イベント・トレース・フレームワークの適用による性能問題の解明

堀川　隆
NEC ネットワーク開発研究本部

イベント・トレース手法は、測定期間内に対象計算機内で起きた現象をつぶさに調べられるため、特に原因のはっきりしない問題を解明する際に有用である。しかし、これまでは個別の問題に応じて ad hoc に利用されることが多く、幅広く活用されてきたわけではなかった。本稿では、イベント・トレース手法により性能分析を行なうために提案したフレームワークに基づいて、原因が不明もしくは推定を誤る可能性の高い性能問題（２種類）の分析を行なった結果を報告する。性質の異なる性能問題について原因を迅速に特定できたことにより、このフレームワークに基づく性能分析の汎用性を示すことができた。

Application of Event Trace Framework for Performance Problem Solutions

Takashi Horikawa
Network Development Laboratories, NEC

Event tracing technique has strong merit in performance analysis; event trace can tells us what happened in the measured system in detail. In spite of this merit, it has not been used versatilely but rather in ad hoc manner. To propose its versatility, this paper applies the event trace framework that the author proposed to difficult performance problems, and then successfully solved them. Since the framework worked out well in the bottleneck detection for the two different types of performance problems, the framework has possibilities as a fundamental baseline for performance analysis.

１．はじめに

問題解決の第一歩は原因の絞込みや特定である。これは、IT システムに限ったことではなく、どのような問題であっても成立する普遍的な真理と思われる。原因の絞込みや特定を間違うと、以降に実施する解決案の策定・適用の作業が無駄になってしまうため、この第一歩は重要である。

IT システムの性能問題では、原因の特定とはボトルネックを見つけることと考えてよい。通常のケースでは、CPU、ディスク、ネットワーク、メモリといった物理資源がボトルネックであり、かつ、それらの使用率はシステム負荷に対してほぼリニアに変化するという素直な性質を示す。このような場合は、物理資源の使用率を調べることでボトルネ

ックを特定できることから、その発見自体は比較的容易である。このため、エンジニアは問題の発生直後からボトルネックに的を絞った対策を施すことが可能となる。

一方、稀ではあるが、物理資源の使用率測定ではボトルネックが判明しない性能問題も存在する。原因（ボトルネック）がエンジニアの予想を超えた所にあることが多く、かつ、個々の問題で所在が異なっていることから、ボトルネック特定に多大な時間を要する難問となる。現実には、ボトルネックを解明できないまま、考えられる対策を手当たり次第に施してみる、という非効率な作業を行なわざるを得ない状況となることが多い。つまり、遭遇することは殆どないが、一旦つまずくと解決には多大な労力を浪費するため、発生頻度の割には重要視すべき問題といえる。

本論文では、このような特異な性能問題のボトルネック発見に、筆者が提案したイベント・トレースのフレームワーク[1]を基礎とする性能分析が有用であることを示す。フレームワークが規定するイベントの時系列を採取して分析し、処理の乱れを検出する点がポイントである。２章では、フレームワークと、それが規定するソフトウエア・イベントを説明する。３章では、ボトルネックを解明したケース・スタディー（２種）を示し、４章でまとめる。

２．フレームワーク

端的には、計算機の性能問題とはユーザからの処理要求（トランザクション）に対するレスポンス時間が予め想定しているよりも大きくなってしまうこと、ボトルネックとはレスポンス時間を増大させている原因といえる。すなわち、性能問題を解決する第一歩は、トランザクション処理時間を増大させている原因を明らかにすることといえる。この考えに基づき、各トランザクションの処理時間内訳を分析するのが、本フレームワークのコンセプトである。

２．１　処理モデルと基本イベント

処理時間の内訳を測定・分析する土台として、UML[2]の一チャートであるシーケンス図をベースとする処理モデルを導入し、基本イベントを抽象的なレベルで規定する。この処理モデルの概要を以下に示す。

- トランザクションの処理は、オブジェクトによって進められる。最も基本的なオブジェクトは、CPU 資源を使って処理を進めるプロセスである。
- ユーザから要求されるトランザクションの処理は、プロセス[1]から開始し、プロセスで終了する。
- プロセスは、必要に応じて他のオブジェクト（他プロセス、ディスクなどの I/O 装置、他の計算機システム）に処理の一部を実行させる（図１）。
- オブジェクトの基本動作は、他のオブジェクトから処理要求を受け取り、必要な資源を確保して処理を開始し、完了時に処理結果を要求元に返すことである。

