Vol.2021-ARC-245 No.12
2021/7/21

Apache Sparkでの異種ストレージ活用に向けた予備評価

今村 智史[1,a)]　風間 哲[1]　吉田 英司[1]

概要：
大量のデータから有益な情報を抽出するためのデータ分析処理の実行には分散処理フレームワークが広く利用されており，特にインメモリ処理を前提としたApache Spark（以下，Spark）が現在主流となりつつある．また，分散処理向けのクラスタシステムを容易に構築できるパブリッククラウドにおいてSparkを利用するケースも近年増えている．Sparkでは，シャッフルと呼ばれるI/Oインテンシブな処理を高速化するために高性能なストレージが必要であるが，一般的には高性能なストレージほど容量単価が高くなる．そのため，多様な種類のストレージが利用可能なパブリッククラウドにおいては，複数種類のストレージを適切に組み合わせることで高いI/O性能を維持しつつストレージコストを削減することが望ましい．
本論文では，Sparkのシャッフル処理において生成される中間ファイルのアクセス特性を調査し，AWS（Amazon Web Services）において2種類のストレージボリューム（HDDとSSD）を併用する場合の性能とストレージコストを評価する．両ボリュームを併用する方法としては，Linuxで利用可能なストレージキャッシング技術bcacheを用いる場合と種類の異なる中間ファイルを両ストレージに手動で分割して格納する場合を比較する．AWSにて構築したクラスタシステムにおいてSpark向けのSortとTerasortベンチマークを用いて評価した結果，これら両アプローチによりSSDのみを使用する場合と同等の性能を達成しつつストレージコストをおよそ半減できることが明らかとなった．

1. はじめに

企業や大学が有する膨大な量のデータからビジネスや研究に有益な情報を抽出するには大規模なデータ分析処理が必要である．こうした分析処理は，複数の計算ノードから構成されるクラスタシステムにおいて分散処理フレームワークを用いて実行されることが一般的である．以前はストレージベースのApache Hadoopといったフレームワークが広く利用されていたが，近年ではインメモリでの高速な分散処理を前提として設計されたApache Spark（以下，Spark）が主流となりつつある．

インメモリ処理を前提としたSparkにおいても，データの形式変換を行うMap処理とデータの集約を行うReduce処理の間でシャッフルと呼ばれるI/Oインテンシブな処理が必要である．シャッフル処理はSparkで使用する全計算ノード間でデータを交換する処理であり，この処理中に生成されるシャッフルデータは耐故障性を保証するためにファイル（以下，シャッフルデータファイル）としてストレージに保存される [1]．また，シャッフル処理中にメモリ容量が不足する場合には，シャッフルデータの一部をストレージ上の一時ファイル（以下，スピル一時ファイル）に退避するスピルと呼ばれる処理が発生する．たとえば，Facebookではメモリ容量の10倍を超えるサイズのデータをSparkを用いて処理することが報告されており [2]，こうしたケースではシャッフル処理およびスピル処理を高速化するために高いI/O性能が必要となる．

また，近年では，パブリッククラウドにおいてSparkを利用する事例が増えている [3–5]．パブリッククラウドでは，あらかじめ用意された仮想マシンインスタンスから複数の計算ノードを短時間で配備できるため，Sparkを実行するためのクラスタシステムを容易に構築できる．さらに，性能と価格の異なる多様なインスタンスとストレージボリュームが利用できるため，ワークロードの特性に応じてシステム構成を柔軟に変更できる．特に，Sparkのシャッフル処理およびスピル処理に関しては，複数種類のストレージボリュームを適切に組み合わせることで高いI/O性能を得つつストレージコストを抑えることが望ましい．

そこで，本論文では，Sparkのシャッフル処理において種類の異なる複数のストレージボリュームを活用するための第一歩として，シャッフルデータファイルおよびスピル一時ファイルのアクセス特性を調査し，AWSにおいて2種類のストレージボリューム（ローカルNVMe SSDと高性能HDD）を併用する場合の性能とストレージコストを評価す

