Vol.2014-HPC-145 No.3
2014/7/28

データ処理平準化による業務アプリ並列実行の高速化

黒松 信行[1,a)]　松田 雄一[1,b)]　上田 晴康[1,c)]

概要：ユーザが既存の業務アプリを修正することなく Hadoop 上で実行することでバッチ処理を高速化する技術として，富士通は NetCOBOL Hadoop 連携機能を開発した．しかし，Reduce タスクの入力データの偏りが原因で実行時間がばらつくために Hadoop の並列効果が低下する問題があった．この問題は Hadoop の仕組みと入力データ中に含まれるキーの出現数の偏りに起因する Reduce-Skew に原因がある．一般に，業務アプリはキーの種類が少ないため出現数が偏ることが多く，NetCOBOL Hadoop 連携機能では Reduce-Skew が生じやすい．我々は業務アプリを修正せずにバッチ処理を高速化する NetCOBOL Hadoop 連携機能の特徴を損なわずに Reduce-Skew を軽減する割合指定 Partitioner を提案する．割合指定 Partitioner は Reduce タスクの入力データ量の平準化と，タスクスケジューリングの最適化により MapReduce ジョブの実行時間を短縮する．実業務で使われるデータを再現して評価した結果、処理完了までの時間を最大で 30.6%短縮した．

キーワード：MapReduce,Hadoop, 並列処理

1. はじめに

既存の業務アプリケーションを Hadoop[5] 上で実行しバッチ処理を高速化するための製品として富士通は NetCOBOL[7] を提供している．元々 NetCOBOL は単一マシンを対象とした COBOL アプリ開発実行基盤である．この NetCOBOL を Hadoop 上で並列実行できるようにする主要技術は NetCOBOL Hadoop 連携機能 [6] である．NetCOBOL Hadoop 連携機能の特徴は，Hadoop 上で実行することを想定せずに作られた業務アプリケーションを修正することなく MapReduce アプリケーションとして実行可能にすることである．ユーザは MapReduce アプリケーションを新規開発する必要がないため低コストでバッチ処理の高速化を実現できる．アプリケーションを修正することなく Hadoop 上で実行する同様の技術に Hadoop Streaming[1] がある．NetCOBOL Hadoop 連携機能は Hadoop Streaming の制約である入出力が標準入出力であること，および入出力が単一である制約を改善した技術である．NetCOBOL Hadoop 連携機能は複数のファイルを入出力としたアプリケーションに適用可能なため，突合せ処理を目的とした COBOL の業務アプリケーションに適用されることが多い．

しかし，一部の業務要件において NetCOBOL Hadoop 連携機能を用いて業務アプリを実行すると，Reduce タスクの実行時間にばらつきが生じるため Hadoop の並列効果が低下することが報告された．

並列効果が低下した原因は Reduce-Skew[2] である．Reduce タスクへデータを振り分ける際に，キーの出現数の偏りが原因で Reduce タスクが処理するデータ量にばらつきが生じる．その結果，Reduce タスクの実行時間がばらつき，MapReduce ジョブの実行時間が長くなる．Reduce-Skew は NetCOBOL Hadoop 連携機能独自の問題ではなく，MapReduce アプリケーションで潜在的に発生する問題である．NetCOBOL Hadoop 連携機能は業務アプリのバッチ処理高速化を目的としている．業務アプリでは取り扱う主キーの種類数が比較的少ないためキーの出現数が偏ることが多く Reduce-Skew が起こりやすい．

そこで，本稿では MapReduce アプリケーションにおける Reduce-Skew を軽減し並列効果を向上することで，MapReduce ジョブの実行時間を短縮する割合指定 Partitioner を提案する．割合指定 Partitioner は NetCOBOL Hadoop 連携機能の特徴である既存の業務データのバッチ処理を低コストで高速化するという特徴を損なうことなく Reduce-Skew を軽減する．さらに，ユーザはジョブ実行の際に提案手法を有効化するだけで済むように設計して NetCOBOL Hadoop 連携機能に組み込んだ．これにより，

[1] 富士通研究所
FUJITSU LABORATORIES LTD
[a)] n.kuromatsu@jp.fujitsu.com
[b)] ymatsu@jp.fujitsu.com
[c)] hal_ueda@jp.fujitsu.com

ユーザは NetCOBOL Hadoop 連携機能を用いた MapReduce ジョブの処理速度向上に対して追加コストを支払う必要がない上，ユーザビリティが損なわれることもない．

