Vol.2022-OS-154 No.2
2022/3/14

複数の時系列性能指標を組み合わせたシステム性能分析手法の提案

山岡　茉莉[1]　奥野　伸吾[1]　平井　聡[1]　岩田　聡[1]　福本　尚人[1]

概要：アプリケーション高速化には，OS や CPU から得られる詳細なシステム性能情報を収集し，性能低下原因などを推定するシステム性能分析が必要である．しかし，膨大な数の性能指標が存在するため，分析実施者が注目すべき性能指標を正しく特定し，それらを用いて分析を実施することが困難である．これに対し我々は，サーバシステムにおける複数の性能指標値を用いたシステム性能分析手法を提案する．本手法の特徴は，時系列データを入力とし，時刻情報を用いて複数の性能指標を紐づけることで複数性能指標間の関連性を考慮しながら，分析に有用な性能指標を選定し分析対象アプリケーションの特性を効率的に発見していく点にある．本報告では手法の提案および分析時に有用な性能指標の選定例や分析対象アプリケーション特性の発見例を示す．

System Performance Analysis Method by Combining multiple Time-seriesed Performance Data

1. はじめに

アプリケーションの効率的な実行方法を考える上で，アプリケーション実行中のマシンに対する分析は必要不可欠である．特にアプリケーション高速化で行われるチューニング時などでは，OS や CPU から得られる詳細な性能指標に基づくシステム性能分析を行うことで性能低下原因を特定している．

例えば，Netflix は Amazon EC2 上で動作するアプリケーションの性能分析に Linux Perf や eBPF (extended Berkeley Packet Filter) といったシステム性能分析ツールを利用している [1,2]．また，Google は独自の性能分析ツールを開発し自社のデータセンタに対して性能分析を実施している [3]．データセンタは数万台規模の計算ノードから構成されるため，性能や利用率のわずかな改善でもデータセンタ全体に対する効果は莫大である．

システム性能分析の実施には，適切な性能情報を収集する必要があるが，単一のプロファイラだけでも扱える性能指標数は膨大である．例えば，システム性能分析でよく用いられる Linux Perf の一機能である perf stat には，使用しているプロセッサにもよるが，扱える性能指標数は 3,000 個を超えることもある．これにより，現状では分析実施者に多くの知識と経験が求められている．

詳細なシステム性能分析を短時間で進めるために，分析対象アプリケーションの性能に影響を与えないよう低オーバヘッドで計測しつつ分析を行い，性能低下原因となっているアプリケーション特性を効率的に特定する技術の開発が考えられる．これを実現するためには，性能低下原因によるシステムの挙動を裏付ける性能指標を見つけ出す方法が挙げられる．しかしながら，この要求を満たすには，特定のプロファイラに固定せず広範囲なシステム性能を計測可能であることと，システム全体の挙動を表現する詳細かつ多様な性能指標間の変化関係を把握することが必要である．

本稿では，アプリケーションが動作するサーバシステムにおける複数の性能指標値を用いた新たなシステム性能分析手法を提案する．本手法は，時系列データを入力とし，時刻情報を用いて複数の性能指標を紐づけることで性能指標の関連性を考慮しながらアプリケーション特性を発見するための効率的な分析を実現する．また，本手法を実装したシステム性能分析ツール「TIDI (Time-series Data Investigator)」を開発し，サーバシステムへの適用を行い，

[1] 富士通株式会社
FUJITSU LTD.

実システム上でその有用性を確認した．

以下，2 章において関連研究について述べ，次に 3 章において本論文で提案する手法を実装した TIDI の設計と実装方法について詳述する．続いて，4 章において本ツールによる提案手法の実サーバシステムへの適用例と有用性検証を行う．最後に 5 章において本論文を総括する．

2. 関連研究

本章では，既存の性能分析ツールを概観し，我々の目的に対する適合性について議論する．

大規模並列計算システムで利用されている多くの性能分析ツールはアプリケーション実行中に収集した性能情報を時系列データではなく時刻情報なしで管理し，アプリケーションの実行後に分析・可視化する手法を採用している．例えば，スーパーコンピュータ「京」で実装されたプロファイラは CPU 性能データを XML 形式に変換できるように改良され [4]，後継機の「富岳」では Score-P [5] や Vampir [6] といった代表的な性能分析ツールと連携できるようになった．CPU の性能情報を分析しようとしている点でこれらのツールは本研究と類似しているが，時系列の情報を分析できない．TAU [7] や Intel VTune [8] はひとつのツールで性能情報の計測から解析までをサポートしているが，評価対象はアプリケーションの 1 回の実行のみである．