[1] 富士通株式会社
[a)] s-imamura@fujitsu.com

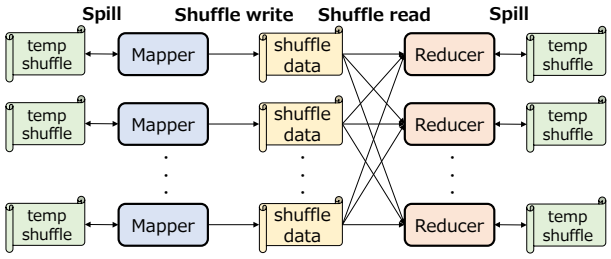


図 1: スピル処理を伴うシャッフル処理の模式図

る．ファイルアクセス特性の調査では，シャッフルデータファイルのサイズがスピル一時ファイルに比べ大きく，かつ，アクセス頻度が低いことを示す．そして，AWS にて構築したクラスタシステムにおいて，Linux で利用可能なストレージキャッシング技術 bcache を用いて両ボリュームを併用する場合とシャッフルデータファイルとスピル一時ファイルを両ボリュームに手動で分割して格納する場合を評価する．この結果，両ボリュームを併用することでローカル NVMe SSD のみを使用する場合と同等の性能を達成しつつストレージコストをおよそ半減できることを示す．

2. 背景

2.1 Apache Spark

Spark は大規模データ分析向けのオープンソース分散処理フレームワークであり，Apache Hadoop の後継として現在では分散処理フレームワークの主流となりつつある．Spark には，機械学習やストリーミング処理，SQL，グラフ処理向けの様々なライブラリが含まれており，Python や R，Scala，Java といったプログラミング言語向けの API がサポートされている．ストレージベースの Hadoop に対し Spark はインメモリ処理を前提として設計されており，Hadoop に比べ最大 100 倍高速であると報告されている [6]．Spark は一般的に HDFS（Hadoop Distributed File System）と組み合わせて使用されることが多く，HDFS 上のファイルからメモリに読み込んだ入力データを RDD（Resilient Distributed Dataset）と呼ばれるデータ単位に分割し，複数タスクで並列処理を行う．

Spark では，データの形式変換を行う Map 処理とデータの集約を行う Reduce 処理の間で計算ノード間のデータ交換を行うシャッフル処理が発生する．そのため，Spark で実行されるワークロードの処理は，シャッフル処理を堺に複数のステージに分割される．また，シャッフル処理中にメモリ容量が不足する場合には，一部のデータをストレージへ一時的に退避するスピル処理が発生する．図 1 にスピル処理が発生する場合のシャッフル処理の模式図を示す．各ワーカノードで稼働する Mapper タスクは指定ディレクトリ内のファイル（以下，シャッフルデータファイルと呼称）にデータを書き込むシャッフルライト処理を行い，Reducer タスクは全ワーカノード上のシャッフルデータファイルからデータを取得するシャッフルリード処理を行う．また，スピル処理が必要である場合には，各タスクはシャッフルデータの一部を指定ディレクトリ内のファイル（以下，スピル一時ファイルと呼称）に退避し，必要に応じて退避したデータを再度メモリに読み出す．このようにシャッフル処理とスピル処理ではストレージに格納されたファイルへのアクセスが発生するため，両処理は I/O インテンシブであることが知られている．なお，シャッフルデータファイルおよびスピル一時ファイルは，*spark.local.dir* パラメタで指定された同一のディレクトリに格納される．

2.2 AWS において利用可能なストレージボリューム

表 1: AWS で利用なストレージボリュームの種類 [7]

名称	種別	最大逐次リードスループット	$/GB/月 [8]
eph	NVMe SSD*	800 MB/s**	0.216**
gp2	汎用 SSD	250 MB/s†	0.120
io2	IOPS SSD	1,000 MB/s‡	0.142+α‡
st1	高性能 HDD	500 MB/s†	0.054
sc1	コールド HDD	250 MB/s†	0.018

