Web サイト評価のための多端末シミュレータ（MTSLW）

A Multi Terminal Simulator on a Linux to evaluate Web Sites

信太晃司， 伊藤剛志， 梅原和芳， 三條光輝， 渡邊高明， 杉村 康

【あらまし】 現在，インターネット上の Web サーバとして, Linux と Apache が普及しつつある．それらのサーバの処理能力評価は，従来，性能指標プログラム (ベンチマーク) を走行させて，得られた結果を個々のシステムで比較するという方法で行われている．しかし，ベンチマークによる評価は実際のシステムの状態を反映していないとの指摘もある．当研究では，処理能力評価を実際の呼データを使用して客観的に評価する多端末シミュレータ (MTS [1]) の手法に着目し， Linux と Apache 上での MTS である MTSLW (A Multi Terminal Simulator on a Linux to evaluate Web Sites) を提案する．

1. はじめに

現在，インターネット上の Web サーバとして，ソースプログラムが開放 (オープンソース) され，無料で，且つ，改造が自由にできる (フリーな) Linux と Apache の普及が急速に拡大しつつある.[2] それらによって実現された Web サーバの処理能力評価は，従来，性能指標プログラム(ベンチマーク)を走行させて，それによって得られた 1 秒(単位時間)当たりの処理された要求 (トランザクション) 数や，単位時間当たりのデータ転送量 (スループット[Mbps]) を，被評価システム間で比較するという方法で行われている.[3]

しかしながら，ベンチマークによる評価は，実際のシステム におけるトランザクションの状態を反映していないとの指摘もある.[4][5]　例えば,文献[5]におけるドライストーンベンチマーク[6]は, OS の動作特性を反映してその命令頻度が決められており，OS 全体の特性を疑似した筈であったが，実 OS 全体の特性を反映したトレースデータを元に評価した結果と比較した場合，トランザクション数/秒の値において 2 倍近く違う結果となったとの報告がある.[5]

それらの結果と同様な現象が Web サーバにおける評価において起こらないかを考えた場合，どの様なトランザクションが，どの様な頻度と順序で,システムに与えられるかがキーポイントと考えられる． なぜならば，例えば Linux においては，ファイルの内容をメモリ上にキャッシュ (バッファキャッシュ[7]) することによって，処理能力を向上する仕掛けを内蔵しており，その処理能力は，現実のシステムで発生しているトランザクションの頻度及び,順序の状態に大きく依存する,と予想されるからである．

従って，実際のシステムにおけるトランザクションの上記の状態を反映した処理能力評価が，非常に重要である．

一方，実際のシステムにおけるトランザクションの状態を収集し，その収集データを元にシステムにトランザクションを与えるシステムとしては，呼データ収集プログラムと負荷発生プログラム(MTS)による手法が,既に存在する. [1]

しかしながら，

(1) それらは,オープンソースでなく，又,フリーでもないため，我々が Apache 等の評価に適用することはできない．更に，

(2) それらは，呼データ収集を，標準化がされていない通信プロトコルの下位層で行うため，複雑であり，又，

(3) そこで用いられているデータベース (DB) は，その内容の更新のためのトランザクション処理を含むため，呼データを収集する前に，DB の内容等のバックアップを取り，MTS の試験の直前に，それらの内容を被システム上に反映することが必要なため，それらの開発工数や試験工数は，決して小さなものでは無かったと予想される．

当研究においては，OS 並びにサーバとして，Linux と Apache を用いることにより，上記(1)の問題点を解決する． 又，呼データ収集は，「要求と,それに対する応答」という単純なやりとりを基本として標準化がかなり進んでいる Hyper Text Transfer Protocol (HTTP) に基づき，通信プロトコルより一つ上のアプリケーション (AP) レベルで呼データを収集することにより，上記 (2) の問題点を解決する．

又，Web サーバの提供サービス形態は, (A) 入力フォーム等を含まない単純な Web サイトと，(B) 入力フォームや DB 等による,サーバ内情報更新を含む,高度な Web サイトに大別できるが，当研究では大半の公開 Web サーバの形態に採用されている上記 (A) のような単純な Web サイトに的を絞ることにより，当面，上記問題点 (3) を回避する．

以上を前提として， Linux と Apache 上でのトランザクションの呼データ収集プログラムと負荷発生プログラム及び補助プログラムからなる,MTSLW を以下に提案する.

2. MTSLW の構成

MTSLW は, Linux 上の Apache において実際に運用中の Web サーバ上で, そのサーバに実際に到着する呼のデータを収集する役目を担う呼データ収集プログラム｛SDCP_MTSLW; Sugimura Lab's Data Collecting Program for an MTSLW:(以下 SDCP と略記) Fig.1 参照｝と，主に,そこで収集された呼データを使用して，任意の OS 上の Web サーバに, 実際の運用状況の疑似呼を提供する負荷発生プログラム ｛SMTSL_MTSLW; Sugimura Lab's MTS on a Linux for an MTSLW: (以下 SMTSL と略記) Fig2. 参照｝から構成される.

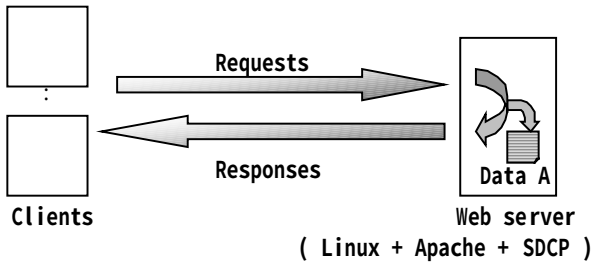

Fig.1 A concept diagram of the SDCP.