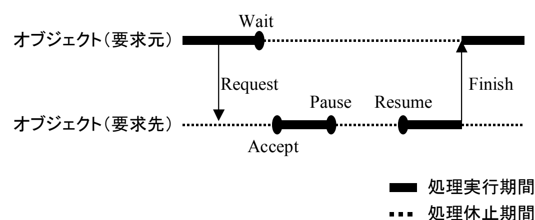

図１：他オブジェクトへの処理要求に関する動作モデル

図１に示した Request、Accept、Finish（必須）、Wait, Pause, Resume（オプショナル）が、この処理モデルにおける基本イベントであり、これらの時系列であるイベント・トレースを採取・分析することで、トランザクション処理時間の内訳を調べることが可能となる。

２．２　測定対象イベント

ここでは、抽象レベル（オブジェクト非依存）の基本イベントと、測定対象となる具体的なイベントとの関係、および、実際のオペレーティング・システム（UNIX[3], Linux[4]）での検出場所を示す。

---

[1] 抽象レベルではスレッドも同義。

a) プロセスに関するイベント

プロセスの CPU 資源使用に関する状態遷移図（図２）に含まれる総ての状態遷移が測定対象イベントである。これらのイベントは、主にプロセスのスケジューラで検出する。

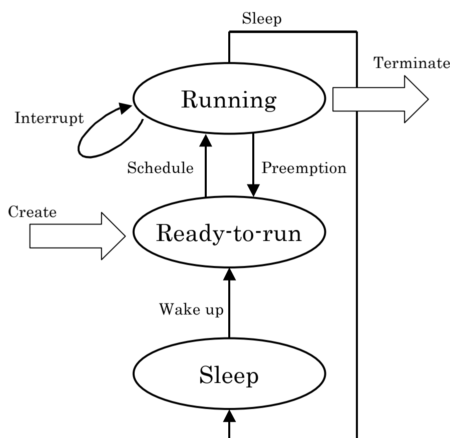

図２：プロセスの状態遷移

UNIX プロセスの状態遷移図[3]を簡略化たもの

b) ディスクに関するイベント

ディスク・アクセスに関するイベントであり、プロセスからのアクセス要求、アクセス開始、アクセス終了が基本的なイベントであり、デバイス・ドライバで検出する。同期アクセスの場合、wait イベント（プロセスがディスク・アクセス完了を待つため sleep 状態に遷移する）イベントも発生する。このイベントはプロセス・スケジューラで検出する。

c) 通信に関するイベント

通常、オペレーティング・システムが提供する共通のメカニズムにより、種々の目的（処理要求、処理終了）のプロセス間通信が行なわれるため、測定段階で抽象的な意味を付与することは困難である。このため、測定では、通信操作（送信・受信）イベントを検出し、抽象的な意味付けはトレース・データの解析段階で行なうようにした。

２．３　性能指標

トランザクション処理を表すシーケンス図には、処理の開始時点と完了時点を繋ぐ線（処理フロー）が存在する。問題発生時のイベント・トレースを解析してシーケンス図を作成し、一連の処理フローに含まれる遅れ時間（下記）を求めれば、ボトルネックを解明することができる。

・オブジェクト処理の開始（再開）待ち時間
・オブジェクト間通信に要する時間

また、複数のトランザクションを並行して処理する場合にのみ発生する性能問題を解明するためには、単一のトランザクションを処理する動作との比較が有効である。この比較で用いる性能指標を以下に示す。

・オブジェクトの処理時間
・オブジェクト間通信の回数、データ量

３．事例

提案するフレームワークを適用し、複数のトランザクションを並行して処理する場合に発生した性能問題（２種類）を解明した。なお、これらの性能問題は、いずれも Linux 2.2 系で発生したものである。

