Hadoopにおけるアウトオブオーダ型並列処理系の実装に関する一考察

山田　浩之† 　合田　和生†† 　喜連川　優††

† 東京大学大学院情報理工学系研究科　〒113–8656 東京都文京区本郷 7-3-1
†† 東京大学生産技術研究所　〒153–8505 東京都目黒区駒場 4-6-1
E-mail: †{hiroyuki,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　Hadoop による並列処理を活用した大規模データの解析が広く普及している．Hadoop は問合せ処理の水平展開による高速化に特徴を有するが，選択性のある問合せを必ずしも高速に処理できない問題がある．本論文では，この問題に対し，Hadoop をベースとしたアウトオブオーダ型並列処理系を提案する．提案する並列処理系は，並列問合せ処理を実行時に複数の処理単位に分解し，処理単位を並行に実行する．これにより，高多重な非同期入出力を発行し，選択性のある問合せの大幅な高速化を図る．本論文では，提案する並列処理系の実装を議論するとともに，20 ノードのクラスタ環境での性能評価実験を示し，アウトオブオーダ型並列処理系の有効性を明らかにする．

キーワード　OoODE，アウトオブオーダ型実行，問合せ処理，並列処理，Hadoop，大規模データ解析

Hiroyuki YAMADA†, Kazuo GODA††, and Masaru KITSUREGAWA††

† Graduate School of Information Science and Technology, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113–8656 Japan
†† Institute of Industrial Science, The University of Tokyo
4-6-1 Komaba, Meguro-ku, Tokyo, 153–8505 Japan
E-mail: †{hiroyuki,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

1. はじめに

社会の情報化が進展するに従い，人々の経済活動や社会活動が膨大なデジタル情報として蓄積されるようになりつつある．このような膨大な情報は，企業の経営やマーケティング，新たな社会的価値の創出などに積極的に活用され始めている．オンライン通信販売やインターネットオークションを手がける米 eBay は，利用者の行動履歴等からなる 40PB 以上のデータアウェアハウスを運用し，それらを用いた解析を行うことで，利用者に適したコンテンツの提供を試みている [1]．また，デンマークの電力会社 Vestas Wind Systems は，気象，潮汐，地理空間データ，衛星写真などからなる数 PB のデータを解析し，風力タービンの最適な設置場所の特定に役立てている [2]．

このような大規模データの解析は，大規模ストレージに接続された集約システムや，複数のサーバからなる並列システムにより従来から行われてきているが，近年，Netezza(注1)を始めとするデータウェアハウス・アプライアンスの登場により，並列システムの利用が一段と進みつつある．データウェアハウス・アプライアンスは，コモディティサーバからなるクラスタを利用し，ハードウェアとソフトウェアを一体化した並列データ処理システムである．データウェアハウス・アプライアンスは，データ処理技術は 1990 年代までの並列データベース技術を応用したものに過ぎないと考えられるが，パッケージ化したデータ処理システムを安価に提供することで，大規模データの解析が一般企業等へと普及する一つのきっかけとなった．

並列処理による大規模データの解析が普及し始めているなか，Hadoop(注2)によりこの流れは加速してきている．Hadoop は Google 社の MapReduce [3] のオープンソース実装であり，高いスケーラビリティや高い耐障害性を特徴とした並列処理系である．Hadoop は，Yahoo や Facebook による積極的な活用・開発により，安定性が広く認知され始めており，さらに，Cloudera 等のディストリビューションが提供するツール群により，導入は著しく容易になっていることから，ウェブ企業におけるデータ解析ソフトウェアのデファクトスタンダードとなりつつある．

Hadoop を始めとする並列処理系における問合せ処理は，データの全走査を基本としている．データの全走査による問合せ処理は，問合せの選択性に関わらず，データ規模に比例した処理

