DEIM2022 J34-5

サーバシステムの性能データ収集および転送における効率化手法の検討

飯山　知香† 　平井　聡†† 　山岡　茉莉†† 　福本　尚人†† 　小口　正人†

† お茶の水女子大学　〒 112–8610 東京都文京区大塚 2 丁目 1 － 1
†† 富士通株式会社　〒 211–8588 神奈川県川崎市中原区上小田中 4 丁目 1 － 1
E-mail: †g1820502@is.ocha.ac.jp, ††{ahirai,yamaoka.mari,fukumoto.naoto}@fujitsu.com,
†††oguchi@ogl.is.ocha.ac.jp

あらまし　多数台のサーバのデータを一元的にリアルタイムで分析するため，分析対象のサーバとそれを分析するサーバは分割されることが多い．しかし，そこで用いられる Linux の性能データなどの時系列データはデータサイズが比較的大きく，扱う際のオーバーヘッドが大きくなる可能性があるため，効率的に扱う手法の実現が求められている．そこで，本研究では，サーバシステムの性能データ収集および転送における効率化手法を検討することを目的とする．その達成のため，データ収集用サーバにおいて採取した性能データを別の解析用サーバに転送する際の，リソース利用量やオーバーヘッドを計測・分析した．これらの結果から，効率化のための課題抽出を行い，その適切な改善案として，データ量削減や転送効率化の手法を検討した．

キーワード　空間・時間・時系列データ処理，コンピュータネットワーク，情報システム，システム構築

Consideration of improvement plans to increase the efficiency of performance data collection/transfer for server systems

Chika IIYAMA†, Akira HIRAI††, Mari YAMAOKA††, Naoto FUKUMOTO††, and Masato OGUCHI†

† Ochanomizu University
2–1–1 Otsuka, Bunkyou-ku, Tokyo 112–8610, Japan
†† FUJITSU LTD.
4–1–1 Odanaka, Nakahara-ku, Kawasaki-shi, Kanagawa 221–8588, Japan
E-mail: †g1820502@is.ocha.ac.jp, ††{ahirai,yamaoka.mari,fukumoto.naoto}@fujitsu.com,
†††oguchi@ogl.is.ocha.ac.jp

1. はじめに

クラウド環境をはじめとして，多数台サーバの共有利用や分散処理利用に関する需要が増えてきている．このような環境で，負荷分散およびシステムやアプリケーションのチューニングを行うには，各サーバの低レイヤを含めた性能データを低オーバヘッドで収集してリアルタイムに分析・提示する手法が必要である．

また，多数台のサーバのデータを一元的にリアルタイムで分析する際，分析対象のサーバとその分析を行うサーバは分割されることが多い．しかし，そこで用いられる Linux の性能データなどの時系列データはデータサイズが比較的大きく，扱う際のオーバヘッドが大きくなる可能性があるため，効率的に扱う手法の実現が求められている．

そこで我々は，データ収集用サーバにて CPU や OS から収集した性能データを時系列化し，データ解析用サーバにて時系列性能データをデータベースに格納し分析する手法を検討中である．今回，時系列データの中でもサイズが大きく処理に時間のかかる CPU のプロファイルデータに関し，時系列 DB およびその API を利用して転送するケースを想定した実機評価を行い，課題やオーバヘッドを見積もり，今後のデータ収集および転送の効率化の改善事項を抽出した．

本論文の構成を以下に示す．2 章ではシステムおよびアプリケーションの性能分析手法に関する関連研究について述べる．3 章で性能データ収集/転送におけるオーバヘッドの計測を行