*インスタンス停止時にデータ消失，**本論文での取得結果
† ボリュームサイズに依存，‡IOPS 指定値に依存

パブリッククラウドでは性能や容量単価の異なる多様な種類のストレージボリュームが利用可能であり，ウェブコンソールやコマンドラインインターフェイスを介して仮想マシンインスタンスに容易に接続できる．本論文では AWS を利用するため，AWS にて使用可能な 5 種類のストレージボリュームを表 1 にまとめる．

eph はインスタンスが配置されたサーバに物理的に接続されたローカル NVMe SSD であり，インスタンスに初めから含まれる形で提供される．本論文の実験にて使用する c5d.4xlarge インスタンス（詳細は 3.1 節を参照）に付属された eph では，fio ツールを用いて計測した逐次リードスループットが最大 800 MB/s であった．また，本表内の eph の一ヶ月における GB あたりの利用料（$0.216/GB/月）は，c5d.4xlarge インスタンスと c5.4xlarge インスタンス（eph が付属されない以外は同一の構成を持つ）の一ヶ月における利用料の差分から算出した値である．ただし，eph ではデータ永続性が保証されておらず，eph に格納されたデータはインスタンス停止時に消失する．

これに対し，eph 以外の 4 種類のボリュームは EBS（Elastic Block Store）と呼ばれるネットワーク接続のストレージボリュームであり，データ永続性が保証される．gp2 は様々なワークロードに対応可能な性能とコストのバランスが取れた汎用 SSD であり，一般的にインスタンスのルートボリュームとして使用される．io2 はユーザが指定した IOPS 値を保証する高性能 SSD であり，最大で 1,000 MB/s

のスループットが得られる一方，容量あたりの利用料に加え指定した IOPS 値に応じて利用料が増加する．最後に，st1 と sc1 は低いコストが特徴の HDD ボリュームであり，st1 は sc1 に比べコストが高い分スループットが高い．

上記のように，各ボリュームは性能面もしくはコスト面でそれぞれ優れた特徴を持つため，高い I/O 性能を得つつストレージコストを抑えるためには複数種類のボリュームを適切に組み合わせることが望ましい．なお，本論文では eph と st1 の併用方法を検討する．

2.3 Spark 向け異種ストレージ活用の関連研究

ここでは Spark 向けに提案された複数種類のストレージを活用する技術を紹介する．Islam ら [9] および Pan ら [10] は，HDFS 上で 3 重複製されたデータの内ひとつが SSD，残りふたつが HDD に格納されることを前提とし，データローカリティと両ストレージの性能差を考慮したタスク管理技術を提案している．これらの技術では HDFS 上に格納されたファイルへの効率的なアクセスを実現できるが，シャッフル処理において生成される中間ファイルは考慮されていない．

一方，HDFS 上に格納された入出力データとシャッフル中間データの I/O アクセス特性分析により，各データの格納先として HDD もしくは SSD を使用する場合の性能差が入出力データに比べシャッフル中間データの方が大きいことが報告されている [1, 11, 12]．この結果から，Zhou ら [1] は，ゲノム分析ワークロードの各ステージの実行時間を予測するモデルを提案し，その予測時間を基に入出力データとシャッフル中間データをそれぞれ HDD と SSD に分割して格納することでストレージコストを削減できることを Google Cloud において実証している．また，Klimovi ら [12] は，対象のワークロードに適したインスタンスタイプに加え入出力データとシャッフル中間データを格納するのに適したストレージ構成をパブリッククラウドにおいて推奨する技術を提案し，AWS における 100 種類以上のワークロードを用いた評価実験を通して性能とコスト両面から適切な構成を高い精度で予測できることを示している．

本論文では，シャッフル中間データが格納されるシャッフルデータファイルおよびスピル一時ファイルに対するアクセス特性を調査し，AWS にて利用できる 2 種類のストレージボリューム（eph と st1）にそれらを分割して格納する場合の性能とストレージコストを評価する．

