様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成　３０　年　　６　月　１４　日現在

機関番号：　８２６２６
研究種目：　若手研究(B)
研究期間：　2016～2017
課題番号：　１６Ｋ２１６７５

研究課題名（和文）ビッグデータ解析環境への効率的なデータステージングの研究

研究課題名（英文）Efficient Data Staging into The Big Data Analytics Platform

研究代表者
谷村　勇輔（Tanimura, Yusuke）

国立研究開発法人産業技術総合研究所・情報・人間工学領域・主任研究員

研究者番号：８０４１５７１０

交付決定額（研究期間全体）：（直接経費）　　3,200,000円

研究成果の概要（和文）：本研究では，ビッグデータ解析処理層（主計算インフラ）とバックエンド層（主ストレージインフラ）において，解析処理層で用いられるデータ構造をできる限り保持したままのデータ・ステージング，主ストレージインフラ側での解析前・解析後処理によるステージングの効率化，同時ステージングにおける性能干渉を防ぐためのスケジューリングに取り組み，SparkとAlluxioを用いたプロトタイプの設計と実装，主ストレージインフラ側での処理を含むAlluxioへのI/O性能評価，I/O性能干渉の軽減策を検討し，主計算インフラの多目的の運用や主ストレージインフラとの相補的な利用に向けた要素技術を開発した．

研究成果の概要（英文）：Data staging between the big data analytics platform (the main computing system) and the backend storage, pre- and post- processing in the backend to achieve efficient data staging, and appropriate scheduling to avoid performance interference by concurrent data staging was studied. The prototype system by using Spark and Alluxio was designed and implemented, and then I/O (staging) performance including storage-side processing was evaluated. The basic results would enable multi-tenant operation of the big data analytics platform and more effective use of the backend storage.

研究分野：　ストレージシステム

キーワード：　データストレージ　ビッグデータ解析　データステージング　資源管理　クラウド

１．研究開始当初の背景

人や物のインターネット（Internet of Anything）や人工知能，ビジネスインテリジェンス等，ビッグデータを扱う応用は大きく広がり，その解析やストレージ技術に注目が集まっている．特に，MapReduce や Google File System などの Google の研究成果を発端に，オープンな開発コミュニティにより発展してきた Hadoop とその周辺ソフトウェアはビッグデータの基盤システムとして広く用いられている．非構造データの「SQL＋α」の処理やストリーミングデータのリアルタイム処理，機械学習のような反復性の高い処理を扱うシステム等，現在は応用に合わせて個々に洗練化された仕組みを利用できるが，その基盤は Hadoop の MapReduce 処理フレームワークと分散ファイルシステム（HDFS）である．これは解析サーバとデータサーバを重ね合わせた構成を基本としており，単一の応用でデータを収集・解析するのに適しているが，マルチテナントの運用や階層的なデータ保管が容易ではない．例えば，複数の応用に用いる場合，応用毎にデータの増加率やアクセス特性等が異なるため，利用頻度の低いデータを退避させる必要が生じうる．あるいは，SSD 等の高速なディスクを用いて解析環境を構築する場合，HDFS の容量が十分でないために外部ストレージへのデータ退避が必要になる可能性がある．しかし，解析環境と外部ストレージシステム間のデータ移動において，データ構造の変換オーバーヘッドが大きい，バースト的なデータ転送が解析処理の性能に干渉する問題がある．

２．研究の目的

背景で述べたように，ビッグデータ解析の普及と応用は拡大を続けているが，現在主流のビッグデータ処理基盤は，外部とのデータ移動において，データ構造変換のオーバーヘッドや実行中の解析プログラムへの性能干渉の問題を抱えており，マルチテナントの運用や階層的なデータの保管が容易ではない．

本研究課題では，バックエンドの安価で巨大なストレージから高速なディスクを持つビッグデータ解析環境への効率的なデータ・ステージングの実現を目指し，解析処理層のデータ構造を保持したままのステージング，バックエンド・ストレージ側での解析前・後処理，性能干渉を軽減したステージング・スケジューリング手法を開発する．これらの開発により，高速なビッグデータ解析環境のマルチテナントの運用や，高度なデータ保存管理機能を有するストレージシステムとの相補的な利用を可能にすることを目指す．

