IEEE 1394を利用した オペレーティングシステムの振舞いの測定手法

山之内 暢彦† 多田好克†

入出力処理を頻繁に行うアプリケーションの性能はオペレーティングシステムに強く依存する．そのアプリケーションの問題を明らかにし性能向上を図るには，オペレーティングシステムの振舞いを測定することが不可欠である．しかし，オペレーティングシステムの測定にはそれ特有の難しさがある．また，既存の手法では，測定のオーバヘッドのために本来とは異なる振舞いを観測することがあり，問題である．我々は，オペレーティングシステムの振舞いの測定に IEEE 1394 を利用した手法を提案する．本手法は，IEEE 1394 コントローラの持つ機能を利用して測定対象の動作状態を「覗き見」するものである．本手法はリアルタイムでの測定を重視したもので，ネットワークサーバ等の実行環境の測定に適している．実験から，測定のオーバヘッドは無視できるほど小さいことが確認できた．本稿では，本手法を用いた測定例として，Web サーバ等の実行環境における振舞いを報告し，手法の有効性を示すほか，本手法が実行環境に与える影響，ユーザプロセスへの適用法について考察を加える．

Measurement Method for Operating System Behavior Using IEEE 1394

Nobuhiko Yamanouchi† and Yoshikatsu Tada†

The performance of application which frequently handles I/O operation strongly depends on operating system (OS). While measuring the system behavior is necessary to improve these applications, we often face difficulties peculiar to operating system. In addition, existing methods incorrectly capture behavior due to their overhead. We propose a method which employs IEEE 1394 to measure OS behavior. Our method is of "peeping" into a measured environment, exploiting the capability of IEEE 1394 controller. Our method places emphasis on realtime measurement. Experimental results show that the overhead imposed by our method is negligible. In this paper we present an example of measurement, by illustrating the behavior of a system in which the Web server is running, and examine effects brought to measured environments, and technique for applying our method to user process.

1. はじめに

ネットワークサーバやデータベースプログラムといった，入出力処理を頻繁に行うアプリケーション（以降，AP とする）は，オペレーティングシステム（以降，OS とする）の機能を利用することが多いため，AP よりむしろ OS の性能がシステム全体を左右する場合が多い．特に高負荷の Web サーバでは，OS が実行時間の大部分を占めるとされている[1)]．したがって，システムの性能上の問題点を明らかにするうえで，AP のみならず OS の振舞いを測定，分析することが不可欠といえる．

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications

AP の測定とは違い，OS の振舞いの測定にはそれ特有の難しさがある．まず，測定を行うユーザプロセスは，OS が提供する保護環境下で動作しているため，OS 内の処理まで知ることができない．また，そのプロセスが動作することによって，スケジューリング等の OS の振舞いが変わる可能性がある．

入出力処理が頻繁に行われる環境では，測定のオーバヘッドが重大な問題となりうる．たとえば，ネットワーク処理は他のホストや通信路との相互動作に依存するため，リアルタイムで測定しなければ本来とは異なる動作を観測する可能性が高い．そのほか，膨大な量になりやすい測定データの扱いも難しい．データをディスク等へ保存することは大きなオーバヘッドをともなうため，避けるべきである．

これらの問題を克服するため，我々は，OS の振舞

いの測定に IEEE 1394 を利用した手法を提案する．この手法は，IEEE 1394 コントローラの持つ機能を利用して，測定対象 PC の動作状態を「覗き見」するものである．ここで「覗き見」とは，別の PC から被測定 PC のメインメモリを読み取ることをいう．その際，測定対象 OS の動作を止めたり，変更したりすることがなく，同時に「覗き見」によって課されるオーバヘッドは無視できる程度であるため，リアルタイムの測定が可能である．

本稿では，2 章で他の測定手法について検討を行ったあと，3 章で我々の提案する測定手法の内容を述べる．本手法において，IEEE 1394 の通信性能は重要な意味を持つ．それについて 4 章で実験結果を示す．5 章では，本手法を用いた例として，3 つの異なる負荷における OS の振舞い，とりわけ割込みの許可・禁止状態について報告する．6 章で本手法によって測定対象が受ける影響を評価し，7 章で考察を行ったあと，8 章でまとめる．

2. 関 連 手 法

振舞いの測定には主に 2 つの方式がある．トレース方式は，測定者の指定したイベントの実行ログを得るものである．この方式では，イベント実行のたびにログの記録処理が必要であるため，測定のオーバヘッドが大きくなりやすい．一方，サンプリング方式は，ある時点の変数を継続的に収集するもので，オーバヘッドは一定である．

Linux カーネルに内蔵のプロファイラはタイマ割込みを利用し，周期的に CPU の命令ポインタをサンプリング，集計するものである．この手法は，プログラムのホットスポットを把握するのには有効だが，タイマ割込みに依存しているため，割込み禁止区間の振舞いを観測することができない．同じくサンプリングを行う Digital Continuous Profiling Infrastructure [2)]（DCPI）は，割込みの制限を受けないが，CPU アーキテクチャ固有の機能に依存している．