(注1)：http://www.ibm.com/software/jp/data/netezza/

(注2)：http://hadoop.apache.org/

時間を要する．この処理特性は，データが大規模化するにつれて無視できない問題になる．このような問題に対して，索引等の並列データベースで培われた技術を並列処理系に応用し，選択性のある問合せの高速化が試みられている [4], [5]．しかしながら，索引を用いたデータアクセスはストレージへの逐次的なランダムアクセスとなり，逐次的なランダムアクセスはストレージが有する帯域を有効に活用できないことから，選択性を有する問合せを必ずしも高速に処理できない．

一方，選択性を有する問合せに焦点を当て，問合せの飛躍的な高速化を狙うデータベース技術への取り組みが行われ始めている．アウトオブオーダ型データベース [6] は，問合せに内在する並列性を考慮し，問合せ処理を非同期に実行することで，高多重の非同期入出力を発行し，ストレージのランダムアクセス帯域を最大限に活用するものである．

問合せ処理の非同期化は，これまで，単一計算機上において有効性が議論されてきた [7], [8]．複数の計算機から構成される無共有並列計算機においても，並列問合せ処理を非同期に実行することにより問合せの高速化が期待されるが，当該実行方式の実現方法および有効性は明らかではない．

以上の背景から，本論文では，並列問合せ処理の非同期に実行する，Hadoop をベースとしたアウトオブオーダ型並列処理系を提案する．アウトオブオーダ型並列処理系は, 並列問合せ処理を実行時に複数の処理単位に分解し，処理単位を並行に実行することにより，高多重な非同期入出力を発行し, 選択性を有する問合せの大幅な高速化を図る．本論文では, 提案する並列処理系の実装を議論するとともに, 20 ノードのクラスタ環境での性能評価実験を示し, アウトオブオーダ型並列処理系の潜在的な有効性を明らかにする．

本論文の構成は以下の通りである．第 2 節でアウトオブオーダ型並列処理系の概要を述べ，第 3 節で Hadoop をベースとしたアウトオブオーダ型並列処理系の実装を述べる．第 4 節で解析タスク及び TPC-H ベンチマークによる評価実験結果を示し，有効性を論じる．第 5 節で関連研究を紹介し，第 6 節で本論文をまとめる．

2. アウトオブオーダ型並列処理系

従来型の問合せ処理は，基本的には逐次的な手続きの実行に基づいている．つまり，問合せ処理は，ある入出力を行い，その入出力結果に対する演算を行う，という処理の繰り返しにより問合せを実行する．すなわち，入出力から当該入出力結果の演算に至る一連の動作は, 事前にプログラムされた決定的な順序に基づくものとなる．ここでは，このような問合せ処理をインオーダ型実行の問合せ処理と呼ぶことにする．

次に，アウトオブオーダ型実行の問合せ処理を考える．アウトオブオーダ型実行の問合せ処理とは，相互に独立した処理には依存関係は存在せず，それらの処理は並行に実行可能であるという基本原理に基づき，問合せ処理を実行することである．すなわち，ある演算において新たな入出力を行う必要が生じると，都度にタスク分解を行い，分解された並行実行可能なタスク上で入出力と関連する演算を行う．このように，問合せ処理にかかる入出力を非同期化することにより，大量の非同期入出力が同時に発行される．これにより，ストレージが有する並列性と，ディスクドライブ及び RAID コントローラ等の高度なスケジューリング機構を効率的に活用することが可能となり，選択性を有する問合せは飛躍的に性能向上することが期待される．

本論文では，並列処理系における並列問合せ処理のアウトオブオーダ型実行，すなわち，並列問合せ処理の非同期化について考える．

並列問合せ処理は，基本的には，演算に必要なレコードをリモートノードから取得し(注3)，当該レコードに対して演算を適用することを繰り返すことにより処理を進めることができる．例えば，選択処理は述語条件に基づくレコードをリモートノードから取得することにより，処理を行うことができる．結合処理においても，外表のレコードをリモートノードから取得し，そのレコードの結合キーを基に，内表のレコードをリモートノードから取得することにより，処理を行うことができる．

