

Hadoop環境における空間分割による並列全 k 最近傍問合せ処理

Processing All k -Nearest Neighbor Queries in Hadoop

横山 拓也[♡] 石川 佳治[◇] 鈴木 優[▲]

Takuya YOKOYAMA Yoshiharu ISHIKAWA
Yu SUZUKI

指定された点に対して最も近い k 個の点を求める k 最近傍問合せは、空間データベースでは基本的な問合せの1つである。これに関連して、データ集合中の各点について、それぞれの k 最近傍を一度に求める問合せを全 k 最近傍問合せという。本研究では、この全 k 最近傍問合せをMapReduceフレームワーク上で行う手法を提案する。空間をセルに分割し、全 k 最近傍問合せの処理をMapReduceの並列分散処理方式に合った形で実行する。分割により生じる問題にMapReduceフレームワークに適した形で対応するための、対象データの分布情報を考慮した改善策についても提案を行う。

A k -nearest neighbor (k NN) query, which retrieves nearest k points from a database is one of the fundamental query types in spatial databases. An *all k -nearest neighbor query* (Ak NN query), a variation of a k NN query, determines the k -nearest neighbors for each point in the dataset in a query process. In this paper, we propose a method for processing Ak NN queries in Hadoop. We decompose the given space into cells and execute a query using the MapReduce framework in a distributed and parallel manner. Using the distribution statistics of the target data points, our method can process given queries efficiently.

1. はじめに

近年、Web上の様々なデータについてジオタグなどにより位置情報（座標情報）が付加されるようになってきている。そして、この位置情報を利用することでユーザの現在位置や気になる場所に他のデータを結びつけるような、新しいサービスが提供され始めている。そのような位置情報を利用したサービスの例として、ソーシャルネットワークサービスにおいてユーザごとに近くにいる別のユーザ5人を求めて定期的に提示するといったものを考える。この例において必要になる、ある座標に最も近い k 件のデータを取り出すという処理は k 最近傍問合せ（ k Nearest Neighbor query; k NN query）と呼ばれている。さらにこの k 最近傍問合せのバリエーションとして、あるデータ集合のすべてのデータについて k 最近傍問合せを行う全 k 最近傍問合せ（All k Nearest Neighbor query; Ak NN query）があり、効率的な問合せ処理方

式が提案されている[4, 6, 12]。ソーシャルネットワークサービスの例でも、個々のユーザごとに別々に k 最近傍を求めるのではなく、バッチ処理的に全 k 最近傍問合せを行うことで全ユーザの k 最近傍を一度に求めることが望まれる。

一方で、近年爆発的に量を増す情報量に対応するために、高度な並列分散処理に基づくクラウドコンピューティング技術が注目され、開発が進んでいる。ユーザの全 k 最近傍問合せを行う例においても、サービスの運営により日々量を増す大規模なデータがクラウドの分散環境上に置かれることは十分に考えられる。そのため、全 k 最近傍問合せ処理は並列分散処理に親和性の高い手法で効率的に行うことが求められる。並列分散コンピューティング技術としては、Googleが自社の大量データを扱うために設計した手法であるMapReduceが有名であり[8]、その考え方を基に開発されたオープンソースとしてApache Hadoop[2]がある。Hadoopは多くの企業において大規模データ処理に用いられた実績がある。

本研究では、Hadoopを使用したMapReduceによる分散処理で効率的に全 k 最近傍問合せに答える手法を提案する。分散処理の実装としてはデータが分布する空間をより細かなセルに分割するという考えを基本としている。分割することで k 最近傍候補データの見落としが生じてしまうが、各データについて得られた k 最近傍情報から追加調査が必要となる範囲を表す最大距離の円を求めることで対処する。また、最大距離の円を確実に求めることができるように、対象データの分布情報を考慮してセルを統合する処理も加える。

2. 関連研究

2.1 MapReduce

本研究では、Apacheによって開発された並列分散コンピューティングフレームワークHadoop[2, 10]の使用を想定している。ここでは、そのHadoopとHadoopにおいて分散処理の基礎となっているMapReduce[5]について説明する。

Hadoopは分散処理対象のファイルが保存される分散ファイルシステムと、分散処理を行うフレームワークMapReduceで構成されている。MapReduceでの分散処理は、入力となるデータを分割して複数のタスク上で同時に処理することで行われる。タスクはMapタスク、Reduceタスクの2種類があり、それぞれでユーザが実装したMap関数とReduce関数が入力データに対して適用される。これらの関数へのデータの出入力はキーバリューペアという形に合わせる必要があり、実装時にはそれに注意してデータ処理を記述する。そのかわり、分散処理をする際に大きな問題となる計算ノード間のデータ転送や障害対策についてはシステム側が面倒を見てくれる。例えば何らかの原因で処理が遅れているタスクがあった時には、そのタスクを別のノードで実行し直したりしてくれる。よって、関数実装時には分散処理ならではの問題について気にする必要はなく、データに対する処理内容だけに集中してプログラミングを行うことができる。

