のファイル構成

ここではＣソースのファイル名とその役割のみを列挙する。

ファイル名	役割
acconfig.h	ディストリビューション情報
bench.c	型変換関数
bench.h	型変換関数定義
config.h	環境設定ファイル
debug.h	デバッグ用関数定義
errexit.c	エラー出力関数
genrand.c	テストファイルの自動生成
get.c	HTML ページ読み込み関数
get.h	HTML ページ読み込み関数定義
getopt.c	オプション設定
gettimeofday.c	現在時刻取得関数
parse_file_list.c	ファイルリスト解析関数
parse_file_list.h	ファイルリスト解析関数定義
random.1.h	乱数生成関数定義(未使用)
random.c	乱数生成関数
random.h	乱数生成関数定義
rexec.c	リモート実行関数
statistics.c	統計関数
statistics.h	統計関数定義
sysdep.c	システム関数
sysdep.h	システム関数定義　定数定義
timefunc.c	時間用統計関数
timefunc.h	時間用統計関数定義
webclient.c	WebClient プログラム
webmaster.c	Webmaster プログラム

今回以下については解析を割合した。

API-test¥	API テスト用
ws20_iss.c	WebStone2.0 ISS 用関数
ws20_ns.c	WebStone2.0 NS 用関数
ws25_iss.c	WebStone2.5 ISS 用関数
ws25_ns.c	WebStone2.5 NS 用関数

CGI-test¥	CGI テスト用
ws20_cgi.c	WebStone2.0 用関数
ws25_cgi.c	WebStone2.5 用関数

2-4　クライアント・サーバ・モデル

WebStone では Webmaster と WebClient のやり取りに socket を用いる。この socket は同期機構としてクライアント・サーバ(client server)方式を採用している。クライアント・サーバ方式では、同期の方法が非対称であり、サーバ側(webmaster)とクライアント側(WebClient)とで、異なるプログラムを書かなければならない。同期は、他のホストからの通信を待ちうけるサーバと、サーバに対してネットワークの接続を要求するクライアントの間でおこなわれる。サーバは、任意のホスト上のクライアントからの接続要求を同時に待ちうけることができる。一方、クライアントは特定のサーバに対して接続を要求する。クライアントが接続を要求したとき、サーバが要求待ち状態にあれば、同期が成立して通信が始められる。もしサーバが要求待ち状態にない場合は、それ以上待つことはせず、即座に同期(connect( ) システムコール)が失敗する。

以上のような socket の同期処理は、複数のシステムコールによって実現される。次の図は、それらのシステムコールの関係を示したものである。

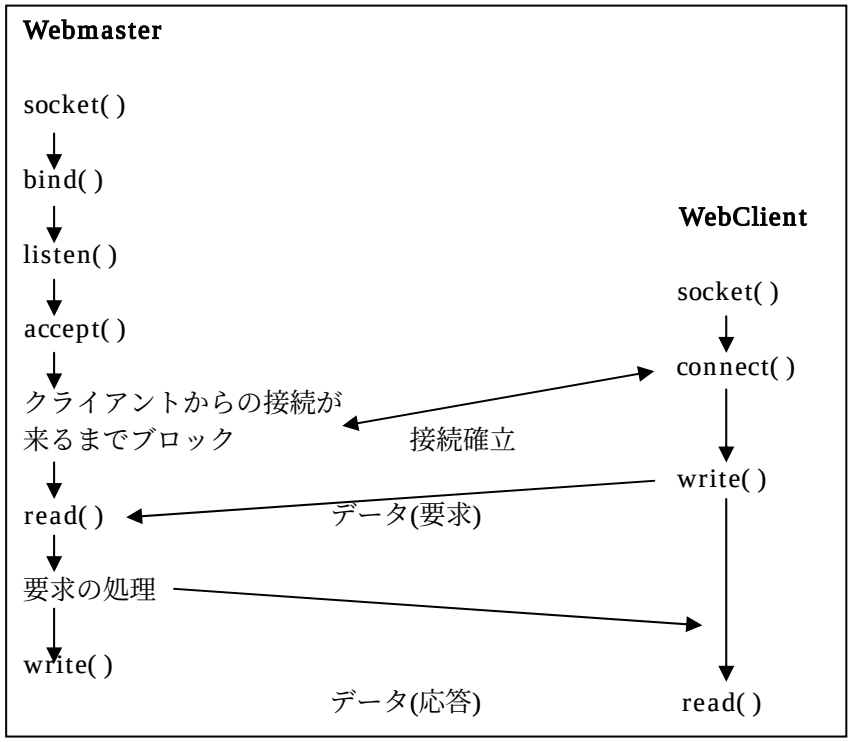

図 2.1 サーバ・クライアント・モデル

クライアント側は比較的単純だが、サーバ側はたくさんのシステムコールを呼ばなければならない。

クライアント側では、まず socket( )で socket を作成し、次に connect( )で同期をとる。通信相手のサーバは connect( )の引数で指定する。同期がとれて通信の準備が整えば、connect( )システムコールが返るので、read( )(受信), write( )(送信) を使って通信を始めることができる。

