Vol.2018-DPS-174 No.33
Vol.2018-CSEC-80 No.33
2018/3/6

フォグコンピューティングのためのライトウェイトなハイブリッドデータ処理フレームワークの検討

野本孝夫† アプドゥハン・ベーナディ†

概要：ビッグデータ処理を実行するクラウドコンピューティングのパワーにより、さまざまなクラウドランタイム環境の開発が進化した。クラウドの能力を、データが作成され処理される場所に近づける Fog Computing という概念は、サービスのパフォーマンスを補完し、向上させることを目指している。 Apache Flink と Apache Spark は、クラウドコンピューティングのために広く使用されているハイブリッド（バッチデータ,ストリーミングデータ）プライマリランタイム環境である。本稿では、Apache Beam の優れた特徴を考慮して、バッチデータとストリーミングデータを使用して Beam 使用時と Beam 不使用時の Flink と Spark の処理特性を検証し、フォグコンピューティングのための実行可能な軽量ハイブリッドデータ処理フレームワークを調査した。結果は有望であり、フォグ・コンピューティングを実現するために関連する問題のよりよい理解を可能とした。

キーワード：Apache Flink, Apache Spark, Apache Beam, ストリーミングデータ処理, バッチデータ処理

Investigation of a Lightweight and Hybrid Data Processing Framework for Fog Computing

Nomoto Takao† Bernady O. Apduhan†

Abstract: With the power of cloud computing to perform big data processing, the development and advancements of various cloud runtime environments had evolved. With the notion of Fog Computing to bring the capabilities of the cloud closer to the place where data is created and processed is envisioned to supplement and augment the service performance. Apache Flink and Apache Spark are the widely used hybrid (batch/streaming data) primary runtime environment for cloud computing. Considering the outstanding features of Apache Beam, in this paper, we conducted experiments on the processing characteristics of Flink and Spark with or without Beam using batch data and streaming data to investigate a viable lightweight and hybrid data processing framework for fog computing. The results were promising and we were able to better understand the platform characteristics and related issues to realize fog computing.

Keywords: Apache Flink, Apache Spark, Apache Beam, streaming data, batch data

1. はじめに

近年、ビッグデータ処理を実行するクラウドコンピューティングのパワーアップにより、さまざまなクラウドランタイム環境の研究と開発が進んでいる。クラウドコンピューティングはインターネット上にデータを保存されるのでデバイス、場所、容量を気にすることなく利用することが出来る。

しかしながら、オペレーションの時間制約が厳しかったり、インターネットの接続状態が悪い環境に適応することはできない。これは、ミリ秒の遅れが致命的な結果を招き、遠隔医療や患者のケアなどのシナリオでは特に課題となる。さらに、集中的なデータの管理は銀行/ビジネス/医療などの情報を完全に把握してしまうためハッカーの攻撃対象になりやすく、個人情報を含む顧客情報や経営情報に流出のリスクが伴う。たとえサービスに障害が出ている場合でもタスクを実行できる信頼性が必要であり、ローカルでの処理能力やストレージを持っておくことは今後重要になってくる。

この問題に対処するために、エッジコンピューティングが進化した。エッジコンピューティングは、ネットワークエッジ/デバイスでデータを処理するため、障害が発生する可能性が低く、応答の遅延を低減するコンピューティングのパラダイムである。しかしながら、高性能の装置が必要とされるため、装置の価格が上昇し、ユーザーにとって負担となる。さらに、インターネット

†九州産業大学 理工学部
Kyushu Sangyo University, Faculty of Science and Engineering

Vol.2018-DPS-174 No.33
Vol.2018-CSEC-80 No.33
2018/3/6

に接続された多くのデバイス/センサーを備えたIOT（Internet of Things）のアプリケーションの普及に伴い、ネットワークトラフィックの負荷が増加する。