システム上で動作する複数のアプリケーションを対象にハードウェアの性能情報を扱うツールも提案されている．CHAMPVis [9] はハードウェアカウンタから取得したデータに対する Top-Down 方式 [10] の性能分析を Web ブラウザ上で実現する．また，Google-Wide Profiling [11] は OProfile [12] ベースのツールで，データセンタ内の複数のマシンに対する継続的なプロファイリングを可能にする．Google は 2 万台以上の計算ノードから構成されるデータセンタに対して Google-Wide Profiling を用いたデータ収集を 3 年間実施し，データ圧縮やメモリ割り当てなどの処理 6 種が多用なワークロード全体の約 30%を占めていることを発見している [3]．本研究で開発する TIDI は特定のプロファイラに固定せず複数の OSS (Open Source Software) を組み合わせることで，より広範囲のデータを収集して解析できるようにする．

ハードウェアの性能情報は性能分析以外にも使用されている．例えば，PMCTrack [13, 14] は OS スケジューラからハードウェアカウンタへのアクセスを抽象化された API で提供しており，CPU のハードウェアカウンタ値を考慮した OS スケジューリングアルゴリズムを CPU アーキテクチャに依存しない形で実装できる．

分散トレーシングはデータ収集時のオーバヘッド削減に関して本研究と関連性が高い．関連する OSS として Dapper [15] や Zipkin [16]，Jaeger [17] が公開されており，

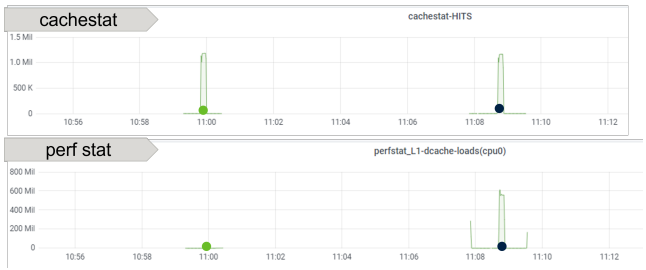

図 1 sysbench 実行時の複数指標値

クラウド環境で広く利用されているマイクロサービスアーキテクチャに基づいたソフトウェアの詳細動作を解析することができる．分散トレーシングでは主にアプリケーションを対象領域としているが，本研究では OS や CPU 等を対象にしている点が異なる．

3. 提案手法および TIDI の設計と実装

本章では，提案する新たなシステム性能分析手法およびそれを実装した TIDI の詳細について述べる．

3.1 時系列データ化された性能指標値を用いたシステム性能分析手法

提案する手法では，複数の計測ツールを同時に使用して任意の時間間隔で複数の性能指標に関して計測を行い，時刻情報を付加した上でデータベース等へ性能指標毎に格納する．その後，可視化された時系列性能指標間の変化関係を確認しながら分析を実施する．ここで計測対象とする性能指標は CPU や OS などに関するものであり，計測範囲の異なる計測ツールを同時に使用して得られる複数性能指標値を組み合わせた分析実施や各性能指標値の変化を時系列的に追うことでアプリケーション特性を発見しやすくする．

図 1 に，実際にサーバシステム上でアプリケーションを実行し，異なる計測ツールの計測を同時に行うことでアプリケーションの特性を容易に推定できる例を示す．実行したアプリケーションはファイル IO を発生させるベンチマーク sysbench [18] であり，指定されたブロックサイズで 4GB のファイルをシーケンシャルに読み込む処理を行った．また，計測ツールは BCC (BPF Compiler Collection) [19] に含まれる cachestat，および Linux Perf に含まれる perf stat を使用した．図中の 11:00 付近のデータはブロックサイズを 16KB に指定して sysbench を実行したときのものであり，11:09 付近のデータはブロックサイズを 1MB に指定したときの結果である．ここで，cachestat では HITS という指標名で OS の page cache ヒット数を計測可能である．また，perf stat では L1-dcache-loads という指標名で CPU の L1 キャッシュロード数を計測可能である．したがって，図 1 から，sysbench はブロックサイズの値を変化させることで page cache のヒット数と CPU キャッシュ

