DEIM Forum 2012 E4-4

Hadoop環境における空間分割による並列全k近傍問合せ処理

横山　拓也† 　石川　佳治††,†,††† 　鈴木　優††

† 名古屋大学大学院情報科学研究科
†† 名古屋大学情報基盤センター
††† 国立情報学研究所

E-mail: yokoyama@db.itc.nagoya-u.ac.jp, ishikawa@itc.nagoya-u.ac.jp, suzuki@db.itc.nagoya-u.ac.jp

あらまし　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．

キーワード　Hadoop，全k近傍問合せ，空間データベース

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; kNN query）と呼ばれている．k最近傍問合せは位置情報を扱う空間データベースで特定の情報を得るためによく行われる問合せである．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; AkNN query）があり，効率的な問合せ処理方式が提案されている[1]〜[3]．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[4]，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[5]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している．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ている．

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[5],[6]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[7]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入出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

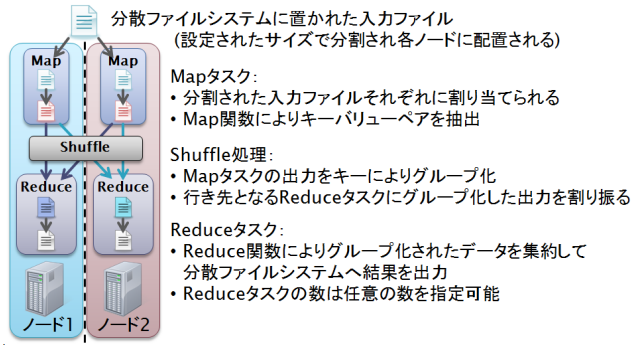


図 1　MapReduce の様子

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．図 1 に 2 つの計算ノードに Hadoop 環境を構築して MapReduce 処理を行う例を示す．

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 [8], [9]．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 [10]．

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. 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 [11]．

全 k 最近傍問合せについては，すでに R-tree 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．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，本論文では扱っていないが，7. で簡単にその違いについて触れる．

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と期待できる．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．と言うのも，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 2 のデータを例にする

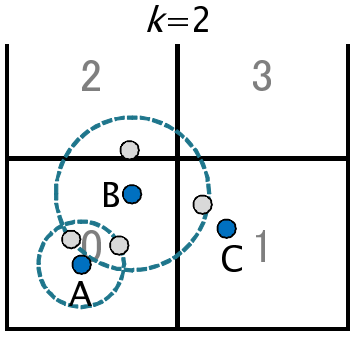


図 2　追加調査範囲の判定

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，それを見つける追加調査が必要になる．

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 の円で描かれる．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（つまり最も遠い）k 最近傍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 2 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 2 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. 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うという問題がある．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ど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 3 に示す．

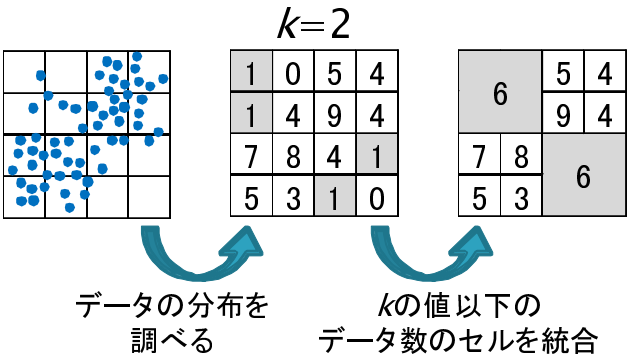


図 3　分布情報によるセル分割の統合

図 3 では空間分割の粒度を $n = 2$ とした 4×4 の分割をベースにし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見てみると，左上の 2 つと右下の 2 つの色付けしたセルで 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することで，セル中のデータ数が k より多くなるようにしている．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（データ分布が完全に均一になるように作られた人工データが対象だった場合など），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

3.4. で示した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，本章の各節で解説する 4 つの処理により実現できる．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 3 つの情報を持つとする．また，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 4 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	バリューの数字を合計
	出力	<セル番号, 合計値>

図 4　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 をキーバリューペアとして出力する．以降，図ではキーとなるデータを下線で示し，バリュー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，バリューの数字を合計して，<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．