３．研究の方法

本研究では，バックエンドの安価で巨大なストレージから高速なディスクを持つビッグデータ解析環境への効率的なデータ・ステージングの実現を目指し，ビッグデータ解析処理層（主計算インフラ）とバックエンド層（主ストレージインフラ）において，解析処理層で用いられるデータ構造をできる限り保持したままデータ・ステージングを行うとともに，主ストレージインフラ側での解析前・解析後処理によるステージングの効率化，同時ステージングにおける性能干渉を防ぐためのスケジューリングに取り組んだ（図１）．

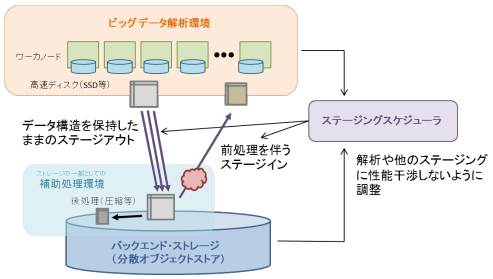

図１　提案概要

まず，利用ケースに基づいた全体設計を行い，解析処理層に Apache Spark，メモリや高速なディスクで構成する中間層（ステージング層）に Alluxio，容量単価の優れたディスクで構成するストレージ層に Hadoop Distributed File System（HDFS）および Ceph RADOS を想定し，提案手法のプロトタイプ実装と評価を行うこととした．なお，これらの３層は従来，１つの計算ノード群に重ねて用意されるのが一般的であるが，本研究では最終的に上位１層を計算ノード群，下位２層をストレージノード群に用意する設計としている（図２）．

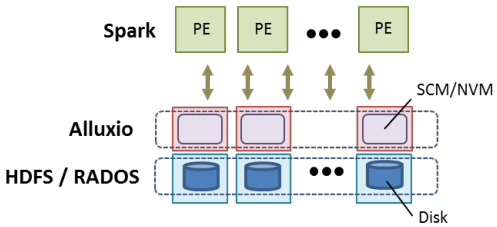

図２　プロトタイプ・アーキテクチャ

次に，上記設計に沿って，(1) Spark のデータ入出力の基本性能評価と高速化を行い，その上で Alluxio を中間層に用いた場合の I/O 性能比較，(2) Alluxio 内部でのデータ処理機構の実装と主ストレージインフラへのデータ処理のオフロードに関する評価，(3) 同時ステージングにおける性能干渉の調査と干渉の軽減策に関する評価を行った．

４．研究成果

本節では３つのサブ課題毎に成果を述べた後，本研究全体の成果についてまとめを述べる．

(1) Spark のデータ入出力における高速なロ

ーカルディスクとリモートの Alluxio ストレージの性能比較

まず，Spark で利用されるデータフォーマット，Alluxio および類似機能を提供する分散メモリストアのデータ構造，バックエンド層のストレージ内のデータ構造について調査し，ステージング対象のデータに対して解析前・解析後の処理を適用することを目標に，データ構造の変換を最小限に抑える手法の検討を進めた．その結果，Spark の DataFrame がサポートしている Parquet は，Spark および Alluxio の両方で利用可能であるため，後述する Alluxio 内部でのデータ処理機構の実装に有用であることが確認できた．

次に，Spark Version 2 以降のデータ入出力において，Alluxio をリモートストレージとして利用する場合の性能を調査した．調査は RDD と DataFrame の両方のデータ抽象化に関して，また複数のデータ型や圧縮手法等に関して行った．そして，Spark のワーカノードに直接接続された高速なローカルディスクを利用する場合と比較して，Spark へのデータ読込ではほぼ同等の性能が得られることを確認した．これは３節で述べた設計が有効である可能性を示唆する結果である．一方，Alluxio へのデータの書込は圧縮がない場合は遅く（表１），その原因調査は今後の課題である．

