1K-1

MapReduce 環境におけるアドホックなクエリを対象とした, Adaptive indexing 適用モデルの提案

奥寺　昇平†　　横山　大作†　　中野　美由紀†　　喜連川　優†

東京大学生産技術研究所†

1. はじめに

近年、我々が生み出すデジタルデータの急増に伴い、ウェブサービスをはじめとする企業等が管理するデータが大規模化している。大規模データを扱う組織では、アドホックな解析を適宜行い、得られた知見を組織が提供するサービスに還元している。よりよいサービスを提供するためには、アドホック解析を中心とするデータ解析サイクルの周期を早くすることが求められる。MapReduce[1]は大規模データにおけるアドホックな解析を行うデータ解析基盤としての重要性を増している。しかし、アドホックな解析は、大規模なデータスキャンを中心としており、処理コストは高い。

MapReduce 環境における大規模データ解析の高速化に関して、著者らは[2]においてクエリに対して適用的にインデキシングを行う手法を提案した。この手法では、同じカラムに対して異なる選択条件が与えられた場合、問い合わせが繰り返されるにつれ処理を高速化する。[2]では，射影処理に関して、提案手法による実行時間の削減効果を報告した。

本論文では、前述の適用的なインデキシング手法を、基本的なデータ処理である集合演算処理と結合演算処理に適用し、実行時間の削減効果を検証した。シミュレーションによる評価を行い、少しずつ選択条件が変化する問い合わせを繰り返し発行し、累積の実行時間を比較した。シミュレーション結果から、適用的なインデックスを用いない場合と比較し、高い性能が得られることを示した。

2．適応的インデキシングを適用した MapReduce 処理

著者らの[2]による手法を説明する前に、MapReduce 環境におけるレンジ条件付きの集合演算処理、結合演算処理の実行フローを説明する。実行する演算処理は下記に示す。

・集合演算:

SELECT partID, AVE(discount) FROM R
WHERE x ≤ discount < y GROUP BY partID

・結合演算:

SELECT S.partID, S.partName, R.discount FROM R, S
WHERE x ≤ R.discount < y AND R.partID = S.partID

Modeling on Adhoc query processing with Adaptive indexing in MapReduce Environment.
Shohei OKUDERA†　Daisaku YOKOYAMA†　Miyuki NAKANO†　Masaru KITSUREGAWA†
† Institute of Industrial Science, University of Tokyo

2.1 MapReduce 処理における集約演算・結合演算

MapReduce 処理は Map フェーズと Reduce フェーズの 2 段階でデータ処理を行い、通常分散ファイルシステム(DFS)上でデータの入出力を行う。Map フェーズではデータを読み込み、選択条件によりレコードを選択し、Reduce フェーズでは Map フェーズが出力したレコードを集計し、集合演算、結合演算を実行した後、データを出力する。各フェーズでは、タスクを複数起動しデータ並列で分散処理を行う。図 1 の上段は、Map タスクの詳細な実行フローを表す。Map タスクは、 データスプリットと呼ばれる分割データ単位を *Scan* し、レコードに変換する。次に、*Map* においてレンジ条件判定を行う。最後に、該当したレコード群を *Collect*、 *Spill*、 *Merge* といったステップを通し、Reduce タスクに分配する．提案手法は Map タスク内で実装されるため、Reduce タスクの実行フローの詳細は従来通りであることから、省略する。

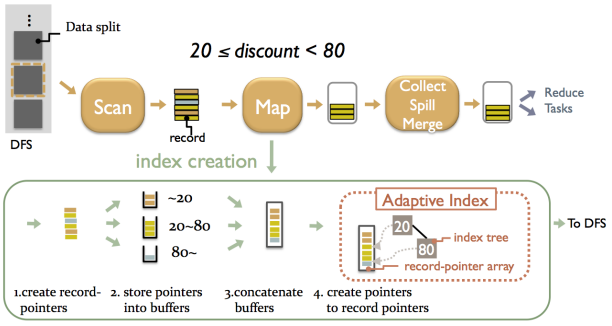

図 1： Map タスクにおける適応的インデックスの作成方法(上段のオレンジの矢印は従来の処理)

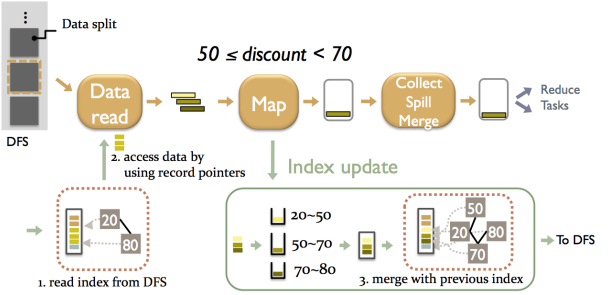

図 2： Map タスクにおける適応的インデックスの更新方法

2.2 適応的インデキシング

本提案手法[2]では、Map タスクにて、データスプリットごとに、クエリの条件節を利用して適応的にインデックスを構築していく。適応的なインデキシング手法は、[4]を基に MapReduce 環境に適用した。インデキシングす

るカラムがソートされていない場合のインデックスの作成と更新方法を説明する。図１に示されるように、インデックスの構成は、全レコードへのポインタ配列と、レコードポインタへのポインタを保持するインデックスツリーの２つである。

適応的インデックスが存在しない時、クエリ実行時にインデックスの作成を行う（図１）。*Map* においてレンジ条件 $20 \leq discount < 80$ の判定に加え、条件を満足しないレコードを含めたすべてのレコードに対してポインタを作成する。作成したポインタをクエリレンジに応じて異なるバッファに格納し、処理終了後にバッファを順番に連結させることでレコードポインタ配列を作成する。次に、レンジの境界となるレコードポインタ *20, 80* を指すポインタを保持するインデックスツリーを作成する。最後にインデックスを分散ファイルシステムに書き込む。

