DEIM2023 2a-4-4

サーバシステムの性能データ収集および転送効率化に向けた改善案の評価

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: †{chika,oguchi}@ogl.is.ocha.ac.jp, ††{ahirai,yamaoka.mari,fukumoto.naoto}@fujitsu.com

あらまし　近年，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．リアルタイムでこれらの性能データを分析するためには，データの収集や転送によるオーバヘッドを抑え，効率的に扱う必要がある．そこで本研究では，ターゲットサーバでのデータ収集およびデータ解析用サーバへのデータ転送時のリソース利用量やオーバヘッドを計測した結果をもとに，効率的な手法を開発することを目的としている．本報告では，データ転送処理のボトルネックの一つである CPU 負荷を削減するための効率化手法として，転送処理の並列化による効果の検証を行う．さらに，並列化した転送処理と CPU に負荷をかけるベンチマークを同時実行し，それぞれに与える影響を分析する．

キーワード　時系列データ，性能データ収集，データ転送，データ圧縮，CPU 負荷率，並列化

Evaluation 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 Kamikodanaka, Nakahara-ku, Kawasaki-shi, Kanagawa 211–8588, Japan
E-mail: †{chika,oguchi}@ogl.is.ocha.ac.jp, ††{ahirai,yamaoka.mari,fukumoto.naoto}@fujitsu.com

Key words　Time-series data, performance data collection, data transfer, data compression, CPU load factor, parallelization

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．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，データ収集を行うサーバと解析を行うサーバが分割された環境を想定して，データ収集・転送時のリソース利用量やオーバヘッドを計測した．その結果，複数コアの情報を 1 つのコアで転送する手法ではコア数に伴い転送処理の遅延が増大することが分かった．そこで本報告では，1 つの CPU コアで行っていた転送処理を各 CPU コアで行うように並列化し，性能を評価・比較することで遅延の改善方法を検討した．さらに，性能情報収集処理が動作環境に与える影響を検証するために，並列化したデータ転送処理と同じ CPU コア上で同時に動作するベンチマーク性能への影響およびデータ転送処理が受ける影響を分析した．

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 Per-

formance 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 評 価 環 境

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．実験環境概要を図 1 に示す．

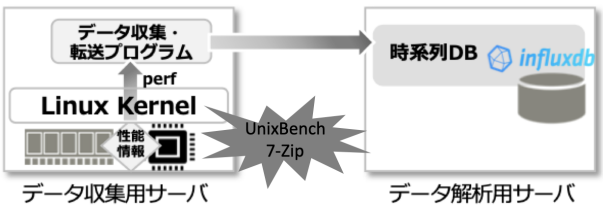


図 1　実験環境概要

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] を使用した．また，CPU 負荷をかける性能ベンチマークとして，UnixBench [10] の Dhrystone と Whetstone, および 7-Zip [11] 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 [12] を使用した．InfluxDB については [13] で解説されている．収集サーバで作成した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．転送するデータには，事前に収集した stress [14] 実行時の perf record のデータを使用した．収集サーバと解析サーバの環境をそれぞれ表 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
UnixBench	5.1.3
7-Zip	22.01
InfluxDB	1.8.3.x86_64

3.2 実 験 概 要

実験の背景を図 2 に示す．CPU コア毎の性能データを収集する一般的な手法として，例えば Linux perf が広く利用されている．perf では，複数 CPU 構成の場合に，全コアのデータを共有メモリ上のテーブルに格納し，それを 1 つのコアがまとめて収集/出力している．前報の実験 [15] では，これに合わせた形式で全コアのデータを 1 コアでデータ転送した場合のオーバヘッド計測等の分析を行った．その結果，複数コアの情報を 1 つのコアで転送する方式では，コア数が増えると転送処理が追い付かなくなることが分かり，転送の効率化が必要であることが分かった．そこで今回の実験では，効率化手法の一つとして，各コアの情報を自コアで転送する並列化を行った．さらに，データ転送処理が同一 CPU コア上で動作するプログラムへ与える影響を調べるため，CPU 負荷をかける性能ベンチマークプログラムを同時動作させ，ベンチマーク性能の変化，および

