オペレーティング・システム　50－10
（１９９１．３．１６）

ハイブリッド・モニタ手法を用いた システム動作の測定・解析

堀川　隆
日本電気（株）Ｃ＆Ｃシステム研究所

システム性能のネックを明確にすることを目的に、ハイブリッド手法により実働計算機システム動作の測定、および、測定データの解析を行なっている。unixワークステーションを対象に、コンパイラ、リンカの実行状況についてケース・スタディを行なった結果、このプログラム実行に関して、ファイル系の性能とプロセッサ性能は、ほぼ同程度に重要であることがわかった。

本手法を発展させることにより、計算機システムにおける種々の性能ネックを明確にできる見通しを得た。

System Performance Evaluation with Hybrid Measurement Approach

Takashi Horikawa
C&C Systems Research Laboratories,
NEC Corporation

System behaviors, including I/O behavior, was measured by a hybrid monitor consisting of both software probe inserted in OS (UNIX kernel) and hardware tracer, to discover various performance bottole necks and to clarify requirements to improve system performance.

A set of trace-data obtained from single-user multi-process workload has been quantitatively analyzed from various aspects; CPU and disk usage, process life time detail, and kernel CPU time detail. The results suggests that a file system performance, not a disk performance alone, will become more important in future multi-processor workstation.

1．はじめに

ユーザの立場からみた計算機システムの性能とは、実用アプリケーション・プログラムを実行させるときの性能である。この実用性能は、プロセッサ系、メモリ系、ファイル系、ディスプレイ系、通信系など、数多くのハードウエア要素性能に依存しているのに加え、ソフトウエアの性質にも大きく左右される。このため、計算機システムの性能要因を解析し、システムとしての性能ネックを解明するためには、プログラムの性質を考慮に入れた性能評価を行なう必要がある。

筆者は、実働計算機システムの動作を測定したデータを解析する方法で性能評価を行なうことにより、ソフトウエアの実行状況も含めたシステム性能要因の解析を行なっている。本稿では、測定ツールとして開発したハイブリッド・モニタ、および、unixワークステーションのファイル系を対象に行なった性能要因解析結果について述べる。

2．計算機システムの性能評価

2．1　性能評価手法

計算機性能の評価は、種々の方法により行なわれている。これらは、大別すると、実働計算機の動作状況を調べる方法、評価対象計算機の動作を抽象化したモデルを用いる方法に分類できる[1]。従来より、「計算機システム全体の性能評価」に多用されているのは、キューイング・モデルを代表とする後者であるが、この方法では、モデルに反映できない性能要因（性能ネック）については原理的に考慮できないため、未知の性能ネックを検出できないという問題がある。これに対し、前者の方法では、実働する計算機システムに存在する性能要因総てを評価対象とするため、このような問題はない。また、後者の方法により得られたなるのである。そこで、筆者は、測定をベースとする計算機の性能評価を進めている。

実働計算機システムの性能を測定する方法として最も多用されているのは、評価対象計算機上でベンチマーク・プログラムを実行させ、その実行時間を計測する方法であろう。評価作業が簡単であり、異なる計算機間の比較も容易である点が長所である。しかし、評価結果として得られるのが測定対象計算機におけるプログラムの実行時間のみである点と、多くのベンチマーク・プログラムで計測できるのはプロセッサ性能のみである点が短所である。

後者に関しては、unixワークステーションの性能指標としてSPECマーク[2]が使われ始めている等、システムとしての性能を測定するベンチマークが重要視されている[3]。ただし、SPECでも、プロセッサ性能中心のベンチマークであるという批判もあり[4]、I/O性能を評価するための標準的なベンチマークを設定することが1つの課題となっている。現状では、I/Oシステムまで含めた計算機システム性能を評価するには、実用アプリケーション・プログラムを実行させたときの動作状況を測定することが不可欠である。

前者の重要度は、性能評価の目的によって異なる。計算機ユーザであれば、各種計算機システムの性能比較が主要な目的であるので、プログラムの実行時間が計測できれば、ほぼ目的を達成できるであろう。しかし、「計算機システムの性能ネックを明確にしたり、新しいアーキテクチャを持った計算機の性能を予測する」という、計算機を設計する立場からの目的に対しては、プログラムの実行時間という評価結果だけでは不十分である。このような目的のためには、プログラムの実行時間だけでなく、プログラムの実行状況を詳細に把握することが必須であると考えている。そこで、筆者は、次章で述べる測定・解析システムを構築して性能評価を行なっている。