表１ Spark プログラムからの String 配列の書込速度の比較

1) RDD を用いた場合

	HDFS	ALLUXIO	OFF HEAP	DISK ONLY
Cache Time (s)	4.142	3.665	3.133	3.418
Total Time (s)	19	18	21	21
Cache size(MB)	1024	1024	1024	1024

2) DataFrame を用いた場合

	HDFS		ALLUXIO		OFF HEAP		DISK ONLY	
Compress	true	false	true	false	true	false	true	false
Cache Time (s)	7.348	7.599	7.026	7.082	8.150	4.932	8.344	4.989
Total Time (s)	26	24	24	22	23	19	24	20
Cache size(MB)	1024	1024	1024	1024	845	1075	847	1075

(2) Alluxio 内部でのデータ処理機構の実装と評価

最初に予備調査として，Ceph RADOS を対象にバックエンド層で解析前・解析後の処理を行う仕組みの実装方法を検討した．RADOS では Luca による仕組みが用意されており，ストレージ側で行う汎用的な処理の実装には有用であることが確認できたが，既存アプリケーションの処理の一部をオフロードするには再実装のコストが大きいとの結論に至った．

これを踏まえて，ストレージノード群ではなく，ステージング層，つまり Alluxio に処理をオフロードする設計とした．Alluxio は(1)の取り組みで確認できたように，Spark と共通のデータフォーマット（Parquet）を扱える利点があり，性能面の問題もない．なお，HDFS には Ceph RADOS の問題はないが，バックエンド側に両方のストレージ，あるいはそれら以外のストレージを使えるという点でも Alluxio に処理をオフロードする設計が妥当だと考えた．

次に，Alluxio（Version 1.7.0）のソースコードを改良し，Alluxio 内部にオンデマンドのフィルタリング機構を試験的に実装した（図３，図４）．しかし，Alluxio 単体での性能評価（図５），Alluxio 側での事前フィルタリングと，Spark（Version 2.2.0）実行環境でのフィルタリングの性能比較（図６）では，Alluxio 内部のデータ処理機構のオーバーヘッドが予想以上に大きく，かつ Spark 実行環境でのフィルタリング処理が十分に高速であったため，解析前・後処理のオフロードについて性能面での優位性を十分に示すには至らなかった．

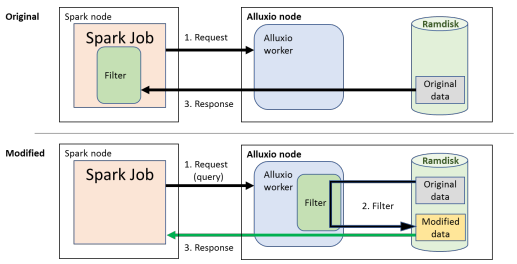

図３ Alluxio 内部におけるオンデマンドのフィルタリング実装（Original が実装（改良）前，Modified が実装後を示している．）

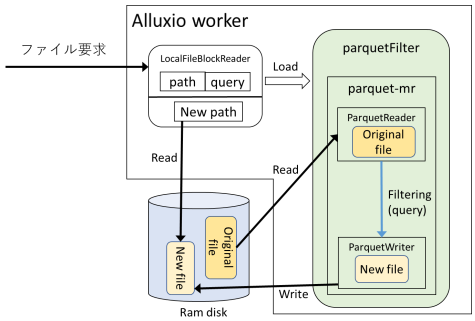

図４ Alluxio Worker 内部のフィルタリングの実装

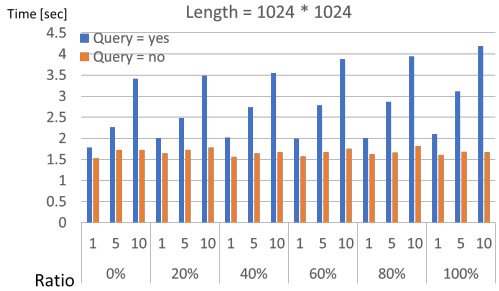

図５ Alluxio 単体での性能評価（Query = yes/no は，Alluxio 内部でのフィルタリングの適用の有無を示している．）

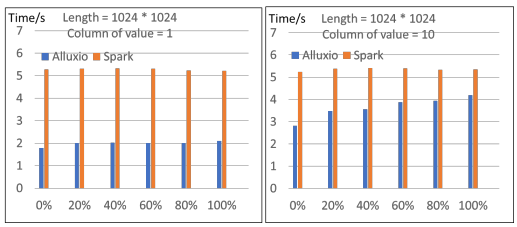

図６ Alluxio 側での事前フィルタリングと Spark 側でのフィルタリングの性能比較（横軸はフィルタリングにおける Selectivity，縦軸は実行時間を示している．左図は Key-Value データの Value(Double 型)が１つの時の結果，右図は Value（Double 型）が 10 個の配列の時の結果を示している．）