次に、各タスクで実行される関数を詳しく解説する。Mapタスクで実行されるMap関数ではデータの抽出を目的としており、分割されたファイルのデータに対して必要に応じて何らかの加工をした上で、<キー, バリュー>というペアの形でデータを出力する。Mapタスクの出力となったキーバリューペアはReduceタスクに渡される前にShuffle処理にてキーによってグループ化され、グループごとにまとめてReduceタスクの1つに転送される。Reduceタスクで実行されるReduce関数ではデータの集約を目的としており、キーでグループ化されたバリューのリストから何らかの集計を行い、その結果を分散ファイルシステムに書き出す。MapタスクとReduceタスクの処理においては他のタスクとの通信が必要ないため、並列にデータ処理をすることができる。

全 k 最近傍問合せは一種の結合処理であると考えられる。MapReduceにおける結合処理は複雑かつ大規模な分析処理のために重要であり、その効率化のための研究が進められている[1, 7]。

[♡] 正会員 名古屋大学大学院情報科学研究科。
yokoyama@db.itc.nagoya-u.ac.jp

[◇] 正会員 名古屋大学情報連携基盤センター／国立情報学研究所
ishikawa@itc.nagoya-u.ac.jp

[▲] 正会員 名古屋大学情報連携基盤センター
suzuki@db.itc.nagoya-u.ac.jp

しかし MapReduce の結合処理は、一般に等結合しか利用できないという制限があり、等結合以外の条件での結合は直接的には行えない。しかし、結合演算はデータ処理において非常に重要であるため、MapReduce 環境で等結合以外の結合をするための研究も進められている [9]。

2.2 全 k 最近傍問合せ

k 最近傍問合せとは、ある座標に対して距離的に最も近いデータ k 個を取得する問合せである。一方、全 k 最近傍問合せとは、データ集合中の各点データに対する k 最近傍を 1 回の処理ですべて求める問合せである。1 節では、ソーシャルネットワークサービスにおいて各ユーザの近くにいたユーザの情報を求めるという全 k 最近傍問合せの例を挙げた。全 k 最近傍問合せはこれ以外にも、データマイニングに先立つ前処理を一括して実現する場合などにも用いられる [3]。

全 k 最近傍問合せについては、すでに R-木や空間充填曲線を利用した効率的なデータアクセスに基づく多くのアプローチが提案されている。しかし、これらは集中化された従来のコンピューティング環境を前提としたものであり、分散処理によって大規模データを処理する MapReduce においては適用できない。そこで本論文では、空間分割を基礎とした MapReduce フレームワークの分散処理に適した全 k 最近傍問合せの処理手法を提案する。

3. 空間分割による分散処理

まず、ここで行う全 k 最近傍問合せの詳細について述べる。問合せの対象となるデータは x, y 座標で表される 2 次元平面上の点データである。また、本研究では単一のデータ集合の中で k 最近傍を求めることを考える。これはデータベース用語における自己結合 (self-join) に相当する。なお、2 つのデータ集合間で k 最近傍を求めるなどの拡張については、[11] で簡単に議論している。

分散処理により上記の全 k 最近傍問合せを行うために、対象となるデータが分布する空間を格子状に区切った $2^n \times 2^n$ 個のより小さな空間 (セル) に分割する。 n は分割の粒度を決める値であり、この値が大きいほどより細かな分割になる。

あるデータの k 最近傍となるデータは空間的に近い場所に集中するので、分割した各セルの中のデータの k 最近傍の多くはそのセルの中で見つかることが期待できる。よって、まずはデータ集合の各データがどのセルに入るかを調べてデータをセルごとにまとめ、その後に各セルに含まれるデータで k 最近傍を求める。セルそれぞれを別々に調べることができるため、全 k 最近傍問合せを MapReduce の複数のタスクを同時実行するという分散処理の形に合わせて行えるようになる。

ただし、空間分割の影響としてこの最初の調査で k 最近傍が確定できないケースが生じるので、それらに対処するための追加調査が必要になる。あるセルにおいて k 最近傍が求まったとしても、そのセルの境界の向こう側、つまり別のタスクで処理されているセルの中のデータを見落としている可能性が残っているからである。例えば図 1 のデータを例にすると、点 B がこのケースに該当する。点 B はセル 0 の領域にあるため、最初の k 最近傍の調査では同じくセル 0 の領域内の点を調べて k 最近傍が求められる。しかし実際には、セル 2 に k 最近傍としてよりふさわしいデータが存在しているため、それを見つかる追加調査が必要になる。

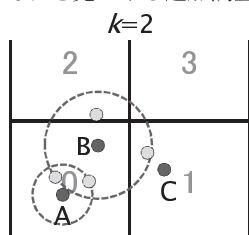


図 1: 追加調査範囲の判定

このように、最初の調査の後に各データについて、近隣セルにより適切な k 最近傍が存在する可能性があるかどうか、また、可能性があるならどのセルまで調査範囲を広げるかを判定する必要がある。この判定は各データの k 最近傍情報から求められる最大距離の円と他のセルとの重なりを調べることで行うことができる。