2．2　性能要因

計算機システムの性能要因を明らかにするために求めるべき性能指標がどのようなものであるのかは重要な問題である。従来より用いられている性能指標には種々のものがあり、どれを重視すべきなのかは、評価対象計算機の利用目的、利用形態によって異なる。主に用いられている指標としては、turn-around time、responce time、throughput がある[5]。

筆者の行なっている性能評価では、レスポンス時間を重視している。現在、さらには、今後の計算機環境の主流である「パーソナル・ユースのワークステーション（PC）をインタラクティブに利用する」ような場合に、ユーザの使い心地を左右する重要な性能指標と考えたためである。従来から行なわれているレスポンス時間の解析では、マルチ・ユーザ環境において、与えられたコマンドに対するレスポンス時間の平均値と分散を求め、計算機の負荷に応じてレスポンス時間が変化する様子を調べている。しかし、計算機を個人で専有して使用するような環境では、ユーザのプログラム実行は、他ユーザのプログラム実行に影響されることがないため、従来から行なわれているレスポンス時間の解析は意味をなさない。筆者は、ユーザから投入される個々のコマンド

を実行する時間そのものが問題になると考え、各コマンドの実行状況を詳細に追及するというアプローチをとった。個々のコマンドが投入されてから実行を完了するまでに行なう処理の内容を調べることにより、オーバーヘッドとなっている処理を明確にすることを目的にしたのである。

一般に、オーバーヘッド削減により期待できる性能向上は、それが全体の処理時間に占める割合に左右される。アムダールの法則が有効なのである。従って、この割合の多いものから削減していくアプローチが有効である。どのような処理をオーバーヘッドとみなすのかが問題となるが、ここではアプリケーション・プログラムを実行している時間以外、すなわち、システム・コールによりOSを実行している時間や、ディスク・アクセスを行なっている時間をオーバーヘッドとみなした。性能解析の第一段階として、このような時間が全体の処理時間に占める割合を調べた。具体的には、ディスクとCPUの利用率（Gantt Profile）や、OSで行なう各処理の実行時間を求めた。また、個々のプロセスが資源待ち状態に陥っている時間もオーバーヘッドと考えられるため、プロセスの待ち時間についても解析を行なった。

3．測定システム

実働計算機動作の測定をベースに計算機システム全体の性能解析を行なうアプローチにおいて重要となるのは、実働システムの動作状況を正確に表した測定データを得ることである。このためには、低オーバーヘッドで測定を行なうことが重要である。オーバーヘッドが大きくなると、個々の処理にかかる時間が変わってしまうため、プロセス間やプロセスとI/O処理との時間関係が実際とは異なってしまうためである。

筆者は、ソフトウエア的な測定ツールとハードウエア的な測定ツールを併用するハイブリッド・モニタ方式により実働システムの動作状況測定を行なっている。ソフトウエア的な情報を低オーバーヘッドで測定するためである。測定対象システムは、33MHzのMC68030を搭載したEWS4800/20、OSはunix[6]である。

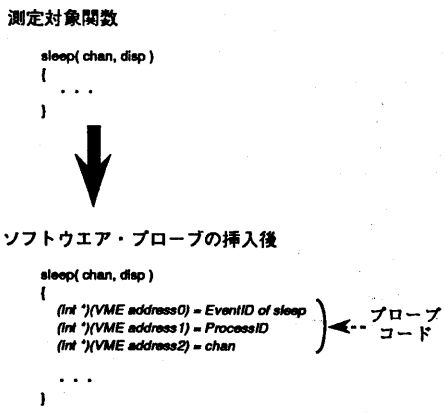

図1:ソフトウエア・プローブの例

イベント	付加情報
プロセスの生成	親プロセスID、新プロセスID
プロセスの終了	プロセスID
プロセスの切り換え	切り換え先のプロセスID
プロセスのsleep	プロセスID、待ち要因
プロセスのwakeup	wakeup対象のプロセスID
システムコールの開始	システム・コールの種類
割り込み例外処理の開始	割り込み例外の種類
システムコールの終了	
割り込み例外処理の終了	
ディスクアクセスの発行	プロセスID、I/Oバッファアドレス
ディスクアクセスの開始	I/Oバッファアドレス
ディスクアクセスの終了	I/Oバッファアドレス