アウトオブオーダ型並列処理系は，この一連のリモートレコードアクセスを非同期に実行することで，並列問合せ処理の非同期化を試みる．すなわち，並列問合せにおける一連の処理を，ディスクドライブまたはネットワークを介したレコードアクセスとそのレコードに対する演算からなる処理単位に分解し，それらの処理単位を並行に実行する．そして，ディスクドライブ及びネットワークにおけるレコードの入出力処理の完了に伴い演算が駆動されるべく制御する．

リモートレコードのアクセス方法には多様な方法が考えられるが，本論文では索引を用いたリモートレコードアクセスを考える．索引を用いたリモートレコードアクセスは，索引によりリモートレコードの位置を特定し，レコードの取得を行うことである．各ノードに所謂ローカル索引がある場合は，全ノードの索引にアクセス（ブロードキャスト）することにより，探索条件に紐づくレコードを取得することができる．この方法は，広く使われるローカル索引を活用できるため，適用機会が多いという利点があるが，全ノードへのブロードキャストアクセスが発生するため，リモートレコード探索のオーバーヘッドが大きくなってしまう．リモートレコードアクセスの効率化として，分割グローバル索引 [10] を利用する方法も考えられる．この方法は，分割グローバル索引のパーティション構成を利用することで，リモートレコードアクセスを選択的に行うことができる．

3. Hadoop をベースとしたアウトオブオーダ型並列処理系の実装

本節では，Hadoop をベースとしたアウトオブオーダ型並列処理系の詳細を述べる．最初にシステム構成と全体の概要を述べ，次に，各構成要素の特徴および実装を述べる．

3.1 システム構成

提案するアウトオブオーダ型並列処理系のシステム構成を図 1 に示す．通常，データ解析を行う並列処理系は，クエリ（ジョ

(注3)：当然、レコードの要求元ノードとレコードの格納ノードが同一の場合はローカルアクセスとなるが，ここではすべてリモートアクセスと表現する．

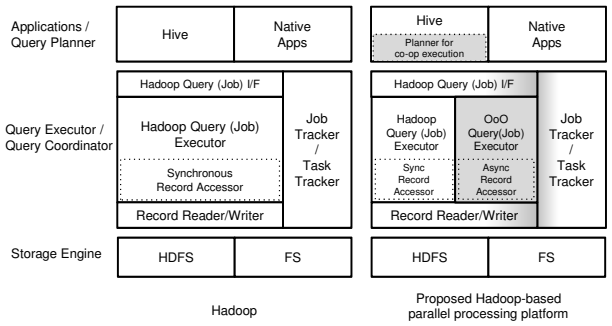


図 1　システム構成（左：Hadoop，右：提案処理系）

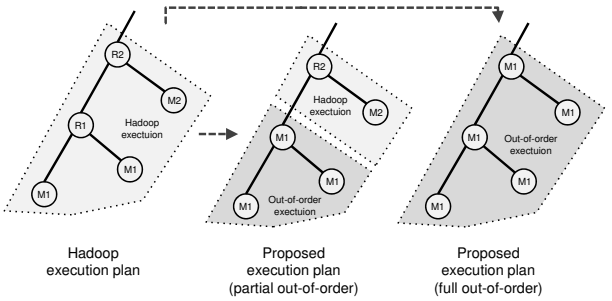


図 2　協調型実行プラン（M，R はそれぞれ Map 処理，Reduce 処理を表し，それに続く数字はジョブのシーケンス番号を表す.）

ブ）コーディネータおよびクエリエクゼキュータを中間層にもち，下位層にストレージエンジン，上位層にアプリケーションプログラムまたはクエリプランナ等をもつ．提案処理系は Hadoop をベースとし，従来の Hadoop のクエリエクゼキュータに加えて，アウトオブオーダ型実行のクエリエクゼキュータを有する．