評価では実際に業務で使われている複数のデータを再現して割合指定 Partitioner の性能を測定した．その結果，Reduce-Skew が改善され処理完了までの時間を最大で 30.6%短縮した．入力データ量が等しくても，データの内容によって処理時間が変化するアプリケーションがあるが本稿ではそのようなアプリケーションは対象外とする．

2. Hadoop の仕組みと Reduce-Skew

Hadoop 上で MapReduce ジョブを実行すると，複数の Map タスクと Reduce タスクが実行される．Map タスクは分割された入力データを処理し，Reduce タスクは Map タスクの出力結果をキー毎に集め集約処理する．

Reduce-Skew の原因は Reduce タスクの入力データ量の偏りである．入力データ量が偏ると Reduce タスクの実行時間にも偏りが発生する．未割当の Reduce タスクがなくなり，各サーバが新規に実行する Reduce タスクがなくなった際に，一部のサーバで実行時間が長い Reduce タスクの実行が続くと MapReduce ジョブの実行も終了しない．その間は Reduce タスク実行を終えたサーバの計算リソースは活用されないため，並列効果が低下し MapReduce ジョブの実行時間が長くなる．

Reduce タスクの入力データ量に偏りが生じる原因は Hadoop において Reduce タスクがどのキーを処理するか決定する Partitioner のアルゴリズムと入力データに含まれるキーの出現数の偏りにある．Hadoop 上で MapReduce ジョブを実行した際の処理の流れを図 1 に示す．入力データを受け取った Map タスクは処理結果を得ると，処理結果に含まれる各キーをどの Reduce タスクへ送信するべきか Partitioner に対して問い合わせる．Partitioner はパーティション値と呼ばれる識別子をキー毎に定めることで振り分け先の Reduce タスクを決定する．パーティション値と Reduce タスク ID は一対一で対応しているため，パーティション値が分かれば振り分け先の Reduce タスクを知ることができる．Map タスクの処理結果を Reduce タスクへ転送するために Shuffle 処理が実行され，Shuffle 処理が Reduce タスクへ必要なデータを全て転送すると Reduce 処理が開始する．Shuffle 処理は Reduce タスクの処理の一部として扱われる．

Hadoop が元々備える Partitioner（標準 Partitioner）ではパーティション値を決定するアルゴリズムとしてキーのハッシュ値を採用している．入力データに含まれるキーの偏りが大きい場合，キーの値によっては出現数が多い複数のキーに対して同一のパーティション値が得られることがある．このパーティション値に対応する Reduce タスクの入力データ量は他の Reduce タスクと比べて多いため，

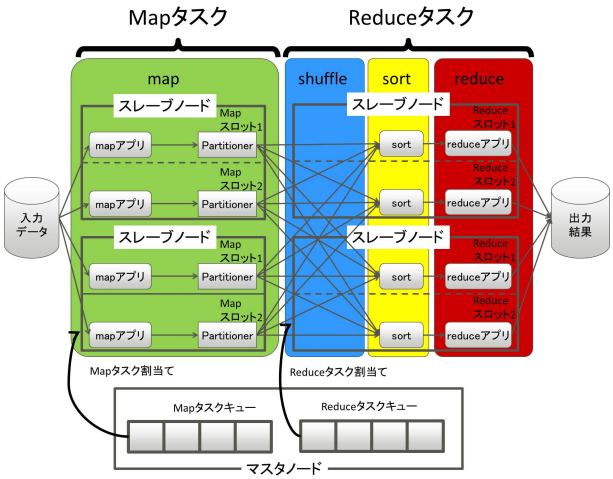

図 1 Hadoop における MapReduce 処理の流れ

処理時間が長くなる．この Reduce タスクが MapReduce ジョブ実行の後半に割り当てられると，他のサーバで処理が終了したにも関わらずこの Reduce タスクの実行が終わらないため MapReduce ジョブが終了しない．その結果，Reduce-Skew が発生し並列効果が低下する．

3. 関連研究

Reduce-Skew を軽減する既存研究は、Reduce タスクの入力データが大きな場合に，入力データを細分化することで特定の Reduce タスクの処理時間が突出して大きくなることを防ぐ手段と，パーティション値の最適化による方式がある．