表1：ソフトウエア・プローブで検出するイベント

3．1　ソフトウエア・プローブ

プログラムの実行状況を把握するためには、プロセス切り換えなどのソフトウエア・イベントを検出する必要がある。一般に、このようなイベントは、測定対象計算機で実行されるプログラムの中に埋め込まれたプローブ命令により検出可能である[1]。ただし、この命令に「検出したデータを記録する役割」も受け持たせるソフトウエア・モニタ方式をとると、測定オーバーヘッドが大きくなる点が問題となる。そこで、筆者は、後述するハードウエア・トレーサと組み合わせて測定を行なうハイブリッド・モニタ方式をとることで、低オーバーヘッドでの測定を可能にしている。

ここで行なった測定で使用したソフトウエア・プローブは、EWS4800/20用unix(Rel.6.0)のカーネルに挿入された命令である。この例を、図1に示す。このプローブの役割は、ソフトウエア・イベントを検出するとともに、プログラムの実行状況を解析するために必要な付加情報を、メモリ・マップされた特定の出力ポートに書き込むことのみである。測定データを記録する役割をハードウエア・トレーサに受け持たせたことにより、ソフトウエア・プローブのオーバーヘッドを低くすることができるのである。

このプローブでは、1）プロセスの状態変化などのソフトウエア実行状態の変化、2）入出力アクセスの開始・終了、をソフトウエア・イベントとして検出した。これらのイベントの内容、および付加情報を表1に示す。

3．2　ハードウエア・トレーサ

対象計算機のソフトウエアには手を入れることなく、バスなどのハードウエア信号をモニタする方法で計算機動作を調べる測定ツールがハードウエア・モニタである。このツールの長所は、測定オーバーヘッドが皆無な点であるが、従来のハードウエア・モニタでは、採取可能なデータ量が少ないという問題点があった。そこで、筆者は、サンプル数を従来のロジック・アナライザより2～3桁拡張した8Mステップのハードウエア・トレーサを開発して測定に利用している[7]。

ここで用いた測定系を図2に示す。3.1で述べたソフトウエア・プローブによるVMEバスへの書き込みをサンプル・クロックとして、アドレス・バスとデータ・バス上の値を採取し、これを時系列（トレース・データ）として記録する。また、書き込みの行なわれた時刻、すなわち、ソフトウエア・イベントの発生時刻を記録するため、測定ツールに持たせたハードウエア・タイマの値をタイム・スタンプとして同時に取り込んでいる。時間分解能は100nSである。このタイマにより、OSの機能としてソフトウエア的に実現されているタイマの分解能（～10mS）より、5桁細かい分解能でイベントの発生時刻を記録することが可能である。

4．測定および解析結果

4．1　測定対象ワークロード

測定をベースとした性能評価により得られる結果は、測定対象プログラム（ワークロード）の種類により大きく左右される。特に、システム全体の性能要因を調べる場合には、実働状況を反映したワークロードを測定することが重要である。

ここでは、プログラム開発における主要な作業であるコンパイルおよびリンクの実行を測定対象ワークロードとして評価を行なった結果を示す。このような作業は、ワークステーション上で行なわれる代表的なものの1つと考えたためである。充分長い時間にわたって計算機動作を測定するため、大規模なオブジェクトを作成するジョブを測定対象ワークロードとした。具体的には、EWS4800のunixカーネルを再コンフィグレーションする1分程度の作業である。これは、makeファイルに記述された内容に従って起動されるコマンド（表2）により実行される。コマンドの実行制御方法を変えて2本のトレース・データを採取した。1つは、通常のmakeによるものであり、1コマンドが終了してから次のコマンドを

コマンド	実行回数	処理の概要
cat	1	複数のソースファイルを連結
rm	7	不必要なファイルを消去
as	5	アセンブラソースよりオブジェクトを作成
cc -E	1	マクロ定義を展開
cc -C	3	cのソースファイルをコンパイル
ld	1	複数のオブジェクトをリンク
(other)	3	ロードモジュールのモード変更など