一方、サーバ側ではまず socket( )で socket を作成した後、bind( )で socket にポート番号を設定する。ポート番号は、外部のホストに対する socket の識別番号のようなものである。前述のとおりクライアントはホスト名とポート番号の対で、通信しようとしているサーバの socket を指定する。ポート番号は 0 から 1023 までは特権ユーザだけが使用できる。それ以外のポート番号は自由に使えるが、一部のポート番号は OS がすでに使用しているものがあるため、一般には 5001 以降の適当な番号を使うとよい。

サーバ側では、bind( )の後、listen( )を呼ぶ。listen( )はその socket がサーバ側の socket として使われることをＯＳに通知し、接続要求の受けつけを開始させる。第２引数として socket が一度に受けつけられる接続要求の数を示す。この値はＯＳによって異なる。

accept( )は、接続要求しているクライアントをひとつ選び、そのクライアントと通信するための socket を新たに作成して同期をとり、socket の識別子を返す。accept( )が返した socket はひとつで双方向の通信に使える。その socket に対して read( )すれば受信、write( )すれば送信の意味になる。元の socket は他のクライアントからの接続要求を待つのに使われる。その socket に対して accept( )を実行すれば、接続されるのを待っている別のクライアントが選ばれ、そのクライアントと通信するための socket が返される。

socket は UNIX で使用されていたが、WIN32 で socket を利用できるようにするため Winsock という規格が作られた。TCP/IP プロトコルスタックの実装方式の違いにより完全に同一のものではないが、以前より移植がしやすくなった。

2-5　Webmaster

WebClient を総括する Webmaster プログラムの説明をする。

1. 必要な変数を定義する。ただし変数の多くはグローバル変数として定義されているため、ここで定義される変数は多くない。Win32 の場合、Winsock の初期化も行う。
2. ParseCmdLine( )関数によりコマンドラインを解析し、オプションから設定を取得する。
3. SetupSyncSocket( )関数により、WebClient との情報のやり取りに使用するソケットを作成する。
4. MakeCmdLine( )関数により、WebClient 用のコマンドラインを作成する。
5. RexecClients( )関数により rexec( )関数を用いて WebClient を各 LAN 上のクライアントマシンで実行させる。
6. 受け取り用のメモリ領域を確保し、WebClient の準備ができるまで(すべての WebClient から「READY」メッセージを受け取るまで)待つ。
7. すべての WebClient の準備ができたら、「GO」メッセージを送信し、WebClient にベンチマークの開始を支持する。
8. テスト時間が設定された環境でのベンチマークでは、WebClient が計測中は、Webmaster はするべき動作がないため、計測終了まで sleep( )関数によりスリープ状態(プログラム停止状態)なる。
9. GetResults( )関数により各 WebClient から結果を収集する。
10. PrintResults( )関数により集計結果を出力する。
11. Webmaster を終了する。

2-6 WebClient

ベンチマーク計測を行う WebCleint プログラムを説明する。
各 WebClient は分岐により子プロセス ClientThread( )を作成する。ClientThread( )は実際に計測を行う関数であり、親プロセス(WebClient)自体は計測を行わず、それらの子プロセスの制御をする。ClientThread( )はコンフィグファイルによって指定された数だけ作成される。