当該並列処理系は，ユーザから与えられた逐次的な手続きからなるアプリケーションプログラムまたは SQL（HiveQL）を入力とし，次の手順で動作する．

A. ジョブ実行プラン（クエリプラン）の生成　逐次的な手続きからなるアプリケーションプログラムが入力の場合は，システム利用者が実行プラン（プログラム）を作成する．SQL が入力の場合は，Hive をベースとしたクエリプランナが SQL をパースし，適切な実行プランを生成する．実行プランは従来の Hadoop による実行とアウトオブオーダ型実行からなる協調型実行プラン（図 2）となる．

B. ジョブ（クエリ）の起動　実行プランを基に Hadoop ジョブを起動する．クエリコーディネータは与えられた実行プランに従いクエリエクゼキュータを選択する．

C. ジョブ（クエリ）の実行　実行プランに従い Hadoop ジョブを実行する．Hadoop 実行のクエリエクゼキュータとアウトオブオーダ型実行のクエリエクゼキュータは協調して問合せを実行する．選択，結合は Hadoop 実行とアウトオブオーダ型実行のいずれかの実行方法で処理が行われ，集約，整列は従来の Hadoop により実行される．

3.2　システムの構成要素

3.2.1　クエリエクゼキュータ

クエリエクゼキュータは，協調型実行プランに従い，通常の Hadoop による実行とアウトオブオーダ型実行を協調して実行する機構を有する．通常の Hadoop は，Hadoop ジョブを連続に実行することにより様々な解析処理を行うが（図 2（左）），当該クエリエクゼキュータは，この処理の一部または全体をアウトオブオーダ型実行により行う（図 2（左，右））．このような実行機構により，提案処理系は問合せの内容に応じた適切な実行方法を選択することが可能となる．

アウトオブオーダ型実行のクエリエクゼキュータは，TaskTracker の拡張 RecordReader 並びに当該 RecordReader から利用されるモジュールからなる．当該クエリエクゼキュータが扱うデータは，Hadoop 配下のレプリカ（拡張レプリカ）として管理され，分割属性値でハッシュ分割されたファイル並びにそのファイルに対する索引からなる．拡張レプリカにおける索引は，ローカル索引とグローバル索引からなる．

3.2.2　クエリプランナ

クエリプランナは Hive をベースとし，拡張として協調型実行プラン（図 2）を生成する機構を備えている．クエリプランナは SQL（HiveQL）から生成された Hadoop の実行プランに対して，拡張レプリカの情報とその統計情報に基づき，当該プランの変更の可否と有効性を判断し，適切な協調型実行プランを生成する．

3.2.3　システムインターフェース

問合せのインターフェースは，手続き型のプリミティブインターフェースと宣言型の Hive インターフェースからなる．

a）　プリミティブインターフェース

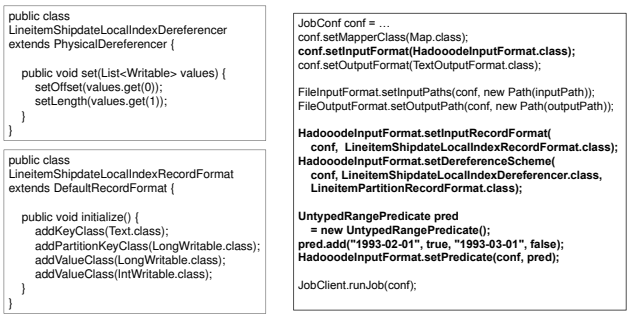
プリミティブインターフェースは，Hadoop における Map 処理と Reduce 処理の手続きプログラムに加えて，リモートレコードアクセスに対する逐次的な手続きプログラムを受け付ける．これらの手続きプログラムは，MapReduce 処理の内容に合わせて，システム利用者が作成する．すなわち，実行プランのアウトオブオーダ型実行への変換は，システム利用者の明示的な指示により行われる．システムはプリミティブインターフェースで与えられた当該手続きプログラムに基づき，問合せ処理のアウトオブオーダ型実行を行う．