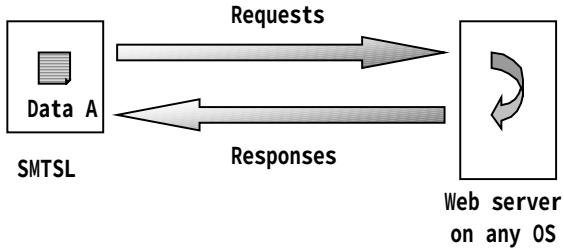

Fig.2 A concept diagram of the SMTSL.

尚, Fig.1 中の Data A は SDCP が作成する呼データファイルである.

以下, 第３章で SDCP について， 第４章で SMTSL について，それぞれ述べる.

3. 呼データ収集プログラム（SDCP)

3.1 呼データ収集個所の特定

Linux 上の Web サーバの httpd プログラム (Apache) を解析し，データの送受信を行っている箇所を特定するために,スーパートレーサ (STDB[8]) を用いて，クライアントからの要求があったときのサーバ内のトレースを行った. この結果 Fig.3 の様に，Apache のソース内の accept 関数で接続要求を受付け，ap_read 関数,ap_write 関数等の中からバッファの読み書きを行うシステムコールが呼び出されていることが判り，そこに呼データ収集のロジックを追加した.

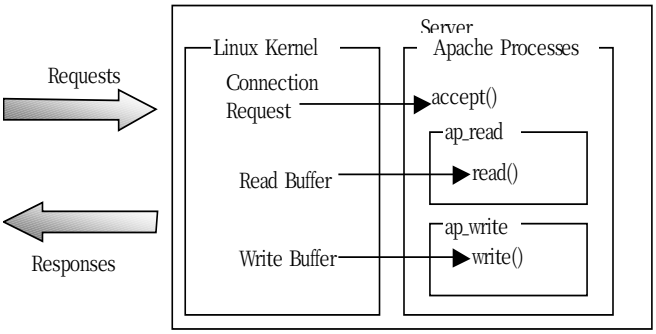

Fig.3 An example of trace results.

3.2 SDCP の呼データファイル管理機構

SDCP を動作させる Web サーバは，通常,24 時間無停止で運用される.従って, SDCP による呼データの収集は, Web サービス提供中に，開始／終了／呼データファイルの切り替えを実現する必要がある.

一方，Apache は，Linux 上の Web サーバとして,複数のシステムプロセス httpd (デーモン) として動作する. そのため, すべての httpd プロセスにおいて，呼データファイルに対して, (A) ファイルのオープンを含む呼データ収集の開始, (B) ファイルのクローズを含む呼データ収集の終了, (C) ファイルのオープンとクローズを含む呼データファイルの切換，を一貫して行うファイル管理機構が必要である.

又，それらの実現機構は，Apache の今後の発展への足かせとならないために,又，SDCP の開発工数の省力化を図るためにも，Apache のユーザインタフェースとは独立した機構とすることが望ましい.

それらの複数プロセス間の相互通信は，一般には，プロセス間通信とよばれ，シグナル，パイプ，セマフォや共用メモリ等を使用して実現されることが多い. しかしながら,それらにおいては，例えば,共用メモリにおいては, shmget の第一引数 key や，セマフォにおいては semget の第一引数 key [9] に見られるように，システム一意な ID を付与して，それらの ID をプロセス間で共用するという手法が用いられる. 従って,それらの ID が，既に そのシステムで使用されている場合は，プログラムを変更しなければならず，不特定多数のシステムへの導入においては，足かせとなる可能性がある.

そこで,SDCP においては,オペレーティングシステムのプロセス間通信機能の内,シグナルのみを用い，その他の通信機構は，既存の STDB[8]のシステムコールを拡張して実現することにより,将来への足かせの極小化を図った.

具体的には，SDCP におけるファイル管理機構は，(a)コンソールから起動されるプログラム群 (以下, SDCP ユーザインタフェースと呼称)，(b) 上記(a)からシグナル(シグナル名=SIGUSR2)により起動される Apache 内の新シグナルハンドラ，及び(c) 上記(a)及び(b)からコールされる新たなシステムコール群の三つにより，実現する．そこでは，SDCP の動作状況等の情報は，カーネル内と Apache 内に設け，それらの情報をシステムコール経由で参照等をすることにより，一貫した,矛盾が無い管理を実現する．

尚，呼データファイルは，複数あるプロセス毎に作成する手法と，一つにまとめる手法の二つが考えられるが，書込み時のデバイスのヘッドの移動を最低限に押さえる(シーク時間を最少にして，アクセスを高速にする)ために，一つのファイルに，追加モードで，複数プロセスから，並行して,順次書込む手法を採用した.詳細について,以下 3.2.1～3.2.3 で述べる．

3.2.1 SDCP 用新システムコール

新しいシステムコールは，現在開発中の Linux との重複を避けるために，新規のシステムコールを用意するのではなく，STDB[8]のシステムコール stdb（番号は 255 番，sched.c 内）に機能を追加するという方法をとった．

該システムコールの第一引数によってその機能を区別する．1から3はすでに使用されているので，10 以上の値を用いる．

stdb(10)はファイルオープンを含む SDCP の開始,

stdb(11)はファイルクローズを含む SDCP の終了,

stdb(12)はファイルのクローズ・オープンを含む呼データファイルの切り替えを行うもので，SDCP のユーザインタフェース部からコールされる．

stdb(15)～(18)と(20)は,下記(D)～(H)に詳細を示すように,その他の情報の伝達や制御を行う．尚，シグナルハンドラからコールされるものには,下記の説明の最後に(*)を付記した．