1. 必要な変数を定義する。Webmaster と同じく、変数の多くはグローバル変数として定義されているため、ここで定義される変数は多くない。Win32 の場合、Winsock の初期化も行う。
2. プログラム終了時に実行する関数(sock_cleanup( ) - WinSock のリソースの解放)を指定し、異常終了に備える。
3. 指定されたスレッドの相対優先順位値を設定する。
4. webclient スレッドをスレッド標準の相対優先順位値より 2 ポイント高い相対優先順位値を指定する。
5. 子プロセスの管理のためのセマフォを作成する。
6. ParseCmdLine( )関数によりコマンドラインを解析し、オプションから設定を取得する。この ParseCmdLine( )関数は Webmaster のそれとは異なる関数で、WebClient 専用のものである。
7. 親プロセス(Webmaster によって実行された WebClient)の場合、ホスト名、IP アドレス、ポート番号のチェックをする。
8. プロキシサーバを使用する場合、サーバの指定を変更する。
9. アドレス解決を行う。
10. 読み込む URL をファイルリストまたはコマンドラインから parse_file_list( )を使い解析し、必要ならばその個々の URL に対して「重み(読み込み頻度)」を指定する。トップページなど、参照のされやすいファイルはその「重み」を大きくすることで、現実の環境に近づけることができる。
11. 親プロセスの場合、測定を開始する準備ができたので、Webmaster にその旨を伝る。
12. 設定された数だけ、子プロセス ClientThread( )を作成、実行する。
13. すべての子プロセスが終了するのを待ち、終了したら WebClient を終了する。

2-7 ClientThread( )

関数 ClientThread( )を説明する。
ClientThread( )は WebClient に実行される、ページ読み込み、時間計測を行う関数である。

1. 計測結果を保存する変数を定義する。
2. オプションに対する設定と変数定義を行う。
3. Webmaster に実行された WebClient の場合、webmaster に準備ができたことを知らせるため、webmaster に接続する。WIN32 ではセマフォで、UNIX ではアラームで連絡する。
4. 時間の計測を開始する。
5. 計測時間終了または規定ループ回数まで、makeload( )関数によってページ読み込みを行う。
6. 時間計測を終了する。

7. Webmaster に実行された WebClient の場合、Webmaster から「GO」メッセージを受け取り次第、Webmaster に計測結果を送信する。
8. Webmaster によって開始された WebClient でない場合、結果を標準出力から出力する。
9. WIN32 の場合、親プロセスに終了したことを伝える。
10. ClientThread( )を終了する。

2-8　コマンドラインオプション

Webmaster と WebClient のオプションは異なる。ただし、通常 WebClient のコマンドラインは Webmaster によって作成されるため、考慮する必要はない。
また WebClient にはオプションが定義されていても、Webmaster から引き渡されないため、使用しないオプションも存在する。

- webmaster 用

必須オプション

-C webclient_path	webclient プログラムへのパス
-f config_file_name	クライアント指定用コンフィグファイル

どちらか一方を指定する

-l roops	UIL を検索するためのループ回数
-t run_time	テストの接続時間(分)

準必須オプション(コマンドラインで指定しない場合、別の方法で指定する必要)

-w webserver URL [URL ...]	Web サーバの URL
-u URL_file_name	webmaster 用 URL ファイルリスト
-U child_URL_file_name	webclient 用 URL ファイルリスト

通常オプション

-a	すべてのクライアントの時間情報を出力
-d	デバッグ文(D_PRINTF)を有効にする
-p port_num	web サーバのポートが 80 でないときのポート番号指定
-s	webclient が得たデータを /tmp/webstone.data.* に保存する
-v	冗長モード
-D debug_file	デバッグファイル指定
-P proxy_server_name	処理にプロキシサーバを使う
-W	web サーバのアドレスがコンフィグファイルにあるとき
-R	すべてのトランザクションを記録する
-S	乱数発生に固定ランダムシードを使用する
-T	トレース文(TRACE)を有効にする

SSL をサポートするとき

-V SSLv3|SSLv23|SSLv2	SSL バージョン
-c cipher	優先 SSL 暗号
-m on|off	クライアントキャッシュモード ON/OFF

隠しオプション

-M network_mask	ネットマスク指定
-X	リモート実行なし

隠し SSL オプション

-r ssl_handshake	SSL ハンドシェイク
-x ssl_mix	SSL mix
-y ssl_port_num	SSL ポート番号指定

- webclient 用

必須オプション

-C masterhost:port	webmaster の動作しているホスト名または IP アドレス ：ポート番号

どちらか一方を指定する

-l roops	UIL を検索するためのループ回数
-t run_time	テストの接続時間(分)

準必須オプション(コマンドラインで指定しない場合、別の方法で指定する必要)

-w webserver URL [URL ...]	Web サーバの URL
-u URL_file_name	webclient 用 URL ファイルリスト
-U URL_file_name	webclient 用 URL ファイルリスト(webmaster からの引渡しがないため使用不可)

通常オプション

-d	デバッグ文(D_PRINTF)を有効にする
-p port_num	web サーバのポートが 80 でないときのポート番号指定
-v	冗長モード(webmaster からの引渡しがないため使用不可)
-D debug_file	デバッグファイル指定
-P proxy_server_name	処理にプロキシサーバを使う
-R	すべてのトランザクションを記録する
-S random_seed	乱数発生に固定ランダムシードを使用する
-T	トレース文(TRACE)を有効にする