い，効率のよいデータ通信を行う必要性を議論した．最後に 4 章でまとめる．

2. 関 連 研 究

本研究と収集対象とする性能データが類似しているツールとして，Score-P と Vampir が挙げられる．Score-P(Scalable Performance Measurement Infrastructure for Parallel Codes) [1] というツールでは，主に並列 HPC アプリケーションの性能分析を行うことができる．Vampir(Visualization and Analysis of MPI Resources) [2] というツールでは，多数台の計算ノードのプロファイルデータや相互通信データなどを統合解析し可視化することができる．これらのツールでは，アプリケーションの実行後に，各ノードで収集した性能データをファイルベースで収集/分析/可視化するため，本研究が目指す実行時の性能データを時系列データとしてリアルタイムで収集・分析する手法とは異なる．既存研究 [3] では，スーパーコンピュータ京などの独自の CPU 性能データを汎用的なデータ形式に変換し，上記の Score-P や Vampir で扱えるようにするための手法が提案されている．分析しようとしている内容は本研究と類似しているが，データ収集と転送・分析を同時に行うことは困難である．

本研究とデータ収集時のオーバヘッド削減手法などの関連性が高い研究として，分散トレーシングという手法がある．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性能データを収集・分析しているのに対し，本研究では低レイヤを対象としている点が異なる．Google 社が 2010 年に発表した論文 [4] を発端に，Twitter 社が Zipkin [5] を，Uber 社が Jeager [6] を開発した．それぞれ OSS として公開され，多くのクラウド基盤で利用されている．

プロファイルデータをデータベースに取り込み分析する手法が本研究と類似している既存研究 [7] では，Google 社のデータセンターで行われているプロファイルデータの収集・分析手法が解説されている．既存研究 [8] では，上記手法を用いてデータセンターのワークロードが分析されている．数千台のサーバからランダムでプロファイルデータ収集を行い，その情報を元にアプリケーションチューニングを行うと共に，パフォーマンス監視やアプリケーション配備 (アフィニティ) 最適化にも利用している．本研究では CPU や OS 等の低レイヤの性能データをオープンなソフトウェア (OSS) を使用/改良して収集/転送/分析することを目指している．

3. 実　　験

3.1 実 験 概 要

性能データ収集/転送の効率化を検討するにあたり，まず予備実験として，データ通信のオーバヘッドを計測する実験を行い，サーバシステムの性能データを収集/転送/蓄積する際のリソース利用量，オーバヘッドを計測・分析した．環境構築として，データ収集用サーバ (以下収集サーバ) とデータ解析用サーバ (以下解析サーバ) の 2 台サーバを用意した．収集サーバには，Linux Kernel の標準的なイベントデータ収集/トレース機能のフレームワークである perf [9] や，様々なベンチマークを使用して CPU やメモリなどに負荷をかけられるツールである stress [10] を使用した．解析サーバには，収集した大量の時系列データを管理する時系列 DB として InfluxDB [11] を使用した．InfluxDB については [12] で解説されている．収集サーバ，解析サーバ実験環境をそれぞれ表 1 と表 2，各使用技術のバージョンを表 3 に示す．

表 1　収集サーバ環境

機種名	Fujitsu PRIMERGY CX2550 M1
CPU	Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz (× 2CPU)
コア数	14
メモリ	128GB
OS	CentOS 7

表 2　解析サーバ環境

機種名	Dell PowerEdge R620
CPU	Intel(R) Xeon(R) CPU E5-2603 v2 @ 1.80GHz (× 1CPU)
コア数	4
メモリ	16GB
OS	CentOS 7

表 3　各使用技術のバージョン

perf	3.10.0-1160.45.1.el7.x86_64.debug
stress	1.0.4
InfluxDB	1.8.3.x86_64

実験概要を図 1 に示す．収集サーバで収集した CPU 性能データ (perf record によって取得したプロファイルデータ) をデータ転送プログラムを用いて転送し，解析サーバ上の InfluxDB に格納した．実験 1 では，データ収集を 1 ミリ秒単位で 1 秒間行い，転送に要する時間を計測した．データ転送を行う際，収集対象の CPU コア数を変更し，データ転送量による転送時間の変化を比較した．実験 2 では，1CPU コア数分のデータを増加させた際の転送時間を実験 1 と比較するため，データ収集の頻度を 100 マイクロ秒単位に変更して計測を行った．実験 3 では，格納されたデータの圧縮率を見るために，解析サーバ上の DB の増加容量を確認した．実験 4 では，収集サーバ上で実行されるデータ転送プログラム，および解析サーバ上で実行される InfluxDB の CPU 負荷率を計測した．計測中の転送処理を継続させるため，128CPU コア数分のデータを転送した．