3. シャッフル処理におけるファイルアクセスの特性調査

3.1 実験環境

本論文の実験では，AWS において同一種類のインスタンスを 4 つ用いて構築したクラスタシステムを使用する．その構成を表 2 にまとめる．Spark を管理するためのマスタノードとして 1 インスタンスを使用し，残り 3 つのインスタンスをワーカノードとして使用する．各ノードには 400 GB の eph（ローカル NVMe SSD）が付属された c5d.4xlarge インスタンスを使用し，さらに 2 TB の st1 EBS ボリューム（高性能 HDD）を各インスタンスに 2 つずつ接続する．st1 のひとつは HDFS に使用し，eph ともうひとつの st1 はシャッフルデータファイルおよびスピル一時ファイルを格納するために使用する．なお，eph の一ヶ月における GB あたりの利用料は表 1 に示した通り st1 の 4 倍である．Spark のパラメタには AWS にて推奨される設定 [13] を採用し，本表に記載されていないパラメタはデフォルト設定である．

表 2: 実験用クラスタシステムの構成

ノード数	マスタノード ×1 + ワーカノード ×3
インスタンスタイプ	EC2 c5d.4xlarge - 16 vCPU，32 GB メモリ，Ubuntu 20.04 - 400 GB **eph** ×1，2 TB **st1** ×2
ソフトウェア	Spark 2.4.5，Hadoop 2.7.7
Spark のパラメタ設定	executor.cores = 5 executor.instances = 9（ワーカあたり 3） executor.memory = 9 GiB（ワーカあたり 27） default.parallelism = 90 (5×9×2)

ベンチマークプログラムには，シャッフル処理がボトルネックとなることで知られる HiBench 7.1.1 の Sort と Terasort を用いる．表 2 に示したクラスタシステムにおいてシャッフル処理中にスピル処理が発生するよう，両プログラムとも入力データサイズを 90 GB に設定する．各入力データはあらかじめ HDFS 上に生成しておくため，この生成時間は各ベンチマークの実行時間に含まれない．

3.2 合計ファイルサイズの調査

本節では，スピル処理を伴うシャッフル処理において生成されるファイルの合計サイズを調査する．シャッフルデータは，*spark.local.dir* パラメタで指定されたディレクトリ以下の “*shuffle_*.data*” という名前のシャッフルデータファイルに格納される．そのため，各ベンチマークの実行中に生成されるシャッフルデータファイルの合計サイズを 1 秒毎に測定し，その最大値をシャッフルデータサイズとする．また，同ディレクトリ内に含まれる全ファイルの合計サイズからシャッフルデータサイズを差し引くことで，シャッフルデータファイル以外の中間ファイル（スピル一時ファイル含む）の合計サイズを算出する．

Sort と Terasort ベンチマークの実行中に測定したシャッフルデータファイルの合計サイズとその他の中間ファイル（スピル一時ファイル含む）の合計サイズを図 2 に示す．このグラフから，両ベンチマークともシャッフルデータファイルの合計サイズがその他の中間ファイルの合計サイズよ