隠しオプション

-s	webclient が得たデータを /tmp/webstone.data.* に保存する

隠し SSL オプション(SSL はすべて隠しオプション)

-C ssl_cipher	SSL 暗号
-m on|off	クライアントキャッシュモード ON/OFF
-r ssl_handshake	SSL ハンドシェイク
-V SSLv3|SSLv23|SSLv2	SSL バージョン
-x ssl_mix	SSL mix
-y ssl_port_num	SSL ポート番号指定

2-9　PrintResults( )

PrintResults( )関数は以下の結果を出力する。

- 各クライアント情報を出力する場合

各クライアント(ClientThread)情報(平均、標準偏差、最小値、最大値)
　接続時間[秒]
　応答時間[秒]
　応答サイズ[bytes]
　body バイトサイズ[bytes]
　総バイトサイズ(ヘッダ＋body)[bytes]
　平均スループット／接続[bytes／秒]

- テスト時間を指定した場合

テスト時間[分]
サーバコネクション率[接続／秒]
サーバエラー率[エラー／秒]
サーバスループット[Mbit／秒]
最小負荷係数(総応答時間／テスト時間[ms])

- 必ず出力されるもの

総 ClientThread 数
平均応答時間(秒)
エラーレベル(エラー数*100／接続)
クライアントスループット平均[Mbit／秒]
合計クライアント応答時間[秒]
総読み込みページ数
合計クライアント(ClientThread)情報(平均、標準偏差、最小値、最大値)
　接続時間[秒]
　応答時間[秒]
　応答サイズ[bytes]
　body バイトサイズ[bytes]
　総バイトサイズ(ヘッダ＋body)[bytes]
　平均スループット／接続[bytes／秒]