収集した CPU 性能データを，InfluxDB に転送可能な形式に変換し，転送するまでの流れを以下に示す．まず収集データから，必要なデータ (時刻，プロセス ID，スレッド ID，実行アドレス) のみを抽出する．収集データは 1CPU コアの 1 秒分のデータであるため，1 ミリ秒単位で収集した場

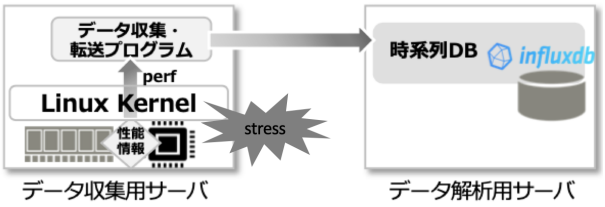

図 1　実 験 概 要

合は 1000 レコード，100 マイクロ秒単位で収集した場合は 10000 レコードのデータが抽出される．次に，抽出データを Python Pandas [13] を利用して DataFrame 形式に格納する．DataFrame 形式に格納したデータのサンプルを図 2 に示す．最後にこのデータを InfluxDB の Python Client モジュールを利用して解析サーバに転送する．上記モジュールには Pandas の DataFrame 形式データを InfluxDB サーバに書き込む API(Influxdb.DataFrameClient) [14] があり，今回はこれを利用している．

時刻(ns単位)	Process ID	Thread ID	EIP(実行アドレス)
2021-12-20T13:18:36.159557443	698085	698085	0x55e3e9dd1151
2021-12-20T13:18:36.160568219	698085	698085	0x7faa24bd7e02

図 2　DataFrame 形式データのサンプル

3.2　結　　果

図 3 に実験 1 でデータ転送に要した時間を示す．1CPU コア数分のデータで転送を実施した際は，データ転送に約 0.051 秒かかった．2，10，20CPU コア数分繰り返して転送を行なった際には，それぞれ約 0.085 秒，約 0.40 秒，約 0.80 秒に増加した (転送時間はすべて 10 回の計測の平均)．つまり，1CPU コア時の転送時間は約 0.05 秒，2CPU コア以上では 1CPU コアあたり約 0.04 秒増加しており，25CPU コア以上の多数コア CPU では perf record の 1 秒間相当のデータを転送するのに 1 秒以上かかることが分かる．繰り返し転送する際，InfluxDB のデータベースへは，measurement(表の名前) を CPU0，CPU1，CPU2，というようにして格納した．1CPU コア数分の転送用データのサイズは約 32KB で変化ないため，それぞれ合計約 64KB，約 320KB，約 640KB 転送された．転送用データのサイズが約 32KB で一定となっているのは，それぞれ 64bit の 4 項目のデータ (時刻，プロセス ID，スレッド ID，実行アドレス) が 1000 レコード分ある 1CPU コア数分の値で不変であるためである．

図 4 に実験 2 でデータ転送に要した時間を示す．100 マイクロ秒単位で CPU 性能データを収集した際には，1CPU コア数分の転送には約 0.21 秒かかった．1 ミリ秒単位でデータ収集および転送を行った時と同様に，2，10，20CPU コア数分の転送も行った結果，それぞれ約 0.57 秒，約 2.0 秒，約 3.9 秒に増加した．つまり，転送データサイズを 10 倍にしても転送時間はそれに比例せず，それぞれ元の 4〜7 倍程度しか増加していないことが分かる．データ転送には InfluxDB の python モジュール API を使用しており，1CPU コアのデータ転送に対して，転