フォグ・コンピューティングの概念は、エッジ・デバイスのすぐ近くに位置する小さなデータ・センター（フォグ・ノードと呼ばれる）の考え方を模倣している。これらのフォグノードは、エッジデバイスからデータを収集し、処理し、エッジデバイスに要求された応答を提供する。フォグノードは処理能力とケイパビリティにある程度の制限があるため、部分処理結果は最終処理のために適切なデータセンターに中継される。フォグノードを使用すると、部分的な処理結果のみが送信されるため、インターネットトラフィックの負荷が軽減され、必要な応答が高速化され、セキュリティリスクが軽減される。このデータセンター/フォグノード/エッジデバイスは、従来の「2 層クラウド中心コンピューティングモデル」の補足として、図 1 に示すいわゆる「3 層クラウドコンピューティングモデル」を形成する。

クラウド内ではストリーミングデータおよびバッチデータの初期入力データがあり、多数のストリーミングデータおよび/またはバッチデータ処理プラットフォーム/エンジンがそれぞれ独自の特徴を有する進化してきた。フォグノードの処理、ストレージ、ネットワークの制限、および関連する処理プラットフォームの処理オーバーヘッドを考慮すると、フォグノードの実行可能なストリーミングデータとバッチデータ処理プラットフォームを検討する必要がある。

本論文では、Apache 処理ソフトウェア、すなわち Flink、Spark、Beam の特徴について説明する。我々は Flink と Spark を同時に実験し、Beam ソフトウェアと組み合わせる。初期の結果は、各ソフトウェアの処理能力についての洞察を提供し、使用したソフトウェアの他の可能性のある問題、またはフォグ・コンピューティングのための軽量でハイブリッドな処理プラットフォームに対する他の処理プラットフォームを探索するためのより良い理解を提供した。

本稿は以下のように提示される。関連する作業についてはセクション 2 で説明し、使用されるストリーミングデータプラットフォームについてはセクション 3 で説明する。セクション 4 では、実験環境とセクション 5 でのバッチ処理の実験と結果について説明した。セクション 7 では実験結果と考察を述べる。最後に、セクション 8 では研究を要約し、将来の研究課題を引用した。

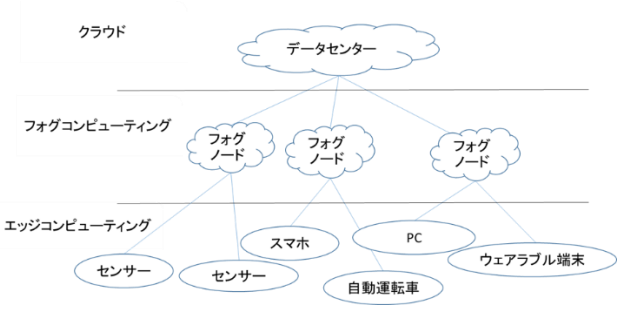

図 1. 3 層クラウドコンピューティングモデル

2. 関連研究

Darren Cottom は、Hazelcast Jet の軽量データ処理技術に関する研究を行った。これは、高速大容量データ処理のためにほぼリアルタイムでデータ集約型アプリケーションを実行する並列ストリーミングエンジンであり、リアルタイム分析が必要なアプリケーションに適している[1]。一方、Tianlun Zhang は、Apache Gearpump を使用した軽量リアルタイムストリーミングエンジンに関する調査を行った。Gearpump は、軽量のリアルタイムストリーミングエンジンでである。このベンチマークは、低遅延で多くのメッセージを処理する可能性を示している[2]。

いくつかの点で同じ目標を共有しいるが、本研究では、Apache Flink と Apache Spark をストリーミング・データ処理プラットフォームとして使用した。また、我々は言語互換性のために Apache Beam との処理を利用し、ストリーミングデータ処理とバッチデータ処理を行った。