最大距離の円とは追加調査の範囲を表すもので、注目データを中心にした半径 r の円である。 r の値はそのデータがセルの中で見つけた k 番目に近いデータまでの距離である。この円の中に k 最近傍の候補となるデータがあれば、それはより近いデータとして k 最近傍に入ることになるため、もし円が他のセルに重なっていればそのセル領域のデータを追加調査する必要がある。図 1 では、点 A と点 B についての最大距離の円が描かれている。点 A については、円は他のセルには重なっておらず、これ以上近いデータが見つかることはないため、追加調査の必要はないと判断することができる。点 B については、円がセル 1, 2, 3 に重なっているため、それらのセルの中により近い k 最近傍データがあるかどうかを調査する必要がある。この追加調査の結果をまとめることで、点 B の k 最近傍を確定することができる。

しかしながらこの追加調査の判定は、そもそも最大距離の円が描けることを前提としているため、それが不可能な状況ではこの方法による調査範囲の判定はできない。その状況とは、分割したセルの中に k 最近傍の候補データが k 個未満しかなかったときである。最大距離の円は k 番目の k 最近傍までの距離を利用して描かれるため、そもそもデータが k 個存在しない状況では描くことができない。図 1 では点 C がこの状況に該当する。 $k=2$ という条件に対してセル 1 の中には 1 つしか k 最近傍候補がないため、最大距離の円が描けない。よって、点 C の k 最近傍を確定させるための追加調査の範囲がわからないという状況になる。

4. 最大距離の円が描けないときの解決策

4.1 ランダム探索の繰り返し

追加調査すべき範囲がわからない状況に対する簡単な解決策として、近隣セルをランダムに探索して k 最近傍候補データを集めることを繰り返すことが考えられる。繰り返しにより探索範囲を少しずつ広げる過程で k 件のデータが集まれば、3 節で説明した最大距離の円による追加調査セルの判定を行い、そのデータについての k 最近傍を確定することができる。しかし、条件を満たすまで近隣セルから探索するセルをランダムに選ぶことを繰り返すので、極端に分布が悪い時 (周囲のセルにほとんどデータがない場合) や、選択の仕方とデータの分布の相性が悪かった場合は、繰り返しの回数が多くなってしまいう問題がある。さらにこの手法では、その繰り返しを何度行うことになるのか事前に知ることができない。回数不明の繰り返しが多いことでこういった問題が生じるのかを以下で述べる。

MapReduce 処理モデルにおいては、計算ノードの割り当てや処理に必要なプログラムの受け渡しといった分散処理を実行するための準備に、分散処理の規模に応じた時間が必要となる。繰り返しの分だけ MapReduce 処理が必要になるために、ランダム探索手法ではこの準備時間が何度も加算されてしまう。さらに繰り返しの度に分散ファイルシステムへの途中結果の入出力が必要になるため、その I/O 時間も加算されていく。そして、それだけの時間をかけて実行する処理そのものは、 k 個のデータが得られなかった一部のデータのみが対象となっているため、比較的短い時間で終わることが予想される。軽い処理を行うためにその処理内容とは別なところで時間をかけるのは効率的ではない。よって、ランダム探索手法の改良として、以下で説明するデータの分布情報を用いたセルの統合手法を検討する。

4.2 データの分布情報を用いたセルの統合

ランダム探索の繰り返しとは異なる解決策として、最大距離の円が描けないそもそもの原因である「分割したセルの中に k 最近傍

の候補データが k 個未満しかない」という状況を発生させないことを考える。そのために格子状に区切ったセル分割をそのまま用いるのではなく、各セルのデータの分布数を調べて、データ数が k の値より少ないセルがあった時に近くのセルと統合することで、各セルが必要データ数を満たした分割にする。このセルの統合処理により、最初の k 最近傍の調査の後、すべてのデータについて最大距離の円が描けるようになるため、追加調査範囲が明確になる。その追加調査範囲を調べて結果をまとめることで、確実にすべてのデータについての k 最近傍を確定することができる。分布情報を利用してセルを統合する例を図2に示す。

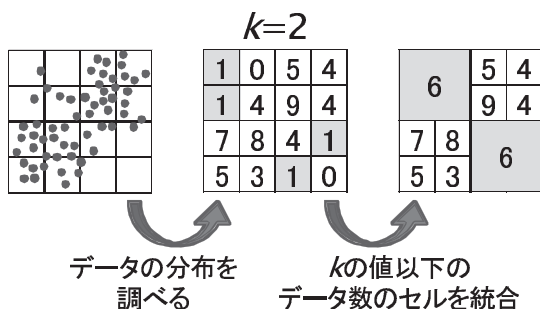


図2: 分布情報によるセル分割の統合

図2では空間分割の粒度を $n=2$ とした 4×4 の分割においてセルの統合を試みている。 $k=2$ という条件と分布情報を合わせて考えると、左上の2つと右下の2つの色付けたセルで k 最近傍候補データが足りないために最大距離の円が描けない状況になることがわかる。これを防ぐために、それらのセルを含むより大きな領域を1つのセルとして統合し、セル中のデータ数が k より多くなるようにする。この統合後のセル分割ならばすべてのセルのデータについて k 最近傍候補データが2個以上見つかるため、最大距離の円を用いた追加調査範囲の判定を行うことができる。