以下,それぞれの機能について述べる．

(A) 第一引数が 10(開始)のシステムコールの処理では，カーネル内の制御表を参照することにより，プロセスをすべて調べ，プロセス名が httpd で，子タスクを持たないプロセスのプロセス識別子(PID)の一覧表を(カーネル内部の新設エリアである) pid[ ]に設定し，プロセス情報のアドレスの一覧を addr[ ]に設定し, sdcp_httpd_n に httpd プロセスの個数を設定し，制御変数 sdcp_sig_comp（シグナル処理完了数）に 0 を設定し，制御変数 sdcp_status に 2(オープン処理要求中)を設定し，第二引数が示す SDCP 用の呼データファイル名をカーネル内部のエリアに保存し,上記 pid[ ]内の全てのプロセスに対して，kill (pid, SIGUSR2);を発行した後,リターンする．

(B) 第一引数が 11(終了)のシステムコールの処理では，制御変数 sdcp_sig_comp(シグナル処理完了数)に 0 を設定し，制御変数 sdcp_status に 3(クローズ処理要求中)を設定し，kill (pid, SIGUSR2)を，上記(A)の pid[ ]の全ての pid に対して発行し，上記(A)で保存したファイル名をクリアして,リターンする.但し,この処理での第二引数はダミーである．

(C) 第一引数が 12(切り替え)のシステムコールの処理では，制御変数 sdcp_sig_comp (シグナル処理完了数)に 0 を設定し，制御変数 sdcp_status を 4(チェンジ処理要求中)に設定し，第二引数が示すファイル名をカーネル内部エリアにオーバーライトし，上記(A)の pid[ ]の全ての pid に対して kill (pid, SIGUSR2)を発行した後,リターンする．

(D) 第一引数が 15 の場合，第二引数が示すエリアに制御変数 sdcp_status を設定し,第三引数が指すエリアに,(カーネル内部に保存されている)ファイル名を設定し,リターンする.(*)

(E) 第一引数が 16 の場合，第二引数の値に制御変数 sdcp_status を変更する．第三引数はダミーである．

(F) 第一引数が 17 の場合，シグナル完了数 sdcp_sig_comp に 1 を加算する.第二，第三引数はダミーである.(*)

(G) 第一引数が 18 の場合，第二引数の指すエリアにシグナル完了数 sdcp_sig_comp を設定し，第三引数の指すエリアにプロセスの個数 sdcp_http_n を設定する．

(H) 第一引数が 20 の場合，第二引数の指すエリアに時間 jiffies を設定し, 第三引数の指すエリアに現在実行中の pid を設定する.(*)

尚,第一引数が 10（開始）の場合，sdcp_status が 0（SDCP が停止中）でなければ，第三引数で与えられるアドレスにエラーコードを格納する.第一引数が 11（終了）又は 12（切り替え）の場合，sdcp_status が 1（SDCP が動作中）でなければ，第三引数で与えられるアドレスにエラーコードを格納する．

3.2.2 Apache 内の新シグナルハンドラ

システムコール stdb の中から発行された SIGUSR2 シグナルを httpd プロセスが受け取ると(あらかじめ sigaction によって登録されている)シグナルハンドラから sig_handl_sdcp が呼び出され，各プロセスのシグナルハンドラはファイルオープン，クローズ，チェンジの動作を行う.これらの動作は,制御変数 sdcp_status の値によって区別する．

まず，シグナルハンドラは，sdcp_status の値を調べるため stdb(15)を呼び出す. sdcp_status の値が 2(オープン処理要求中)なら以下の(1)の処理を，3(クローズ処理要求中)なら(2)の処理を，4(チェンジ処理要求中)なら(3)の処理をそれぞれ行う．

(1) 新シグナルハンドラの呼データファイルオープン処理: stdb(15)で取得済みの呼データファイル名に対してファイルオープンをし，得られたファイルポインタをsdcp_fpに保存する．その後stdb(16)を呼び出すことにより,(ユーザインタフェース部に) 処理の完了を告げ，その後 OS へリターンする．尚，Apache に埋め込まれた SDCP は，sdcp_fp が 0 でない場合，所定の箇所で送受信データを sdcp_fp のファイルに書込む.

(2) 新シグナルハンドラの呼データファイルクローズ処理: 呼データファイル sdcp_fp に対してファイルクローズを行い，sdcp_fp には 0 を設定する.stdb(16)を呼び出し，処理の完了を告げ，終了する.

(3) 新シグナルハンドラの呼データファイル切換処理: sdcp_fp に対してファイルクローズを行い，stdb(15)で取得済みの呼データファイルに対してファイルオープンを行い，ファイルポインタを sdcp_fp に設定する．stdb(16)を呼び出し，処理の完了を告げ，終了する.

3.2.3 SDCP ユーザインタフェース

SDCP におけるユーザインタフェースは，sdcp_open，sdcp_close，及び，sdcp_changeの三つのプログラムからなる．

sdcp_open はファイルのオープンを含む呼データ収集の開始，sdcp_close はファイルのクローズを含む呼データ収集の終了，sdcp_change はファイルのクローズ，オープンを含む呼データファイルの切り替えを行う.

ファイル名はユーザインタフェース内で，作成時のタイムスタンプを用いて sdcp_yymmddhhmmss（年の下二桁：yy，月：mm，日：dd，時：hh，分：mm，秒：ss）とし，Apache とは別の HDD 上に格納してアクセス速度向上を可能とするため，/root/sdcp/data/の配下に置く．該ディレクトリの所有者名は nobody，アクセス権は 744 である． 以下，sdcp_open， sdcp_close， sdcp_change 及びエラー処理のそれぞれについて述べる.

