7M-02

メニーコアシステムにおける分散ストリーム処理システムの性能評価 – 遅延に関する評価 –

牧田 直樹 † 杉浦 健人 †† 石川 佳治 ††

† 名古屋大学工学部電気電子・情報工学科 †† 名古屋大学大学院情報学研究科

1 はじめに

Web サービスのトラフィック解析や e コマースでの取引など，ビッグデータに対するリアルタイムな解析・処理の要求が増加している．特に金融取引や Web ショッピングやスマートフォン向けアプリでの購入・課金処理は迅速かつ正確に行われる必要があるため，そういった場面でもストリーム処理は重要視されており，その性能にも目が向けられている．また，IoT デバイスの普及による流通データ量の大規模な増加によって，ストリーム処理は今後ますますその重要性を高めていくことが予想される．

こうした背景から，様々な分散並列ストリーム処理システムの研究開発が学術・産業両面で活発に行われている．

主な分散並列ストリーム処理 OSS（open source software）はシェアードナッシングアーキテクチャを採用しており，複数のマシンを使用することで性能のスケールアウトを可能としている．シェアードナッシングアーキテクチャでは，複数のマシンで処理を分散して実行するが，システムを構成するマシン間ではリソースを共有せず，それぞれが独立して処理を行う．

その一方で，近年ではメニーコア CPU による各マシン単位での性能のスケールアップの潮流が見られる．ストリーム処理システムがメニーコア CPU にスケーリングされていれば，シェアードナッシングアーキテクチャにおいても，メニーコア CPU の搭載によって 1 マシンあたりのスループットの向上を見込める．それによって，マシンリソースの削減によるコストの軽減も期待できる．

しかし，メニーコア CPU を搭載したマシンにおける既存の分散並列ストリーム処理 OSS の基本的な性能評価に関する知見は十分とは言えない．そこで，本研究では既存の分散並列ストリーム処理 OSS のメニーコア CPU における処理性能を調査し，特にレイテンシの面からその評価を行う．評価対象の分散並列ストリーム処理 OSS として，Flink [1], Samza [2], Storm [3], Spark Streaming [4] を扱う．

Performance Evaluation of Distributed Stream Processing System in a Many-Core Systems: Evaluation of Delays

Naoki Makita†, Kento Sugiura††, and Yoshiharu Ishikawa††

†Department of Information Engineering, School of Engineering, Nagoya University

††Graduate School of Informatics, Nagoya University

2 調査対象となる OSS

調査対象となる OSS について，処理方式，並列処理の実現方法に着目し，その特徴を述べる．

ストリーム処理フレームワークの処理モデルは，大別して one-at-a-time 方式と micro-batch 方式に大別できる．one-at-a-time 方式ではデータ要素の 1 つ 1 つに対し決められた処理を適用するのに対し，micro-batch 方式では短い時間間隔の間に到着したデータ要素をまとめて 1 つの micro-batch とし，この micro-batch 毎に決められた処理を適用する．

2.1 Flink

Flink は one-at-a-time を処理モデルとしている．

Flink プログラムは実行されると，ストリームと変換オペレータで構成されるストリーミングデータフローにマップされる．このストリーミングデータフローは DAG（Directed acyclic graph）と似た構造を持ち，1 つ以上のソースで始まり，1 つ以上のシンクで終わる．ストリームは 1 つ以上のストリームパーティションに分割され，各オペレータは 1 つ以上の独立したサブタスクに分割される．オペレータのサブタスクは互いに独立しているため，異なるスレッド，あるいは異なるマシンまたはコンテナで実行される．つまり，オペレータのサブタスクの数は，その特定のオペレータの並列処理数を表す．各サブタスク間で，シャッフリングによるキーに基づいたグループ化を行える．

2.2 Samza

Samza は one-at-a-time を処理モデルとしている．

各ストリームは 1 つ以上のパーティションに分割され，各パーティション中のタプル（メッセージ）はある順序で並べられたシーケンスとなる．ジョブを複数に分割したタスクと呼ばれる並列処理の単位によって，入力ストリームのうちの 1 つのパーティションからデータを消費する．タスクは，メッセージオフセットの順序で各パーティションからのメッセージを順番に処理する．パーティション間では順序を定義しないため，各タスクは並列で処理できる．

2.3 Storm

Storm は one-at-a-time を処理モデルとしている．

Storm は Topologies と呼ばれる spout と bolt のグラフに基づいて処理を行う．spout は外部ソースからストリームを生成するノードであり，bolt は単純な変換を担うノードである．spout から出力されるストリームをシャッフルあるいはキーによるグループ化によって bolt の各タスクに割り振り，各タスクがそれらを処理して出力ストリームを生成する．spout，bolt のタスク

表 1　評価用サーバの構成

機器名	Dell PowerEdge R640
OS	Ubuntu 18.04.3 LTS
CPU	Intel(R) Xeon(R) Gold 6262V CPU @ 1.90GHz
メモリ	256GB
ストレージ	480GB

は並列実行される．

2.4 Spark Streaming

Spark は micro-batch を処理モデルとしている．

Spark では 1 つのデータソースから読み込んだデータを 1 つの RDD（Resilient Distributed Dataset）として扱い，1 つの RDD は複数のパーティションに分割され，1 パーティションを 1 タスクが変換処理する．このタスクが Spark クラスタ内の各ノードに分散配置され，並列で変換処理が行われる．各タスク内ではシャッフルを伴わない変換処理が行われ，シャッフルが必要になった時点でそのタスクは終了し，シャッフル後は新しいタスクで処理が行われる．

3　評価手段

メニーコア CPU における基礎的な性能調査として，本研究では並列（処理スレッド）数・キーの偏り・時間窓の幅の 3 点をパラメータとして変化させ，レイテンシを計測する．

ここではレイテンシを，「入力レコードがシステムに入力される時間」と「そのレコードに対応する結果が生じる時間」との間の期間として定義する．より具体的には，「ある特定のタプルが属する時間窓において初めてタプルが到着したときの時間」と「ある特定のタプルが属するすべての時間窓において結果の出力が終了するまでの時間」の差である．

Flink，Samza，Storm は one-at-a-time を処理モデルとしているのに対し，Spark は micro-batch を処理モデルとしているため，Spark で計測されるレイテンシは比較的大きくなることが予想される．

本研究では評価用のベンチマークとして，既存研究を参考に単純なデータセットを自作する予定である．

評価に使用するマシンの構成を表 1 に示す．

4　おわりに

本稿では，メニーコア CPU を搭載したマシンにおける既存の分散並列ストリーム処理 OSS の並列処理方式について議論した．今後は，レイテンシ計測のための実装を行い，実験により性能を測定する予定である．

謝辞

本研究は，JSPS 科研費（16H01722, 19K21530）の助成，および，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務による．

参考文献

[1] P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, and K. Tzoumas, “State management in Apache Flink®: consistent stateful distributed stream processing,” *PVLDB*, vol. 10, no. 12, pp. 1718–1729, 2017.

[2] S. A. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. H. Campbell, “Samza: stateful scalable stream processing at LinkedIn,” *PVLDB*, vol. 10, no. 12, pp. 1634–1645, 2017.

[3] A. Toshniwal, J. Donham, N. Bhagat, S. Mittal, D. Ryaboy, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, and M. Fu, “Storm @Twitter,” in *Proc. SIGMOD*, pp. 147–156, 2014.

[4] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, “Discretized streams: Fault-tolerant streaming computation at scale,” in *Proc. SOSP*, no. 1, pp. 423–438, 2013.