鶴[3)] は，ソフトウェアで実装された命令トレーサで，振舞いを網羅的に把握することができるが，オーバヘッドがきわめて大きいためリアルタイムでの測定ができない．同じトレーサでも，ハードウェアで実装された TinyTOPAZ [4)] や，参考文献 5) で用いられたハードウェアモニタがある．これらはリアルタイム性を犠牲にするものではないが，特殊なハードウェアを用いており，可搬性，入手性に問題がある．

いずれの手法においても，長時間の測定を行うとデータの量が膨大になり，その記録にかかるオーバヘッドが問題となりやすい．TinyTOPAZ では，外部ハードウェアにハードディスクを内蔵させることで，記録の負担を軽減している．

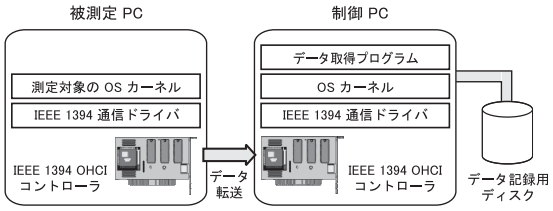

図 1 測定手法の装置構成
Fig. 1 System structure of our measurement method.

測定によるオーバヘッドは，動作しているシステムの性能低下として現れる．性能低下の割合は，各参考資料によると，DCPI で 1.0〜3.0%，TinyTOPAZ で 1.5%，鶴で 330〜625 倍とされる．

3. 測 定 手 法

本研究で提案する手法は，測定対象コンピュータ（以降，被測定 PC とする），測定制御用コンピュータ（以降，制御 PC とする），およびそれらを接続する IEEE 1394 コントローラで構成される（**図 1**）．被測定 PC では，測定対象の OS，AP を動作させ，一方，制御 PC では IEEE 1394 コントローラの Physical Read 機能を使って「覗き見」を行い，得たデータをハードディスクへ記録する．

以下では，まず IEEE 1394 と Physical Read について説明し，それから IEEE 1394 を用いた測定手法の内容を述べる．

3.1 IEEE 1394 の概要

高速シリアルバス規格の IEEE 1394（FireWire, i. Link とも呼ばれる）は，400 Mbps の通信速度を持つ通信手段である．通信モードには，パケット再送の手順が規定された Asynchronous モードと，周期的なデータ転送が規定された Isochronous モードの 2 つがある．本手法では，データ転送のタイミングに制限がない Asynchronous モードを使用する．

Asychronous モードのパケットヘッダには 64 ビットのアドレスがあり，上位 16 ビットが通信ノード等の識別，下位 48 ビットがノード内アドレスに使われる．

3.1.1 Physical Read 機能

Physical Read 機能は，IEEE 1394 を介したメインメモリのリモート読み取りを可能にする機能である．Physical Read の処理は次の順に行われる．

(1) メモリ内容を要求するノードが，相手先に Read Request パケットを送信する．

(2) それを受け取ったノードは，パケットヘッダ中のアドレス（ノード内アドレス 48 ビットのうち下位 32 ビット）を物理アドレスとして解釈し，メインメモリから内容を読み取る．

(3) 読み取った内容から Read Response パケットを構成し，それを要求元に送り返す．

Physical Read の重要な利点は，上記 (2)，(3) の処理をすべて IEEE 1394 コントローラが行う点にある．一般的な NIC では，パケットの受信時にハードウェア割込みが発生し，ソフトウェアがパケット処理を行う．一方，Physical Read ではいっさい割込みが発生しない．

Physical Read 1 回あたりの最大サイズは通信速度によって異なり，400 Mbps のとき 2,048 バイトである．4 バイト境界にアラインされる点を除き，読み取りアドレスに制限がない．

Physical Read 機能は，IEEE 1394 コントローラのなかでも Open Host Controller Interface [6]（OHCI）規格に準拠したものが搭載されている．現在流通している PC 用コントローラのほぼすべてが OHCI に準拠しており，デバイスの入手は容易である．

3.2 測定への応用

OS の振舞いを把握する有効な方法の 1 つは，OS 内のデータを取得することである．Physical Read を使えば，被測定 PC の動作を止めることなく，OS 内のデータを直接取得できる．

OS 内のデータには，メモリ空間を管理するマップをはじめ，割込みの統計情報，プロセスの管理情報等が存在し，それらから OS の振舞いの一側面を把握することができる．また，それらのデータを継続的にサンプリングすることで，値の変化や分布を測定できる．

測定内容によって，CPU のレジスタ内容，ルーチンの実行回数等が必要とされる場合がある．しかし，それらの情報はメモリ上に存在しておらず，Physical Read で値を直接得ることはできない．その際，ソフトウェアによる支援が有効である．たとえば，値が変化する位置に，その情報をメモリに記録するごく短い命令列（プローブコード）を挿入する．被測定 PC からは，Physical Read を使いそのメモリを読み取る．

たとえば，ユーザプロセスかカーネル，どちらを現在実行中なのかを判別するには，CPU の実行モード（カーネルモードか，ユーザモードか）を記録する必要がある．その場合には，システムコールの出入口にプローブコードを挿入することで対応できる．