(1) sdcp_open: まず,日付データから呼データファイルのファイル名を決定する．そのファイル名を第二引数に，エラー変数 err のアドレスを第三引数に設定し，stdb(10)を呼び出す.結果が正常なら一秒間 sleep し，その後 stdb(18)を呼び出し，sdcp_comp (シグナル完了数)と sdcp_httpd_n (プロセスの個数)の情報を得る.sdcp_comp が sdcp_httpd_n に等しくなければ，再度スリープする．(30 回 sleep したらエラーリターンする.) 等しければ， stdb(16) を呼び出し sdcp_status を 1 (SDCP 動作中) に設定する．ユーザインタフェースとシグナルハンドラ,及びシステムコールの関係を Fig. 5 に,状態遷移図を Fig. 6 に,それぞれ示す.

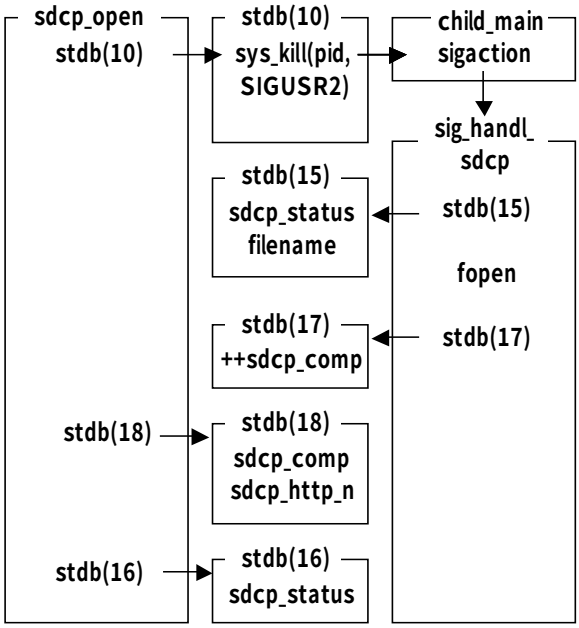

Fig.5 Open Processing.

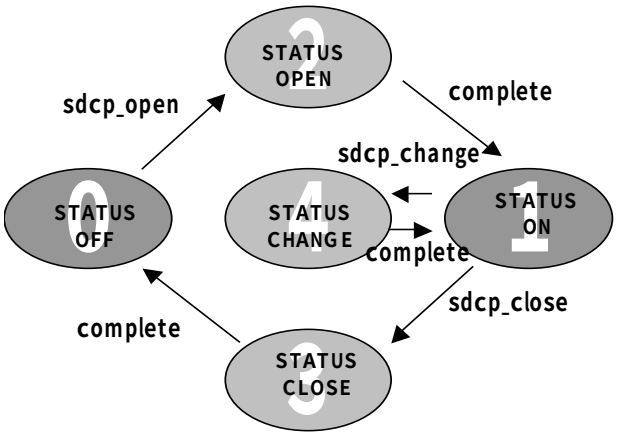

Fig. 6 SDCP's Status Transition Chart.

(2) **sdcp_close:** 第三引数にエラー変数 err のアドレスを設定し，stdb(11)を呼び出す.エラーがあればエラー処理を行う.正常なら一秒間 sleep し，その後 stdb(18)を呼び出し，sdcp_comp (シグナル完了数)と sdcp_httpd_n (プロセスの個数) の情報を得る.sdcp_comp が sdcp_httpd_n に等しくなければ，再度スリープする.(30 回 sleep したらエラーリターンする.) 等しければ， stdb(16) を呼び出し sdcp_status を 0 (SDCP 停止中) に設定する．ユーザインタフェースとシグナルハンドラ,及びシステムコールの関係を Fig. 7 に示す.

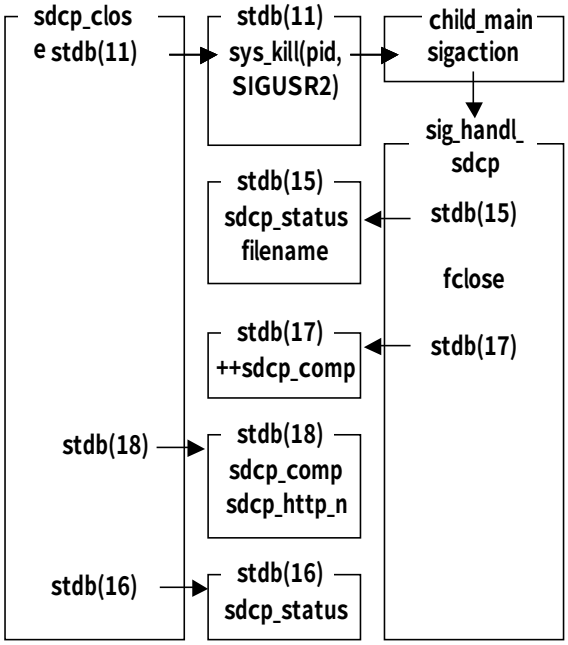

Fig.7 Close Processing.

(3) **sdcp_change:** まず,日付データからファイル名を決定する. そのファイル名を第二引数に, エラー変数 err のアドレスを第三引数に設定し, stdb(12)を呼び出す.エラーがあればエラー処理を行う.正常なら一秒間 sleep し, その後 stdb(18)を呼び出し, sdcp_comp(シグナル完了数)と sdcp_httpd_n(プロセスの個数) の情報を得る.sdcp_comp が sdcp_httpd_n に等しくなければ, 再度スリープする.(30 回 sleep したらエラーリターンする.) 等しければ, stdb(16)を呼び出し,sdcp_status を 1(SDCP 動作中)に設定する. ユーザインタフェースとシグナルハンドラ,及びシステムコールの関係を Fig. 8 に示す.