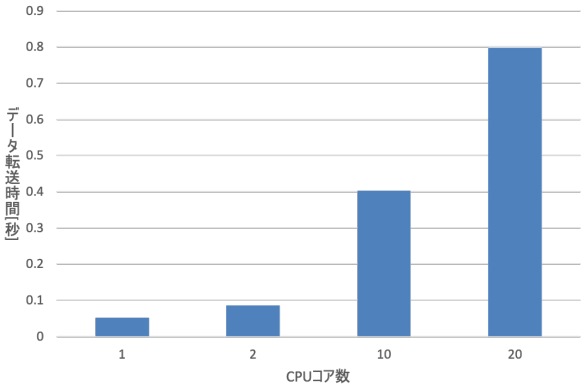

図 3　データ転送時間 (1 ミリ秒単位)

送データのサイズに関わらず 1 度だけ呼び出しを行っている．よって，データ本体の転送以外の，InfluxDB を載せたサーバ (解析サーバ) とのネゴシエーション等のコストが大きく，時間がかかっていると推察される．データ収集の頻度が元の 10 倍細かくなっているため，1CPU コア数分の転送用データサイズは 10 倍になり，CPU コア数を増加させた際の合計データサイズもそれぞれ元の 10 倍になっている．

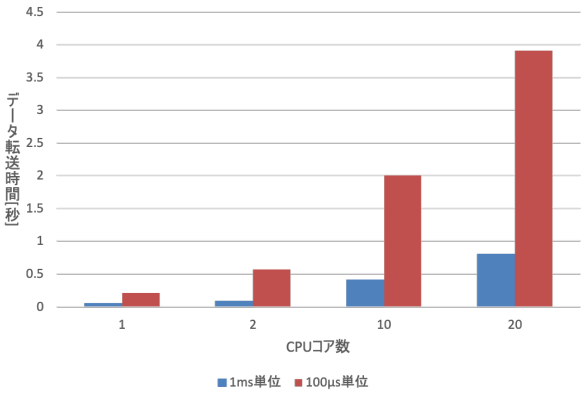

図 4　データ転送時間比較 (1 ミリ秒/100 マイクロ秒単位)

表 4 に実験 3 の結果を示す．データ転送を行う前後で，解析サーバのデータ格納用ディレクトリ下のファイルサイズを計測した結果，約 13.9KB 増加していた．転送データには 1 ミリ秒単位で収集した 1CPU コア数分のデータを使用したため，収集サーバからは約 32KB 転送されている．つまり，解析サーバでのデータ格納時に，半分以下のサイズに圧縮されたことが分かる．圧縮効果が高くなっている要因として，今回取得した perf のプロファイルデータのプロセス ID やスレッド ID が，同一値または近い値で連続していることが考えられる．

表 4　転送前後の DB ファイルサイズ

転送前のファイルサイズ	58bytes
転送後のファイルサイズ	13895bytes

実験 4 では，収集サーバ上で実行されるデータ転送プログラムの CPU 負荷率を計測した．ユーザレベル，システムレベル

での計測結果をそれぞれ図 5 と図 6 に示す．データ転送プログラムが実行される CPU コアは，転送開始時に指定できるため，負荷率の計測はその指定した CPU コアでのみ行った．計測開始から約 6 秒後に転送を開始し，その約 4.61 秒後に終了した．転送実施中の CPU 負荷率はユーザレベルで約 60〜73%，システムレベルで約 0〜6%だった．転送前後の CPU 負荷率はどちらもほとんど 0%になった．これらの結果より，収集サーバにおいて，データ転送プログラムの CPU 負荷の飽和による性能低下は発生していないが，データ転送を行う CPU コアでは 60%以上の負荷がかかっており，CPU 負荷改善が必要であることが分かる．