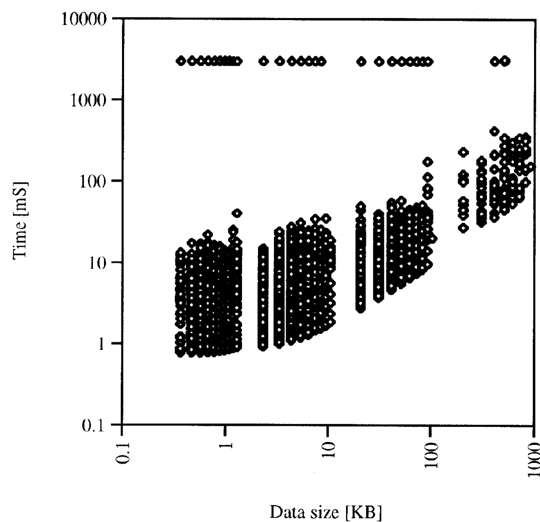

図３：各トランザクションの処理時間とデータ量の関係

３．１　Web ベンチマーク[5]

a) 問題の概要

Web サーバ１台、負荷発生用クライアント１台というシステム構成で SPECweb'96 ベンチマークを実行させた所、サーバとクライアントの両方に Linux 2.2 系を用いた場合の

み性能が悪かった。サーバとクライアントのどちらか一方をLinux 2.4系（当時は2.3系）にすると問題は発生しなかった（ネットワークのスループット値から期待される性能に近い値となる）ため、ボトルネックの所在が不明であった。
各 web トランザクションが要求したファイルのサイズ（転送サイズ）とレスポンス時間の関係をクライアント側で調べた（従来手法）ところ、特定の問題トランザクションのみレスポンス時間が延びている現象が確認できた（図３）が、それ以上の分析は実施できなかった。

b) 処理モデルと測定内容

まず、クライアントとサーバの両者を含むWebトランザクションの処理モデル（図４）を設定した。次に、図中の７点によって区切られる６区間の経過時間を測定した。サーバとクライアントにまたがっている区間もあったため、両マシンから同期させてイベント・トレースを採取し、転送サイズと各区間の経過時間の関係を調べた。

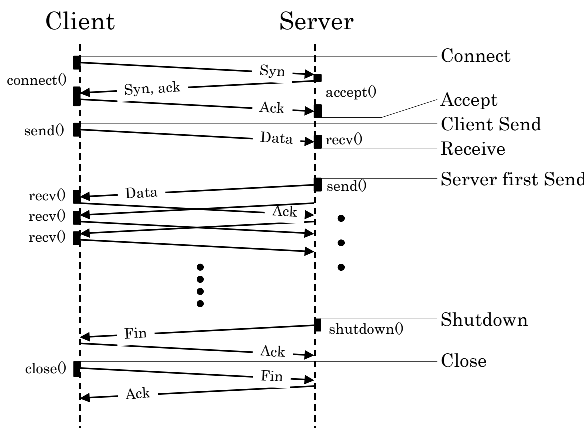

図４ ：Webトランザクションの処理モデル

c) 分析結果

分析の結果、問題トランザクションの処理時間が延びているのは、Connect－Accept区間であることが判明した（図５）。

そこで、この区間を更に詳細にモデル化（図６）した上でイベント・トレースを採取し、問題トランザクションの処理時間が延びる原因箇所の特定を試みた。この結果、Linux 2.2系では、TIME_WAIT状態のポートに対してクライアント・マシンがコネクション要求（Synパケット）を行なうと、サーバは応答（Syn, ackパケット）を返さないことが分かった。このような要求が生じるのは、クライアントが高頻度でweb要求を行なうため、同じポート番号を再利用するまでの時間がTIME_WAIT 時間よりも短くなっていたためである。