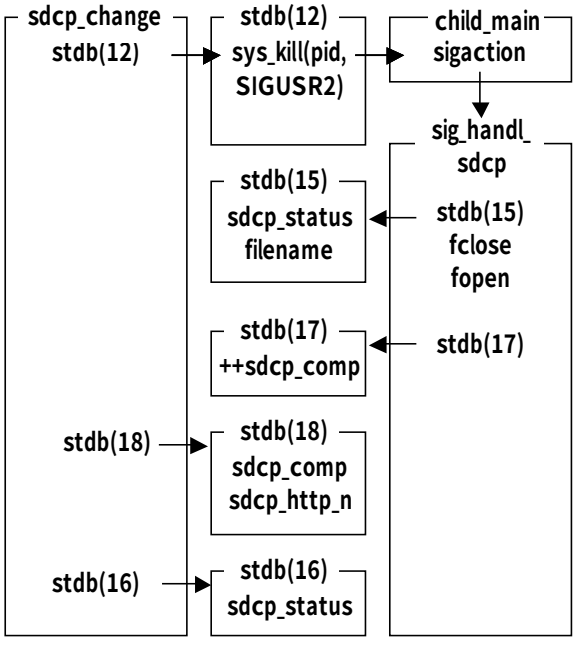

Fig.8 Change Processing.

(4) エラー処理: 各ユーザインタフェースによってコールされた stdb のシステムコールがエラーでリターンした場合は, 第 3 引数に返却されるエラーコードに従って, コンソールへ下記 List.1 のメッセージを出力する.

List.1 Error code and explanation.

値	コンソールへの出力	意味
0	Normal return	正常リターン
1	Already file open!	すでにオープン状態である
2	File isn't open yet!	オープン状態にないときにクローズ要求をした
3	File isn't open yet!	オープン状態にないときにチェンジ要求をした
4	SDCP is waiting for signal completion:OPEN	オープン処理中
5	SDCP is waiting for signal completion:CLOSE	クローズ処理中
6	SDCP is waiting for signal completion:CHANGE	チェンジ処理中
7	SDCP : sleep over!	30回sleepしても完了しない

3.3 SDCP が収集する呼データ等

SDCP が収集する呼データは, Apache の ap_accept 関数の位置で収集するものと, それ以外の場所で収集するものの 2 種類に大別でき, それらは, 以下の通りである.

ap_accept 関数:

- 応答した httpd のプロセス ID 番号
- 応答した時間 (10[ms])
- クライアント IP アドレス

ap_read, ap_write 関数:

- データを取得した関数名
- 応答した Apache のプロセス ID
- 取得した時間 (10[ms])
- 受信 or 送信データの内容.

以下,それぞれの関数の場所における SDCP の処理等について述べる.

3.3.1 ap_accept() 関数

ソースファイル http_main.c の child_main 関数内にある ap_accept のコール直後に次の処理を加えた.

戻り値 (ソケットディスクリプタの値) を, (プロセスの内部情報としての) sdcp_csd に格納し, 呼データファイルが開かれており(即ち sdcp_fp≠0), ソケット接続があれば, 時間とプロセス ID は stdb(20), クライアント IP アドレスは,accept()の第二引数(sa_client 構造体)からソケット情報取得関数[10]を使用して,それぞれ取得する.

呼データファイルへの出力は, 出力後, fflush 関数(ファイルフラッシュ)によって, 確実な書込みが行われるようにする."尚, 誤動作防止のため, ソケット接続が切断さ

れたと推測される箇所(accept の処理に行く前)で, sdcp_csd の値を 0 (ソケット接続なし) と設定する.(sdcp_csd が 0 の場合は,呼データのファイルへの書込みを行わない.)

3.3.2 ap_read() 関数

ソースファイル buff.c 内の ap_read 関数内の read システムコールが発行されている箇所の直後に以下の処理を加えた.

sdcp_csd と fb->fd(read を行うファイルディスクリプタ)が同じで, 出力ファイルが開かれている(sdcp_csd≠0)ならば, 以下の出力を行う.

- ap_read から出た情報である事を示すヘッダ
- stdb(20)で得た時間
- プロセス ID
- 入力バッファ(read の第二引数)内のデータ
- データの終わりを示すフッタ

データのサイズは read システムコールの戻り値で判断する. accept の場合と同様, 書込み後にファイルフラッシュを行う.

3.3.3 ap_write() 関数

ソースファイル buff.c 内の ap_write 関数内の write システムコールを発行している箇所の直後に以下の処理を加えた.

sdcp_csd と fb->fd(write を行うファイルディスクリプタ)が同じで, 出力ファイルが開かれているならば, 以下の出力を行う.

- ap_write から出た情報である事を示すヘッダ
- stdb(20)で得た時間
- プロセス ID
- 入力バッファ(write の第二引数)内のデータ
- データの終わりを示すフッタ

データのサイズは write システムコールの戻り値で判断できる.又,上記の accept や ap_read 関数の場合と同様, 書込み後にファイルフラッシュを行う.

尚, 当データの収集は, SMTSL での被評価システムの正常性確認のためであり, SDCP のデフォルトでは,全ての write データを収集する. そのため, デフォルトの状態で,多量のデータを write する場合は, Apache サーバマシンの HDD に高負荷を与える可能性がある. それを回避するため, オプションとして, write データの Contents 部分の一定データと全データ長を収集することを, 更に, 検討中である.