Spark 内部のフィルタリング実装と同等の手法を Alluxio 内部のデータ処理機構に実装したり，Spark がサポートしないフィルタリング以外の独自処理を Alluxio 側で実行したりする場合には，依然として，オフロードに性能面での優位性があると思われるが，それを明らかにすることは今後の課題である．

(3) 同時ステージングにおける性能干渉の影響の調査とその軽減

まず，Spark アプリケーションが Alluxio に対して Read や Write を行う操作と Alluxio がストレージ層に対して Read や Write を行う操作が同時に行われた際のぞれぞれの性能への影響を評価した．その結果，I/O 操作の同時実行により，最大で 15%の性能低下が見られ（図７），データ・ステージングを適切にスケジューリングすることの有効性を確認できた．

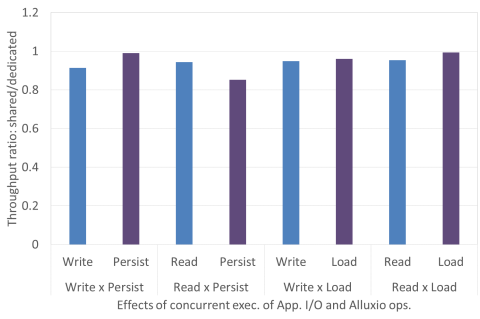

図７ Spark アプリケーションのステージング I/O とバックエンド層内の I/O が競合した場合の性能干渉の度合い

次に，お互いの性能干渉を防ぐため，複数のステージングのタイミングを半自動的に調整する試みを検討した．その結果，ストレージ側でのデータ処理を前提とする場合は，I/O 監視だけでなく，オフロードしたデータ処理の速度予測が重要になること，Spark 実行環境におけるキャッシングの仕組みとの連携が重要になることが明らかになった．

(4) まとめ

３つのサブ課題の取り組みを通して，ビッグデータ解析環境として Spark を利用する場合に，Alluxio および Parquet を用いることで，解析処理層のデータ構造を保持したままのステージング，そしてバックエンド・ストレージ側での解析前・後処理が可能になることを示した．また，その際に性能干渉を軽減するスケジューリングの有効性を示した．一方，システム全体の効率や汎用性に関する定量的な評価，具体的なスケジューリングアルゴリズムの開発には，複数の解決すべき問題が明らかになった．これらへの取り組みは今後の課題としたい．

５．主な発表論文等

（研究代表者、研究分担者及び連携研究者には下線）

〔学会発表〕（計３件）

① Yusuke Tanimura, Towards Efficient Data Staging for Multi-Tenant Big Data Analytics, he 25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC), 査読無, 2016, 1-2.

② Kaihui Zhang, Yusuke Tanimura, Hidemoto Nakada, Hirotaka Ogawa, Understanding and Improving Disk-based Intermediate Data Caching in Spark, The 6th Workshop on Scalable Cloud Data Management in 2017 IEEE International Conference on Big Data, 査読有, 2017, 1-10.

③ Kaihui Zhang, Yusuke Tanimura, Hidemoto Nakada, Hirotaka Ogawa, Storage-Side Processing for Spark with Tiered Storage, 情報処理学会 第 163 回ハイパフォーマンスコンピューティング研究会, 査読無, 2018, 1-6.

６．研究組織

(1)研究代表者

谷村 勇輔（TANIMURA, Yusuke）

産業技術総合研究所・人工知能研究センター・主任研究員

研究者番号：80415710

(2)研究分担者

(3)連携研究者