3.3 利　　点

OS の振舞いの測定において，本手法の利点は以下のとおりである．

- OS の動作を止めたり，変えたりすることがない．被測定 PC 上で測定を制御したり，測定データを処理したりする必要がないため，OS の本来の動きと同じ振舞いを測定することができる．
- 長時間の測定が可能である．OS 内データのサンプリングを続けると，測定データは膨大なものになる．本手法では，測定データを高速な IEEE 1394 を使って転送し，ディスク等への保存を制御 PC が行う．そのため，被測定 PC の動作に影響を与えることなく大量のデータを管理できる．
- OS の動作状態に左右されずに測定が可能である．Physical Read は CPU とは独立して動作する．そのため，被測定 PC が過負荷状態でも測定動作が遅延することはない．また，割込み駆動のサンプリングとは違い，割込み禁止区間等のクリティカルセクション内の振舞いを測定することができる．
- 測定に特殊な装置が必要ない．以上の利点を得るために，従来は特殊な測定装置を PC に接続し測定が行われてきた．本手法で用いる IEEE 1394 コントローラは広く流通しているデバイスであり，容易に本手法を適用できる．

3.4 測定による動作の乱れ

測定を行うことにより，OS の動作に乱れが生じ，本来の動作とは異なる振舞いを測定することがある．本手法がもたらす乱れの発生源は以下の 2 つに限定される．

Physical Read　IEEE 1394 コントローラがメインメモリとの間でデータ転送を行っている間，システムバスが占有される．その際，CPU からのメモリアクセスや他のデバイスの処理が遅れる可能性がある．

メモリアクセスの方式がライトバックである場合，Physical Read が誘引するライトバックによってシステムの本来の動作が待たされる可能性がある．Physical Read がキャッシュシステム上の最新のデータを得るには，ライトバックが Physical Read の動作と同時かそれより前に完了している必要がある．一部アーキテクチャ（本稿の実験で使用したシステムを含む）の場合，キャッシュシステムが Physical Read のバスアクセスをつねにスヌープ（監視）しており，もしデータがダーティーであればライトバックが自動的に行われる．したがって，Physical Read のたびにライトバックのオーバヘッドが課される可能性がある．

プローブコード　プローブコードのオーバヘッドに

よって，被測定 PC の本来の動作が遅れる可能性がある．また，プローブコードがメモリアクセスを行うことでキャッシュシステムに乱れが生じる．

プローブコードの機能とそのオーバヘッドはトレードオフの関係にある．プローブコードがレジスタ内容をメモリに書き出すだけのものであれば，コードは数命令で済み，単体のオーバヘッドは小さい．また，オーバヘッドは実行頻度にも依存する．測定を行う者は，コード内容と実行頻度を調整することで，プローブコード全体のオーバヘッドを許容できる範囲に抑えることが可能である．

これらのオーバヘッドは，結果的に被測定 PC の性能低下として現れる．5 章で測定例のプローブコードを示し，6 章で実験をとおし性能低下の程度を示す．

4. IEEE 1394 の通信性能

この章では，Physical Read の通信性能について述べる．測定で OS 内データのサンプリングを行う場合，どの程度の反復間隔でデータを読み取れるかが重要である．その間隔が短いほど，より小さな振舞いを観測することができる．実験の結果，Physical Read を最小 20.7 マイクロ秒の間隔で行えることが分かった．

4.1 実験環境

実験には Intel Pentium III 600 MHz（600E 型），128 M バイト SDRAM 搭載の PC/AT 互換機を 2 台用い，それらに IEEE 1394 コントローラ（TI 社製 OHCILynx 搭載，PCI ボード）を装着した．被測定 PC では Linux 2.2.14，制御 PC では Linux 2.0.36 を動作させた☆．IEEE 1394 の通信ドライバには参考文献 7)，8) のものを使用した．

制御 PC 側のユーザプロセスから容易に Physical Read を利用できるようにするため，デバイス `/dev/rmem` を導入した．これは，IEEE 1394 通信ドライバとユーザプロセスとの間で通信データの受渡しを行うものである．

計測は `/dev/rmem` を使い，ユーザプロセスから行った．このプロセスは，被測定 PC のメインメモリ上に確保した 2,048 バイトの領域を Physical Read を使って繰り返し読み取る．スループット等は，計 256 M バイトを読み取る時間から算出した．

☆ 制御 PC 側の Linux のバージョンが古いのは，フル機能の IEEE 1394 ドライバがそのバージョンにしか対応していないためである．被測定 PC 側には，Physical Read を有効にするだけの簡易ドライバを使用した．

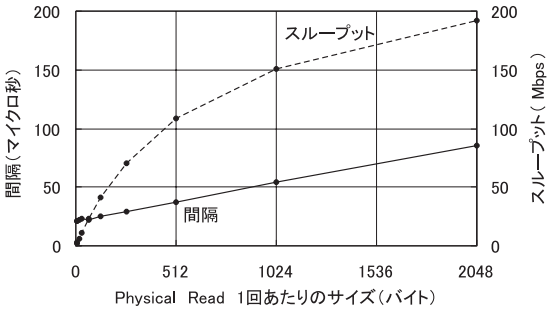

図 2 Physical Read の通信性能
Fig. 2 Performance of Physical Read.