2-10　データフロー

ここでは計測データの流れを示す。

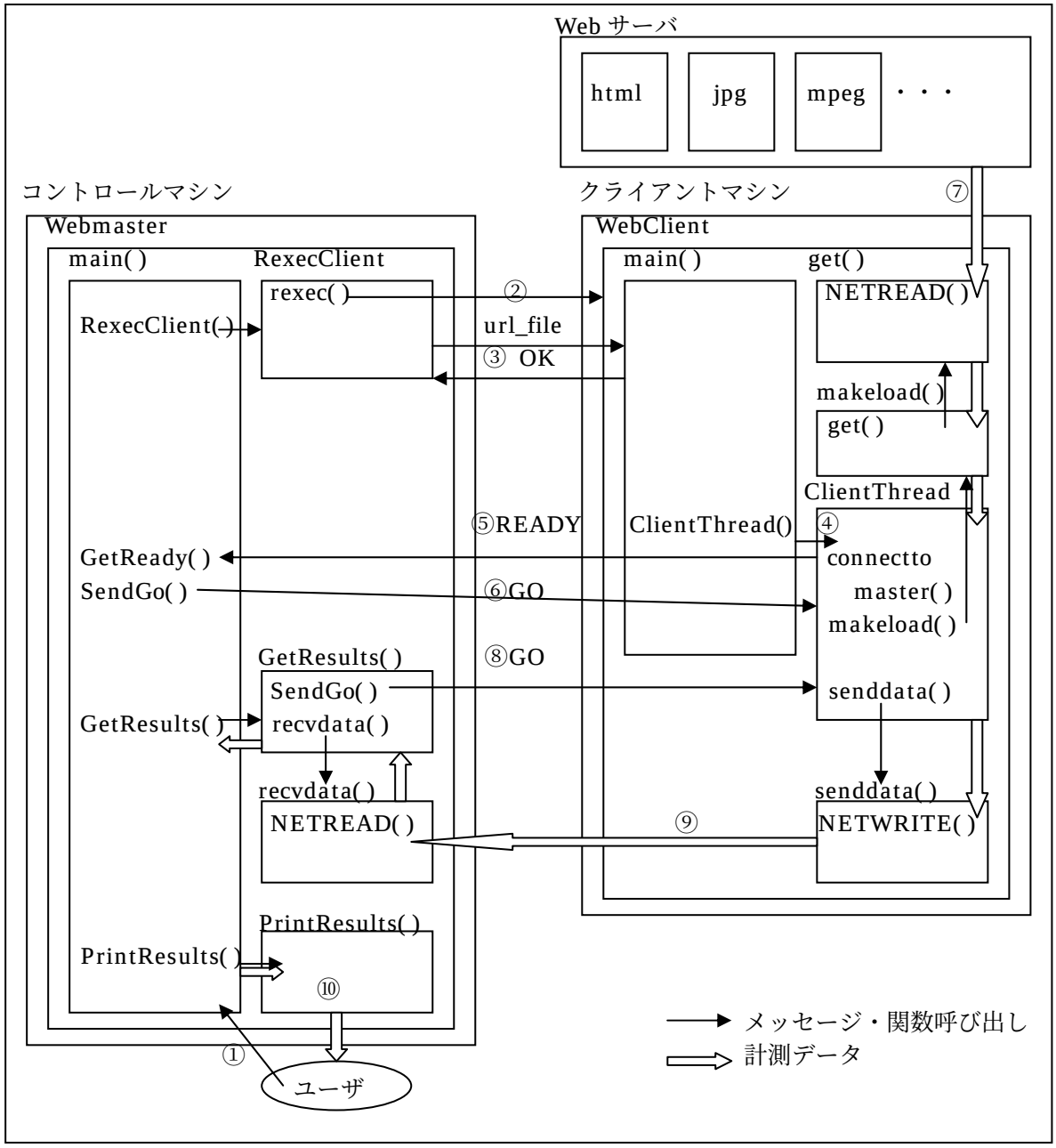

図 2.2 主なデータフロー

● 流れの解説
(番号は図 2.2 の番号に対応)

① ユーザは Prel またはコマンドラインから Webmaster を実行する。
② Webmaster は WebClient を実行し、ファイルリストを送信する。
③ WebClient は正常に開始すると Webmaster に「OK」メッセージを送信する。
④ WebClient は子プロセス ClientThread( )を実行する。

⑤ ClientTherad( )は Webmaster との接続を確認し、「READY」メッセージを送信する。
⑥ Webmaster はそれに対し、「GO」メッセージを送信し、ClientThread( )にベンチマークの開始を伝える。
⑦ ClientThread( )はメッセージを受け取ると制限時間が来るまで Web サーバからのページ読み込みを行う。
⑧ 制限時間が来ると Webmaster は ClientThread( )に「GO」メッセージを送信し、計測結果を送信するように伝え、受信準備をする。
⑨ ClientThread( )はベンチマーク結果を送信し終了する。ClientThread( )がすべて終了すると親プロセスである WebClient は終了する。
⑩ Webmaster は ClientThread( )からすべての結果を受信すると結果を集計し、ユーザにそれを返す。

2-11　バグ

今回の解析で 2 個のバグが発見された。ここにそれを記す。

- bench.c (オリジナル版 181 行目、解析版 206 行目)

double_to_text( )内
誤)
```
181     sprintf(double_as_text, "%17.01f¥t", the_double);
                                          ^
```
正)
```
181     sprintf(double_as_text, "%17.0lf¥t", the_double);
                                          ^
```
問題点
　double 型に変換した変数を出力する部分であるため、変換仕様は%lf でなければなりません。

- webclient.c(オリジナル版 730 行目、解析版 880 行目)

usage( )内
誤)
```
730     returnerr("Maximum of %d files on the command line.¥n");
                                                          ^
```
正)
```
730     returnerr("Maximum of %d files on the command line.¥n",MAXNUMOFFILES);
                                                          ^
```
問題点
　%d に対する変数の指定がありません。

第３章　　HTML リンク抽出機能の拡張

３-１　開発動機

WebStone 2.5 では HTML ファイルをバイナリとしてしか認識しないため、そのページのコンテンツは全く無視されている。このため本来 HTML ファイルであればそのページを表示するために必要な画像などのコンテンツを同時に読み込むが、現バージョンではこれを行わない。このままでは実際の Web ページからの読み込みを完全にシミュレートすることにはならないため、本研究では HTML ファイルを解析しその Web ページに含まれるコンテンツを同時に読み込めるようにした。またフレーム構造の場合、HTML ファイルを再帰的に読み、解析することで実際にブラウザで表示するときと同じデータ転送量を再現した。

３-２　開発環境および実験環境について

本研究で使用したマシンスペック、言語及びＯＳは

・ハードウェア
CPU
AMD Athlon(tm) XP 1500+ (1.33GHz)
メモリ
256MB DDR-RAM

・言語
Microsoft Visual C++ 6.0 Professional

・ＯＳ
Microsoft Windows XP Professional

以上である。対象ＯＳに Windows を選択した理由は一般に最も普及しているＯＳいう理由からである。

３-３　プログラム

HTML ファイルの読み込みに再帰性を持たせ、Web ページ内のコンテンツを同時に読み込めるようにする関数を作成した。本プログラムは Web サーバから読み込むファイルを指定する makeload( )関数に追加する。