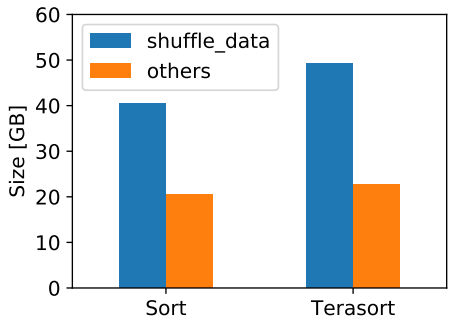


図 2: Sort と Terasort ベンチマーク実行時のシャッフル処理において生成されるファイルの合計サイズ

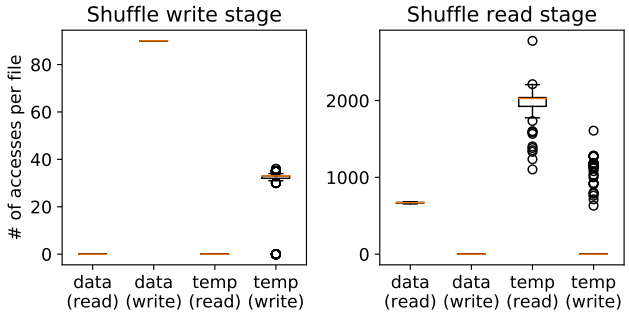
り約 2 倍大きいことが分かる．

3.3 ファイルアクセス回数の調査

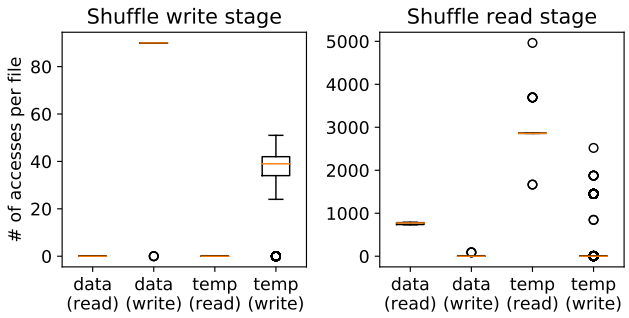
次に，シャッフル処理において生成されるシャッフルデータファイルとスピル一時ファイルに対するアクセス回数を調査する．シャッフルデータは上述の通りシャッフルデータファイルに格納される一方，スピルデータは同ディレクトリ内の “*temp_shuffle_**” という名前のスピル一時ファイルに格納される．そのため，シャッフルデータファイルとスピル一時ファイルに対して発行されるリード/ライトシステムコールの回数を Linux の strace コマンドを用いてそれぞれ測定する．Sort と Terasort ベンチマークの各ステージ実行中に測定したファイル毎のリード/ライトシステムコールの回数を箱ひげ図として図 3 に示す．

Sort ベンチマーク（図 3a）では，シャッフルライト処理を行うステージ（以下，シャッフルライトステージと呼称）にて各シャッフルデータファイルに対し約 90 回のライトが発生し，シャッフルリード処理を行うステージ（以下，シャッフルリードステージと呼称）にて約 700 回のリードが発生している．また，シャッフルライトステージでは，各スピル一時ファイルに対して約 30 回のライトが発生しており，シャッフルデータファイルに比べアクセス回数が少ない．これに対し，シャッフルリードステージでは，スピル一時ファイルに対し数百から数千回のリードおよびライトが発生しており，シャッフルデータファイルに比べアクセス回数が数倍多いことが分かる．

Terasort ベンチマーク（図 3b）においても Sort ベンチマークと同様の傾向が見られる．シャッフルライトステージでは，シャッフルデータファイルとスピル一時ファイルに対してそれぞれ数十回のライトが発生している．シャッフルリードステージでは，シャッフルデータファイルに対して約 900 回のリードが発生している一方，スピル一時ファイルに対してはその数倍多くのリードおよびライトが見られる．



(a) Sort ベンチマーク



(b) Terasort ベンチマーク

図 3: シャッフル処理を行うステージにおけるファイル種類毎のファイルアクセス回数

4. 性能とストレージコストの評価

4.1 評価方法

前節のファイルアクセス特性調査の結果から，サイズが大きくアクセス頻度が低いシャッフルデータファイルを低速で安価なストレージボリュームに格納し，サイズが小さくアクセス頻度が高いスピル一時ファイルを含むその他の中間ファイルを高速で高価なボリュームに格納することで，高い I/O 性能を維持しつつストレージコストを抑えられることが予想できる．そこで，本節では，Spark のシャッフル処理に対して eph（ローカル NVMe SSD）と st1（高性能 HDD）の 2 種類のストレージボリュームを併用する場合の性能とストレージコストを評価する．