ところで，MapReduce 処理では Combiner という処理をオプションで使うことができる．Combiner とは，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．集計によって 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となっている．今回のような単純にデータを合計するという作業では，中間的にデータを合計しても最終的な合計値は変わらないため，この Combiner を使うことができる．その 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 5 のように行った．

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	バリューの数字を合計
	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	バリューの数字を合計
	出力	<セル番号, 合計値>

図 5　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．統合する際にキーとなるアルゴリズムを図 6 に示す．

```
入力：データ数が k 以下のセルのリスト l
while リスト l にデータがある do
  c ←リスト l の最初のデータ
  m ← 0
  sum ← 0
  while sum ≦ k do
    m ← m + 1
    c を含む 2^m × 2^m のセル領域を設定
    sum ← セル領域のデータ数の合計値
  end while
  設定されたセル領域を統合
  for セル c' in セル領域 do
    if c'がリスト l にある then
      リストからセル c' を削除
    end if
  end for
end while
```

図 6　セル統合のアルゴリズム

このアルゴリズムでは，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 3 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 7 において色付けした領域のように，空間分割の階層

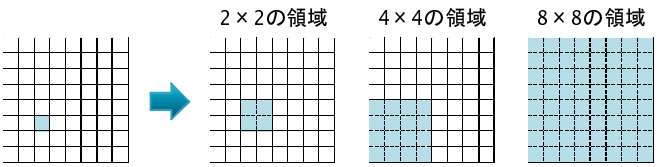


図 7　空間分割の階層

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．

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 8 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 8　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. 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 9 に示す．

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．

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 9　MapReduce3：追加調査範囲の判定と k 最近傍の更新

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 10 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>

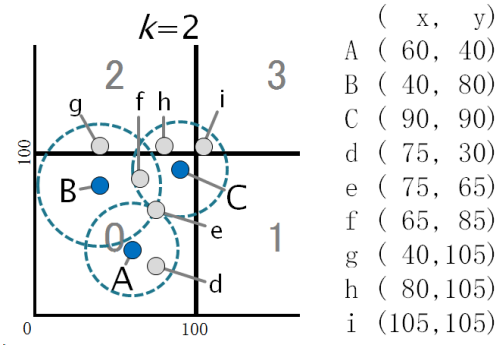
図 10　MapReduce4：k 最近傍の統合

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．

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 11 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．



	(x, y)
A	(60, 40)
B	(40, 80)
C	(90, 90)
d	(75, 30)
e	(75, 65)
f	(65, 85)
g	(40,105)
h	(80,105)
i	(105,105)

図 11　データ分布の例

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 12 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. 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．

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

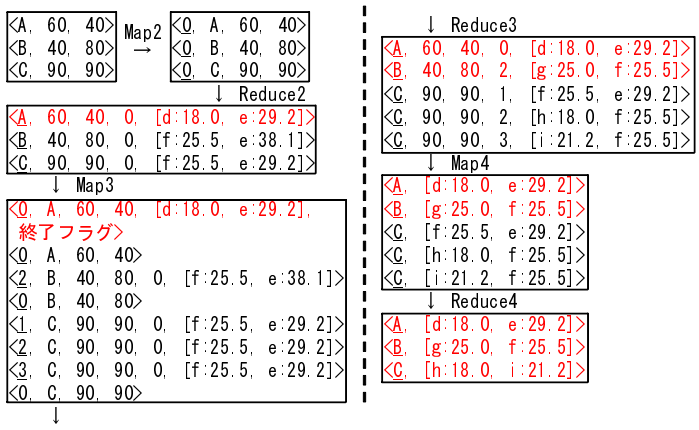


図 12　点 A,B,C のデータの流れ

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 10,000,000 個で約 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．

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

表 1　計算ノードの構成と Hadoop の設定

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なので，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．