一度適応的なインデックスを作成した後のクエリでは、インデックスを利用したデータアクセスとインデックスの更新を行う（図２）。次のクエリのレンジ条件が $50 \leq discount < 70$ で与えられたとする。データの取得はインデックスを利用して $20 \leq discount < 80$ を満たすレコードを取得する。*Map* では、選択条件の判定を行うと同時に、レコードポインタ配列をクエリレンジに応じてレンジ分割し、レンジの境界となるレコードポインタ *50, 70* を指すポインタをインデックスツリーに追加する。最後に、インデックスを分散ファイルシステムに書き込む。

3. シミュレーション環境

[3]の実行時間に関するコストモデルを基に、シミュレーションを用いた評価を行った。[3]のコストモデルでは、各タスクでのステップ（例. *Scan*、 *Map*）の実行時間をI/O 時間、CPU 時間の単純和で表現し、ステップの実行コストを集計することで MapReduce のフェーズ、ジョブの実行コストを求めている。今回、この実行コストにインデキシングに関するコストを加えた。各ステップにおける I/O、CPU コストは、実環境で計測を行い求めた。計測したサーバーのスペックは、Intel Xeon E5530 2.40GHz 2P/8C、DDR3 24GB, HDD×1 である。使用した HDD の連続読込性能、連続書込性能は、共に 115MBps であった。シミュレーションでは、10GBps のネットワーク環境で接続された 10 台の前述のサーバーを計算ノードとして利用し、ノードあたり 4 タスクが同時実行可能とした。また、１タスクは 1 コアで実行することを想定している。

4. 集合演算、結合演算における適応的インデキシングの評価

集合演算、結合演算における提案手法の評価を行った。

4.1 セットアップ

データセットは17個のカラム，平均レコード長170バイトのリレーションR（1TB）と、10個のカラム、平均レコード長168バイトのリレーションS（100GB）を用いた。問い合わせは第2章で示した集合演算と結合演算である。クエリレンジを決定する変数 x, y は、初期の選択率が0.01%で、レンジが直前のレンジに含まれるように80%ずつ縮小していくように変化させた。提案手法では、ソートされていないカラム *discount* に対してインデックスを構築する。Mapタスクに与えられるデータスプリットサイズは512MB, Reduceタスク数は1とした。

4.2 シミュレーション結果

図３は、発行したクエリ数と集合演算処理に要するジョブ累積実行時間の推移を表したグラフである。累積実行時間とは、N 回目までのジョブ実行時間の累積和である。従来の MapReduce 処理(original)は、クエリ数に比例して累積実行時間が増加する。初回のクエリにかかる時間が 690 秒、20 個のクエリを発行する時間が 13,500 秒超であった。一方、提案手法(Adaptive indexing)では累積実行時間が微増で収まっている。 初回のクエリでは 720 秒であったが、20 個のクエリの処理では 2,200 秒であった。以上の結果から、提案手法の初回のクエリの実行時間はインデックス作成により従来の処理に比べ 30 秒増加したものの、２回目以降の累積実行時間が大きく減少することを確認した。

図４は、発行したクエリ数と結合演算処理に要するジョブ累積実行時間の推移を表したグラフである。従来のMapReduce処理は、クエリ数に比例して累積実行コストが増加し20個のクエリを処理するために15,500秒かかったが、提案手法の累積実行コストはゆるやかに増加し、20個のクエリを実行する時には3,900秒である。この結果から、適応的インデキシングにより、累積実行コストを大幅に削減できることがわかった。

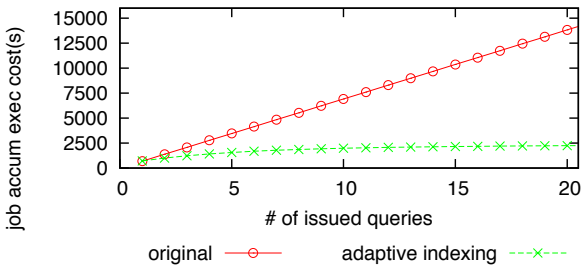

図 ３:集合演算処理における累積実行時間の推移

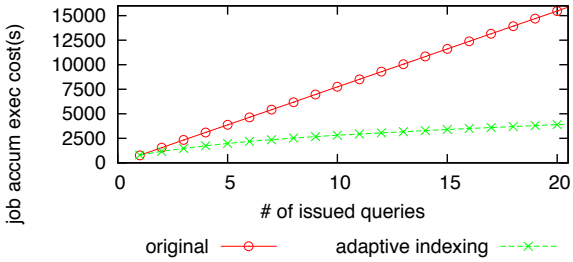

図 ４:結合演算処理における累積実行時間の推移

5. まとめ

本稿では、 MapReduce 環境でのアドホック解析処理に適応的インデキシングの適用を行い、集合演算／結合演算処理に要する累積実行コストをシミュレーションで評価した. その結果から、適応的インデックスを用いることで、従来の処理に比べて累積実行時間を大きく削減できることを確認した。

References

[1] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, 2008.
[2] 奥寺昇平,横山大作,中野美由紀,喜連川優: “MapReduce環境におけるアドホックなクエリを対象とした, Adaptive indexing適用に関する一検討”, 電子情報通信学会データ工学研究会,電子情報通信学会技術報告, Vol. 112, No. 346, DE2012-38, pp. 131-136 (2012.12).
[3] H. Herodotou. Hadoop Performance Models. Technical report, Duke Univ., 2010. http://www.cs.duke.edu/starfish/files/hadoop-models.pdf.
[4] S.Idreos, M.Kersten and S.Manegold: Database Cracking. CIDR 2007