この分布情報を用いたセルの統合手法ではランダム探索手法とは異なり、MapReduce 処理の回数を必要最小限に抑えることができるため、処理の準備時間やファイルシステムへのI/Oも最小限となる。ただし、セルを統合するために必要となるデータの分布情報を得るためのMapReduce 処理を最初に行う必要がある。しかし、この1回の調査とその結果に基づくセルの統合により、何度起こるか不確定なMapReduce 処理をなくすることができるので、採用に値する手法であると考えられる。

また、セルの統合以外にもこの分布情報は使い道がある。例えば、あるセルにはデータが存在しないことを事前に知ることができていれば、 k 最近傍を求める処理においてそのセルが最大距離の円に重なった時でも、データが存在しないため調べる必要はないと判断することができ、計算量を減らすことに繋がる。なお、あまりないことだと思われるが、もし最初にセルに分割した段階ですべてのセルが必要データ数を満たしていたならば、分布情報に基づいたセルの統合の必要がなくなるため、その分布情報を得るための処理の分だけ時間を損することになってしまう。

5. MapReduce 処理モデルの詳細

分布情報を利用した空間分割を基礎とした全 k 最近傍問合せは、以下で解説する4つの処理で実現できる。入力となるデータ集合は、各行が1つのデータを表し、〈データID, x 座標, y 座標〉の情報を持つとする。セルの分割の粒度 n や、対象データの分布する領域の範囲、求めたい k の数は事前に決まっているとする。

5.1 分布調査とセルの統合

ここでは対象となるデータ集合が分布する空間を $2^n \times 2^n$ 個のセルに分割して各セルのデータ数を調べ、その結果によりセルを統合することが目的となる。分布調査部分はMapReduce による処理を行い、その結果をローカルに取得しセルの統合を行う。まず、

MapReduce により分布調査をするためのMap関数とReduce関数の処理を図3に示す。

Map	入力	データ集合
	処理	データの座標から該当するセル番号を計算
	出力	<セル番号, 1>
Reduce	入力	セル番号ごとにグループ化されたデータ
	処理	バリュウの数字を合計
	出力	<セル番号, 合計値>

図3: MapReduce1: 分布調査

この処理はいわゆるワードカウントとほぼ同じことをしている。Map関数では各データからカウントしたい項目（ここでは座標から計算されたセル番号）を抽出し、それと数字の1をキーバリュウペアとして出力する。以降、図ではキーを下線で示し、バリュウとなるデータは下線をつけないことで示す。Shuffle 処理により、Reduce 関数にはセル番号でグループ化されたデータが渡されるので、バリュウの数字を合計して、<セル番号, 合計値>という形で出力する。このMapReduce 処理にて得られた<セル番号, 合計値>というデータが各セルのデータ数を示す分布情報となる。

なお、MapReduce ではCombiner という処理をオプションで利用できる。これは、Map関数の出力をShuffle 処理に送る前に、そのMap関数が使われたMapタスク上でデータを中間集計するものである。集計により、Shuffle 処理でのデータ転送量とReduce タスクでのデータ処理量を減らすことが目的である。今回のような単純なデータの合計では、中間的にデータを合計しても最終的な値は変わらないため、Combiner を利用できる。Combiner の動作としてはReduce 関数とまったく同じものを利用できるため、実際のMapReduce による分布調査は図4のようになる。

Map	入力	データ集合
	処理	データの座標から該当するセル番号を計算
	出力	<セル番号, 1>
Combiner	入力	セル番号ごとにグループ化されたデータ
	処理	バリュウの数字を合計
	出力	<セル番号, 合計値>
Reduce	入力	セル番号ごとにグループ化されたデータ
	処理	バリュウの数字を合計
	出力	<セル番号, 合計値>

図4: MapReduce1: 分布調査 (Combiner を使用)

次にこの分布情報からデータ数が k 以下のセルを統合する処理を行う。この統合処理は分散処理が必要なほどの規模にはならないので、MapReduce は用いずにローカルで処理をする。分布情報から求めたデータ数が k 以下のセルのリストを入力とし、それらのセルを図2のように $2^n \times 2^n$ のセル領域をベースに統合することを目標としている。まず、リストからセルを1つ取り出しそのセルを含む 2×2 のセル領域を設定し、その領域のデータ数を求める。設定した領域内のデータ数が k 以下なら、さらに大きな 4×4 の領域を設定し、その領域のデータ数を求める。このように領域内のデータ数が k より大きくなるという条件を満たすまで m の値を大きくすることで領域を拡大していき、条件を満たした時のセル領域を統合する。ただし、この $2^m \times 2^m$ の領域は中に統合したいセルを含んでいれば自由に決めていいわけではない。領域設定は図5において色付けた領域のように、空間分割の階層構造に従って行うこととする。領域の統合を終えた後は、リストからその領域に含まれるセルを削除する。この処理をリストが空になるまで繰り返す。