3. ストリーミングデータのプラットフォーム

3.1 Apache Flink

Apache Flink は分散ストリーミング処理プラットフォームのオープンソースソフトウェアでストリーム処理とバッチ処理を実行可能であり、複数イベント処理、機械学習、SQL ライク API を提供する All-in-one 構成となっている。
耐障害性に優れており、各処理をステートフルで扱っており、障害が発生した際は処理を自動的に復旧させる機能を有している。Flink の特徴として以下の 6 つが挙げられる。

Vol.2018-DPS-174 No.33
Vol.2018-CSEC-80 No.33
2018/3/6

第一には高パフォーマンス＆低レイテンシ。第 2 はイベントタイムのサポートできることである。リルタイムで処理を行うため遅延などの影響は必ず発生する。Flink は受信したイベントに対しての時間を 3 つの概念で扱うことが出来る(システムの時間、イベント発生元の時間、イベントを取得した時間)。第 3 は正確に 1 回だけ実行することである。Flink で処理されるイベントは、Exactly-once のポリシーに基づいて処理される。障害発生後も、前回処理した内容を保持されており、その途中状態から処理を再開することが可能。第 4 はストリームウィンドウが柔軟なことである。スライディングウィンドなどの Window API も提供し、トリガーを利用して、条件をカスタマイズすることが出来る。第 5 は Backpressure タイプのストリーム処理である。過負荷となりイベントを処理しきれない場合、イベント処理を中断し、全体がバックアップしないようにするフロー制御の機能を保持している。第 6 は分散スナップショットを利用した耐障害性である。Chandy-Lamport アルゴリズムを利用した分散スナップショットにより、高スループットを維持しつつ、耐障害性を実現している。使用言語は Java、Scala の API クライアントが用意されている。API の構成を図 2 で示す[3]。

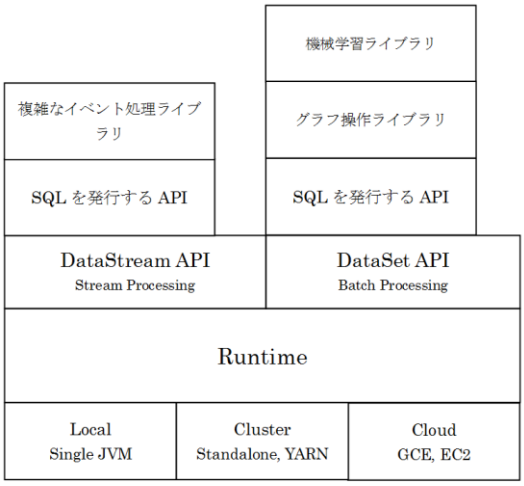

図 2. Flink の API の構成 [3]

3.2 Apache Spark

Apache Spark は分散処理を行うオープンソースのフレームワークで、Java や Scala、Python などいろいろな言語の API が用意されている。Spark は図 3 のような構成となっており、構造化データや表形式データを扱う Spark SQL、ほぼリアルタイムでストリーミングデータを処理する Spark Streaming、機械学習を行う MLlib、グラフ処理を行う GraphX がある。Spark は Hadoop が抱える繰り返し処理を多数含む処理では時間がかかることやスケールアウトによるシステム増築によって運用が複雑になる状態の背景から開発されたのである。Hadoop の MapReduce の部分に置き換わることを目指し、Hadoop の苦手であった繰り返し処理を HDFS に特殊なキャッシュを設けて、インメモリを使用した高速な分散処理を行うことが可能になったのが Spark である[4]。

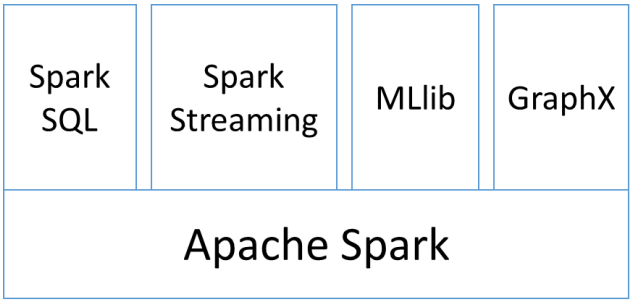

図 3. Spark の構成 [4]