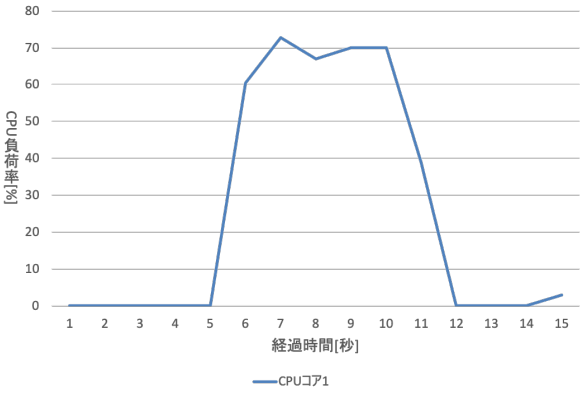

図 5　データ転送プログラムの CPU 負荷率 (ユーザレベル)

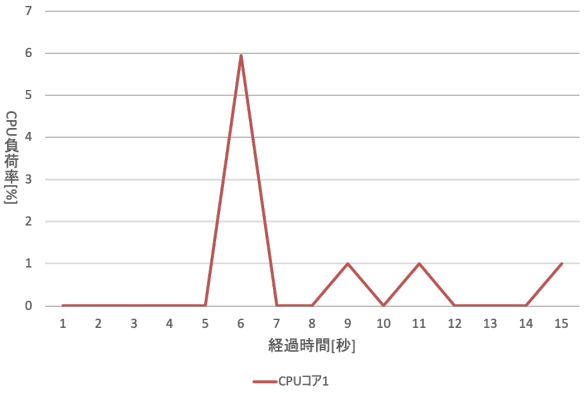

図 6　データ転送プログラムの CPU 負荷率 (システムレベル)

実験 4 では，解析サーバ上で実行される InfluxDB の CPU 負荷率も計測した．ユーザレベル，システムレベルでの計測結果をそれぞれ図 7 と図 8 に示す．InfluxDB が実行される CPU コアは転送毎に変化するため，解析サーバ上のすべての CPU コアで負荷率を計測した．今回は CPU コア 1 で実行されていることが読み取れる．計測開始から約 4 秒後に転送を開始し，その約 4.61 秒後に終了した．ユーザレベルでの負荷率は，転送実施中は約 13〜33%，転送前後ではほとんど 0%になった．システムレベルでの負荷率は，転送中かその前後かによらず，どの CPU コアでも約 0〜4%だった．これらの結果より，解析サーバでは InfluxDB の CPU 負荷率は低く抑えられており，今回行ったデータ転送の実験に対しては処理能力に余裕があることが分かる．

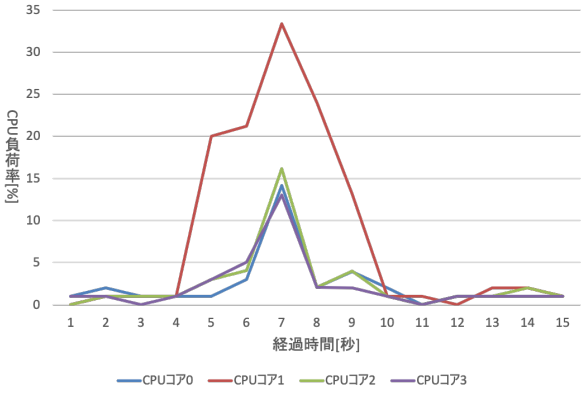

図 7　InfluxDB の CPU 負荷率 (ユーザレベル)

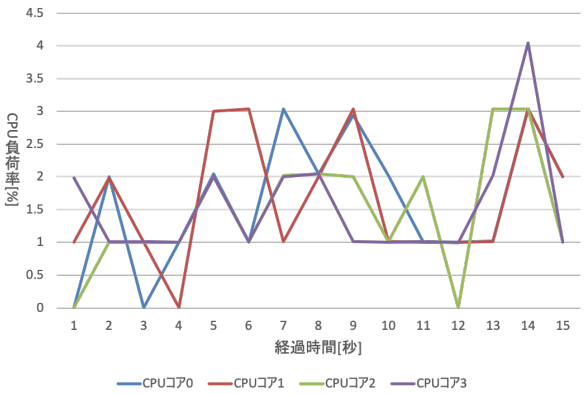

図 8　InfluxDB の CPU 負荷率 (システムレベル)