表2：測定対象ワークロードの概要

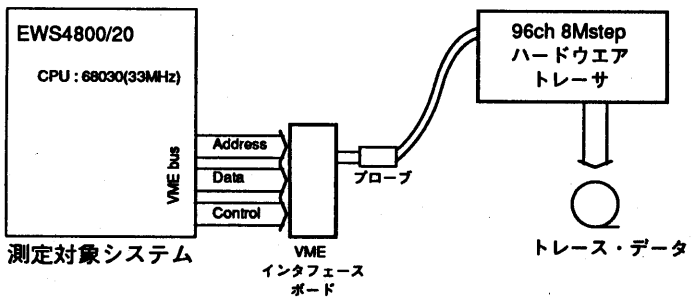

図2：ハードウエア・トレーサを用いた測定系の概要

起動する制御である。この場合、アクティブになるユーザ・プロセスの数は1つに制限される。もう1つは、独立に実行可能なコマンドを並行して実行させる制御である。本稿では、前者をseq-make、後者をpara-makeと呼ぶ。2通りの動作状況を比較することにより、システムとしての性能ネックをより明確にできると考えた。

4．2　ディスクとCPUの使用状況

Gantt profileとして知られる統計データである。ディスクとCPUをリソースとみなし、各リソースがビジーになっている時間を調べたものである。本来のGantt profileは、結果を表示するのに、測定期間に対する割合を用いているが、ここでは、絶対時間を単位とした。

結果を図3に示す。コマンドの実行時間（総ての要素の和）は、seq-makeとpara-makeの間で10%程度の差が生じているが、CPUの動作時間（CPU onlyとCPU and DISKの和）およびディスクの動作時間（CPU and DISKとDISK onlyの和）については、両者に大きな差はない。すなわち、para-makeでは、CPUとディスクが同時に動作する時間が増えたため、全体の処理時間を短縮できたことがわかる。seq-makeにおいて、CPUとディスクが同時に動作している時間が少ない、ということは、CPUとディスクが交互にアクティブになって処理を進めていることを意味している。従って、1プロセスのみが動作しているような状況では、CPUとディスクの並列動作は期待できないことがわかる。

また、ディスクの動作時間は、CPUの動作時間より若干多いことから、このコマンドの実行時間は、CPUよりもディスク性能に依存していることがわかった。コマンドの実行時間を短縮するためには、CPUの性能を上げるよりも、同程度にディスクの性能を上げるほうが、効果は若干高いと予想できる。

4．3　プロセスの実行待ち要因

unixでは、通常、プロセスは親プロセスのforkシステム・コールにより生成され、exitシステム・コールにより消滅する。この間、プロセスは、CPU上で走行したり、ディスクをアクセスすることにより処理を進めていく。すなわち、CPUやディスクは、プロセスを実行するために必要な資源であると見なせる。また、このような処理の実行を直接行なうような資源以外にも、排他制御を行なうためのロックも資源の一種と考えられる。プロセスの実行に必要な資源が他のプロセスにより使用されているときには、そのプロセスは資源を獲得できるようになるまで資源待ち状態に陥る。

筆者は、プロセスが資源待ち状態に陥っている期間もオーバーヘッドの一種と考え、これについて解析を行なった。解析内容は、各プロセスが生成されてから消滅するまでの間に、1）CPU、2）ディスク、3）ロック資源、に関する待ちの時間がどの程度生じているのかを調べたものである。プロセスがCPU待ちに陥るときは、プロセスが自らsleep（システム・コールではなく、カーネル内部の関数）を行なうことなく他のプロセスに切り換えられたときである。この状態は、再び当該プロセスの実行を開始

	CPU使用中	CPU待ち	CPUは使用しない
ディスク使用中	CPUrDISKr	CPUwDISKr	DISKr
ディスク待ち	CPUrDISKw	CPUwDISKw	DISKw
ディスクは使用しない	CPUr	CPUw	wait another resource