Smirti ら [3] はジョブの実行中にあるキーが他のキーと比べて出現数が多いことが分かると，このキーをサブキーに細分化して複数の Reduce タスクで分散処理する Chopping により Reduce-Skew を解消した．ただし，Chopping を適用できるのは，同一のキーが異なる Reduce タスクで分散して処理されることが許される場合に限る．適用可能なアプリケーションが限定されるため，この手法では NetCOBOL Hadoop 連携機能の機能性が低下する問題がある．

一方で，Rares ら [4] は MapReduce ジョブの実行開始から Map タスクの処理が始まるまでの間に入力データの一部をサンプリングすることで入力データに含まれるキーの出現数を傾向を調べ，各 Reduce タスクの入力データの偏りを平準化するようにパーティション値を最適化する手法を提案した．しかし，この手法は入力データの一部をサンプリングして出現数を抽出するため，最適な平準化を実現することができない．

しかし，いずれも MapReduce の実行時間のばらつきを考慮したキーの割当て方式は提案されていない。

4. 提案手法

割合指定 Partitioner は MapReduce ジョブの実行時に

発生するタスク毎の実行時間にばらつきが発生したとしても，処理を平準化するための手法である．各キーの振分け先を決定する際に，通常の Reduce タスク（以降は main-Reduce タスクと呼ぶ）の他に，入力データ量が小さい remain-Reduce タスクを用意し，remain-Reduce タスクを実行の最後に割り当てるようにスケジューリングする．これにより，main-Reduce タスク実行時に実行時間にばらつきが発生し，本来の想定よりもばらつきが大きくなったとしても，入力データ量が小さいため処理時間が短い remain-Reduce タスクが他の Reduce スロットに割り当てることで処理の平準化を図る．remain-Reduce へ振り分けるキーの決定方法は出現数の合計が一定の割合（$rate$）以下になるように出現数の少ない主キーから選ぶ．

従って，割当て方は主キーの偏り，main-Reduce タスクの数，remain-Reduce タスクの数，remain-Reduce タスクへ割り当てる割合によって変化する．

4.1 その他の方式

MapReduce ジョブ実行時間を短縮するための三つの最適化機能を NetCOBOL Hadoop 連携機能に追加した．三つの最適化を 4.1.1 節から 4.1.3 節で説明する．なお，4.1.1 節および 4.1.2 節の最適化にはキーとその出現数の情報が実行時に必要である．この情報は mainkeyList と呼ばれるファイルに記述し，ジョブ実行前に各サーバに配布する．mainkeyList の作成方法を 4.2 節で述べる．

4.1.1 Reduce タスクの入力データ量の最適化

一つ目は Reduce タスクが処理する入力データ量の最適化である．Reduce タスクは入力データ量の差によって実行時間にばらつきが生じる．この実行時間のばらつきが Reduce-Skew を生んでいた．そこで, ユーザが MapReduce ジョブを投入してから実際に実行が開始されるまでの間に，各 Reduce タスクに対して入力データ量が平準化されるようにデータの振分け先を決定する．データの振分け先の決定には BinPacking アルゴリズムを用いた．

ただし，データの振分けはキー単位でなければならないため，入力データで一部のキーの出現数が極端に多い場合は理想的な平準化を実現できずに入力データ量のばらつきが残ってしまう．

4.1.2 Reduce タスクのスケジューリングの最適化

二つ目は Reduce タスクのスケジューリングの最適化である．Hadoop には Map タスクのバックグラウンドで実行される Reduce タスクの Shuffle 処理が Map タスクによって隠蔽される特徴がある．割合指定 Partitioner は隠蔽される Shuffle 処理時間を最大化することで，実行時間の短縮を実現した．Shuffle 処理が Map タスクにより隠ぺいされる仕組みを図 2 に示す．Map タスクのバックグラウンドで実行される Reduce タスクとは，各 Reduce タスクに最初に割り当てられる Reduce タスクである．従って，