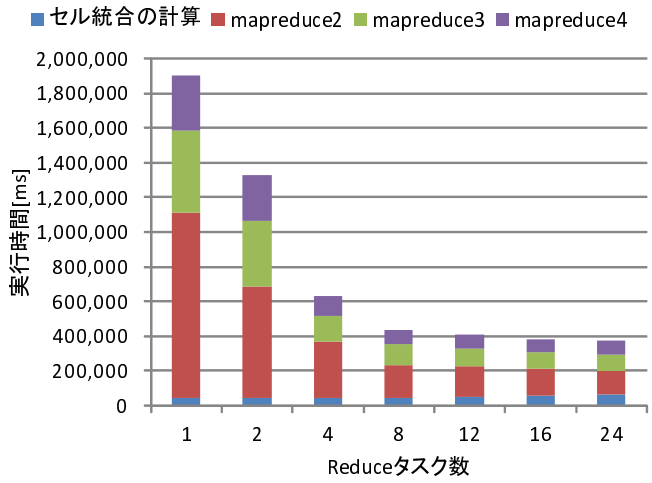


図 13　Reduce タスク数の変化と全 k 最近傍問合せの総処理時間

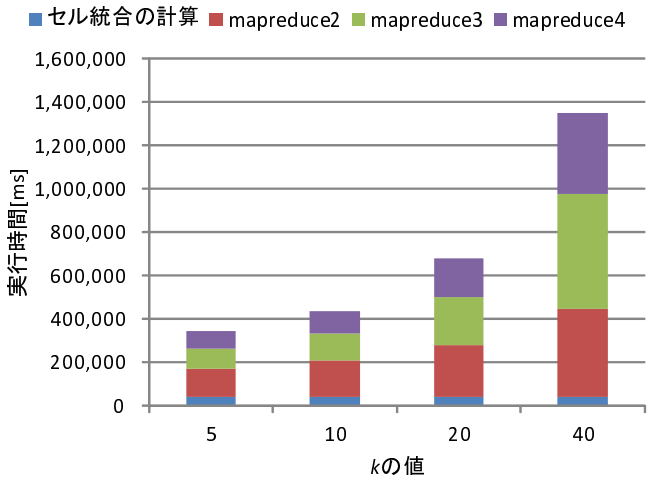


図 15　k の値の変化と全 k 最近傍問合せの総処理時間

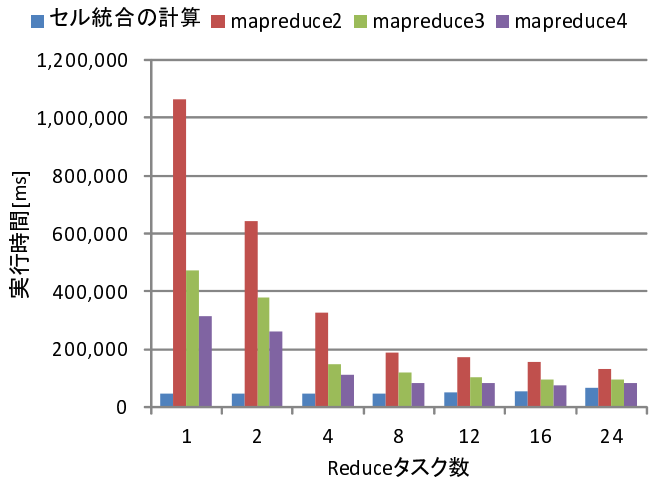


図 14　Reduce タスク数の変化と全 k 最近傍問合せの個別の処理時間

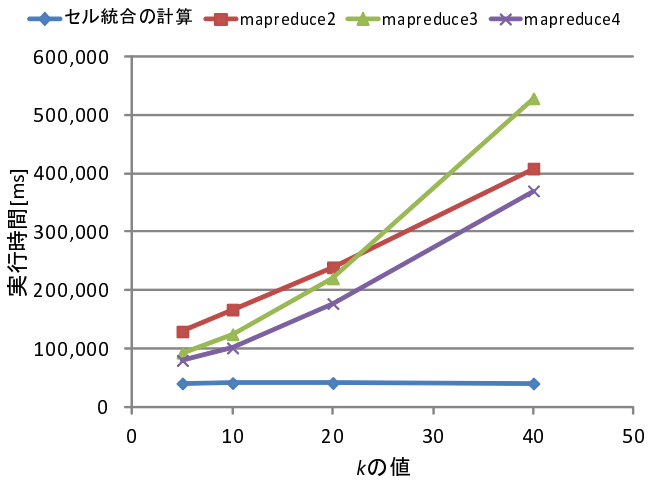


図 16　k の値の変化と全 k 最近傍問合せの個別の処理時間

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 1,2,4,8,12,16,24 と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 13,14 に示す．