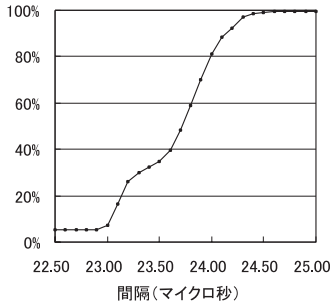

図 3 Physical Read の反復間隔の累積分布
Fig. 3 Cumulative distribution function of intervals of repetitive Physical Reads.

4.2 結　　果

Physical Read のスループットと反復間隔を**図 2** に示す．グラフの横軸は Physical Read 1 回あたりのサイズを表す．

Physical Read の間隔は，最小 20.7 マイクロ秒（4 バイト時），最大 85.3 マイクロ秒（2,048 バイト時）である．これは，既存の手法で用いているタイマ割込みの周期 10 ミリ秒に比べはるかに短く，より小さな振舞いの観測が可能といえる．一方，仮に CPU の動作クロックを 1 GHz としたときのクロック周期 1 ナノ秒とは開きがあるため，命令レベルの振舞いを網羅的に測定することは難しい．

8 バイトの Physical Read を 1 万回行ったときの間隔のばらつきを累積値として示したものが**図 3** である．99%以上の Physical Read が 24.6 マイクロ秒以内に完了しており，安定的に Physical Read を続けられることが分かる．2,048 バイトのときも同様で，99%が 93.2 マイクロ秒以内に終わっている．若干のばらつきが出た理由は，(1) IEEE 1394 コントローラや PC 本体等のハードウェアに起因した揺らぎが発生した，(2) 測定はユーザプロセスで行ったため，常駐しているデーモンプロセスや他のハードウェア割込みの影響を受けた，2 点と我々は考える．

5. 測定例：割込み許可・禁止の振舞い

本手法の具体的な利用方法を示すために，その一例として，割込みの許可・禁止の振舞いを測定した．一般に，割込みの禁止時間が長い場合，システムの反応が遅れ，その結果性能が低下する可能性がある．この測定例では，割込みによる性能低下の存在を確かめる．

割込みの許可・禁止の振舞いは，従来のタイマ割込みを用いたサンプリング手法では観測が難しい．また，トレース方式ではログデータを扱う際のオーバヘッドが大きいため，ネットワーク処理の測定には適さない．そのため，割込みの許可・禁止状態に依存せず，低いオーバヘッドで測定できる本手法が適していると我々は考える．

測定の結果，Web サーバが過負荷のとき，割込みの禁止時間は全体の 11%に達した．しかし，禁止されてから許可されるまでの平均時間が短く，割込み禁止による性能への影響は小さいと考えられる．

この章では，測定例の詳細について述べ，いくつかの考察を加える．

5.1 測定内容

この実験で測定した項目を以下に示す．

- 割込みの許可・禁止状態
- ユーザプロセス，カーネル（割込みハンドラを除く），割込みハンドラの各実行時間
- プロセス数，実行キューの長さ

ユーザプロセス等の実行時間やプロセス数は，OS の全体的な振舞いを把握するために測定した．

割込みが禁止されていると，再び許可されるまで割込みの発生が遅れ，それだけハードウェアに対する OS の反応が悪くなる．したがって，割込みの禁止時間は短いほうが良いとされる．今回は，割込み禁止の原因を明確にするため，割込みハンドラの top half（以降，TH とする），bottom half（以降，BH とする）を区別して測定を行った．

ここで，TH，BH について説明を加える☆．Linux の割込みハンドラは，一部割込み禁止の状態で実行される TH と，TH の直後に割込み許可の状態で実行される BH に分かれる[9]．これは割込みに対する反応を素早くするためで，時間のかかりやすい処理は BH で実行される．ネットワークハンドラの BH では，遅延されたパケットの送信，受信したパケットの分配，プロトコル処理等が行われる．

☆ Linux の用語では，他の Unix 系 OS の TH，BH と実行順序，意味ともに異なるので注意されたい．

測定に際し，種類の異なる 3 つの負荷をかけて行った．

qsort 計 40M バイトの浮動小数点数をクイックソートする．計算中心の負荷である．メモリのアクセス量，範囲は広いが，システムコールはほとんどない．

make Linux カーネルの構築．計算処理が多く，システムコール，ディスクとの入出力処理も頻繁に行われる．

httpd Apache 1.3.20 を動作させ，サーバが処理しきれない量のリクエストを別の PC から送る．入出力処理が中心である．

そのほか，負荷をかけていないアイドル状態（以降，**nop** とする）の測定を行った．

5.2 測定方法

測定に際し，トレース方式，サンプリング方式のどちらを採用するか考慮する必要がある．この実験ではサンプリング方式を採用した．理由は，トレース方式のオーバヘッドが無視できなくなるからである．ネットワーク処理では，割込みの許可・禁止が頻繁に行われるため，それだけトレースのオーバヘッドが大きくなる．一方，ネットワーク処理は他のホストとの相互動作に依存するため，リアルタイムの測定が不可欠である．したがって，トレース方式では誤った測定結果を得やすい．