3.3 Apache Beam

Apache Beam は分散処理を定義するためのプログラミングモデルで、バッチ処理・ストリーミングデータ処理の両方に対応しており対応言語は Java、Python の 2 言語である。プログラム中で Apache Beam SDK のクラスを import し、Apache Beam SDK の API を用いてデータ処理プログラムを作成すると作成したプログラムをさまざまなデータ処理エンジン上で動かすことが出来き、データ処理エンジンを Runner と呼ぶ。今回の実験は Beam で作成したプログラムを Apache Flink と Apache Spark を Runner として使用した[5]。

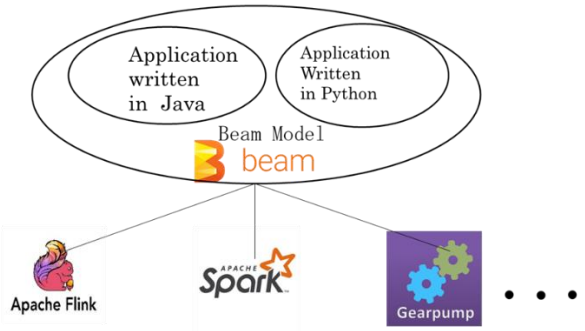

図 4. Beam のイメージ図

Vol.2018-DPS-174 No.33
Vol.2018-CSEC-80 No.33
2018/3/6

4. 実験環境

4.1 使用マシンの仕様

本研究では OpenStack Private Cloud で実験を行い、3 つのノードを使って各ソフトウェアを実行した。各ノードの仕様を以下に示す。

表 1. Kafka サーバーの仕様

CPU	2 core
メモリ	8GB
ストレージ	100GB
OS	CentOS7
処理ソフト	Apache Kafka

表 2. Beam サーバーの仕様

CPU	2 core
メモリ	8GB
ストレージ	100GB
OS	CentOS7
SDK	Apache Beam
処理ソフト	Apache Flink/Spark

表 3. Flink サーバーの仕様

CPU	2 core
メモリ	8GB
ストレージ	100GB
OS	CentOS7
処理ソフト	Apache Flink

表 4. Spark サーバーの仕様

CPU	2 core
メモリ	8GB
ストレージ	100GB
OS	CentOS7
処理ソフト	Apache Spark

4.2 実験で使用データ

今回の実験で使用したデータはストリーミングデータ、バッチデータともに Twitter を使用した。ストリーミングデータは、キーワードを「Cloud」と指定し、キーワードを含むツイートを使用した(2017/11/29 10:00~11:00)。バッチデータは、九州産業大学 理工学部 情報科学科の twitter の 2014/03/27～2018/01/29 までのデータを使用した[6]。