3.3.4 SDCP の呼データの出力内容例

List.2 に示す STDB のホームページに Web ブラウザ(クライアント)がアクセスしたときの SDCP の呼データの出力内容例を List. 3 及び List. 4 に示す. List.3 の最初の 3 行が, accept()関数における呼データであり, その後の 14 行が, ap_read()関数における呼データであり, List. 4 での残りの行が ap_write( )関数における呼データである. #記号が付いている部分は, 収集データの区切りとして出力されている (上記 3.2.2, 3.3.3 における) ヘッダとフッタである.

List.2 Contents of the STDB ' s homepage.

```
<html>
<head>
<title> STDB's Home Page.</title>
</head>
<body>
<big><big><big>
<center>Welcome to "A super tracer and an analyzer for analyzing <br>
detail behavior of a Linux on a Pentium family processor<br>
 (STDB)"'s Home Page. <br>
<hr size="4">
</big>
<br>
<a href="./d_text/japanese.html">Japanese</a>.<br>
<br>
<a
href="./d_text/english.html">English</a>.<br></big></big>
<br>
</CENTER>
<hr size="4">
NTT Information Sharing Platform Laboratories.<br>
</body>
</html>
```

List.3 An example of data collected by the SDCP(1/2)

```
Accept IP address : 172.16.53.242 ,pid = 563 ,jiffies = 88556 * 10 [ms]

###   The buffer that passed ap_read   ###
pid = 563, jiffies = 88558 * 10 [ms]

GET /index-e.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
Accept-Language: ja
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)
```

List.4 An example of data collected by the SDCP(2/2)

```
Host: 172.16.53.241
Connection: Keep-Alive

########################################
### The buffer that passed ap_write   ###
pid = 563 , jiffies = 88560 * 10 [ms]

HTTP/1.1 200 OK
Date: Fri, 01 Mar 2002 06:43:29 GMT
Server: Apache/1.3.12 (Unix) (Red Hat/Linux) PHP/3.0.15
mod_perl/1.21
Last-Modified: Fri, 01 Mar 2002 02:02:23 GMT
ETag: "1bcba-1f6-3c7ee12f"
Accept-Ranges: bytes
Content-Length: 502
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

########################################
```

3.3.5 SDCP の正常性確認

SDCP における呼データの収集漏れがないかをチェックするため，その試験においては tcpdump を起動し，そのデータと呼データとの比較により，正常性を確認した.

4. 負荷発生プログラム（SMTSL)とその補助プログラム (SEDIT,CHECKN)の構成

4.1 ab の流用性と SMTSL の構成

Linux には，負荷発生用のプログラムとして Apache HTTP server benchmarking tool (ab) が用意されており，そのソースコードを解析することにより，流用できる部分がないかについて,検討を行った. その結果,現状の ab には,以下の特徴があることが判明した.

1) ab は 1 種類の web アクセスを何度も繰り返すだけである.(URL は一つだけ)
2) 要求時の keepalive 指定にかかわらず,web サーバ側から keepalive がくれば,回線断を行わず,write(1 電文分)と read(1 電文分)を一組として繰り返す.
3) keepalive が来なかった場合,write と read の一組の後，サーバ側からの回線断を契機に,接続をやり直す.
4) プロセスは一つしか使用していない.
5) read と write は,select によって制御されているので，並行動作が行われているが，connect や close 中には read や write のイベントがあっても処理されない.

尚,上記の ab の特徴を踏まえた上で,SMTSL においては,以下を実現する必要がある

(A) プロセスの複数化

ab.c は 1 プロセスで複数クライアントを擬似している．そのため負荷を発生する側の CPU を 100%活用できない可能性がある．それを回避するために，SMTSL では，あらたに設ける起動パラメータ情報ファイルより起動パラメータをリードし,そのパラメータで指定されている同時接続数分のプロセスを fork することにより，connect 等を含めた並行実行を実現する.

(B) メモリ上の送受信データの極小化

SMTSL の処理能力限界を，被評価システムの処理能力限界より高くするためには，SMTSL が使用するデータは，送信を開始する前に,メモリ上に予めロードしておくことが望ましく,且つ,その量は，できるだけ少ない方がベターである.

又，SDCP は，複数のプロセス毎の呼データ等を高速に格納するために，一つのファイルに混在させて{ Fig. 9 (A) }に格納するが，SMTSL が呼データを複数のプロセスから送信して，被評価システムに高負荷を与えるためには，(a) 呼データを回線を切断せずに連続して送出する単位(セション)毎に予め整理したセション毎データを別途作成し，(b) SMTSL は，それらを用いて，逐次，空いたプロセスが，未送信セションのデータを送信することにより，処理を単純化することが望ましい.

一方，SMTSL 及び被評価システムの正常性の確認のためには，被評価システムから受信したデータの内容(Contents)を，SDCP が write したデータの Contents と比較チェックしなければならない.

しかし，それらのチェックを全て，SMTSL において，被評価システムに高負荷を与えながらオンラインで行うことは，SMTSL の送信データより，SMTSL の受信データの方が遙かに量が多いと考えられる為，比較用のデータを予めメモリ上に用意することを含め,実現は，かなり難しいと予想される.

又，比較チェックの結果，異常の発生を検知した場合，その異常のパターンは予期しないものであろうから，その異常検出の妥当性の確認には，最終的には，目視による確認が必要と考えられる.