サンプリング方式でも，従来のタイマ割込みを使った手法は利用できない．割込み禁止中はサンプリングを行えず，まったく誤った結果しか得られないからである．そのため，CPU の状態に依存せずに行える本手法が，割込みの許可・禁止状態の測定に適している．

本実験では，割込み禁止ルーチン等にプローブコードを挿入し，割込み禁止の回数と時間比を挿入した位置別に得られるようにした．これにより，割込み禁止時間が長い箇所を特定することができる．トレース方式と同様，割込み禁止のたびにプローブコードが実行されるが，ログの記録処理がいらないため，コードはわずか数命令に収まる．

5.2.1 プローブコード

カーネル内のいくつかの場所にプローブコード（機械語で 2〜7 命令）を挿入し，CPU の動作モード，命令ポインタ等をメインメモリ上に保存するようにした．制御 PC では 100 マイクロ秒周期で Physical Read を 2 回発行し，プローブコードが保存した値とプロセス数に関する変数，計 82 バイトを繰り返しサンプリングする．

プローブコードは以下の位置に挿入した．

(1) プロセス切換えルーチン
(2) システムコール，シグナル，例外，TH，BH の

```
プロセス切換えルーチン（include/asm-i386/system.h）
#define switch_to(prev,next,last) do { \
  ■ プロセス管理構造体のアドレスを保存
  __fp_task_addr = (unsigned long)next; \
  ■ PID を保存
  __fp_id_mode = \
     __FP_ID_MODE(next->pid, __FP__KERNEL)}; \
  ■ これ以降はもとの処理
  asm volatile("pushl %%esi\n\t" \
               "pushl %%edi\n\t" \
  ■ 以下省略（レジスタの保存，スタックの切換え処理など）

割込み禁止ルーチン（include/asm-i386/system.h）
#define __cli() do { \
  __asm__ __volatile__ ( \
    ■ CPU の割込み許可フラグをクリア（もとの処理）
    "cli\n\t" \
    ■ 現在の実行アドレスを保存
    "call 1f\n" \
    "1:\tpop %0" \
      : "=g" (__fp_eflags_addr) : : "memory"); \
  ■ 割込み禁止状態であることを保存
  __fp_eflags = __FP_EFLAGS_CLI; \
} while(0)
```

図 4 プローブコードの例
Fig. 4 Probe codes.

出入口

(3) 割込み許可，禁止ルーチン

これらのうち，プロセス切換えと割込み禁止のルーチンを**図 4** に示す．コード中，下線部が測定のために追加した箇所で，■印以降はコメントである．`__fp` で始まる変数は，プローブしたデータの保存用で，メインメモリ上に置かれており，測定のとき Physical Read により取得される．`__FP` で始まる識別子は，CPU のモード等を表すマクロ（`#define`）である．プロセス切換えルーチンでは，プローブコードによって切換え後のプロセス ID がメモリに保存される．割込み禁止ルーチンでは，それが実行されたコード位置と，割込み禁止状態を示す値が保存される．

5.2.2 実験環境

httpd の実験では，クライアントとして FreeBSD 4.4 が動作する Pentium II 400 MHz，192M バイト SDRAM 搭載の PC/AT 互換機を使用し，サーバ，クライアント間の接続に 100BASE–T を用いた．HTTP の負荷発生には S-client [10] を使用し，100,000 トランザクションの処理が完了するまで測定を行った．S-client は，サーバに過度の負荷を与えるプログラムで，4,096 バイトの同一ファイルを繰り返し取得する．

その他の実験環境は 4 章に示したものと同じである．

5.3 結果

実行時間の比を**表 1** に示す．httpd でカーネルの実行時間の割合が高く，Web サーバのカーネルへの依存度が高いことが分かる．Apache を含め，複数プロセス型 Web サーバの重大な問題点として，システムコールとスケジューリングのオーバヘッドが繰り返し指摘されている [11),12)]．しかし，この実験では，それらの実行時間を含む「割込み以外」が 3 割を下回っており，スケジューリング等による影響が必ずしも支配的ではないことが分かる．むしろ，割込みハンドラ（TH + BH）の実行時間の比が大きい．

表 1 実行時間の比
Table 1 Percentage of execution time by an idle state and three workloads.

		nop	qsort	make	httpd
ユーザ		0.0	98.8	92.9	44.1
カーネル	TH	*0.1*	*0.1*	*0.2*	*9.2*
	BH	*0.0*	*0.0*	*0.1*	*21.5*
	割込み以外	*0.0*	*1.1*	*6.5*	*25.2*
	計	0.1	1.2	6.8	55.8
アイドル		99.9	0.1	0.2	0.1

(%)

表 2 割込み許可・禁止時間の比
Table 2 Percentage of elapsed time with interrupts enabled/disabled.