eph と st1 を併用する 1 つ目の方法として，Linux に含まれるストレージキャッシング技術 bcache を利用する．bcache は，2 種類のストレージデバイスをキャッシュ/バッキングデバイスとして組み合わせ単一のブロックデバイスを構築し，キャッシュ制御を行う技術である．本実験では，キャッシュポリシを writeback，キャッシュ追い出しアルゴリズムを LRU（Least-Recentry Used）に設定し，eph と st1 をそれぞれキャッシュ/バッキングデバイスとして使用する．バッキングデバイスとしての st1 の容量は各ベンチマークのシャッフル処理において生成される全ての中間ファイルを格納できる値（図 2 に示したシャッフルデータファイルとその他の中間ファイルのサイズの合計）に設定

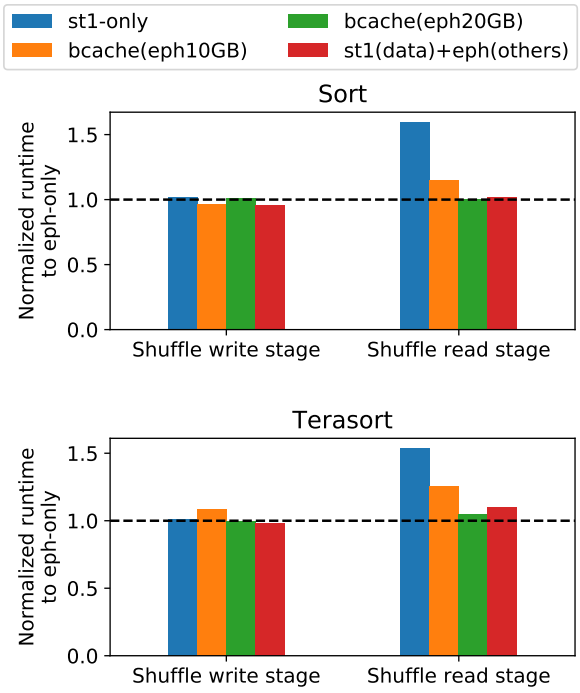


図 4: Sort と Terasort ベンチマークのシャッフル処理を行うステージ毎の正規化実行時間

し，キャッシュデバイスとしての eph の容量は 10 GB もしくは 20 GB に設定する．図 2 に示した通りシャッフルデータファイル以外の合計ファイルサイズが約 20 GB であるため，前者の場合はその半分がキャッシュされるのに対し，後者の場合はそのほぼ全てがキャッシュされる．また，bcache は逐次 I/O アクセスのキャッシングをバイパスする *sequential_cutoff* と呼ばれる機能を有し，デフォルト設定では有効化されている．しかしながら，この機能を有効化した場合には Sort と Terasort ベンチマークのシャッフル処理に対するキャッシングの効果が極めて小さかったため，本節の実験においてはこの機能を無効化する．

また，eph と st1 を併用するもう 1 つの方法として，シャッフルデータファイルとその他の中間ファイル（スピル一時ファイル含む）をそれぞれ異なるディレクトリに格納できるよう Spark のソースコードを修正し，各ディレクトリに st1 と eph をそれぞれマウントする．なお，本実験では，bcache デバイスを含め全てのストレージボリュームを EXT4 ファイルシステムでフォーマットし所定のディレクトリにマウントする．

4.2 性能評価

4 通りのストレージ構成にて計測した Sort と Terasort ベンチマークの各ステージの実行時間を図 4 に示す．なお，縦軸の各ステージの実行時間は，eph のみを使用する場合の実行時間で正規化されている．Sort ベンチマークのシャッフルライトステージでは，図 3a に示した通りシャッフルデータファイルとスピル一時ファイルのアクセス回数