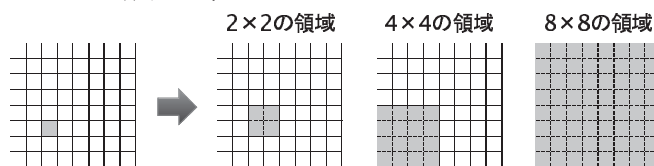


図5: 空間分割の階層

セルの統合についての情報は、<元のセル番号, 統合後のセル番号>という形のマップデータとして出力する。このマップデータは以降の MapReduce 処理でサブデータとして読み込み、セル番号を指定する際に使用する。

5.2 最初の全 k 最近傍の計算

ここでは、セルの統合情報を用いて k 最近傍の計算に必要なデータ数を満たしつつデータをセル領域ごとにまとめ、そのセル領域内における各データの k 最近傍を求めることが目的となる。この処理を行う MapReduce の Map 関数と Reduce 関数の処理を図 6 に示す。

Map	入力	データ集合
	処理	データの座標と統合情報から該当するセル番号を計算
	出力	<セル番号, ID, 座標>
Reduce	入力	セル番号ごとにグループ化された<ID, 座標>
	処理	グループ化されたデータの中でのすべてのペアに対して距離計算を行い k 最近傍を求める
	出力	<ID, 座標, セル番号, k 最近傍リスト>

図 6: MapReduce2: 最初の全 k 最近傍の計算

Map 関数はデータ集合を入力として、各データについて座標情報とセルの統合情報からセル番号を求める。そして、求めたセル番号をキー、ID と座標をバリューとしたキーバリューペアを出力する。Shuffle 処理により、Reduce 関数にはセル番号でグループ化されたデータが渡されるので、そのセルのすべてのデータの組合せで距離計算を行う。これによりセル領域内における各データの k 最近傍が求まるので、それをリストとして追加して出力する。 k 最近傍リストの中身は、 k 件の<ID, 距離>のペアである。

5.3 追加調査範囲の判定と k 最近傍の更新

ここでは、各データに対して 3 節で解説した最大距離の円を用いた追加調査範囲の判定をし、必要なものに対しては追加調査を行い k 最近傍を更新することが目的となる。この処理を行う MapReduce の Map 関数と Reduce 関数の処理を図 7 に示す。

Map	入力	MapReduce2 の結果
	処理	詳しくは本文にて解説
	出力	以下の 2 種類のデータ 1. 各データの k 最近傍リスト <セル番号, ID, 座標, k 最近傍リスト> 2. k 最近傍計算のために必要になる座標データ <セル番号, ID, 座標>
Reduce	入力	セル番号ごとにグループ化された上の 2 種類のデータが混ざった集合
	処理	混ざった 2 種類のデータを分けた後にそれらのデータのすべてのペアに対する距離計算 既存の k 最近傍より近いデータがあれば更新する
	出力	<ID, 座標, セル番号, 更新した k 最近傍リスト>

図 7: MapReduce3: 追加調査範囲の判定と k 最近傍の更新

Map 関数は前の MapReduce2 の結果を入力とする。セルの統合により必要なデータ数を満たしているため、各データについて最大距離の円を描き追加調査範囲の判定をすることができる。円と他のセルとの関係により以下のどちらかの処理を行う。

- 円が他のセルと重ならなかった場合（追加調査不要）：他に k 最近傍候補はないとわかったので、追加調査が必要ないことを示す終了フラグをバリューの末尾に追加した<セル番号, ID, 座標, k 最近傍のリスト, 終了フラグ>というキーバリューペアを出力する。Reduce 関数で行われる追加調査では終了フラグが付けられたデータは基本的に無視するが、出力するデータはその他のデータと同じ形式にする。
- 円が他のセルと重なった場合（追加調査必要）：重なったセルに k 最近傍としてよりふさわしいデータがある可能性があるため、追加調査でそれらのセルを調べる必要がある。よって、<重なったセルのセル番号, ID, 座標, k 最近傍のリスト>とい

うキーバリューペアを出力する。複数のセルに重なっていたときは、その数の分だけキーだけが異なるペアを出力する。

また、 k 最近傍を更新するためには、各調査先セルにおいて距離計算の相手となるデータが必要となる。そのため、追加調査の要不要に関わらずに各データから<セル番号, ID, 座標>という形の距離計算用のキーバリューペアを出力する。その後の Shuffle 処理により、セル番号でグループ化されたデータが Reduce 関数に渡される。しかしその中には、追加調査のためにそのセルに送られたデータとそのセルに含まれる座標データが混在している。よって、まず調査すべきデータと座標データを分離することが必要になる。その上でそれらの間で距離計算を行い、より近い k 最近傍データがあれば置き換えることで、 k 最近傍を更新する。

5.4 k 最近傍の統合

ここまでの処理で追加調査により k 最近傍を更新することができた。しかし、調査範囲が複数のセルに渡ったデータについては各セルにおいて更新後の k 最近傍データが得られているため、最後にそれらの k 最近傍データを統合する必要がある。この処理を行う MapReduce の Map 関数と Reduce 関数の処理を図 8 に示す。