の使用方法に変化が出る特性を持つことを容易に認識できる．

3.2 TIDI の設計と実装

我々は，3.1 節で述べた分析手法をシステム性能分析ツール TIDI (Time-series Data Investigator) として設計および実装を行った．本節ではこれについて述べる．

3.2.1 設計

システム性能分析ツール TIDI は，以下の点を考慮して設計を行った．

(1) 対象 OS として Linux，対象 CPU として複数の IA (Intel Architecture) プロセッサをサポートすること．
(2) 分析対象のアプリケーションの実行結果に影響を与えにくいリアルタイムな計測と分析が行えること．
(3) 機能追加をしやすくすること．

(1) を実現するべく，CPU アーキテクチャ毎に異なる箇所を変数化しておき，計測前に計測可能な計測指標を自動で探索しその結果をもとに計測を行う機能を実装した．例えば，既存の計測ツールである perf stat による計測を行う際には，perf stat で計測できるイベント内容が CPU アーキテクチャ毎に異なることを考慮する必要がある．これに対し TIDI では，perf stat 計測前に perf list を実行しその出力を利用することで，計測時にユーザの計測環境にあった計測内容を指定できるようにした．

(2) については，分析対象のアプリケーションを実行しているデータ収集用のサーバとは別に，データ管理用のサーバを用意することで対応した．具体的には，膨大な数の時系列性能指標値をリアルタイムにデータ管理用のサーバへ転送することで，データ収集用サーバの DISK 容量を圧迫しないようにした．これにより，分析対象のアプリケーション実行結果への影響を抑え，アプリケーション実行中に同時に分析を行うことが可能となる．

(3) は，データ収集および転送に関する部分を共通化するためのフレームワークを作成することで実現した．これにより，新しい計測ツールの取り込みなど機能追加をする際に実装が容易になる．ツール毎に異なる箇所に関しては設定ファイルを作成し，それを読み込む形で設計した．

3.2.2 実装

TIDI の全体構成を**図 2** に示す．TIDI は大きく測定環境と分析環境に分けられる．測定環境は「データ収集サーバ」と「管理サーバ」の 2 サーバで構成され，分析環境は計測結果確認用の GUI 操作可能なマシンで構成される．TIDI は python で実装されており，データ収集サーバにて python ファイルを実行することで設定確認などの計測準備および計測を開始する．TIDI は以下の流れで処理を行う．

(1) 実行環境など必要な設定を読み込む
(2) ユーザが計測に使用したいツールをインタラクティブに指定する

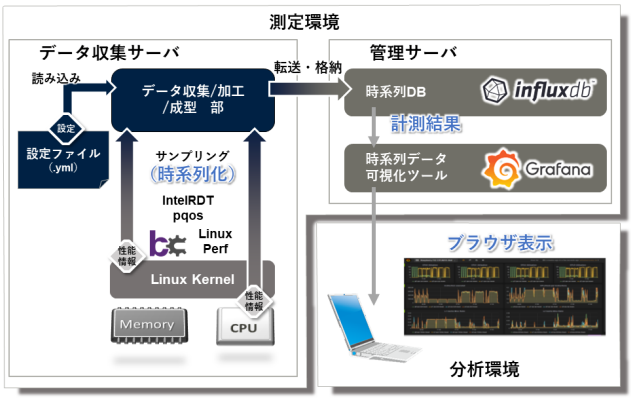

図 2 TIDI のシステム構成

(3) 計測ツール毎の設定を読み込み，必要があればインタラクティブにパラメータ等の設定を行う
(4) 計測ツール毎に計測，管理サーバへのデータ転送，および InfluxDB へのデータ格納を開始

ここで，パラメータは yaml ファイルで管理しており，これを編集することで自由に設定を変更することが可能である．また，InfluxDB に格納された性能指標値は Grafana を用いることで，分析環境上で動作するブラウザにて視覚的に確認することができる．