4.3 実験環境の構造

環境の構造を図 5 に示す。

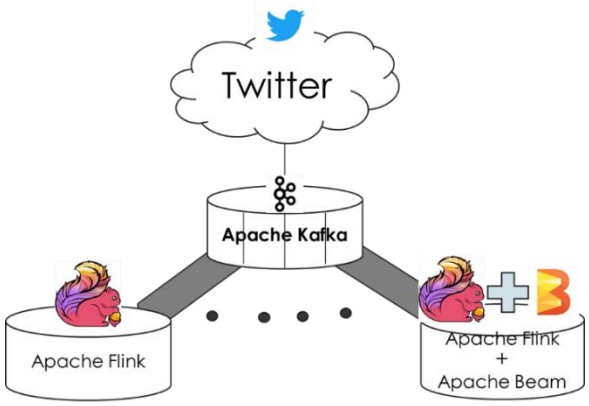

図 5. 実験環境の構造

5. バッチデータ処理の実験と結果

5.1 Spark vs. Flink

Twitter のデータを使用し、Flink と Spark で WordCount の実験を行い、処理速度の測定を行った。結果を図 6 に示す。

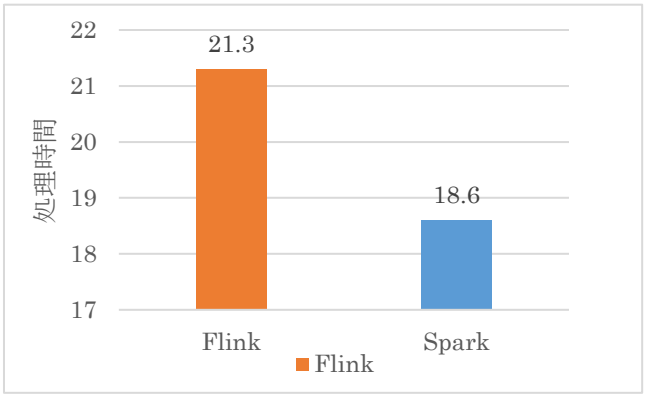

図 6. Flink vs. Spark の実験結果

5.2 Flink+Beam vs. Spark+Beam

上記と同様の実験を Flink+Beam と Spark+Beam で WordCount の実験を行い、処理速度の計測を行った。結果を図 7 に示す。

Vol.2018-DPS-174 No.33
Vol.2018-CSEC-80 No.33
2018/3/6

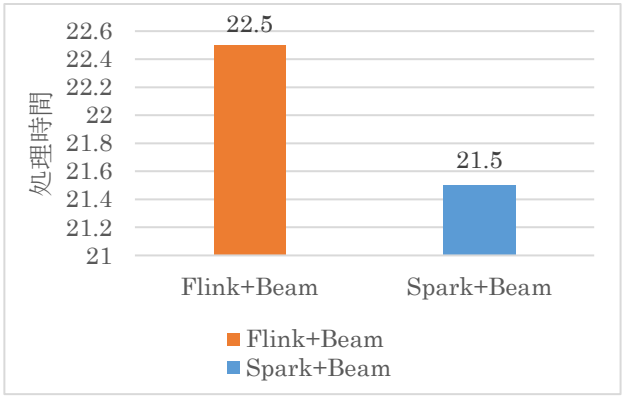

図 7. Flink+Beam vs. Spark+Beam の実験結果

6. ストリーミングデータ処理の実験と結果

6.1 Flink vs. Flink+Beam

Twitter のストリーミングデータを使用し、Flink と Flink+Beam で同時に WordCount の実験を行った。実験をはじめ 1 時間で処理できた Tweet 数を計測した。結果を図 8 に示す。