Map	入力	MapReduce3 の結果
	処理	ID をキーとして k 最近傍リストを抽出
	出力	<ID, k 最近傍リスト>
Reduce	入力	ID ごとにグループ化された k 最近傍リスト
	処理	グループ内の k 最近傍リストを統合
	出力	<ID, 統合した k 最近傍リスト>

図 8: MapReduce4: k 最近傍の統合

MapReduce3 の結果を Map 関数の入力とする。Map 関数では、各データから ID をキーとして、 k 最近傍データをバリューとして抽出したキーバリューペアを出力する。あるデータが前の MapReduce 処理で複数のセルを調査先としていれば、Shuffle 処理で調査先別に更新した k 最近傍リストが ID によりグループ化される。それらの k 最近傍リストを Reduce 関数において統合したものが、そのデータについての正しい k 最近傍リストとなる。もともと追加調査が必要なかったデータや追加調査先が 1 つのセルだけだったデータは、ID でグループ化を行っても 1 つしか k 最近傍リストを持たないため、それをそのデータの k 最近傍リストとして出力する。

ここまでの処理により、すべてのデータについての k 最近傍が確定するため、全 k 最近傍問合せは終了となる。

5.5 MapReduce 処理モデルの具体的なデータの流れ

最後に、後半 3 つの MapReduce 処理により全 k 最近傍を求める具体的なデータの流れを、簡単な例を用いて示す。図 9 が例で用いるデータの分布であり、大文字で示した点 A, B, C に注目する。この 3 点はデータ分布の代表的なものであり、すべての点はこのどれかのパターンに該当する。

- 点 A: 最大距離の円が他のセルと重ならない: MapReduce2 で k 最近傍が確定。
- 点 B: 最大距離の円が 1 つのセルと重なる: MapReduce3 でセル 2 を追加調査することで k 最近傍が確定。
- 点 C: 最大距離の円が複数のセルと重なる: MapReduce3 でセル 1, 2, 3 を追加調査。MapReduce4 でセル 1, 2, 3 それぞれの調査結果を統合することで k 最近傍が確定。

図 10 が点 A, B, C が各 Map と Reduce の出力において取る値である。MapReduce が進むことで点 A, B, C の k 最近傍が順に確定していくことが分かる。

6. 実験

5 節で説明した MapReduce プログラムを実際に Hadoop 環境で動作させてその実行時間を測定することで、提案した分布調査に基づくセルの統合手法の性能を見る。

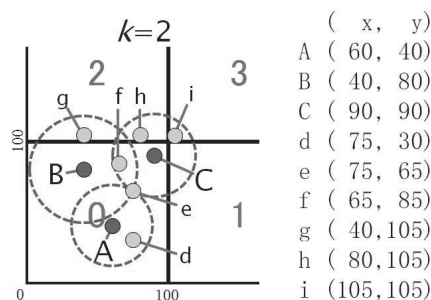


図 9: データ分布の例

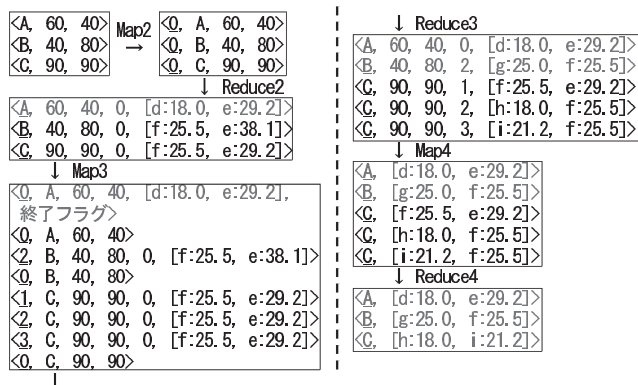


図 10: 点 A, B, C のデータの流れ

6.1 対象データ

実験は人工データを対象にして行った。この人工データの個々のレコードには、データを一意に特定する ID とランダムに生成した x, y 座標データが含まれる。生成時に特に座標に対する偏りは与えていないため、データは対象となる空間全体に均一に分布している。実験ではレコード数が 1,000 万件で約 350MB のサイズの人工データを用いた。

6.2 Hadoop 環境

実験で用いる Hadoop 環境は表 1 に示す構成を用いた。計算ノードは 3 台用意し、1 台は Hadoop のマスターとスレーブを両方動作させ、残る 2 台はスレーブのみを動作させている。

表 1: 計算ノードの構成と Hadoop の設定

計算ノード	OS	Linux 3.0.0-14-server Ubuntu 11.10
	CPU	Intel(R) Xeon(R) CPU E5620 @ 2.40GHz
	CPU コア数	4 × 2
	メモリ	32GB
	HDD	500GB
	ネットワーク	1 ギガビットイーサネット
Hadoop	バージョン	0.20.203.0
	レプリカ数	1
	各ノードの Map タスク数の最大値	8
	各ノードの Reduce タスク数の最大値	8

Hadoop の設定については以下でより詳しく説明する。

- レプリカ数: Hadoop では、何らかの障害によってデータを保存するノードがダウンした時にデータが失われないように、データを複数のノードに複製するレプリカという機能がある。デフォルトの設定ではレプリカ数は 3 となっており、データが 3 つのノードに複製されて保存されることになっている。しかし、実験で用いる Hadoop 環境は小規模なものであるため、障害対策よりも速度を重視してレプリカ数を最小の 1 とした。