関数　searchurl( )
仕様
- HTML4.0.1 規格基準
- スタイルシート未対応
- フレーム構造対応
- .htm .html ファイルに対して解析を実行
- 正常終了した場合１を、エラー終了の場合は０を返す

詳細

HTML ファイルを解析する際、高速検索を行うため１度メモリ内にロードする。その際、コメント文は除去し、無駄な検索を省く。

次に</HEAD>までを読み飛ばす。これはコンテンツはすべて<BODY> ~ </BODY>間に記述されるためである。

BODY の解析では本来タグを検索すべきであるが、本プログラムでは URL を指定する属性である BACKGROUND="*URL*"、 SRC="*URL*"、 DATA="*URL*" のみを検索し URL を抽出している。これはマッチングの個数を減らし、全体の検索スピードを上げるためである。また、検索開始までにコメント文は除かれているためコメント内のこれらの属性は検索されることはない。

URL を発見した場合、その URL が絶対パスであるか、相対パスであるかを調べ URL を修正する。発見後すぐにそのファイルを読み込み、この関数を再帰呼び出しする。

現バージョンでは同じ URL が発見された場合でも、そのファイルを再読み込みする。

ちなみに拡張子、タグ、属性は大文字、小文字どちらでも検索可能である。

関数 searchhtm( )

仕様

- 拡張子が htm html なら１を、そうでないなら０を返す
- 大文字、小文字どちらでも検出可能

３-４ 動作テスト

今回作成したプログラムの性能評価を行った。

本実験では通信速度によらない純粋な解析速度を測定するため、実際にある Web ページを自身のハードディスクに保存し、その HTML ファイルを解析するのにかかる時間を計測した。また Web コンテンツの読み込みは行わないがフレームについては読み込む。

３-４-１ 測定対象

以下が今回測定した Web ページである。

1. www.mindcraft.com/index.html
2. mindcraft.com/index.html
3. www.shimane-u.ac.jp/index.html
4. www.cis.shimane-u.ac.jp/index.html
5. www.cis.shimane-u.ac.jp/~tanaka/index.html
   /c1.html
   /c2.html
   /c3.html
   /c4.html

１～４はフレームを使用していないページ、５はフレームを使用したページである。

3-4-2　測定

測定では同ページに対する測定を１１回行い、１回目を除く２回目から１１回目までの計１０回の平均を求め比較、検証した。

表 3.1 測定結果

URL	file	file size	commennt size	non-comme size	header size	body size	links	平均時間 (ms)
www.mindcraft.com	index.html	14175	4672	9266	532	8730	13	24.9
mindcraft.com	index.html	21251	5476	15368	849	14519	15	23.2
www.shimane-u.ac.jp	index.html	9879	3030	6711	130	6581	57	42.1
www.cis.shimane-u.ac.jp	index.html	3939	513	3314	226	3088	12	23.7
www.cis.shimane-u.ac.jp/~tanaka/	index.html	461	0	409	63	346	4	
	c1.html	396	0	391	38	353	0	
	c2.html	495	0	484	63	421	0	
	c3.html	614	141	416	65	351	0	
	c4.html	261	0	230	47	183	1	
	合計	2227	141	1930	276	1654	5	32.7

表 3.1 より処理時間は、本条件ではそのページの BODY サイズによらないことがわかる。それに比べ、リンク数とフレーム数が大きいものについては処理時間がかかっていることがわかる。この結果を基に検証を行った。