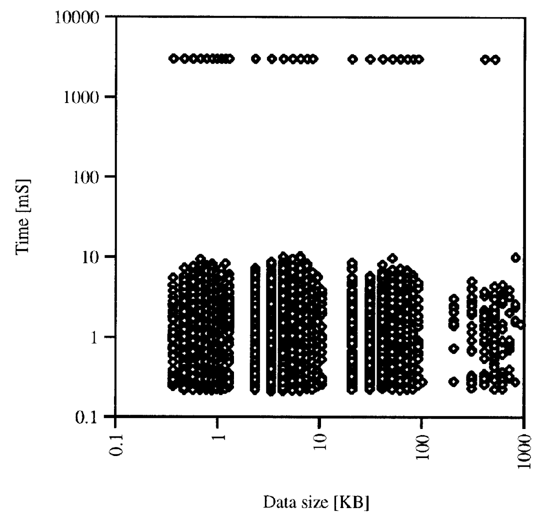

図５ ：Connect-Accept区間の
経過時間とデータ量の関係

他の区間には経過時間が 1 秒を超えるトランザクションは見られなかった。

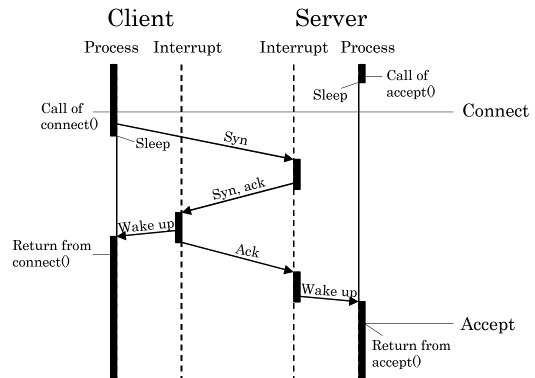

図６ ：Connect-Accept区間の
詳細な動作モデル

クライアントにLinux 2.4系を用いた場合に問題が生じなかったのは、利用可能なポー

ト番号の範囲（クライアント側）が Linux 2.2 系よりも大きかったため、同じポート番号を再利用するまでの時間が長くなっていたためであることも分かった。この結果より、Linux 2.2 系のクライアントであっても、ポート範囲を広くすることで問題が解消できるとの見通しが得られ、この方法で実際に回避することもできた。

３．２　複数ファイルの並行読み込み[6][7]

a）問題の概要

Linux 2.2 系を搭載したシステムにおいて、同一ディスク上に存在する複数の大容量ファイルを並行にアクセスすると、１ファイルづつ順にアクセスした場合よりも極端に時間がかかった。

表１：各処理の実行時間　[単位：秒]

Kernel release	2.2.17	2.2.17
実行方法	逐次	並行
処理 i 実行時間	6	564
処理 ii 実行時間	6	568
処理 iii 実行時間	6	556
処理 iv 実行時間	6	568
処理 v 実行時間	6	556
処理 vi 実行時間	6	548
処理 vii 実行時間	6	557
処理 viii 実行時間	6	569
処理 ix 実行時間	6	569
全処理実行時間	55	569

処理 i ～ ix は、各々100M バイトのファイルを読み込むプロセスを表す。
全処理実行時間とは、逐次の場合は全処理の合計時間、並行の場合は実行時間の最大値。

同一ディレクトリ（ディスク）上に約 100M バイトのファイルを９個用意し、各々に対して逐次および並行にアクセスさせた場合（表１）、両者の実行時間には 10 倍程度の違いがあった。なお、測定に際しては、ファイル・システムを一旦アンマウントしてから再度マウントすることで、メモリ中にバッファされたデータをクリアしているため、総てのファイル・データはディスクから読み込むようにしている。

原因の予想として、当初、１）メモリ不足によるスワップ操作の頻発（thrashing）、２）ディスク・シーク時間の増大（複数ファイル並行アクセスのため、アクセス対象の disk ブロックが飛び飛びとなる）、が挙げられたが、以下に示す分析の結果、両方とも違っていたことが分かった。

b）処理モデルと測定内容

この処理は、各ファイル読み込みプロセスによるディスク・アクセスが処理時間の大半を占める（＝ボトルネック）ことが明白なため、ボトルネック資源を特定するだけでは問題解明にならない。そこで、逐次と並行アクセスの両者について、ディスク・アクセス時間、回数、データ量、を比較する方法を用いた。

表１の実行時間を調べた際の動作（逐次および並行の２種類）についてイベント・トレースを採取し、上記の性能指標を調べた。

表２：逐次アクセス
[単位：秒、回、K バイト]