4. TIDI の評価

本章では，時系列データを入力としたシステム性能分析手法を実装した TIDI により，本手法の有用性を評価する．

4.1 評価環境

TIDI 評価時に使用したサーバシステム環境および TIDI 実装時に使用したソフトウェアを**表 1** に示す．本環境は，測定環境に含まれるデータ収集サーバ 1 台と同スペックの管理サーバ 1 台，および分析環境に含まれる Grafana 表示結果確認用マシンで構成される．本評価では**表 2** に示すとおり CPU に関連する 4 種類のデータを収集した．なお，紙面の都合上，perf stat と pqos [20] の結果についてのみ示す．

また，分析対象アプリケーションとして LMbench [21,22] の lat_mem_rd を使用した．LMbench はメモリバンド幅やメモリレイテンシなど，OS とハードウェアシステムの性能を計測するためのツールである．lat_mem_rd は LMbench に含まれるメモリレイテンシ計測用ベンチマークであり，Array Size の配列の範囲内にて指定された stride 幅でのメモリアクセスを繰り返す処理を行う．指定された最大 Array Size まで Array Size を増加させながら計測を行うことで，様々な Array Size でのメモリレイテンシを計測する事が可能である．**図 3** は lat_mem_rd を実行する際に stride を変化させた際の結果例を示しており，stride の値を変化させるとメモリレイテンシ計測結果も変化する傾向

表 1　評価環境

測定環境	CPU	Intel(R) Xeon(R) Gold 5120 CPU
		14 core (Hyper-Threading : OFF)
		動作周波数：2.20GHz （ターボ・ブースト利用時 最大周波数：3.20GHz）
		L1：I-cache と D-cache がそれぞれ 32KB/core
		L2：1,024KB/core
		L3：19.25MB (shared)
	OS	Linux 5.8.0-44-generic x86_64
	ディストリビューション	Ubuntu 20.04.3 LTS
	開発言語	Python 3.7.6
	時系列データ転送 API	python3-influxdb 5.2.0-1.1
	時系列データベース	influxdb 1.8.3-1
	時系列データ可視化ツール	grafana-enterprise 7.2.2
分析環境	OS	Windows10 20H2