それらを勘案した結果，SMTSL においては，その起動時の運転モードとして，上記の正常性確認に主体を置いた（ア）正常性監視モードと，被評価システムに高負荷をかけることを第一目標とした（イ）高負荷モードの二つのモードを設け，両モードにおいては,受信データ長のチェックを行うが，受信データの Contents のチェックはおこなわず，且つ,正常性監視モードにおいては，受信データを単に{Fig. 9 (D1) }のファイルに書き出すことにより，SMTSL によって予めメモリ上にロードされるセショ

ンデータ{ Fig. 9 (B1) }中には, SDCP の write データの長さと呼データファイル{Fig.9 (A) }内の位置のみを格納することを実現し，更に，目視確認を確実に行うこと主体として，詳細のチェックは，後述のオフラインで走行する正常性チェックプログラム(CHECKN)で，SMTSL の受信データと SDCP で収集された write データの Contents の比較を行うことにより，セション毎データ内に SDCP の write データの内容を含めることを不要とする.

尚，上記のセション毎のデータの整理は，オフラインで走行する SDCP データ変換プログラム(SEDIT)で行う.

これまで紹介したプログラム群と,それらの入出力ファイルの概念を Fig. 9 に示す.

以下，4.2～4.4 章で，それぞれ，SEDIT，SMTSL 及び，CHECKN について述べる.

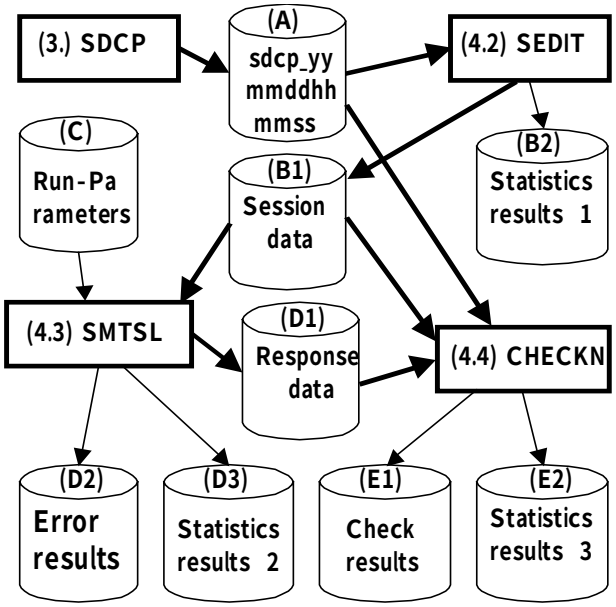

Fig. 9 I/O Files of the SMTSL.

4.2 SDCP データ変換プログラム (SEDIT)

SEDIT は，SDCP が出力した呼データファイルの内容{ Fig.9 (A) }を編集して，SMTSL が必要とするデータのみをセション毎にまとめて，セションデータファイル{ Fig.9(B1) }を生成する.

具体的には，呼データファイルの内容では，Fig. 10 の左側に示すように，SDCP によって，複数あるプロセスの PID 毎に，accept に関するデータ，read に関するデータ，write に関するデータが混在する形で格納されているが，SEDIT は，Fig.10 に示す様に，それらのデータを，同一 PID に関するセション毎のデータにまとめ，各セション毎のデータを accept した時間順にセションデータファイルに出力する.

尚，一つの read に対して， 応答データが長い場合は，複数の write が対応するが，それらは，一つの write に集約される.

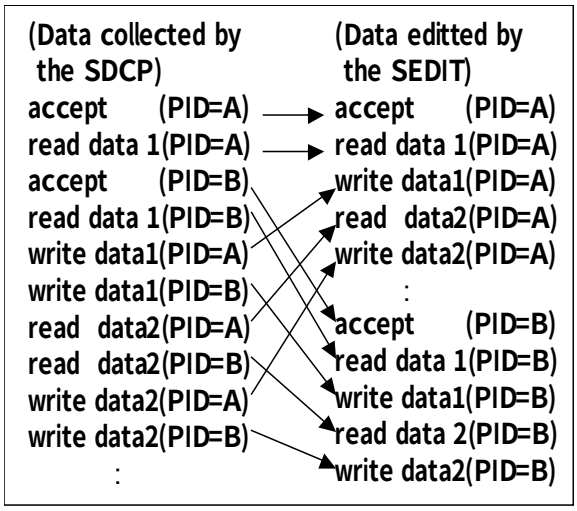

Fig. 10 An example of editting by the SEDIT.

4.3 負荷発生プログラム (SMTSL)

SMTSL は，起動されると，List. 5 に示す起動パラメータとセションデータを{ Fig. 9 (C) 及び(B1) }のファイルより読込み，起動パラメータで指定された値に基づいて,複数の子プロセスを生成し，セションデータに基づいて，複数の呼を並行して,被試験システムへの要求として連続的に与え，その応答データ長とセションデータ内の write データ長とに不一致があれば,セション番号とセション内番号を含むエラー情報を{Fig.9 (D2) }のファイルに出力し，起動パラメータ中の起動モードが正常性監視モードであれば，応答データにセション番号とセション内番号を付加して,{Fig.9 (D1) }のファイルに出力し，起動パラメータ中で指定した運転停止状態を検出すると，List. 6 に示す統計情報を{ Fig.9 (D3) }のファイルに出力したあと，プログラムを停止する.

List.5 A list of run parameters of the SMTSL.

1. 運転モード {高負荷モード or 正常性確認モード}
2. 被測定システムの IP アドレス
3. 被測定システムのポート番号
4. 同時接続数 (子プロセス数)
5. セション間隔 [秒] (デフォルトは零)
6. 運転停止条件:
   - (A) 全セションデータを処理した時点で停止
   - (B) 指定トランザクション数のレスポンス時点停止
   - (C) データ長エラーが指定数発生した時点で停止

List.6 Statistics data.