3-5　検証

測定結果に基づき、さまざまな条件の HTML ファイルを生成しそれに対して測定を行った。

検証１　BODY サイズによる変化

テストファイル
　ヘッダサイズ　150 bytes
　リンク数　10
　フレーム構造未使用

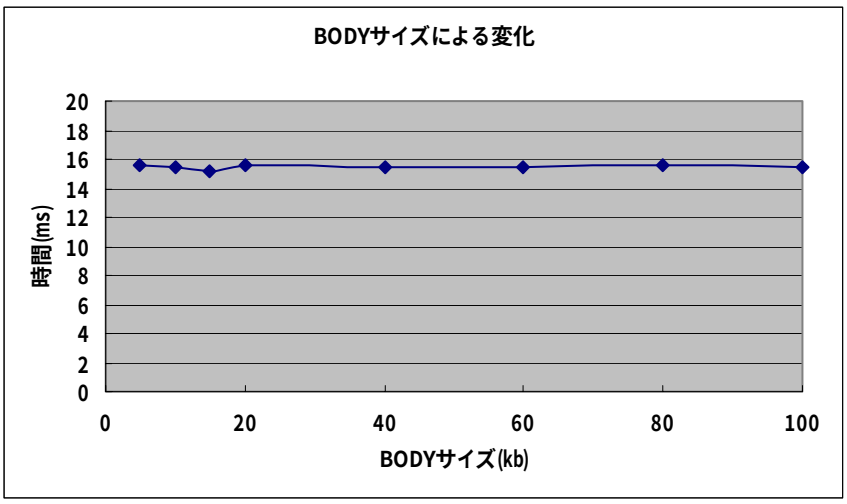

図 3.1 BODY サイズによる変化

案の定、BODY サイズによる変化は見られなかった。これ以上のサイズの場合は不明であるが、通常 100KB を超える HTML ファイルはありえないため、これ以上の検証は必要ないと考える。
BODY サイズによる変化が見られないことから検索速度については問題ないといえる。よって検索効率の向上はこれ以上必要ない。

検証２　リンク数による変化

テストファイル
　ヘッダサイズ　150 bytes
　BODY サイズ　40 KB
フレーム構造未使用

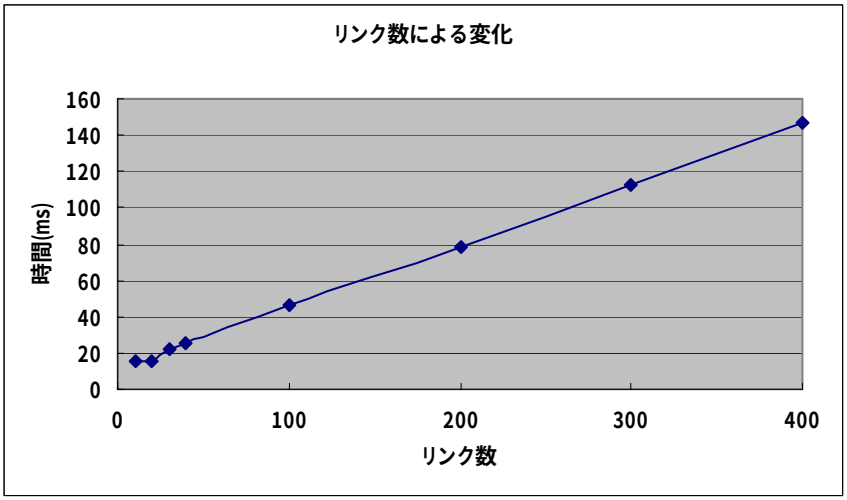

図 3.2 リンク数による変化

リンク数の変化については比例関係が認められた。これは関数 searchurl( )が URL を取得した際、関数 searchhtm( )を呼び出すためと考えられる。searchhtm( )の処理には時間はかからないが、関数のロードに時間がかかっているものと思われる。この点を解消するには serchhtm( )を searchurl( )内に関数展開する方法が考えられる。しかしプログラムが見難くなってしまうため、今回は割合した。

検証３　フレーム数による変化

テストファイル
　ヘッダサイズ　150 bytes
　各 BODY サイズ　40KB／フレーム数
　合計 BODY サイズ　40KB
　各リンク数　40／フレーム数
　合計リンク数　40
　フレーム階層は１