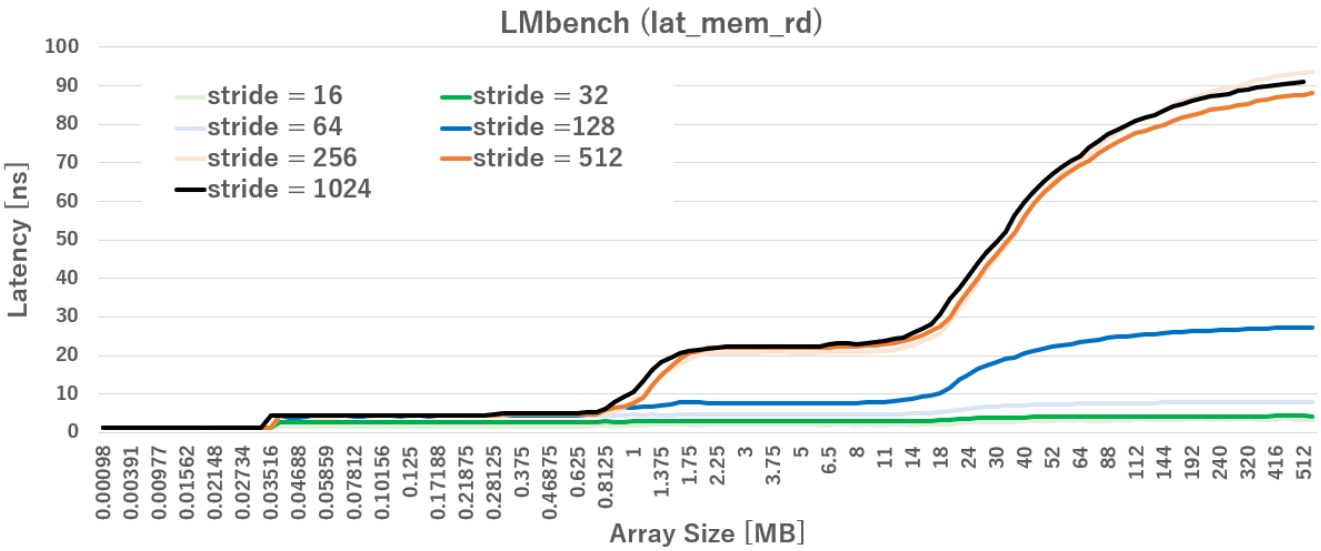

図 3　lat_mem_rd 実行結果

表 2　収集した計測内容

計測内容	データ収集元
CPU 使用状況	/proc/stat
page cache	cachestat
PMU カウンタ値	perf stat
CPU キャッシュ使用状況	pqos

がある事が分かる．

以降の実験では，複数の性能指標を用いて，stride 値の変化によりメモリレイテンシが変化した原因の特定をすることで lat_mem_rd のアプリケーション特性を発見した．なお，本評価では perf stat と pqos から得られる性能指標値の収集に加え，lat_mem_rd 自体の出力も TIDI で収集しデータベースに転送した．これは，分析時に「どのタイミングでどの Array Size に関するメモリレイテンシ計測をしているか」を明確にするためである．

4.2　アプリ特性の発見

表 1 のサーバシステムのうちデータ収集サーバ上で lat_mem_rd を実行し，TIDI による複数性能指標値の収集および分析を行うことで lat_mem_rd の特性を発見した．本評価では最大 Array Size を 512MB で固定し，stride を 64B，128B，256B の順に変化させ，以降に記載した手順で lat_mem_rd の CPU キャッシュ使用特性を発見した．

(1) 複数の計測ツールを用いて，システム全体を対象とし広範囲を計測
(2) ベンチマーク実行状況によって変化するメモリアクセス領域を特定

まず，(1) では，異なる計測ツールである perf stat の TopdownL1 と pqos を同時に計測し，lat_mem_rd がメモリネックなアプリケーションであることを裏付けた．**図 4** および**図 5** はそれぞれ，stride を変化させながら lat_mem_rd を実行している際に TIDI で TopdownL1 と pqos について計測した結果を示している．図 4 からは，stride 値に関わらず Backend Bound の値が 1 付近を保つ傾向にあることが分かり，図 5 からは，stride 値の変化に伴ってローカルメモリバンド幅を示す MBL[MB/s] の値や CPU キャッ