1. SMTSL の起動パラメータの全て
2. 呼発生開始時間
3. 運転終了時間
4. 全セッション数
5. セッション当り平均トランザクション数
6. 単位時間当たりのトランザクション数
7. 単位時間当たりの全/要求/応答の平均データ転送量[Mbps]
8. 平均/最大/最小応答時間
9. 1 トランザクション当たりの全/要求/応答/応答の Contents の平均, 最小, 最大転送データ長 [バイト]
9. 応答データ長不一致数
10. 応答データ長不一致率[%]

4.4 正常性チェックプログラム (CHECKN)

CHECKN は, SDCP が出力した呼データファイル{Fig. 9 (A)}と, SEDIT が出力したセションデータ{Fig. 9 (B1)}と, SMTSL が出力した応答データ{Fig. 9 (D1)}を入力して, SMTSL の受信データと SDCP の write データの Contents の比較を行い, 相違を検出した場合, そのセション番号, セション内番号, SMTSL の受信データの Contents, 及び, SDCP の write データの Contents を, チェック結果としてファイル{Fig. 9 (E1)}に出力すると共に, List. 7 に示す統計結果をファイル{Fig.9 (E2)}へ出力する.

List.7 A list of statistics results of the CHECKN

1. 全セション数
2. 不一致トランザクション数
3. 不一致トランザクション率[%]
4. close 応答トランザクション数と率[%]
5. contents length 無の応答トランザクション数と率[%]
6. クッキー有要求トランザクション数と率[%]
7. クッキー有応答トランザクション数と率[%]

5. むすび

以上の手法により, Pentium を使用した Red Hat Linux 6.2 を例にとって, 処理能力評価を実際の呼データを使用して客観的に評価するシステム MTSLW が実現可能であることを示した. 尚, 当研究は, 現在進行中であり, 将来の同 Linux 7.3 での実証, 並びに, GPL[11]に基づき, 以下の URL でのソースプログラムやマニュアル等の今年度末公開を目指している.

http://www.ibaraki-ct.ac.jp/ece/yas/mtslw/index.html

6.謝辞

素晴らしい Linux を実現してくださった Linus Torvalds 様と仲間の皆様, Apache を開発された Apache Group の皆様, 並びに, GPL の元で種々のプログラムを提供してくださった皆様に, 深謝致します.

【参考文献】

[1] 伊土誠一, 中川豊, 杉村康,"運用データを利用した大規模 オンライン情報システム用信頼性保証方式", 信学論, Vol.J 79-D-I,No.12,pp1192-1202, 1996 年 12 月
[2] 土田米一, "Linux 白書 2001-2002",インプレス(株), 2002 年 12 月.
[3] "ラボベンチマーク結果: Windows NT 対 Linux", <http://www.zdnet.co.jp/pcweek/news/9906/28/c-022.html>, 2000 年 6 月.
[4] ラインホルト P.ワイカー,"ほころびが見えはじめたベンチマークテスト", 日経バイト, pp.245, May 1991.
[5] 杉村康,"リアルタイムにおける RISC 性能の解析", 信学論, Vol.J79-D-I,No.2,pp.88-100, 1996 年 2 月.
[6] R.P.Weiker,"Dhrystone: A synthetic systems programming benchmark", Commun. ACM. Vol.27,No.10, pp1013-1030 , Oct. 1984.
[7] R.Card, E.Dumas, F.Meel,"Linux 2.0 カーネルブック", オーム社, 1999 年 5 月.
[8] 杉村康,伊土誠一, " Pentium 系 Linux の詳細動作を解析するスーパートレーサ(STDB) ", Linux Conference 2001 in japan, 4 月 2001 年.＜http://www.ibaraki-ct.ac.jp/ece/yas/stdb/index-e.html＞
[9] 葛西重雄訳,"Linux プログラミング", ソフトバンク, 1999 年 3 月.
[10] 篠田陽一訳,"Unix ネットワークプログラミング (Vol.1) ネットワーク API: ソケットと XTI", ピアソンエデュケーション, 1998 年 7 月.
[11] " GNU 一般公有使用許諾書,<http://www.sra.co.jp/public/doc/gnu/gpl-2>

【著者紹介】

信太晃司: 茨城高専・情報電気電子専攻科二年在学中. Web サイト評価のための多端末シミュレータ(MTSLW)を開発中.
伊藤剛志: 茨城高専・情報電気電子専攻科二年在学中. Web サイト評価のための多端末シミュレータ(MTSLW)を開発中.
梅原和芳: 平成 14 年 3 月茨城高専・電子情報工学科卒. Web サイト評価のための多端末シミュレータの情報収集プログラム(SDCP)の開発に従事した. 現在, 同校情報電気電子専攻科一年在学中,マルチプロセッサ Linux 用スーパートレーサを研究中.
三條光輝:平成 14 年 3 月茨城高専・電子情報工学科卒. Web サイト評価のための多端末シミュレータの情報収集プログラム(SDCP)の開発に従事した.
渡辺高明: 平成 14 年 3 月茨城高専・電子情報工学科卒. Web サイト評価のための多端末シミュレータの情報収集プログラム(SDCP)の開発に従事した. 現在, 茨城大学工学部情報工学科在学中.
杉村康: 昭和 46 年鹿児島大・工学部・電子工学科卒. 同年日本電信電話公社（現 NTT）入社.以来ＴＳＳ系／ＲＴＳ系ＯＳの制御プログラム等の開発・性能評価に従事.平成 13 年 4 月より, 国立茨城高専・電子情報工学科/情報電気電子専攻科教授, 工学博士(九大). 電子情報通信学会, 情報処理学会, ＩＥＥＥ, 日本リヌックス協会各会員.

「完」