次に，当該インターフェースの利用による実行の流れを示す．

A. 手続きプログラムの作成　レコードアクセスのための手続きプログラムを作成する．レコードアクセスのための手続きプログラムは，索引及び実表のレコード書式プログラムと，索引から実表への参照方式プログラムからなる．レコード書式プログラムは，データ構造におけるレコードの書式であり，レコードのキー属性値の型，レコードの分割属性値の型，レコードのキー属性値に紐づけられた値の型を規定するプログラムである．また，参照方式プログラムは，データ構造から別のデータ構造への参照の方法を規定するプログラムである．

B. ジョブの起動　Hadoop のジョブプログラムに追加情報を設定し，ジョブを起動する．追加情報とは，アクセスする索引と実表のレコード書式，索引から実表への参照方式，及び選択条件からなる．

C. ジョブの実行　システムは，与えられた選択条件と索引の



```
public class
LineitemShipdateLocalIndexDereferencer
extends PhysicalDereferencer {

    public void set(List<Writable> values) {
        setOffset(values.get(0));
        setLength(values.get(1));
    }
}
```

```
public class
LineitemShipdateLocalIndexRecordFormat
extends DefaultRecordFormat {

    public void initialize() {
        addKeyClass(Text.class);
        addPartitionKeyClass(LongWritable.class);
        addValueClass(LongWritable.class);
        addValueClass(IntWritable.class);
    }
}
```

(a) ローカル索引のレコード書式プログラム（上）とローカル索引から実表への参照方式プログラム（下）

```
JobConf conf = …
conf.setMapperClass(Map.class);
conf.setInputFormat(HadooodeInputFormat.class);
conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(inputPath));
FileOutputFormat.setOutputPath(conf, new Path(outputPath));

HadooodeInputFormat.setInputRecordFormat(
    conf,  LineitemShipdateLocalIndexRecordFormat.class);
HadooodeInputFormat.setDereferenceScheme(
    conf, LineitemShipdateLocalIndexDereferencer.class,
    LineitemPartitionRecordFormat.class);

UntypedRangePredicate pred
    = new UntypedRangePredicate();
pred.add("1993-02-01", true, "1993-03-01", false);
HadooodeInputFormat.setPredicate(conf, pred);

JobClient.runJob(conf);
```

(b) アプリケーションプログラム

図 3　プリミティブインターフェース

レコード書式に基づき索引を探索し，結果（該当する実表レコードのオフセットとレコード長）の値を参照方式プログラムに設定する．システムは，参照方式プログラム及び索引アクセスから得られた値に従い，実表への参照を並行に実行する．

図 3(a) にプリミティブインターフェースを利用した LINEITEM 表へのレコードアクセスにおけるプログラム例を示す．図 3(a)（上）に LINEITEM.l_shipdate 索引のレコード書式を，図 3(a)（下）に索引から実表 LINEITEM への参照方式プログラムを示す．l_shipdate は日付の属性であり，l_shipdate 索引の分割属性は l_orderkey，値は実表 LINEITEM へのオフセットとレコード長としている．また，図 3(b) にプリミティブインターフェースを利用したアプリケーションプログラムの例を示す．通常の Hadoop アプリケーションプログラムに太字で示した数行の手続きを追加することで，アウトオブオーダ型実行の指示をシステムに与えることが可能となっている．

b） Hive インターフェース

Hive インターフェースは，Hive と同様に SQL（HiveQL）を受け付ける．SQL は前述のクエリプランナにより実行プランへの変換が行われ，アウトオブオーダ型実行に必要な手続きは，システムにより自動的に生成される．