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 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 15,16 に示す．

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. 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，繰り返しを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全 k 最近傍問合せ処理を行えるよう拡張することが挙げられる．つまり，2 つのデータ集合 A と B に対して，データ集合 A のすべてのデータについてデータ集合 B の中から k 最近傍データを取得できるようにする．このためには，MapReduce2 と 3 において 2 つのデータ集合を読み込み，それらの間で距離計算を行うことと，2 つのデータ集合の分布情報を利用してセルを統合することが必要となる．

距離計算のための改良については，提案手法の MapReduce3 の Reduce 関数で行った処理が参考になる．その Reduce 関数では，k 最近傍を更新するデータと距離計算用の座標データが混在した入力データに対し，それらを分けた後で距離計算を行っていた．同様のデータの仕分けをデータ集合 A と B で適用することで，2 つのデータ集合間での k 最近傍計算が行える．

2 つの分布情報を利用したセルの統合は，1 つの分布情報を利用する場合と少し異なる．具体的にあるセルを統合する必要があるかどうかは，データ集合 A のデータの有無とデータ集合 B のデータ数が k 以上か未満かで判断する．表 2 にその組合せを示す．

結論から述べると，データ集合 A のデータがあり，かつデータ集合 B のデータ数が k 未満の場合だけセルを統合する必要がある．セル統合の目的は，最初の k 最近傍調査ですべてのデータが k 個以上の k 最近傍候補を得られるようにすることである．そのため，たとえデータ集合 B のデータ数が k 未満でも，そのセルに k 最近傍を求めるデータ集合 A のデータがなければ，セルを統合する必要はない．

データ集合 A	データ集合 B	セル統合の必要性
データあり	データ数が k 以上	なし
データあり	データ数が k 未満	あり
データなし	データ数が k 以上	なし
データなし	データ数が k 未満	なし

表 2　あるセルのデータの分布状況とセル統合の必要性

その他の課題としては，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が挙げられる．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計算により求めることができれば，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「地球環境情報統融合プログラム」による．

文　献

[1] Yun Chen and Jignesh M. Patel, “Efficient Evaluation of All-Nearest-Neighbor Queries”, Proc. of ICDE, pp.1056-1065, 2007.
[2] Tobias Emrich and Franz Graf and Hans-Peter Kriegel and Matthias Schubert and Marisa Thoma, “Optimizing All-Nearest-Neighbor Queries with Trigonometric Pruning”, SSDBM, pp.501-518, 2010.
[3] Jun Zhang and Nikos Mamoulis and Dimitris Papadias and Yufei Tao, “All-Nearest-Neighbors Queries in Spatial Databases”, SSDBM, pp.297-306, 2004.
[4] 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.
[5] The Apache Software Foundation: Hadoop Homepage, http://hadoop.apache.org/.
[6] Tom White,“Hadoop”, オライリー・ジャパン, 2010.
[7] Jeffrey Dean and Sanjay Ghemawat, “MapReduce Simplified Data Processing on Large Clusters”, OSDI, pp.137-150, 2004.
[8] Foto N. Afrati and Jeffrey D. Ullman, “Optimizing Joins in a Map-Reduce Environment”, EDBT, pp.99-110, 2010.
[9] Dawei Jiang and Anthony K. T. Tung and Gang Chen, “MAP-JOIN-REDUCE: Towards Scalable and Efficient Data Analysis on Large Clusters”, IEEE TKDE (accepted for publication).
[10] Rares Vernica and Michael J. Carey and Chen Li, “Efficient parallel set-similarity joins using MapReduce”, SIGMOD, pp.495-506, 2010.
[11] Christian Böhm and Florian Krebs, “The k-Nearest Neighbour Join: Turbo Charging the KDD Process”, Knowledge and Information Systems, Vol.6, No.6, pp.728-749, 2004.