	nop	qsort	make	httpd
許可	99.9	99.9	99.5	89.4
禁止	0.1	0.1	0.5	10.6

(%)

表 2 は，割込み許可・禁止の状態を時間の率で示したものである．httpd の割込み禁止の比は 10.6%と高く，それだけ割込みハンドラの起動が遅延しているように見える．しかし，各ルーチンについて詳しく見た結果，割込みの禁止による性能への影響は大きくないことが分かった．詳しくは次のとおりである．

httpd の測定のうち，割込みの禁止を行う主なルーチンを**表 3** に示す．表中，右の列は割込み禁止時間全体に対する各ルーチンの比を表す．また，この比が高いルーチンについて，禁止状態が継続する時間の平均値を禁止・許可命令の実行回数から見積もった（**表 4**）．禁止状態が続く時間は 10 マイクロ秒以下で，割込みの平均発生間隔 62.4 マイクロ秒よりも十分小さい．したがって，割込みの遅延が発生していたとしても，それによる影響は大きくないと推測できる．

ここで，表 3 にあげたルーチンについて考察を加える．TH 関連ルーチンの比が高いが，もともと TH は割込み禁止状態で開始され，実行頻度が高く，そのためこの振舞いは容易に予想できるものといえる．一方，スケジューリングを行う `schedule` の比が 23.8%に達

表 3 割込み禁止を行う主要なルーチン
Table 3 Major routines which disable interrupts.

ルーチン	比（%）
TH 関連	46.3
IRQ0x05_interrupt	*34.7*
handle_IRQ_event	*11.0*
その他	*0.5*
schedule	23.8
kmem_cache_free	7.0
kfree	4.1
kmem_cache_alloc	3.9
kmalloc	2.2
do_sigaction	1.9
__get_free_pages	1.6
do_bottom_half	1.5
mod_timer	1.1
do_gettimeofday	1.0

表 4 割込み禁止時間
Table 4 Interrupt disabled time.

ルーチン	実行回数	禁止時間（マイクロ秒）
IRQ0x05_interrupt	1,110,042	2.3
handle_IRQ_event	1,117,192	0.7
schedule	255,070	6.9

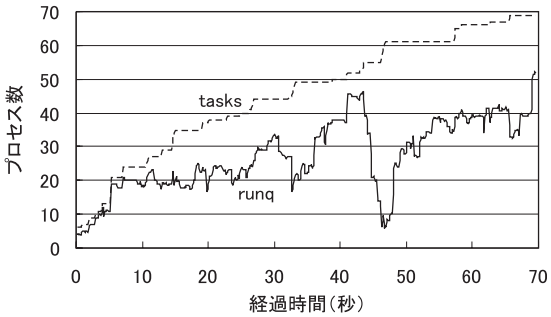

図 5 httpd プロセスの推移
Fig. 5 Transition of httpd processes.

したことには注意を要する．

schedule の振舞いを説明するために，**図 5** に httpd のプロセス数（図中，tasks）と実行キューの長さ（図中，runq）を示す．Apache は負荷の状態に応じて自身のプロセス数を増減させる機構を持つが，tasks，runq いずれも時間の経過とともに増加傾向を示し，runq は最大で 54 に達する．schedule 中には runq に対して線形時間の処理があり，このため割込み禁止時間の比が大きくなったといえる．

6. 測定が与える影響の評価

3.4 節 で述べたとおり，プローブコードの挿入と Physical Read を行うと被測定 PC の性能が低下することが予想できる．反対に，被測定 PC の動作によって Physical Read が遅延され，正しいデータの取得を行えない可能性がある．いずれもその影響は小さいほうが良い．この章では，これらの影響について実測し定量的に示す．

表 5 測定による性能低下
Table 5 Performance degradation imposed by measurement.

	qsort	make	httpd
プローブ挿入	0.1	0.2	0.5
＋ Physical Read	0.2	0.4	1.8

（%）

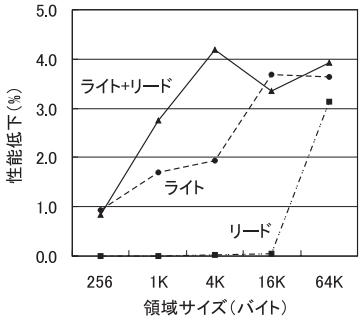

図 6 メモリ性能の低下
Fig. 6 Degradation of memory performance.

6.1 被測定 PC の性能低下

5 章 の実験における被測定 PC の性能低下の程度を**表 5** に示す．表中の値は，測定を行っていないときの実行時間に対する増加率である．性能低下は，プローブコードの挿入時，たかだか 0.5%でさらに Physical Read を行ったとき 2%を超えていない．この値は，プローブの挿入を行わない DCPI（性能低下率 1.5〜3.0%）や，データの取得を特別なハードウェアで行う TinyTOPAZ（同 1.5%）と比べても高くない．qsort に比べ httpd の性能低下が大きい理由には，(1) プローブコードが挿入されているシステムコール，割込みハンドラの実行頻度が高い，(2) 入出力処理によるシステムバスの占有率が高いことがあげられる．