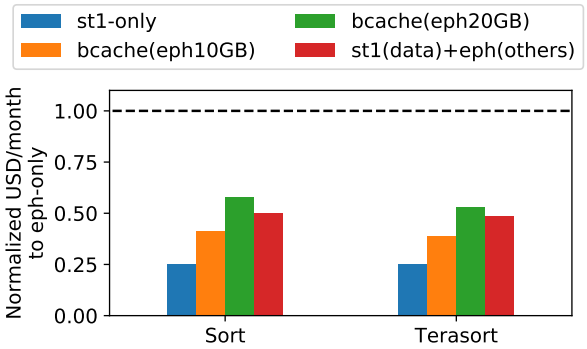


図 5: Sort と Terasort ベンチマーク実行時の正規化ストレージコスト

がいずれも少ないため，ストレージ構成が変化してもほとんど実行時間に差がない．これに対し，シャッフルデータファイルが約 700 回，スピル一時ファイルが数千回アクセスされるシャッフルリードステージでは，st1 のみを使用する場合に実行時間が約 59%長くなる．これに比べ，bcache を用いて eph と st1 を併用する場合には実行時間が大幅に削減される．特に，キャッシュデバイスとしての eph の容量を 20 GB に設定した場合には，シャッフルデータファイル以外のアクセス回数の多い中間ファイルがほぼ全てキャッシュされるため，eph のみを使用する場合と同等の性能が得られる．さらに，シャッフルデータファイルとその他の中間ファイルをそれぞれ st1 と eph に手動で分割して格納する場合にも eph のみを使用する場合と同等の性能が得られた．

また，Terasort ベンチマークにおいても Sort ベンチマークと同様の実験結果が得られた．シャッフルライトステージでは，いずれのストレージ構成でも実行時間にわずかしか差が見られない．これに対し，ストレージ構成の影響が顕著に見られるシャッフルリードステージでは，キャッシュデバイスとしての eph の容量を 20 GB に設定した bcache を用いる場合とシャッフルデータファイルおよびその他の中間ファイルを st1 と eph に手動で分割して格納する場合に eph のみを使用する場合と同等の性能が得られた．

4.3 ストレージコスト評価

次に，Sort と Terasort ベンチマークを実行する際の各ストレージ構成のコストを比較する．ここでは各ストレージボリュームを一ヶ月間インスタンスに接続し続ける状況を想定し，表 1 に示した一ヶ月における GB あたりのコストとそのボリュームに格納するファイルの合計サイズ（単位は GB）の積により各ストレージ構成の一ヶ月におけるストレージコストを算出する．4 通りのストレージ構成にて Sort と Terasort ベンチマークの実行に要するストレージコストを図 5 に示す．なお，縦軸のストレージコストは eph のみを使用する場合のコストで正規化されている．

st1 のみを使用する場合は図 4 に示した通り性能が最も低い一方で，eph のみを使用する場合に比ベストレージコストが 75%安価である．また，st1 と eph を併用する bcache では，キャッシュデバイスとして使用する eph の容量によってコストが変動するものの，ストレージコストをおよそ半減できる．また，シャッフルデータファイルとその他の中間ファイル（スピル一時ファイル含む）を st1 と eph に手動で分割して格納する場合でもストレージコストを半減できることが分かる．この場合には，キャッシュ容量を 20 GB に設定した bcache よりストレージコストが安価となる．これは，キャッシュデバイスとしての eph の容量がワークロードから利用できず st1 の容量が全シャッフル中間ファイル分必要である bcache に対し，st1 と eph いずれの容量も利用できるためである．

5. おわりに

本論文では，分散処理フレームワークとして広く利用される Spark の I/O インテンシブなシャッフル処理およびスピル処理に着目し，AWS において 2 種類のストレージボリューム（ローカル NVMe SSD と高性能 HDD）を併用することの有効性を検証した．それぞれの処理中に生成されるシャッフルデータファイルとスピル一時ファイルへのアクセス特性を調査した結果，シャッフルデータファイルは比較的サイズが大きくアクセス頻度が低いことが明らかとなった．また，この結果を基に，AWS にて構築した Spark 環境において Sort と Terasort ベンチマークを実行する際の性能とストレージコストを評価した．その結果，bcache を用いて両ボリュームを併用する場合もしくはシャッフルデータファイルとその他の中間ファイルを両ボリュームに手動で分割して格納する場合に，eph のみを使用する場合と同等の性能を達成しつつストレージコストをおよそ半減できることを示した．