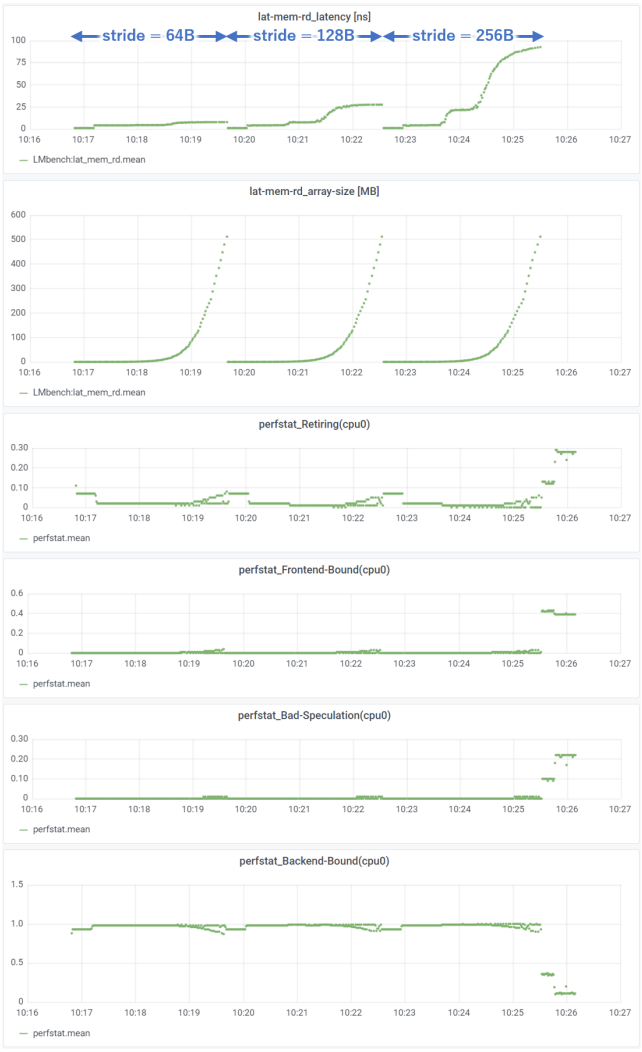

図 4 手順 (1) の計測結果（TopdownL1）

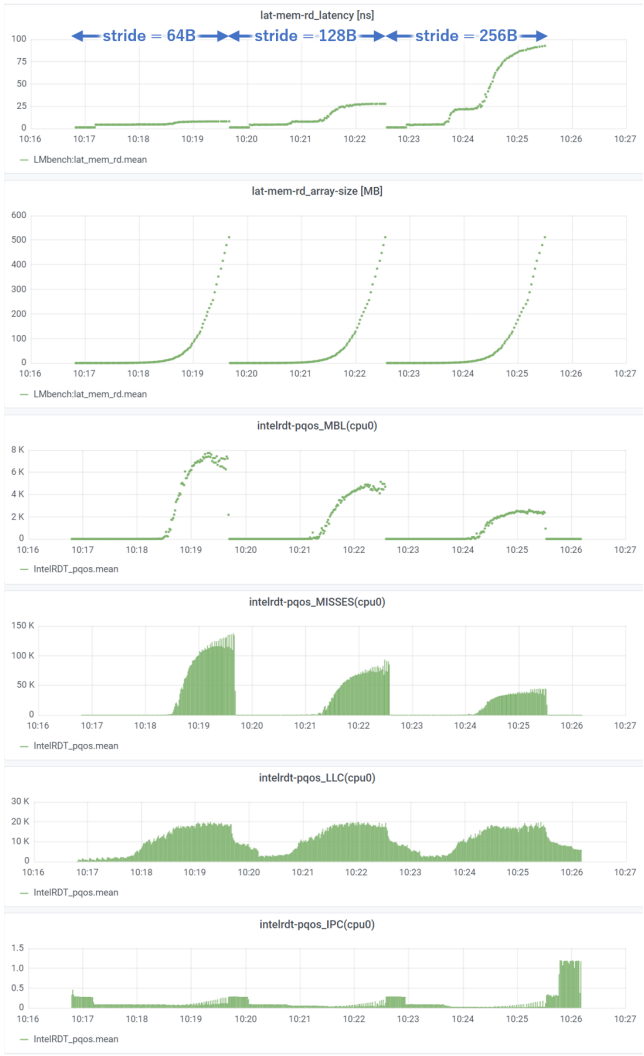

図 5 手順 (1) の計測結果（pqos）

シュのミス数を示す MISSES の値が変化していることが分かる．Backend Bound の値が 1 に近いときはメモリネックとなっている可能性が高いことから，lat_mem_rd はメモリネックなアプリケーションであると言える．また，L1 キャッシュのようなより低層のキャッシュメモリは低レイテンシで応答可能なため単位時間当たりのメモリアクセス回数が増加することをふまえると，stride 値を変化させるとメモリバンド幅が変化するのは，stride 値の変化に伴い CPU キャッシュの使用領域が変化するためであると推測できる．

次に (2) にて，キャッシュアクセス領域の変化を時系列で追うために CPU キャッシュに関する複数の性能指標の計測結果を比較することで，stride 値の変化に伴い CPU キャッシュの使用領域が変化していることを確認した．確認のために計測した性能指標は perf stat の mem_load_retired.l1_hit，mem_load_retired.l2_hit，mem_load_retired.l3_hit であり，計測結果を**図 6** に示す．ここで，mem_load_retired.l1_hit はメモリアクセス要求時に L1 キャッシュがデータ保持元として該当していた回数を示しており，同様に mem_load_retired.l2_hit は L2 キャッシュが該当していた回数，mem_load_retired.l3_hit は L3 キャッシュが該当していた回数を示している．図 6 より，L1 キャッシュ使用状況は stride 値によらないが，L2 キャッシュと L3 キャッシュに関しては使用状況に変化があることが分かる．使用状況変化の 1 点目は L2 キャッシュの使用状況である．stride = 64B のときは array size = 512MB まで L2 キャッシュを使用しているが，stride = 128B のときは array size = 20MB 程度まで，stride = 256B のときは array size が非常に小さい段階までしか L2 キャッシュを使用できていない．2 点目は，L2 キャッシュと L3 キャッシュの併用状況である．stride = 64B のときは L3 キャッシュを使用しておらず，stride = 128B のときは array size = 20MB 程度までは L2 キャッシュと L3 キャッシュを併用している状態である．一方で stride = 256B のときは L2 キャッシュと L3 キャッシュの併用は見られない．