- 各ノードの Map タスク数の最大値: この値で、各計算ノードにおいて同時に実行できる Map タスクの数の最大値を制限している。計算ノードの CPU コア数や入力データのサイズや実行する Map 関数の内容に応じてチューニングすることで、よりよい性能を発揮できる可能性がある。デフォルトでは 2 という値が設定されているのに対し、実験では CPU コア数に応じて 8 という値を設定した。
- 各ノードの Reduce タスク数の最大値: この値で、各計算ノードにおいて同時に実行できる Reduce タスクの数の最大値を制限している。Map タスク数と同様で、チューニングにより性能を発揮する可能性がある。やはりデフォルトでは 2 という値が設定されているのに対し、実験では CPU コア数に応じて 8 という値を設定した。

6.3 実験 1: 分散処理数を変化

提案した手法の分散能力を確認するために人工データを用いて、処理における Reduce タスク数を 1 から 24 の間で変化させたときの全 k 最近傍問合せにかかる処理時間を測定した。このとき、分割の粒度 n は 8、全 k 最近傍の k は 5 とした。この実験の結果を図 11, 12 に示す。

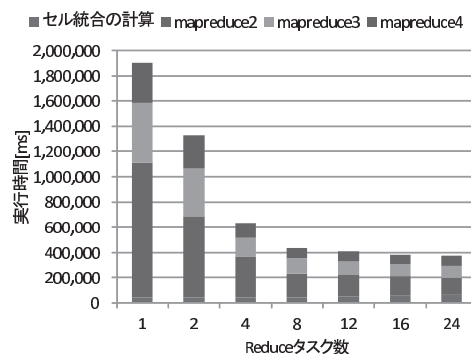


図 11: Reduce タスク数に対する問合せの総処理時間

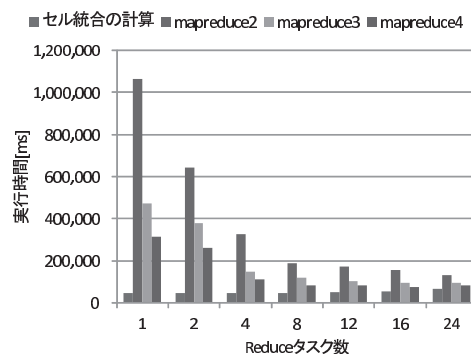


図 12: Reduce タスク数に対する問合せの個別の処理時間

Reduce タスク数を多くすればするだけ全体の処理時間が短くなるという結果が得られ、提案手法の分散能力を確認できた。全 k 最近傍問合せの各処理について個別に見てみると、最初に行うセルの統合の計算は他 3 つの処理とは異なる振る舞いをしていることがわかる。他 3 つでは Reduce タスク数を増やすことで処理時間が減少しているのに対し、セルの統合の計算では、Reduce タスク数を増やすことで処理時間が増加している。つまり、分散処理をしない方が高速に動作していることになる。この原因は、分布情報を得るために使われている MapReduce1 の処理で、他の MapReduce 処理では適用していない Combiner の機能を使っているためだと思われる。Combiner の集計処理により Shuffle 処理に渡されるデータ量と Reduce タスクでのデータ処理量が減少する。データ処理にほとんど時間がかかっていない状況で使用する

る Reduce タスク数を増やしたために、タスクの準備時間が増えてしまいよりより時間がかかってしまった。

6.4 実験 2: k の値を変化

人工データを対象に、セル分割の粒度 n を 8, Reduce タスク数を 24 と固定した状態で、 k の値を 5, 10, 20, 40 と変化させたときの処理時間を調べた。この実験の結果を図 13,14 に示す。

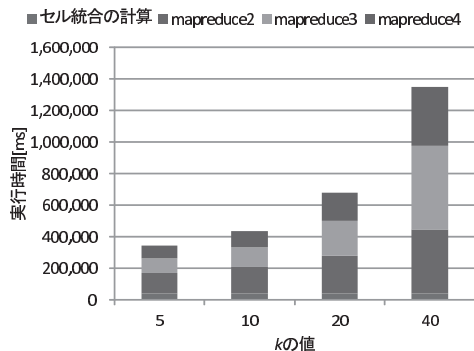


図 13: k の値に対する問合せの総処理時間

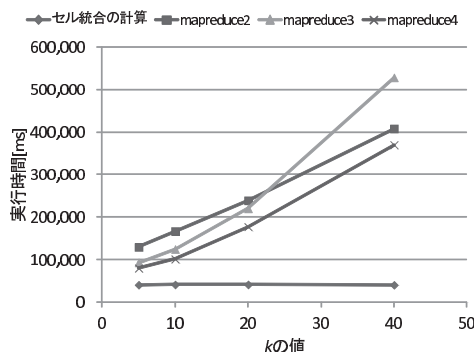


図 14: k の値に対する問合せの個別の処理時間

結果として、 k の値が増えることで総処理時間が増加した。個別の処理で見ると、特に MapReduce3 と MapReduce4 の時間の増加が大きい。処理時間増加の理由としては、 k 最近傍の数が増えたために 1 レコードあたりのバイト数が大きくなり、データ転送時間や I/O コストが増えたことが考えられる。また、MapReduce3 と MapReduce4 で時間の増加が大きかったのは、より多くの k 最近傍を求めた結果として各データの最大距離の円が大きくなり、追加調査するデータが多くなったからである。