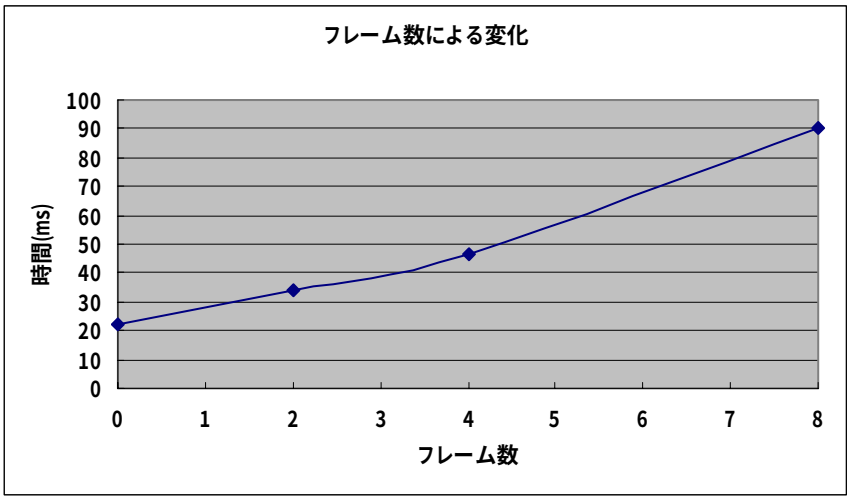

図 3.3 フレーム数による変化

フレーム数(ページ分割数)にも比例関係が見られる。各ファイルの BODY サイズの合計とリンク数の合計を一定にしているため、この変化に影響を与えたのはフレーム数であると言える。フレームの場合、searchurl( )を再帰呼び出しするので、この変化は当然と言える。フレームについては４分割までが普通であるので、これ以上の分割数の検証は必要ないと思われる。

検証４　フレーム構造の深さによる変化

テストファイル
　ヘッダサイズ　150bytes
　合計フレーム数　8
　各ページ BODY サイズ　5KB
　合計 BODY サイズ　5*8 = 40KB
　各ページリンク数　10
　合計リンク数　10*8 = 80

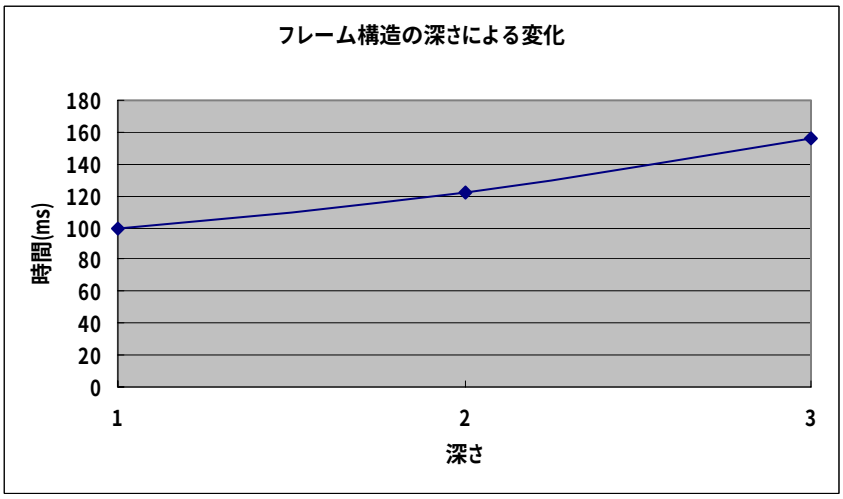

図 3.4 フレーム構造の深さによる変化

フレーム階層にも比例関係が見られる。各ファイルの BODY サイズの合計、リンク数の合計そしてフレーム数の合計を一定にしているため、この変化に影響を与えたのはフレーム階層であると言える。通常フレーム分割の階層は１であるので、これ以上の階層の検証は必要ないと思われる。

第４章　　まとめ

４-１　まとめ

本研究によって、より現実に即した環境でのサーバの性能を測定できるようになった。
またネットワークプログラミングについて理解を深めることができた。

４-２　今後の課題と展望

今回の拡張によって HTML の解析にかかるオーバーヘッドがわずかながら発生し、正確なスループットが測定できなくなったため、オーバーヘッドを減らし、さらに正確な測定ができるようにする必要がある。
また Mindcraft 社が公言しているように以下の点については改善の余地がある。

- SSL のサポート
- POST のサポート
- HTTP1.1 のサポート
- Cookie のサポート
- データバスアクセスを伴う動的作業負荷
- 認証
- HTTP1.0 keep-alive のサポート
- 異なったヘッダを送信する機能
- 長い URL のサポート
- さらに多くのシステム用のコンパイル前の実行ファイルの作成

謝辞

本研究を御指導して頂いた田中教授に心から感謝致します。
また、この１年共に頑張った本研究室の平田純一さん、貫目洋一さん、辰己圭介さん、驛範之君、森岡慶彦君に感謝致します。
本研究の著作権を田中章司郎教授に譲渡致します。

参考資料・参考文献

1）Mindcraft Inc.　http://www.mindcraft.com/

2）(株)アンク, 「ＨＴＭＬタグ辞典　第４版」, 株式会社 翔泳社, 439pp, 2001

3）B.W.カーニハン／D.M.リッチー　石田晴久 訳,
「プログラミング言語Ｃ　第２版　ANSI 規格準拠」, 共立出版株式会社, 343pp, 1989

4）FreeBSD Ports: Benchmarks　http://www.freebsd.org/ja/ports/benchmarks.html

5）http://crypto.stanford.edu/~nagendra/projects/WebStone/