5. まとめと今後の課題

本論文では，アプリケーションが動作するサーバシステ

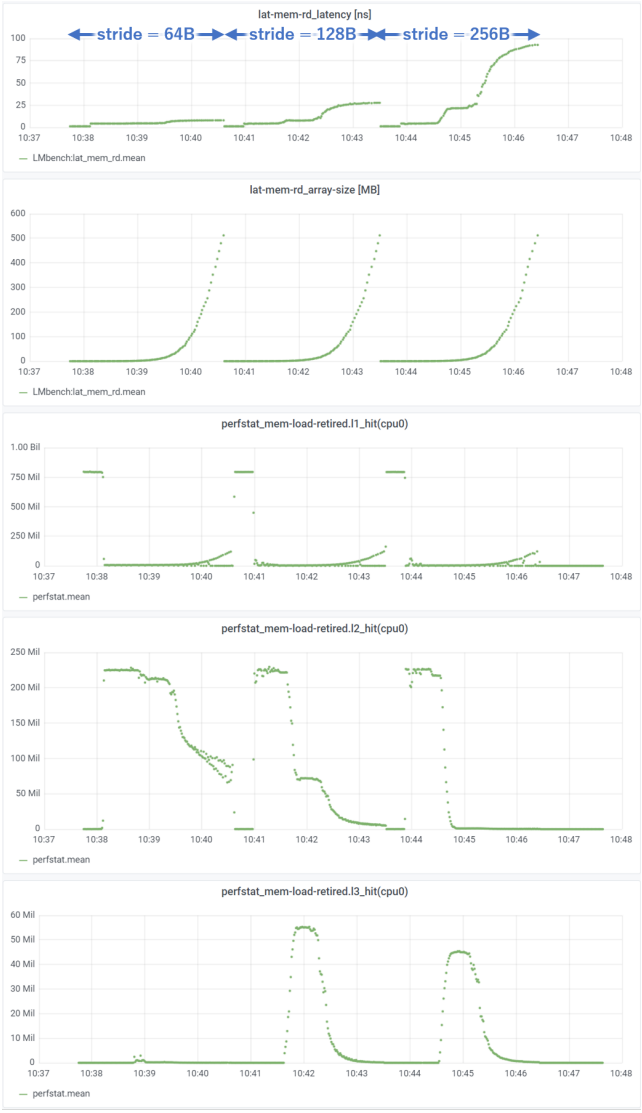

図 6 手順 (2) の計測結果

ムにおける複数の性能指標値を用いた新たなシステム性能分析手法を提案した．これにより時刻情報を用いて複数の性能指標を紐づけ，複数性能指標間の関連性を考慮したことでより詳細にアプリケーション挙動を追うことが可能になった．実際に，本手法を実装したシステム性能分析ツール TIDI を開発し，実システムに適用することでその有用性を確認できた．

本稿では，lat_mem_rd を分析対象のアプリケーション例として評価を行った．このベンチマークは同一のメモリアクセス処理を繰り返す単純な仕組みだったため，目視でも性能指標間の関連性を確認でき，アプリケーション特性の発見を容易に行うことができた．しかしながら，より複雑な処理を行うアプリケーションの場合は，目視での性能指標間の関連性確認によるアプリケーション特性の発見は困難となる．今後はより複雑な処理を行うアプリケーションも分析対象として取り組み，指標値の関連性を基に分析時に有用な指標値を自動で抽出するよう TIDI の改良を行う．

また，ARM アーキテクチャなど，Intel 以外のアーキテクチャでも TIDI を使えるように改良を行う．これにより，異なるアーキテクチャ間でのアプリケーション特性比較などができるようになる．