4. 評価実験

アウトオブオーダ型並列処理系の有効性を検証するために，解析タスク及び TPC-H ベンチマークを用い，性能評価実験を行う．

4.1 実験環境

評価実験には，20 台のデータ処理ノードと 1 台の管理ノードからなるクラスタを用いた．ハードウェア環境および基本ソフトウェア環境は表 1 のとおりである．各ノードが有する 24 本のデータ用のディスクドライブから，22D+2P（RAID6）の LU（ストライプサイズ: 512KB）を構成した．LU 上に ext4 ファイルシステムを構築し，すべての問合せ処理は当該ファイルシステム上で行った．

4.2 評価対象システム

本実験では, Hadoop と本論文で提案する Hadoop ベースの並列処理系の比較を行う．両システムの共通基盤である Hadoop は，バージョン 0.20.2 を使用した．Hadoop は “Real World Cluster Configuration”(注4) に従い設定し，最大 Mapper 数，最大 Reducer 数はそれぞれコア数と同数にした．管理ノードでは，NameNode 及び JobTracker が動作し，データ処理ノードでは DataNode 及び TaskTracker が動作する．解析タスクにおける Selection タスクはプリミティブインターフェースによりジョブを実行し，その他の実験においては Hive を経由してジョブを実行する．

表 1　実験環境

クラスタ構成	
ノード数	20 データ処理ノード（＋ 1 管理ノード）
インターコネクト	DELL PowerConnect 5548 (1Gbps)
ノード構成	
CPU	Intel Xeon E5-2680 2.70GHz x 2 (16 コア)
メモリ	64GB
HDD (for OS)	SAS 15Krpm x 2
HDD (for data)	SAS 10Krpm x 24
RAID Controller	PERC H710P
OS	CentOS (Linux kernel 2.6.18)

本実験では，実験結果として 3 種類の結果を示す．**Hadoop** は通常の Hadoop を表す．**Hadoop w/ index** は **Hadoop** に対して索引を利用したリモートレコードアクセスを行う問合せ処理を追加したシステムを表す．**OoO Hadoop** は提案するアウトオブオーダ型並列処理系を表す．各問合せ実行時間は Hadoop ジョブの起動時間の除いた時間とする．

アウトオブオーダ型実行における実行多重度(注5)の調整には，オペレーティングシステムのスレッドを使用する．特に言及が無い限り，提案処理系の実行多重度は 1400 とする．また，ノード間通信の実装には TCP を用いる．提案処理系においては，クエリプランナのパラメタの調整を事前に行い, 本実験のすべての問合せで, 問合せの選択率に応じた最適な問合せ処理を選択する．

4.3 解析タスクによる性能評価

最初の性能評価は，多くの MapReduce 関連論文で用いられる解析タスク [11] を使用する．当該タスクは，ログファイルの解析と HTML 文書の処理等を行う 4 つの解析タスクからなる．本論文では，図 4 に示す Selection タスク及び Join タスクを用いる．

解析タスクを行うデータセットは，HTTP サーバのアクセスログを保持する UserVisits 表，HTML 文書を保持する Document 表，Document 表に対して付与されたメタデータを保持する Ranking 表からなる．このデータセットのスキーマ情報の詳細は [11] を参照されたい．提供されるデータジェネレータにより，約 8 億件の UserVisits レコード（約 100GB）と約 950 万件の Rankings レコード（約 560MB）を各ノードごとに作成した．Hadoop では，各ノードから hadoop コマンドを利用

(注4)：http://hadoop.apache.org/docs/r0.20.2/cluster_setup.html
(注5)：本論文では，説明の便宜上，問合せにおける同時処理中のデータアクセス数を問合せ処理の実行多重度と呼ぶ．実行多重度が 1 の **OoO Hadoop** は，**Hadoop w/ index** と同じ処理となる．

```
SELECT pageURL, pageRank
FROM Rankings
WHERE pageRank > X;
```

(a) Selection タスク

```
SELECT sourceIP, COUNT(pageRank),
       SUM(pageRank), SUM(adRevenue)
  FROM Rankings AS R, UserVisits AS UV
  WHERE R.pageURL = UV.destURL AND
      R.pageRank >= X
      UV.visitDate >= ``2000-01-15'' AND
      UV.visitDate <= ``2000-01-22'';
```