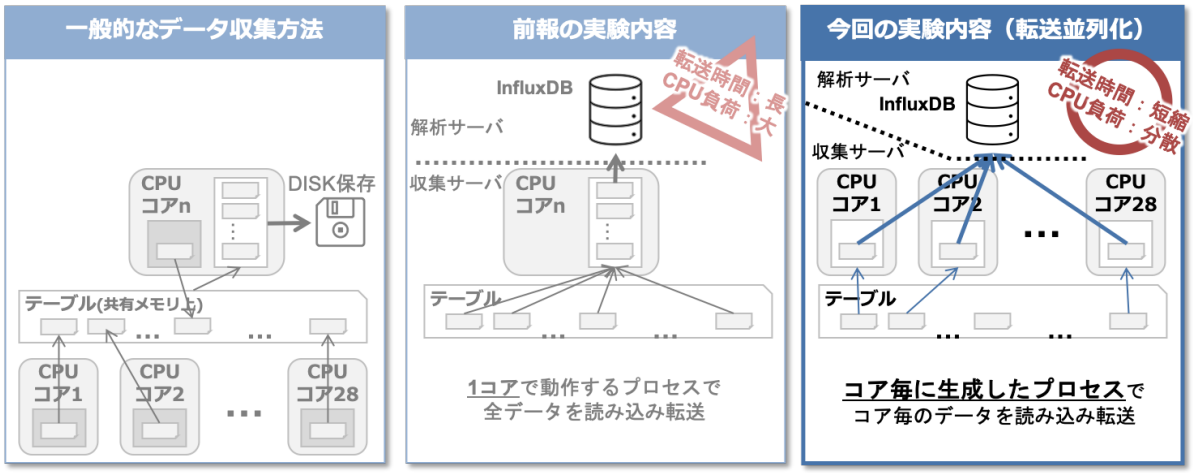


図 2　実験の背景

データ転送処理が受ける遅延を計測し分析した．

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，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．この時，それぞれ 64bit のデータが 4 項目あるため，1CPU コアに関する転送データサイズは約 320KB となる．次に，抽出データを Python Pandas [16] 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 3 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) [17] 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 3　DataFrame 形式データのサンプル

3.3　転送処理並列化

3.3.1　実験内容

各コアの情報を自コアで転送する並列化の効果を検証するため，以下の実験 1〜4 の計測を行った．実験 1 では，CPU コア数を変更しながらデータ転送時間を計測した．実験 2 では，データ転送処理部分の CPU 負荷率を計測した．実験 3 では，InfluxDB の CPU 負荷率を計測した．実験 4 では，InfluxDB の I/O 負荷を計測した．以降，転送する全ての CPU コアのデータを 1 つのコアで転送する場合を 1 コア転送版，各 CPU コアの情報を自コアで転送する場合を並列転送版と記述する．