参考文献

[1] Gregg, B.: Using Linux perf at Netflix, Presented at Kernel Recipes 2017 (`https://www.brendangregg.com/Slides/KernelRecipes_Perf_Events.pdf`) (2017).
[2] Gregg, B.: BPF Performance Analysis at Netflix, Presented at AWS re:Invent 2019 (`https://d1.awsstatic.com/events/reinvent/2019/REPEAT_1_BPF_performance_analysis_at_Netflix_OPN303-R1.pdf`) (2019).
[3] Kanev, S., Darago, J. P., Hazelwood, K., Ranganathan, P., Moseley, T., Wei, G.-Y. and Brooks, D.: Profiling a Warehouse-Scale Computer, *Proc. 42nd Annual International Symposium on Computer Architecture, ISCA 2015*, pp. 158–169 (2015).
[4] 阿部文武，中村朋健，志田直之：プロファイラにおける汎用的な CPU 性能情報と表示機能，情報処理学会研究報告，Vol. 2016-HPC-153, No. 18, pp. 1–8 (2016).
[5] Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A. D., Nagel, W. E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B. and Wolf, F.: Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir, *Proc. 5th International Workshop on Parallel Tools for High Performance Computing*, pp. 79–91 (2011).
[6] Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M. S. and Nagel, W. E.: The Vampir Performance Analysis Tool-Set, *Proc. 2nd International Workshop on Parallel Tools for High Performance Computing*, pp. 139–155 (2008).
[7] Shende, S. S. and Malony, A. D.: The Tau Parallel Performance System, *The International Journal of High Performance Computing Applications*, Vol. 20, No. 2, pp. 287–311 (2006).
[8] Intel Corporation: Intel® VTune™ Profiler, `https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html`.
[9] Pentecost, L., Gupta, U., Ngan, E., Beyer, J., Wei, G.-Y., Brooks, D. and Behrisch, M.: CHAMPVis: Comparative Hierarchical Analysis of Microarchitectural Performance, *Proc. 2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools, ProTools 19*, pp. 55–61 (2019).
[10] Yasin, A.: A Top-Down Method for Performance Analysis and Counters Architecture, *Proc. 2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS-2014*, pp. 35–44 (2014).
[11] Ren, G., Tune, E., Moseley, T., Shi, Y., Rus, S. and Hundt, R.: Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers, *IEEE Micro*, Vol. 30, No. 4, pp. 65–79 (2010).
[12] Cohen, W. E.: Tuning Programs with OProfile, *Wide Open Magazine* (2004).
[13] Saez, J. C., Casas, J., Serrano, A., Rodríguez-Rodríguez, R., Castro, F., Chaver, D. and Prieto-Matias, M.: An OS-Oriented Performance Monitoring Tool for Multicore Systems, *Proc. Euro-Par 2015: Parallel Processing*

Workshops, pp. 697–709 (2015).
[14] Saez, J. C., Pousa, A., Rodriíguez-Rodriíguez, R., Castro, F. and Prieto-Matias, M.: PMCTrack: Delivering Performance Monitoring Counter Support to the OS Scheduler, *The Computer Journal*, Vol. 60, No. 1, pp. 60–85 (2017).
[15] Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S. and Shanbhag, C.: Dapper, a Large-Scale Distributed Systems Tracing Infrastructure, Technical report, Google, Inc. (2010).
[16] Twitter, Inc.: OpenZipkin, `https://zipkin.io/`.
[17] The Linux Foundation: Jaeger, `https://www.jaegertracing.io/`.
[18] Kopytov, A.: sysbench, `https://github.com/akopytov/sysbench`.
[19] The Linux Foundation: BCC BPF Compiler Collection, `https://www.iovisor.org/technology/bcc`.
[20] Aleksiński, M.: intel-cmt-cat, `https://github.com/intel/intel-cmt-cat`.
[21] McVoy, L. and Staelin, C.: LMbench - Tools for Performance Analysis, `http://lmbench.sourceforge.net/`.
[22] McVoy, L. and Staelin, C.: lmbench: Portable Tools for Performance Analysis, *Proc. USENIX 1996 Annual Technical Conference*, *USENIX ATC 96* (1996).