今後は，シャッフルデータファイルとスピル一時ファイル以外に Spark 実行中に生成されるファイルのアクセス特性を調査し，3 種類以上のストレージボリュームを併用する方法やデータ配置の自動最適化手法を検討していく．今回は 2 種類の中間ファイルのみを対象としたため 2 種類のボリュームを併用できる bcache によって望ましい効果が得られたが，3 種類以上のボリュームを組み合わせる必要がある場合には bcache の構成が複雑になり十分な効果が得られなくなることが予想できる．

参考文献

[1] Zhou, P., Ruan, Z., Fang, Z., Shand, M., Roazen, D. and Cong, J.: Doppio: I/O-Aware Performance Analysis, Modeling and Optimization for In-memory Computing Framework, *Proceeding of 2018 IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS '18, pp. 22–32 (2018).

[2] Zhang, H., Cho, B., Seyfe, E., Ching, A. and Freedman, M. J.: Riffle: Optimized Shuffle Service for Large-Scale Data Analytics, *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18 (2018).

[3] Amazon: Amazon EMR の Apache Spark，Amazon（オンライン），入手先 ⟨https://aws.amazon.com/jp/elasticmapreduce/details/spark/⟩（参照 May, 2021）.

[4] Google: Hadoop や Spark のクラスタを Google Cloud Platform に移行する，Google（オンライン），入手先 ⟨https://cloud.google.com/hadoop-spark-migration?hl=ja⟩（参照 May, 2021）.

[5] Microsoft: Apache Spark とは - Azure HDInsight，Microsoft（オンライン），入手先 ⟨https://docs.microsoft.com/ja-jp/azure/hdinsight/spark/apache-spark-overview⟩（参照 May, 2021）.

[6] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S. and Stoica, I.: Spark: Cluster Computing with Working Sets, *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, p. 10 (2010).

[7] Amazon: Amazon EBS ボリュームの種類，Amazon（オンライン），入手先 ⟨https://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/ebs-volume-types.html⟩（参照 May, 2021）.

[8] Amazon: Amazon EBS の価格，Amazon（オンライン），入手先 ⟨https://aws.amazon.com/jp/ebs/pricing/⟩（参照 May, 2021）.

[9] Islam, N. S., Wasi-ur Rahman, M., Lu, X. and Panda, D. K. D.: Efficient Data Access Strategies for Hadoop and Spark on HPC Cluster with Heterogeneous Storage, *Proceeding of 2016 IEEE International Conference on Big Data*, Big Data '16, pp. 223–232 (2016).

[10] Pan, F., Xiong, J., Shen, Y., Wang, T. and Jiang, D.: H-Scheduler: Storage-Aware Task Scheduling for Heterogeneous-Storage Spark Clusters, *Proceeding of 2018 IEEE 24th International Conference on Parallel and Distributed Systems*, ICPADS '18, pp. 1–9 (2018).

[11] Ruan, X. and Chen, H.: Improving Shuffle I/O Performance for Big Data Processing using Hybrid Storage, *Proceeding of 2017 IEEE International Conference on Computing, Networking and Communications*, ICNC '17, pp. 476–480 (2017).

[12] Klimovic, A., Litz, H. and Kozyrakis, C.: Selecta: Heterogeneous Cloud Storage Configuration for Data Analytics, *Proceeding of 2018 USENIX Annual Technical Conference*, USENIX ATC '18, pp. 759–773 (2018).

[13] Shanmugam, K.: Best practices for successfully managing memory for Apache Spark applications on Amazon EMR, Amazon (online), available from ⟨https://aws.amazon.com/jp/blogs/big-data/best-practices-for-successfully-managing-memory-for-apache-spark-applications-on-amazon-emr/⟩ (accessed May, 2021).