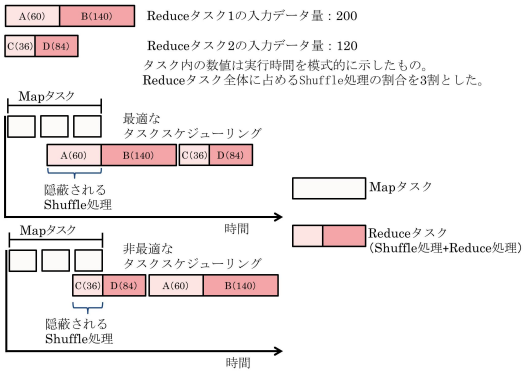

図 2 Shuffle 処理が隠ぺいされる仕組み

Shuffle 処理時間の隠蔽が可能な Reduce タスクの数は多重度の数に等しい．Shuffle 処理は転送するデータ量が多い程長いため，入力データ量が多い Reduce タスク程この効果が高い．そこで，割合指定 Partitioner はデータ量が多い Reduce タスクを優先して割り当てることで隠蔽可能な Shuffle 処理時間を最大化した．図 2 において，最適なタスクスケジューリングを施した場合，Reduce タスクの処理時間は $B + C + D = 260$ であるが，最適なタスクスケジューリングではない場合は $D + A + B = 284$ の時間がかかる．これを実現するためには，タスクスケジューリングを最適化する仕組みが必要である．Hadoop の標準のスケジューラでは，Reduce タスク ID が小さい順に割り当てられる．Reduce タスク ID にはパーティション値が使われる．そこで，入力データ量が多い Reduce タスクに小さなパーティション値を与えることで最適なタスクスケジューリングを実現し隠蔽可能な Shuffle 処理時間を最大化した．これにより，一つ目の最適化でも平準化できずに Reduce タスクの入力データ量にばらつきが生じる場合でも，処理時間の差を抑え並列効果を向上することができる．

4.1.3 Reduce タスク数の最適化

三つ目は Reduce タスク数の最適化である．業務アプリを実行する度に現在の多重度を調べ，最適な Reduce タスク数を自動的に設定する．

Reduce タスク数は MapReduce ジョブ実行前にユーザが設定する静的な値である．一般に Reduce タスク数は実行環境における多重度の倍数が適していると言われている．しかし，Hadoop の実行環境は定常的に業務アプリを実行する複数のサーバで構成されているため，サーバが故障して多重度が低下したり，性能を向上させるためサーバを追加することで多重度が増加することがある．多重度が変化すると，ユーザは業務アプリを実行する前に MapReduce ジョブの設定ファイルを書き換える必要がある．

割合指定 Partitioner は Reduce タスク数をジョブ実行直前にユーザに透過的に設定するため，サーバの故障または追加による台数の変化があったとしてもユーザが設定ファイルを変更しなくても常に最適な Reduce タスク数で

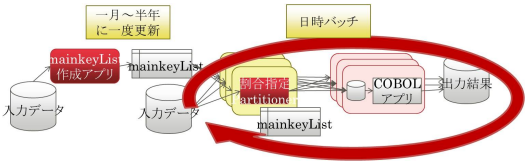

図 3 割合指定 Partitioner のユースケース

MapReduce ジョブが実行される．ただし，本報告時点では Reduce タスク数の最適化は機能が未実装であるため，本稿では手法の提案のみに留める．

4.2 入力データ中のキーの出現数の確認方法

割合指定 Partitioner を使ったジョブ実行の前に，入力データ中のキーとキーの出現数が記述された mainkeyList を作成する必要がある．割合指定 Partitioner は mainkeyList の情報から 4.1.1 節と 4.1.2 節の最適化を行う．

mainkeyList の作成は専用の MapReduce ジョブによる並列処理で行う．NetCOBOL Hadoop 連携機能は作成した mainkeyList を MapReduce ジョブ実行時に Hadoop のサイドデータとして各サーバへ配布し，割合指定 Partitioner は mainkeyList に含まれる主キーとその出現数を見て，主キーのパーティション値を最適化する．

mainkeyList の作成は入力データを全て確認するため，mainkeyList の作成は時間を要する．しかし，業務アプリでは入力データの傾向が変化することは少なく，本機能は日時バッチの高速化を対象として想定しているため mainkeyList の作成は一月から半年に一度で十分である．そのため，全体の実行時間から見れば mainkeyList 作成にかかる時間は少ない．想定するユースケースを図 3 に示す．

我々はユーザがこれら一連の処理を意識しなくとも割合指定 Partitioner を使えるように NetCOBOL Hadoop 連携機能の中に組み込んで実験を行った．ユーザは設定ファイルの項目を書き換えるだけで mainkeyList 作成アプリの実行および MapReduce ジョブ実行完了までの時間を短縮できる．