3.3.2　実験結果

実験 1 の結果を図 4 と図 5 に示す．図 5 の各 CPU コア数における平均転送時間を表 4 に示す．

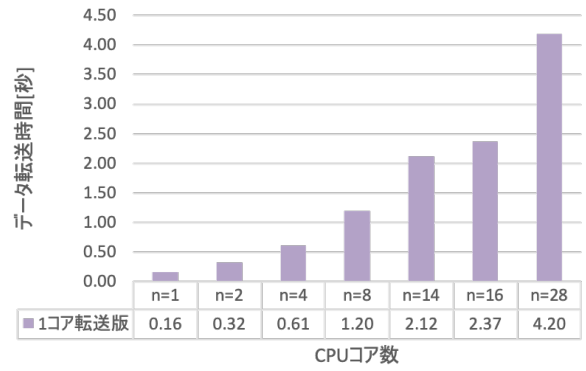


図 4　1 コア転送版のデータ転送時間

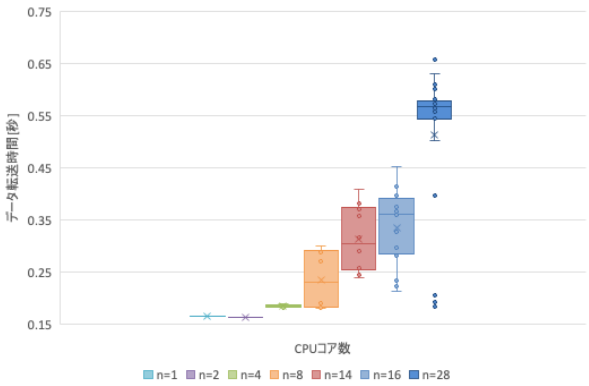


図 5　並列転送版のデータ転送時間

図 4 より 1 コア転送版では，8 コア以上のデータ転送に 1 秒以上かかっていることが読み取れる．一方並列転送版は，表 4 より 28 コアのデータ転送でも 0.51 秒で転送可能であり，並列化による効果が得られていることが分かる．しかし，図 5 より各 CPU コア数 n において，並列転送版のデータ転送時間が 1 コア転送版の 1/n よりも長くなっていることから，並列化にはオーバヘッドがあることが推察される．例えば n=8 において，並列転送版の平均転送時間 (0.24 秒) は 1 コア転送版の 1/8(0.15 秒) よりも長く，約 1/5 となっている．また，並列転

送版では，転送を実行するコア間で転送時間にばらつきがあり，CPU コア数が増えるとばらつきも同時に大きくなっていることが読み取れる．この要因として，コア数が増えて InfluxDB 側の CPU 負荷率や Disk 書き込み時の iowait が増加した場合に，InfluxDB 側の処理が追いつかなくなっている可能性が挙げられる．

表 4　図 5 各 CPU コア数における平均転送時間

n=1	0.17 秒
n=2	0.17 秒
n=4	0.19 秒
n=8	0.24 秒
n=14	0.31 秒
n=16	0.34 秒
n=28	0.51 秒

並列化のオーバヘッドをより詳細に分析するため，CPU コア数 n=14 で実験 2〜4 を行った．

実験 2 のデータ転送処理部分の CPU 負荷率の計測結果を表 5 に示す．

表 5　データ転送処理部分の CPU 負荷率

1 コア転送版		50.3 %
並列転送版	平均	31.6 %
	最大	40.0 %
	最小	23.6 %

また，並列転送版の n=14 の時の各コアの CPU 負荷率とデータ転送時間を抽出した結果を図 6 に示す．図中の各点がコアを表している．コアごとにデータ転送時間にばらつきがあり，データ転送時間が長くなるほど CPU 負荷率は低くなっている (Idle が増えている) ことが読み取れる．CPU 負荷率の収集は，転送処理時間に加えて，該当プロセスが CPU に割り当てられていた時間 (task-clock) を perf 機能を利用することで取得し，以下の計算式で算出した．

CPU 負荷率=(該当プロセスの転送処理部分が CPU に割り当てられていた時間)/(転送処理時間)*100(%)

Python のコードに，libpfm4 [18] という perf 機能を Python から呼び出すためのライブラリ (モジュール) を利用して，上記の CPU 負荷率収集を実装した．

実験 3 の InfluxDB の CPU 負荷率の計測結果を図 7 に示す．並列転送版では，InfluxDB が動作している解析サーバ上の 4 つのコア全てが同じタイミングで 30%程度になっていることが読み取れる．CPU コア数 n=14 での転送時間が約 0.32 秒であることから，処理中の CPU 負荷率は 100%程度で CPU ネックになっており，ばらつきの要因となっていることが推察できる．

実験 4 の InfluxDB の I/O 負荷の計測結果を図 8 に示す．図中の線が重なっているが，主に sda，sda2，dm-0 が動作していた．sda(sda2) および dm-0 は InfluxDB のデータが格納される論理デバイスである．並列転送版において，InfluxDB が動作している解析サーバでは 1 秒間で約 2.74MB の書き込み処理