純粋に Physical Read による影響を知るために，被測定 PC のメモリアクセス性能がどの程度低下するか調べた．具体的には，被測定 PC，Physical Read の両方から同一領域のメモリへアクセスを反復し，衝突が頻繁に起こる状況を作り出した．

図 6 は，被測定 PC が累積 256 MB のリード・ライトを終えるまでの時間をもとに，性能の低下率を示したものである．Physical Read の最大サイズは 2,048 バイトだが，領域のサイズがそれを超えるときは，領域の終端まで Physical Read が繰り返し実行される．

被測定 PC のメモリアクセスの性能低下は，いずれの領域サイズでも 5%を超えておらず，十分許容できる範囲内といえる．領域が 16 KB 以下のときリードの性能低下はほとんど見られないが，その原因は 16 KB の L1 データ・キャッシュの効果だと考えられる．

図 6 はメモリアクセスの衝突が頻繁に起こる最悪ケースの結果であり，実際の測定ではより小さくなると推測できる．5 章で述べた測定例では，Physical Read による性能低下は，表 5 の値からたかだか 1.3%と見積もることができる．

6.2 Physical Read の遅延

前節で被測定 PC に対する影響を述べたが，反対に，CPU によるメモリアクセスによって Physical Read が遅延される可能性がある．そこで，前節と同じ方法で Physical Read にかかる時間と反復間隔の分布を調べたが，メモリアクセスのあるなしに関係なく同じ結果が得られた．CPU の動作にかかわりなく Physical Read を安定して反復することが可能だといえる．

7. 本手法に対する考察

7.1 仮想記憶

Physical Address のアクセス先は被測定 PC の物理アドレスで指定される．そのため，仮想記憶によってデータが移動もしくはスワップアウトされる領域には本手法を直接適用することはできない．逆に，データがメインメモリに固定されているカーネルには本手法をそのまま適用することができる．

カーネル内のデータアクセスが仮想アドレスを通して行われる場合，測定者は，仮想アドレスから物理アドレスへの変換をあらかじめ行っておけばよい．たとえば Linux では，仮想アドレスからオフセット（通常 0xc0000000）を差し引けば物理アドレスが得られる．

仮想記憶によってデータが移動する環境，たとえばユーザプロセスでは，データが移動しないよう固定化（ピンダウン）を行うことで本手法を適用することができる．ただし，被測定 PC 上の OS が固定化の機能を提供していない場合，固定化を行うカーネルドライバ等を新たに導入する必要がある．たとえば，測定者は，ユーザプロセスのまとまった連続領域に測定対象データが配置されるよう指定し，コンパイルを行う．また，測定データが置かれた領域を固定化するカーネルドライバを導入し，ユーザプロセスから呼ぶようにする．測定の際は，Physical Read の制御プログラムにその領域の物理アドレスを入力すればよい．

7.2 メモリ外のデータの取得

3.2 節で述べたとおり，CPU レジスタ等のメインメモリ外のデータを取得するには，データの値が変化する位置にプローブコードを挿入するのが有効である．しかし，変化の時期がまったく不定であったり，データが CPU 命令によって得られにくい場合（メモリキャッシュの状態等），本手法を使ってデータを得ることは難しい．その際は別の手法を検討すべきだが，データ転送に本手法を利用できると考える．

7.3 メモリキャッシュとの関係

CPU のメモリアクセスがライトバック方式である場合，Physical Read で最新のデータを得るには，アクセス先のラインのライトバックが Physical Read と同時かそれより前に発生している必要がある．一部アーキテクチャではライトバックが自動的に行われるが，そうでなければソフトウェアによる特別な対応が必要である．たとえば，Physical Read のアクセス先のデータを変更したら，毎回ライトバックの命令を発行すればよい．あるいは，領域をあらかじめライトスルーに指定し，キャッシュの影響を受けないようにしておけばよい．

なお，前章までの実験でキャッシュに対する特別な対応は導入されていない．使用した PC のキャッシュシステムがライトバックを自動的に行うからである．具体的には，キャッシュシステムはメモリバス上のアドレス線をつねにスヌープ（監視）しており，Physical Read のアクセス先がキャッシュにヒットした場合，キャッシュシステムはすぐにライトバックを開始する．PCI バスコントローラは，そのライトバックの信号を受けてから，IEEE 1394 コントローラにデータを送る．

7.4 イベントトレーサへの応用

5 章で示した測定例は，同じデータを継続的にサンプリングすることにより統計的な結果を示したものといえる．一方，本手法をイベントトレースに応用することができる．我々が開発したイベントトレーサ[13]は，本手法をデータ転送手段として用いており，カーネル内の任意の位置で時刻と指定された変数内容を記録することができる．ただし，トレースされるのはユーザが指定した位置に限っており，命令レベルのトレースを実現するものではない．

8. おわりに

本稿では，OS の振舞いの測定に IEEE 1394 を利用する手法を提案した．本手法は，入手性の高い IEEE 1394 コントローラの機能を使って，被測定 PC 上のメモリを「覗き見」するというものである．また，本手法を実際に用いて，従来の手法では観測が難しい割