以上の結果から，有用な転送効率化/CPU 負荷改善手法を検討する．多数コアの CPU 性能データを時系列で切り出して転送するには，今回使用した手法よりも効率的な転送手法を検討する必要がある．また，本研究ではデータ収集と転送を同時に行うことを想定しているため，出来るだけ CPU 負荷などの外乱が少ない効率化手法である必要がある．現時点では，有用な転送効率化手法として，データ転送用プログラム (ライブラリ，モジュール) の高速化，転送データの簡易分析，転送データの圧縮などの手法を検討している．収集サーバでの CPU 負荷率を改善するためには，転送専用の CPU コアの用意，または CPU 負荷を低減させるためのプログラム改善，あるいは転送処理の並列化などを検討している．

4.　まとめと今後の予定

性能データの収集および転送の効率化手法を検討するにあたり，まずサーバシステムの性能データを収集/転送/蓄積する際のリソース利用量やオーバヘッド，CPU 負荷率を計測・分析した．その結果，通常の InfluxDB の python モジュールを使用した転送方法では，多数コア CPU において 1 秒間相当のデータの転送に 1 秒以上かかってしまうため，データ収集と転

送を同時に行うにはより効率化が必要であることが分かった．1CPU コア数分の転送データサイズが転送時間に比例しなかったことから，データ本体の転送以外にかかる InfluxDB サーバとのネゴシエーション等のコストが大きく，時間がかかることも分かった．また，転送先の解析サーバでの DB 増加量は転送したデータの半分以下であったため，今回取得した perf のプロファイルデータは圧縮効果の高いデータだったと考えられる．転送実施中の収集サーバでの CPU 負荷率が高いため，CPU 負荷を分散または低減させる必要があることも分かった．

本研究では，転送効率化のため，データ転送用プログラム (ライブラリ，モジュール) の高速化や転送データの簡易分析，転送データの圧縮などの手法を検討した．CPU 負荷改善のためには，転送専用の CPU コアの用意，または CPU 負荷を低減させるためのプログラム改善あるいは転送処理の並列化などの手法を検討した．今後は，今回検討した転送効率化および CPU 負荷改善手法を試行していくとともに，InfluxDB サーバとのネゴシエーション等の詳細や，プロファイルデータによる圧縮率の差異についても調べていく．

文　　献

[1] score-p. https://www.vi-hps.org/projects/score-p.

[2] vampir. https://vampir.eu.

[3] 阿部文武, 中村 朋健, and 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. 研究報告ハイパフォーマンスコンピューティング *(HPC)*, 2016(18):1–8, 2016.

[4] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. *Google Technical Report dapper-2010-1*, 2010.

[5] zipkin. https://zipkin.io.

[6] jeager. https://www.jaegertracing.io/.

[7] G.Ren, E.Tune, T.Moseley, Y.Shi, S.Rus, and R.Hundt. Google-wide profiling: A continuous profiling infrastructure for data centers. *IEEE Micro*, 30(4):65–79, 2010.

[8] S.Kanev, J.P.Darago, K.Hazelwood, P.Ranganathan, T.Moseley, G.Wei, and D.Brooks. Profiling a warehouse-scale computer. *ISCA '15: Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 768, 2015.

[9] perf. https://perf.wiki.kernel.org/index.php/Main_Page.

[10] stress. https://linux.die.net/man/1/stress.

[11] influxdb. https://www.influxdata.com/products/influxdb/.

[12] S.N.Z. Naqvi, S. Yfantidou, and E. Zimányi. Time series databases and influxdb, université libre de bruxelles. 2017.

[13] pandas. https://pandas.pydata.org.

[14] Dataframeclient. https://influxdb-python.readthedocs.io/en/latest/api-documentation.html#dataframeclient.