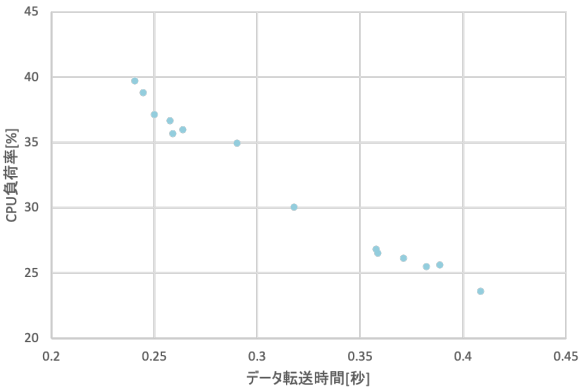


図 6　データ転送処理部分の CPU 負荷率 (並列転送版，CPU コア数 n=14)

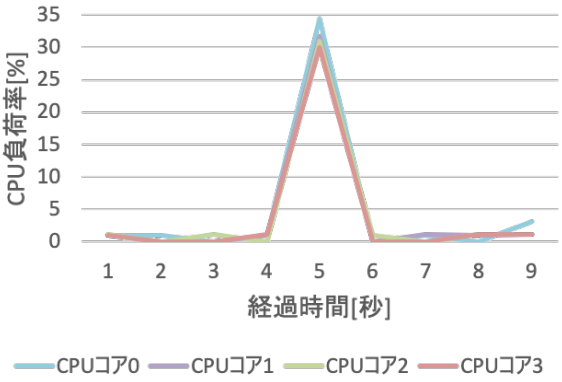


図 7　InfluxDB の CPU 負荷率 (並列転送版)

が行われていることが読み取れる．このタイミングで圧縮した転送データを一度に書き込んでいることが想定されるが，Disk 性能を計測する fio ベンチマーク [19] ではランダム書き込み性能が 1280MB/s であることから，Disk ネックになっている可能性は低い．

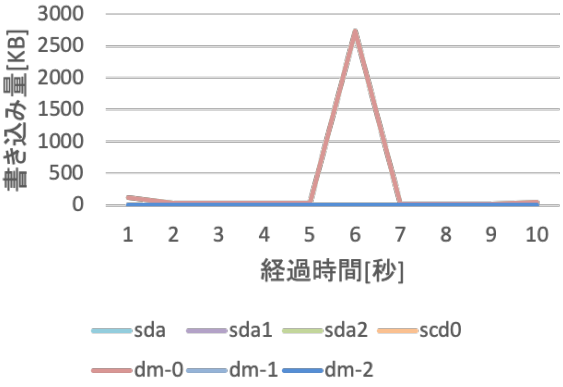


図 8　InfluxDB の I/O 負荷率 (並列転送版)

3.4　ベンチマーク同時動作

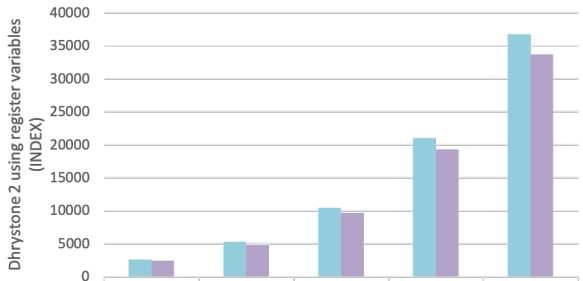
3.4.1　実 験 内 容

データ転送処理が同一 CPU コア上で動作するプログラムへ与える影響および転送処理が受ける影響を調べるため，以下の

実験 5〜7 の計測を行った．実験 5〜6 では，CPU 負荷ベンチマークである UnixBench を使用した．実験 5 では，UnixBench のテストケースの一つで整数演算処理を行う dhry2reg を使用した．実験 6 では，UnixBench のテストケースの一つで浮動小数点数演算処理を行う whetstone-double を使用した．実験 7 では，ベンチマークとして 7z 形式ファイルの圧縮・解凍を行う 7-Zip を使用した．