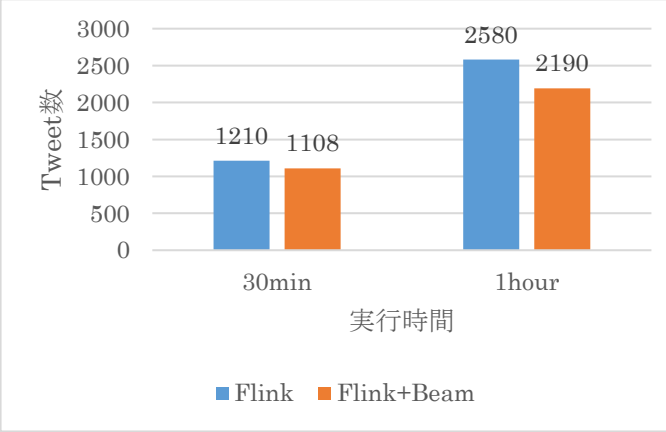

図 8. Flink vs. Flink+Beam の実験結果

6.2 Spark vs. Spark+Beam

上記と同様の実験を Spark と Spark+Beam で行った。実験結果を図 9 に示す。

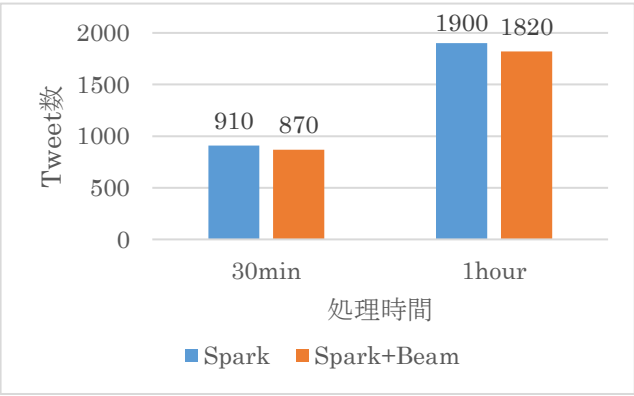

図 9. Spark vs. Spark+Beam の実験結果

6.3 Flink vs. Spark

上記の実験を Flink と Spark で行った結果を図 10 に示す。

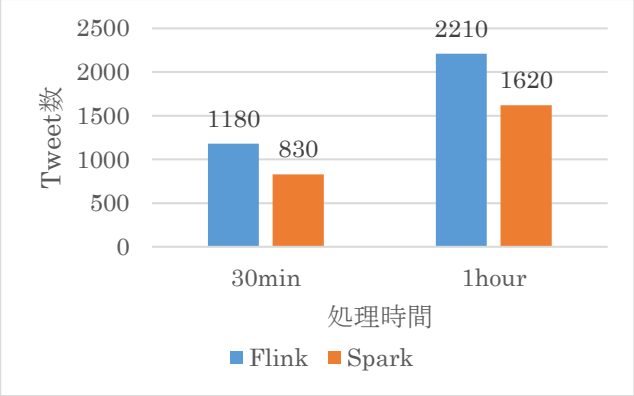

図 10. Flink vs. Spark の実験結果

6.4 Flink+Beam vs. Spark+Beam

上記と同様の実験を Fliknk+Beam と Spark+Beam で行った。実験結果を図 11 に示す。

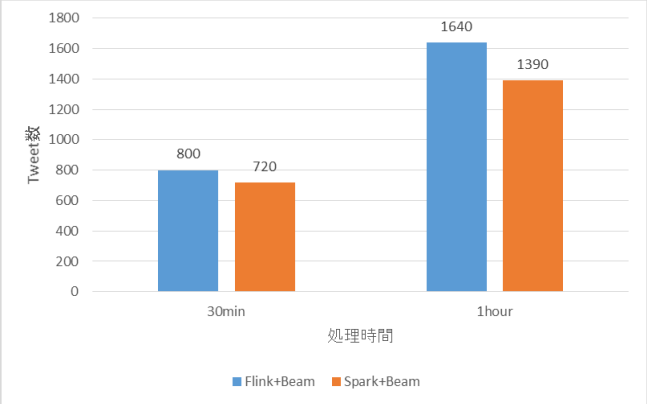

図 11. Flink+Beam vs. Spark+Beam の結果

7. 実験結果の考察と結論

7.1 バッチデータ処理の実験の考察

バッチ処理実験の結果より、Flink と Spark を比較すると、Spark のほうが 2.7 秒早く処理することが出来た。Beam を使用した場合でも Spark+Beam のほうが 1 秒早く処理することが出来た。どちらも Spark を使用したほうがより早く処理することが出来た。理由としては、Spark はバッチ処理、ストリーミング処理ともに可能だが、もともとバッチ処理をメインに開発されており、ストリーミング処理よりもバッチ処理のほうが得意なので、Flink よりも早く処理することが出来たと考えられる。実験結果のまとめ