表3：プロセス状態の分類

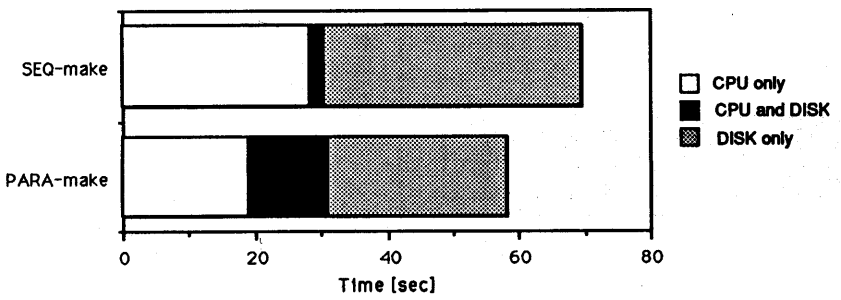

図3：CPUとディスクの使用状況(Gantt chart)

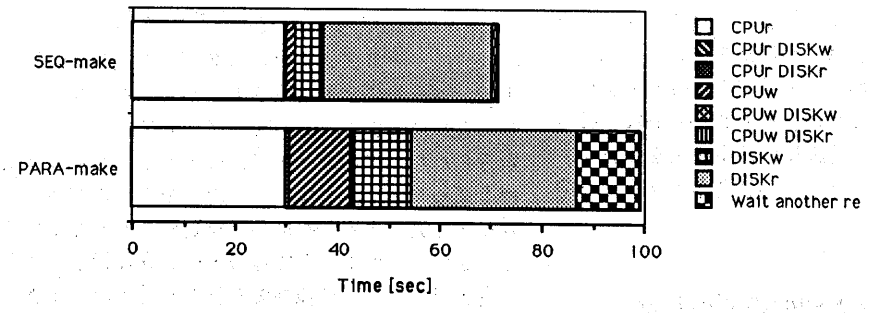

図4：プロセス実行状況の解析（全プロセスの合計）

するときに解除される。また、ディスク待ちに陥るときは、プロセスが発行したI/O要求を直ちに実行できず、I/O発行待ちのキューにつながれたときである。この状態は、問題となっているI/O処理が起動されたときに解除される。ロック資源待ちは、sleepの実行により開始され、他プロセスからのwakeupにより解除される。この場合、どのような資源を待っていたのかは、sleep関数に与えられる引き数から判別できる。

ここで行なった解析では、個々のプロセスがCPUとディスクの使用状況をもとに、プロセスの状態を表3に示す通りに分類し、各々の状態にいる時間を調べた。全プロセスの結果を合計したものを図4に示す。para-makeはseq-makeより、コマンドの実行時間は短いが、プロセスの生成から消滅までの時間の合計は長くなっている。これは、複数のプロセスが並行して処理を行なっているため、あるプロセスが待ち状態に陥っている時間と他のプロセスがCPUやディスクを用いて処理を行なっている時間や、プロセスの待ち時間同志の重複が生じるためである。

seq-makeで資源待ち時間が少なくなっている理由は、4.1で述べたとおり、seq-makeではアクティブなプロセス数が1つであるため、他のプロセスとの間で資源の競合を起こすことがないためである。これに対し、para-makeでは、複数のプロセスが動作することから資源の競合が発生し、資源待ち時間が多くなっている。特に、CPUやディスク以外の資源、すなわち、排他制御のためのロック待ち時間が多くなっている点が注目される。

プロセスの待ち時間を詳細に解析した結果を図5に示す。排他制御のためのロック待ち時間の大半は、ファイル・システムを管理するためのデータ（inode）に対するロックを確保するための待ちによるものであることがわかった。プロセッサ待ち時間は、マルチプロセッサ構成を採用することにより削減できるのに対して、ディスク待ち時間やinode待ち時間は、簡単には削減できない。すなわち、将来のワークステーション、特にマルチプロセッサ構成になった場合は、ファイル系の性能がシステム性能を左右する重要な要因になる可能性が大きいといえるのである。

4．4　OSのオーバーヘッド

OSを実行している時間を削減するためには、種々のOSサービスの内、費やした時間の大きいものを調べることが重要となる。ここで行なった解析では、OS（unixカーネル）の実行時間を、1）ファイ