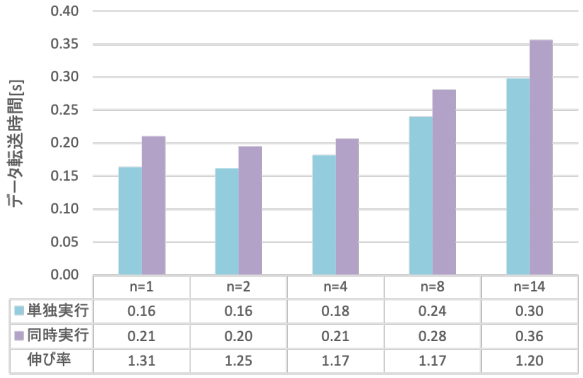
3.4.2 実験結果

実験 5 の Dhrystone(dhry2reg) での結果を図 9 と図 10 に示す．dhry2reg ベンチマークとデータ転送プログラムを同時に実行することにより，それぞれ単独で実行した時と比べてベンチマーク値は 7〜8%程度劣化し，データ転送時間は 1.17〜1.31 倍程度伸びていることが読み取れる．



	n=1	n=2	n=4	n=8	n=14
単独実行	2611.2	5263.8	10497.2	21035.0	36761.4
同時実行	2432.5	4846.5	9714.3	19351.8	33745.4
劣化率	0.93	0.92	0.93	0.92	0.92

図 9　ベンチマーク値比較 (dhry2reg)



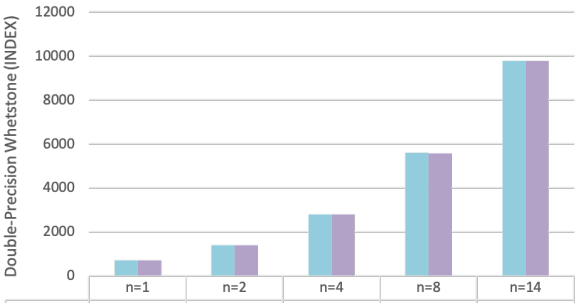
	n=1	n=2	n=4	n=8	n=14
単独実行	0.16	0.16	0.18	0.24	0.30
同時実行	0.21	0.20	0.21	0.28	0.36
伸び率	1.31	1.25	1.17	1.17	1.20

図 10　データ転送時間比較 (dhry2reg)

実験 6 の Whetstone(whetstone-double) での結果を図 11 と図 12 に示す．whetstone-double ベンチマークとデータ転送プログラムを同時に実行することにより，それぞれ単独で実行した時と比べてベンチマーク値はほとんど劣化しないが，データ転送時間は 1.13〜1.31 倍程度伸びていることが読み取れる．

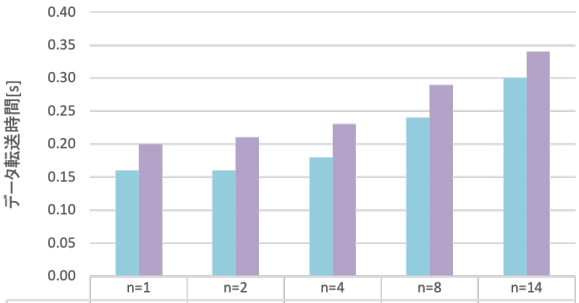
実験 7 の 7-Zip での結果を図 13 と図 14 に示す．7-Zip ベンチマークとデータ転送プログラムを同時に実行することにより，それぞれ単独で実行した時と比べてベンチマーク値は 7〜8%程度劣化し，データ転送時間は 1.04〜1.38 倍程度伸びていることが読み取れる．

同時実行によるベンチマーク値の劣化率とデータ転送時間



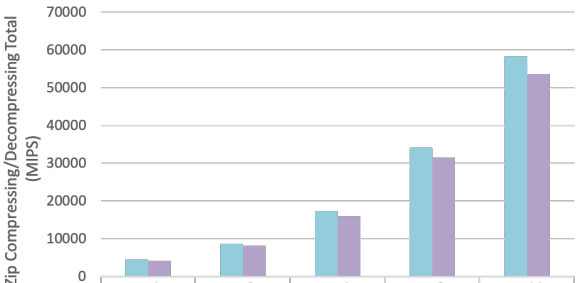
	n=1	n=2	n=4	n=8	n=14
単独実行	700.1	1399.2	2798.2	5595.1	9767.6
同時実行	699.6	1398.8	2795.4	5581.3	9767.6
劣化率	1.00	1.00	1.00	1.00	1.00