込みの許可・禁止の振舞いについて報告した．測定結果から，過負荷の Web サーバが動作している環境では，割込み禁止の時間比が 10%強と高いが，それが性能に与える影響は大きくないことがいえる．

本手法が特に有効といえる測定対象は，ネットワークサーバ等の入出力処理が頻繁な環境である．この環境下では，本来とは異なった振舞いを観測しないために，リアルタイムで測定を行うことが重要である．本手法は測定による性能低下がきわめて小さいため，リアルタイム性を阻害することがない．また，割込み駆動の手法とは違い，本手法は OS のクリティカルセクション内の振舞いを測定することができる．このことは，OS の実行時間が長いサーバ環境の測定において重要といえる．

本手法は，測定のリアルタイム性やデバイスの入手性を重視したもので，他のあらゆる手法を置き換えるものではない．たとえば，CPU レジスタの全内容やメモリキャッシュの振舞い等を網羅的に測定することは難しい．また,「覗き見」の反復間隔はマイクロ秒以上であるため，高速な CPU の振舞いを命令レベルで把握することはできない．命令トレーサを使えばこれらの測定は可能といえるが，逆にリアルタイムで測定することは難しい．

測定例として Web サーバの振舞いを示したが，過負荷状態のままであると，時間とともにプロセス数が増加していき，HTTP スループットの低下が見られた．このことは複数プロセス型 Web サーバに特有の問題点といえるが，詳しい原因の追求は今後の課題である．

謝辞 IEEE 1394 通信ドライバの最新版を提供していただいた電気通信大学の兵頭和樹氏に深く感謝いたします．

参 考 文 献

1) Barford, P. and Crovella, M.: A Performance Evaluation of Hyper Text Transfer Protocols, *Proc. Joint International Conference on Measurement and Modeling of Computer Systems*, pp.188–197 (1999).
2) Anderson, J.M., Berc, L.M., Dean, J., Ghemawat, S., Henzinger, M.R., Leung, S.-T.A., Sites, R.L., Vandevoorde, M.T., Waldspurger, C.A. and Weihl, W.E.: Continuous Profiling: Where Have All the Cycles Gone?, *ACM Trans. Comput.Syst.*, Vol.15, No.4, pp.357–390 (1997).
3) 森本洋行，小宮山彰一郎，毛利公一，吉澤康文：性能評価のための命令トレーサの開発，電子情報通信学会論文誌（D–I），Vol.J84–D–I，No.6, pp.584–593 (2001).
4) Horikawa, T.: TinyTOPAZ: A Hybrid Event Tracer For UNIX Servers, *Proc. Symposium on Performance Evaluation of Computer and Telecommunication Systems* (*SPECTS*), pp.203–210 (1999).
5) Torrellas, J., Gupta, A. and Hennessy, J.: Characterizing the Caching and Synchronization Performance of a Multiprocessor Operating System, *Proc. 5th International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS*), pp.162–174 (1992).
6) The Promoters of The 1394 Open HCI: 1394 Open Host Controller Interface (1997).
7) 兵頭和樹，中山泰一：IEEE 1394 を用いた PC クラスタシステム—通信機構の設計と評価，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.41, No.SIG 8 (HPS 2), pp.39–47 (2000).
8) 山之内暢彦，兵頭和樹，南 将朝，中山泰一：IEEE 1394 による PC クラスタシステムの設計，情報処理学会第 58 回全国大会講演論文集，3F–01 (1999).
9) Bovet, D.P. and Cesati, M.: *Understanding the Linux Kernel*, O'Reilly (2001).
10) Banga, G. and Druschel, P.: Measuring the Capacity of a Web Server, *Proc. USENIX Symposium on Internet Technologies and Systems* (1997).
11) Nahum, E., Barzilai, T. and Kandlur, D.: Performance Issues in WWW Servers (Extended Abstract), *Proc. Joint International Conference on Measurement and Modeling of Computer Systems*, pp.216–217 (1999).
12) Banga, G., Druschel, P. and Mogul, J.C.: Better Operating System Features For Faster Network Servers, *Performance Evaluation Review*, Vol.26, No.3, pp.23–30 (1998).
13) 山之内暢彦，多田好克：IEEE 1394 を利用した OS プロファイラの開発，情報処理学会研究報告，No.2001–OS–87, pp.25–32 (2001).

(平成 13 年 11 月 7 日受付)
(平成 14 年 11 月 5 日採録)

山之内暢彦（学生会員）

1975 年生．2001 年電気通信大学大学院情報システム学研究科博士前期課程修了．2002 年同研究科博士後期課程中退．修士（工学）．オペレーティングシステム，ネットワーク処理，システム性能評価に興味を持つ．ACM 会員．

多田　好克（正会員）

1985 年東京大学大学院工学系研究科情報工学専門課程博士課程修了．工学博士．同年電気通信大学電子情報学科着任．1992 年より電気通信大学大学院情報システム学研究科．並列・分散システムの記述法に興味を持ち，オペレーティングシステムをはじめとするシステムソフトウェアの実現法に関する研究に従事．ACM，電子情報通信学会会員．