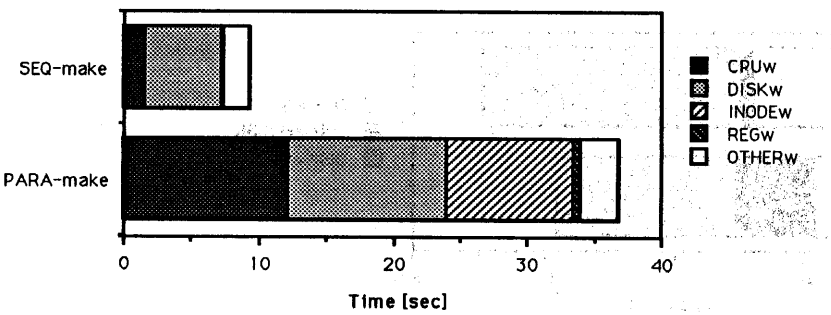

図5：　プロセス待ち時間の解析

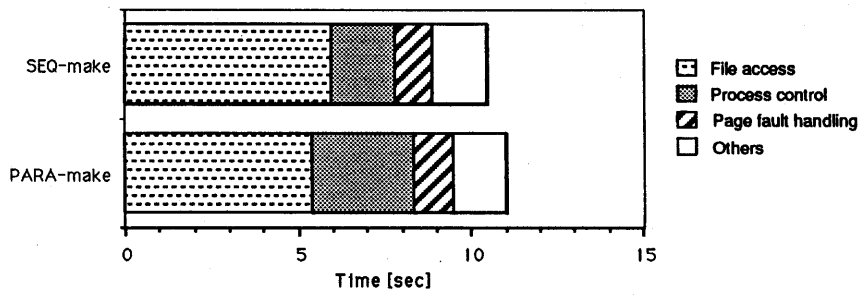

図６:カーネル実行時間 の解析

ル関係のシステム・コール処理時間、２）プロセス管理関係のシステム・コール処理時間、３）その他のシステム・コール処理時間、４）ページ・フォルトの処理時間に分類し、内訳を調べた。
結果を図６に示す。OSの実行時間は約１０秒であり、CPU 時間全体（約３０秒）の1/3であった。このうち、1/2以上がファイル関係のシステム・コールを処理する時間である。すなわち、OSのサービス時間からみても、ファイル系の性能は、システム背能上、重要であるといえる。

５．おわりに

システム性能のネックを明確にすることを目的に、実働計算機システム動作の測定、および、測定データの解析を行なっている。ハイブリッド・モニタ手法によると、測定オーバーヘッドが小さいため、個々のディスク・アクセスやシステム・コールのレベルで詳細な動作を測定することができた。
unixワークステーションを対象に、コンパイラ、リンカの実行状況についてケース・スタディを行なった結果、このプログラム実行に関して、ファイル系の性能とプロセッサ性能は、ほぼ同程度に重要であることがわかった。
この方法によると、システム性能上、重要な要因を明確にできるため、計算機システム全体のアーキテクチャを検討するのに有用である。今後は、ファイル系の動作を、より詳細に解析していくとともに、他の部分の動作も調べていく予定である。

参考文献

[1] Philip Heidelberger and Stephen S.Lavenberg, "Computer Performace Evaluation Methodology," IEEE trans. on Computers,1984,Vol.c-33,No.12,pp.1195-1220.
[2] SPEC News Letter vol.1 Issue1,1989 fall および vol.2 Issue 1,1990 winter.
[3] 「特集：ベンチマーク」、情報処理、vol.31、no.3、1900年3月.
[4] 門田、「SPECベンチマーク、コンピュータをシステム・レベルで評価」、日経エレクトロニクス、1990年４月３０日号、no.498、pp.277-287.
[5] Domenico Ferrari,Giuseppe Serazzi and Alessandro Zeigner,Measurement and Tuning of Computer Systems,Englewood Cliffs, NJ:Prentice-Hall,1983.
[6] Maurice J.Bach,The Design of th UNIX Operating System,Englewood Cliffs, NJ:Prentice-Hall, 1986.
[7] 堀川、大鷹、大野、加藤、「ハードウエア・トレーサを用いた計算機アーキテクチャ評価システム」、情報処理学会OS研究会報告、87-OS-34-3、1987年2月.