6.5 実験のまとめ

実験により、提案手法は分散処理により総処理時間を減らせる性能を持っていることが確認できた (実験 1)。また、 k の値の増加に対しては追加調査が必要なデータが増えていくため、その追加調査の処理にはより時間がかかることが確認された (実験 2)。

最後に、4 節で議論した最大距離の円が描けないときの対応についての結論を述べる。データの分布情報を得てセルを統合するという処理は他の処理に比べると非常に高速であり、その速さは下手に分散処理をしない方が早く結果が得られるほどである。特に処理対象となるデータが大きいほど、セル統合にかかる時間の総処理時間に対する割合が小さくなる。この処理によって確実に最大距離の円を描けるようにし、繰り返しの発生させることなく続く 3 つの MapReduce ジョブで確実に全 k 最近傍を求めるのは理にかなった処理であると言える。

7. おわりに

本論文では、全 k 最近傍問合せを MapReduce フレームワーク上で行う手法を提案した。空間分割により MapReduce の並列分

散処理方式に合った形で実行し、分割により生じる k 最近傍データの見落としには、追加調査が必要となる範囲を表す最大距離の円を求めることで対処した。さらに確実に最大距離の円が描けるように、対象データの分布情報を考慮したセルの統合手法を採用した。実験では、動作する Reduce タスクの数や k の値を変化させたときの処理時間を計測し、提案手法の処理時間の変化を調べた。

今後の課題としては、提案手法をより一般化して、異なるデータ集合間での全 k 最近傍問合せ処理へと拡張することが挙げられる。また、処理データ量に応じた適切な並列処理の数やセル分割の粒度を計算により求めることも挙げられる。実験で確認したように、データ処理時間に対して並列数が多すぎると、逆に総処理時間が多くなってしまう。対象データのサイズやその他の条件に応じた適切な並列処理数やセル分割の粒度を事前に導出できれば、常に最適な環境で分散処理を行えるようになる。

【謝辞】

本研究の一部は科学研究費 (22300034) および「地球環境情報統合プログラム」による。

【文献】

- [1] F.N. Afrati and J.D. Ullman, "Optimizing Joins in a Map-Reduce Environment", *EDBT*, pp.99–110, 2010.
- [2] The Apache Software Foundation: Hadoop Homepage, <http://hadoop.apache.org/>.
- [3] C. Böhm and F. Krebs, "The k -Nearest Neighbour Join: Turbo Charging the KDD Process", *Knowledge and Information Systems*, 6(6), pp.728–749, 2004.
- [4] Y. Chen and J.M. Patel, "Efficient Evaluation of All-Nearest-Neighbor Queries", *ICDE*, pp.1056–1065, 2007.
- [5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *OSDI*, pp.137–150, 2004.
- [6] T. Emrich, et al., "Optimizing All-Nearest-Neighbor Queries with Trigonometric Pruning", *SSDBM*, pp.501–518, 2010.
- [7] D. Jiang, A.K.T. Tung, and G. Chen, "MAP-JOIN-REDUCE: Towards Scalable and Efficient Data Analysis on Large Clusters", *IEEE TKDE*, 23(9), pp.1299–1311, 2011.
- [8] 西田 圭介, Google を支える技術: 巨大システムの内側の世界, 技術評論社, 2005 年.
- [9] R. Vernica, M.J. Carey, and C. Li, "Efficient Parallel Set-similarity Joins Using MapReduce", *SIGMOD*, pp.495–506, 2010.
- [10] T. White, "Hadoop", オライリー・ジャパン, 2010.
- [11] 横山 拓也, 石川 佳治, 鈴木 優, Hadoop 環境における空間分割による並列全 k 最近傍問合せ処理, *DEIM*, 2012 年.
- [12] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao, "All-Nearest-Neighbors Queries in Spatial Databases", *SS-DBM*, pp.297–306, 2004.

横山 拓也 Takuya YOKOYAMA

2012 年名古屋大学大学院情報科学研究科修士。大学院においては大规模データの並列分散処理の研究を行う。現在、富士通関西中部ネットテック (株) に勤務。

石川 佳治 Yoshiharu ISHIKAWA

名古屋大学情報基盤センター教授。データベース、データ工学、情報検索等に興味を持つ。日本データベース学会、情報処理学会、電子情報通信学会、人工知能学会、ACM、IEEE 各会員。

鈴木 優 Yu SUZUKI

名古屋大学情報基盤センター特任助教。情報検索、Web、データ工学等に興味を持つ。日本データベース学会、情報処理学会、電子情報通信学会、人工知能学会、ACM、IEEE CS 各会員。