を表 5 に示す。

表 5. バッチデータ処理実験結果のまとめ

Batch 処理	処理速度
Flink	21.3 秒
Beam+Flink	22.5 秒
Spark	18.6 秒
Beam+Spark	21.5 秒

7.2 ストリーミングデータ処理実験の考察

ストリーミング実験の結果より、Flink と Flink+Beam の比較をすると、Flink のみのほうが Flink+Beam よりも 390tweet 多く処理することが出来た。やはり Flink の API を使って直接プログラムを動かしたほうがより多くの tweet を処理することが出来た。Beam の場合処理を行う際、一度 Flink にプログラムを対応させる必要があるので少し遅くなったと考えられる。次に Spark と Spark+Beam の比較を行うと、Flink と同様に Spark のみのほうが Spark+Beam よりも 80Tweet 多く処理することが出来た。理由としては Flink と Flink+Beam の比較の同様の理由が挙げられ Beam の場合プログラムを Spark で実行できる形に変換する必要があったため処理が遅くなったと考えられる。次に Flink と Spark の比較をすると、Flink のほうが Spark よりも 590Tweet 多く処理することが出来た。理由としては、Spark はバッチ処理をメインに開発されており、ストーミング処理もバッチ処理を組み合わせ擬似的にストリーミングデータ処理を行っている。一方 Flink はストリーミングデータ処理をメインに開発されており、真のストリーミングデータ処理が可能なので、Flink のほうが多くの Tweet を処理することが出来たと考えられる。次に、Flink+Beam と Spark+Beam の比較を行うと、Flink+Beam のほうが Spark+Beam よりも 250Tweet 処理することが出来た。理由としても上記までの結果と同様の理由が挙げられる。それぞれの実験結果のまとめを表 6 に示す。

表 6. ストリーミングデータ処理実験結果のまとめ

Streaming 処理	処理 Tweet 数	
	30 分間	1 時間
Flink	1210	2580
Flink+Beam	1108	2190
Spark	910	1900
Spark+Beam	870	1820

8. まとめと今後の課題

8.1 まとめ

本研究においては、Apache Flink、Apache Spark、Apache Beam の 3 つのソフトを使用しバッチ処理、ストリーミングデータ処理の実験を行った。バッチ処理の場合 Spark を使用した場合のほうがより早く処理できた。Beam を使用した場合でも同様の結果が出た。しかし、Beam を使用した場合 Flink、Spark のみで処理を行うよりも、少しだけ処理な遅くなってしまった。ストリーミングデータ処理の場合は Spark よりも Flink のほうがより多く処理することが出来た。また、Beam を使用した場合も同様の結果が出た。今回も Beam を使用した場合のほうが少し処理が遅かった。

Flink、Spark のみで処理を行うほうがより早く処理することが出来るという結果が出た。Beam を使用するメリットとしては、Beam でプログラムを記述することができれば、Flink、Spark 以外の処理ソフト(Apache Gearpump.など)にも対応しており、それぞれのソフトの API を理解していなくても Beam の API を理解して入ればさまざまなソフトを使用することが出来る。以上の結果より、バッチデータ処理は Spark を使用したほうがより早く処理することが出来た。ストリーミングデータ処理の場合は、Flink を使用することでより早く処理することが出来た。どちらの場合でも、Beam を使用することで処理は遅くなったが、Beam を使用するメリットを考えると処理は少し遅くなっても使用する価値はあると考えられる。

8.2 今後の課題

今後の課題としては、軽量処理の特性に関する知識と知識の深化が望まれる。同様に、他のランタイム環境を調査し、他のデータを使用して

実験する予定である。

参考文献

[1] Darren Cottom　Introducing Hazelcast Jet - A New Lightweight, Distributed Data Processing Engine, <https://hazelcast.com/press-release/introt-jet-new-lightweight/>
[2] Tianlun Zhang　Apache Gearpump -Litweight Real-time Streaming Engine- <https://github.com/gearpump/gearpump>
[3] Apache Flink , http://mogile.web.fc2.com/flink/flink-docs-release-1.0.3/
[4] Apache Spark, https://spark.apache.org/
[5] Apache Beam, https://beam.apache.org/
[6] Twitter, https://twitter.com/
[7] 野本孝夫, フォグコンピューティングのためのライトウェイトストリーミングデータ処理のフレームワークの検討, 学士卒業論文,九州産業大学情報科学部, 2018 年 1 月.