	Disk 使用時間	アクセス回数	アクセスデータ量
処理 i	5.985	2001	104956
処理 ii	6.144	1707	104940
処理 iii	6.043	2082	104940
処理 iv	6.061	1692	104940
処理 v	6.055	2076	104940
処理 vi	6.030	2079	104940
処理 vii	6.081	1689	104940
処理 viii	6.102	2076	104940
処理 ix	6.059	1750	104940
合計	54.560	17152	944476

表３：並行アクセス
[単位：秒、回、K バイト]

	Disk 使用時間	アクセス回数	アクセスデータ量
処理 i	64.555	21421	104952
処理 ii	64.021	21568	104940
処理 iii	64.555	21421	104952
処理 iv	64.021	21568	104940
処理 v	61.827	21640	104940
処理 vi	62.056	20998	104940
処理 vii	63.605	21339	104940
処理 viii	63.950	21020	104940
処理 ix	61.894	20137	104940
合計	570.483	191112	944484

c) 分析結果

逐次アクセスと並行アクセスの結果を、各々表２、３に示す。並行アクセスは、逐次アクセスに比べ、アクセス時間とアクセス回数は両者とも 10 倍程度に増大するが、ディスクから読み出しているデータ量は変わらない、という結果が得られた。これより、並行アクセス時の実行時間が増大している原因は、ディスク・アクセス１回当りのアクセス・サイズが小さくなっていることが原因であることがわかった。根本的には、Linux 2.2 系でファイルの並行アクセスを行った場合、アクセスのマージ処理（ディスク上で連続するブロックへの複数アクセスを１つにまとめる）がうまく機能していない疑いが強い。

幸い、Linux 2.4 系の kernel では、この不具合は解消されていることが判明したため、強いて Linux 2.2 系の問題を解消する必要性は薄れた。Linux 2.4 系での逐次と並行アクセスの差異（表４）は、シーク時間増大のためと考えられる範囲内に収まっている。

表４：Linux 2.4 系における
各処理の実行時間 ［単位：秒］

Kernel release	2.4.0	2.4.0
実行方法	逐次	並行
処理 i 実行時間	6	24
処理 ii 実行時間	6	26
処理 iii 実行時間	6	32
処理 iv 実行時間	6	48
処理 v 実行時間	6	53
処理 vi 実行時間	6	62
処理 vii 実行時間	6	73
処理 viii 実行時間	6	78
処理 ix 実行時間	6	81
全処理実行時間	55	81

４．まとめ

本論文では、筆者の提案しているイベント・トレース・ベースの性能分析手法の概要および適用事例を示した。従来手法では解明できない、もしくは、原因の特定を誤った可能性の高い問題（２種類）について原因を速やかに解明できたことは、本手法の汎用性および有効性を示していると考えている。

ここで示した事例（性能問題）の原因は、両者とも OS 内にあったが、同様の問題がアプリケーション・プログラム内に存在しないとは言い切れない。今後、既存コンポーネントの組み合わせでシステムが構築されるケースが多くなり、システム内に性能的なブラック・ボックスが多数存在している状況では、問題原因の特定がますます重要になると予想される。提案手法は、このような状況下で特に有用になるものと期待される。

参考文献

[1] T. Horikawa, A Framework for Performance Evaluation Based on Event Tracing, IPSJ Journal, vol. 42, no. 1, pp. 68-78, Information Processing Society of Japan, 2001.

[2] G. Booch, I. Jacobson, J. Rumbaugh, and J. Rumbaugh, The Unified Modeling Language User Guide, Addison-Wesley, 1998.

[3] M. J. Bach, The Design of the Unix Operating System, Prentice-Hall, 1986.

[4] D. P. Bovet, Understanding the Linux Kernel, O'Reilly & Associates, 2000.

[5] T. Horikawa, A Performance Measurement and Analysis Method based on Event Tracing, CMG 2001 Proceedings, pp. 105-116, Computer Measurement Group.

[6] 堀川、Linux 用ソフトウエア・イベント・トレーサ Linux Conference 2001、http://lc.linux.or.jp/lc2001/papers/SoftwareEventTracer-paper.pdf.

[7] T. Horikawa, Efficient Initial Investigation of Performance Problems Using Event Trace-Based Measurement, CMG 2002 Proceedings, pp. 547-555, Computer Measurement Group.