図 11　ベンチマーク値比較 (whetstone-double)



	n=1	n=2	n=4	n=8	n=14
単独実行	0.16	0.16	0.18	0.24	0.30
同時実行	0.20	0.21	0.23	0.29	0.34
伸び率	1.25	1.31	1.28	1.21	1.13

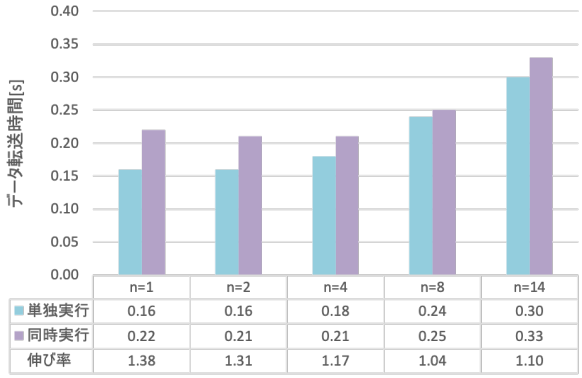
図 12　データ転送時間比較 (whetstone-double)



	n=1	n=2	n=4	n=8	n=14
単独実行	4468	8664	17246	34196	58377
同時実行	4105	8035	15955	31442	53685
劣化率	0.92	0.93	0.93	0.92	0.92

図 13　ベンチマーク値比較 (7-Zip)

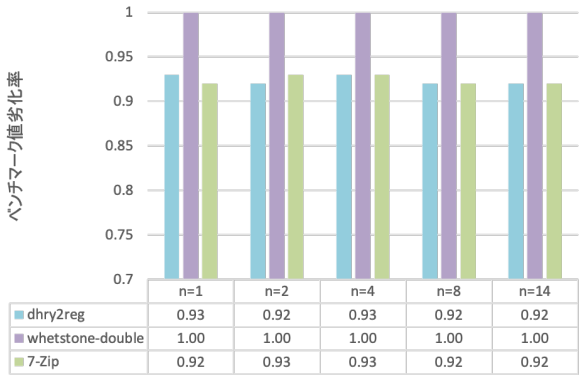
の伸び率を各ベンチマークで比較した結果をそれぞれ図 15 と図 16 に示す．図 15 より，各 CPU コア数 n の場合にてベンチマークごとに劣化率が異なることが分かる．また，CPU コア数 n を増やしても各ベンチマークでの劣化率は変化しない．これより，ベンチマーク値に与える影響は，ベンチマークにより影響の有無や度合いが異なるが，CPU コア数 n には依存しないことが読み取れる．図 16 より，今回使用した全てのベンチマークによりデータ転送時間が伸びていることが確認できる．また，使用 CPU コア数 n の増加に伴いその伸び率は減少していることも確認できる．これに加え，実験 1〜4 から，単独実行の場合，本実験環境では使用コア数 n の増加に伴う InfluxDB 側の CPU 負荷により，データ転送時間が伸びていた (転送中の Idle 時間が増加していた) ことが分かっている．このことか



	n=1	n=2	n=4	n=8	n=14
単独実行	0.16	0.16	0.18	0.24	0.30
同時実行	0.22	0.21	0.21	0.25	0.33
伸び率	1.38	1.31	1.17	1.04	1.10

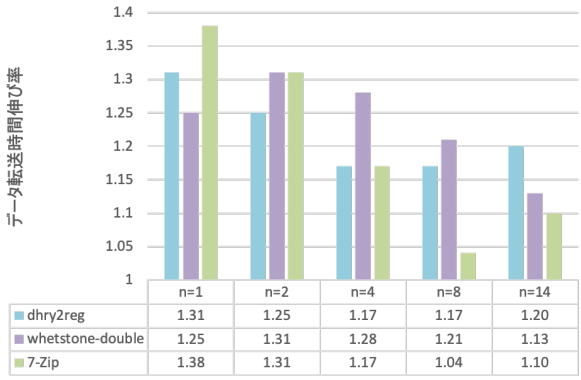
図 14　データ転送時間比較 (7-Zip)