5. 評価実験

標準 Partitioner を使った場合の MapReduce ジョブ実行時間に対して，割合指定 Partitioner を使うことでどの程度処理完了までの時間を短縮できるかを評価する．4.1.3 節の最適化は未実装のため，評価実験では 4.1.1 節と 4.1.2 節の最適化のみを行った．4.1.1 節の最適化は mainkeyList に含まれているキーのみ対応するが，今回は入力データのうち 50%の割合を最適化して性能を測定した．

割合指定 Partitioner は入力データに含まれるキーの出現数の偏りによって性能が変化する．そこで，実際の業務で使われるデータを再現して割合指定 Partitioner の性能を評価した．再現したデータは NetCOBOL で Reduce-Skew

表 1 実験サーバのスペック

CPU	3.3GHz 8core	HDD	SATA 600GB × 3
RAM	94GB	ネットワーク	10GbE

表 2 実験結果

データセット	標準 Partitioner	割合指定 Partitioner	短縮率
実事例 1	848.6 秒	589.2 秒	30.6%
実事例 2	66.4 秒	70.2 秒	-5.7%
実事例 3	281.2 秒	233.4 秒	17.0%
経理業務 1	256.9 秒	222.3 秒	13.5%
経理業務 2	99.2 秒	77.1 秒	22.3%

が発生していた実事例データおよび富士通研究所の経理業務で使われる業務用データである．実験にはトランザクションファイルとマスタファイルの突合せを行う業務アプリを用いた．実験に使ったサーバのスペックを表 1 に示す．実験ではマスタサーバを一台とスレーブサーバを 5 台用意した．一台のスレーブサーバが実行する Reduce タスク数を 3 にしたため，Reduce タスクの多重度は $3 * 5 = 15$ である．実験結果を表 2 に示す．

実験結果から実事例 1 において最大で 30.6%実行完了までの時間を短縮していることが分かった．一方で，実事例 2 においては標準 Partitioner と比べて実行時間が長くなっている．この原因は mainkeyList に記述されたキーの数が多いため，キー毎のパーティション値を決定する際に探索に時間がかかったことで Map タスクの実行時間が増加したためである．他のデータセットでも Map タスクの実行時間の増加はあったものの，Reduce タスクの入力データ量平準化による実行時間短縮効果の方が高かったため MapReduce ジョブ実行完了までの時間が短くなっていた．一方で，実事例 2 では MapReduce ジョブの実行時間が 70 秒程度と短かったため，平準化による短縮効果（約 7 秒）よりも Map タスクの実行時間の増加（約 10 秒）の方が大きかった．Map タスクの実行時間の増加は実装方法の不備が原因であることが分かっているため今後改善する．

6. まとめ

NetCOBOL Hadoop 連携機能を用いて業務アプリを実行した際に生じる Reducec-Skew を軽減する手段として，割合指定 Partitioner を開発した．割合指定 Partitioner は Reduce タスク数，Reduce タスクの入力データ量，Reduce タスクのスケジューリングの三つの最適化を施すことで，MapReduce ジョブ実行完了までの時間を削減する．そのうち，二つの最適化を施し実際の業務で使われるデータを再現して割合指定 Partitioner の性能を評価した結果，MapReduce ジョブ実行完了までの時間を最大で 30.6%短縮することを確認した．残る一つの最適化は今後の課題とする．

参考文献

[1] Apache: Hadoop Streaming.
[2] Kwon, Y., Balazinska, M., Howe, B. and Rolia, J.: SkewTune: Mitigating Skew in Mapreduce Applications, *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, New York, NY, USA, ACM, pp. 25–36 (online), DOI: 10.1145/2213836.2213840 (2012).
[3] Ramakrishnan, S. R., Swart, G. and Urmanov, A.: Balancing Reducer Skew in MapReduce Workloads Using Progressive Sampling, *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, New York, NY, USA, ACM, pp. 16:1–16:14 (online), DOI: 10.1145/2391229.2391245 (2012).
[4] Vernica, R., Balmin, A., Beyer, K. S. and Ercegovac, V.: Adaptive MapReduce Using Situation-aware Mappers, *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, New York, NY, USA, ACM, pp. 420–431 (online), DOI: 10.1145/2247596.2247646 (2012).
[5] White, T.: Hadoop 第 3 版，O'REILLY (2013).
[6] 上田晴康，榎田　勉，立岩恭也：NetCOBOL の Hadoop 連携機能の開発と実践事例，デジタルプラクティス，情報処理学会 (2014).
[7] 富士通：NetCOBOL：富士通ミドルウェア.