ら，同時実行で使用コア数 n を増やした場合，転送するタイミングがベンチマークプログラムの影響を受けてばらつくことで，本実験環境では InfluxDB 側の CPU 負荷が平準化され，使用コア数 n の増加によるデータ転送時間への影響が少なく見えていることが推察できる．以上の結果から，転送処理は，ベンチマークによらずに影響を受けるが，使用 CPU コア数 n の増加に伴い受ける影響が少なくなる傾向があることが分かる．ただしこの傾向は InfluxDB サーバの性能に依存すると推測される．



	n=1	n=2	n=4	n=8	n=14
dhry2reg	0.93	0.92	0.93	0.92	0.92
whetstone-double	1.00	1.00	1.00	1.00	1.00
7-Zip	0.92	0.93	0.93	0.92	0.92

図 15　ベンチマーク値劣化率比較



	n=1	n=2	n=4	n=8	n=14
dhry2reg	1.31	1.25	1.17	1.17	1.20
whetstone-double	1.25	1.31	1.28	1.21	1.13
7-Zip	1.38	1.31	1.17	1.04	1.10

図 16　データ転送時間伸び率比較

4.　まとめと今後の予定

本報告では，データ収集を行うサーバと解析を行うサーバが分割された環境を想定したデータ転送の効率化手法として，各コアのデータを自コアで転送する並列化転送の評価を行った．さらに，並列化した転送処理と CPU に負荷をかけるベンチマークを同時実行し，それぞれに与える影響を分析した．CPU コア数を変更しながらデータ転送時間を計測した実験 1 からは，14 コアのデータ転送に 1 コア転送版では 2.12 秒かかるのに対し並列転送版では 0.31 秒で済むなど，転送時間の短縮が可能であることが分かった．実験 1 と，データ転送処理部分の CPU 負荷率を計測した実験 2 からは，データ転送時間はコアごとにばらつきがあり，コア数の増加に伴ってばらつきも大きくなっていることが分かった．InfluxDB の CPU 負荷率を計測した実験 3 からは，処理中の InfluxDB の CPU 負荷率が 100%で CPU ネックになっていることが推察された．InfluxDB の I/O 負荷を計測した実験 4 からは，InfluxDB の書き込み処理量には余裕があり，Disk 側のボトルネックではないことが分かった．ベンチマークへの影響および転送処理への影響を計測した実験 5〜7 からは，データ転送プログラムとベンチマークの相互影響の度合いはベンチマークによって異なることが分かった．また，ベンチマークプログラムの影響によって InfluxDB 側の CPU 負荷が平準化され，使用 CPU コア数 n の増加によるデータ転送時間への影響が少なくなっていることが推察された．今後は，InfluxDB サーバの CPU コア数を増やした環境での評価や，ベンチマークによる相互影響の受けやすさ/与えやすさの詳細分析，使用 CPU コア数 n の増加によるデータ転送時間の伸び率の変化についての詳細分析を行っていく．また，転送プログラムの改善として，コアごとの転送処理のタイミングをずらす処理や，転送側で CPU 負荷の低いコアを検出してそのコアに転送させる処理の実装も行う予定である．

謝　　辞

本研究の一部はお茶の水女子大学と富士通株式会社との共同研究契約に基づくものであり，JST CREST JPMJCR22M2 の支援を受けたものである．

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] Unixbench. https://code.google.com/archive/p/byte-unixbench/.

[11] 7-zip. https://7-zip.org.

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

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

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

[15] 飯山知香, 平井聡, 山岡茉莉, 福本尚人, and 小口正人. 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. 第 *14* 回データ工学と情報マネジメントに関するフォーラム *(DEIM2022)*, 2022.

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

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

[18] libpfm4. https://sourceforge.net/p/perfmon2/libpfm4/ci/master/tree/.

[19] fio. https://fio.readthedocs.io/en/latest/fio_doc.html.