(b) Join タスク

図 4　解析タスク

し，当該データを HDFS へロードした．提案処理系の実行に必要な実表および索引は，HDFS にロードしたデータから独自のレプリカビルダ(注6)により作成した．レプリカビルダにより作成される実表の分割キーは主キーとした．

4.3.1　Selectioin タスク

Selection タスクは，Ranking 表の選択問合せである．図 4(a) の SQL に相当するアプリケーションプログラムを作成しジョブを実行した．

図 5 に Selection Task の実験結果を示す．横軸は問合せの選択率を表し，縦軸は実行時間（対数）を表す．**Hadoop** は処理対象データの全走査を行うため，問合せの選択率に大きく依らず長い実行時間を要していることがわかる．**Hadoop w/ index** はローカルの索引を利用したアクセスを行うため，選択率 0.001%では， **Hadoop** に対し多少の性能の向上がみられるものの，選択率が 0.01%以上では，ディスクドライブに対する逐次的なランダムアクセスにより，**Hadoop** よりも大幅に性能が低くなっていることがわかる．**OoO Hadoop** は，選択率 0.1%以下ではアウトオブオーダ型実行の効果により，他のシステムに対して大幅に性能が向上していることがわかる．**OoO Hadoop** は，選択率 0.01%において, **Hadoop** に対して約 28 倍, **Hadoop w/ index** に対して約 65 倍の性能向上の達成が確認された．

図 6 に，**OoO Hadoop** が活用する問合せ処理の実行多重度を変化させた場合の問合せの実行時間を示す．活用する問合せ処理の実行多重度が向上するにつれて，問合せの実行時間が大幅に減少していることがわかる．このように，豊富なストレージ資源を有する実験環境においては， 高い実行多重度により問合せを実行することで，ストレージ資源の性能を最大限に活用できることがわかる．

4.3.2　Join タスク

Join タスクは，図 4(b) に示す Ranking 表と UserVisits 表の結合問合わせである．

図 7 に Join Task の実験結果を示す．**Hadoop** は対象データの全走査を行う再分配結合，すなわち，並列ソートマージ結合であり，処理データの絶対量は各選択率で大きな差がないため，

(注6)：紙面の都合から，レプリカビルダに関する説明は省略する．

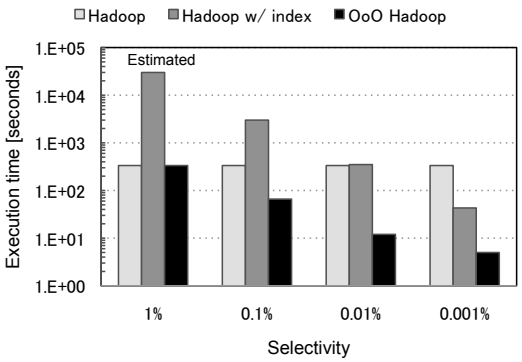


図 5　Selection タスクにおける選択率ごとの実行時間の比較

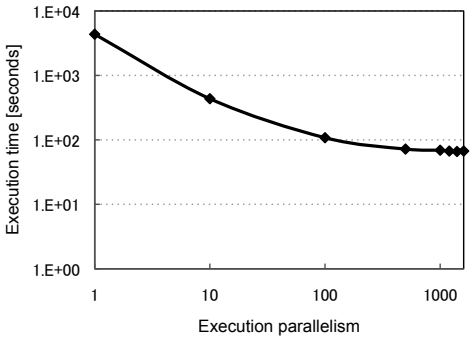


図 6　Selection タスクにおける実行多重度による実行時間の変化（選択率 0.1%）

問合せの選択率に依らず長い実行時間を要していることがわかる．**Hadoop w/ index** は索引を利用した結合，すなわち，並列ネステッドループ結合に類する結合処理を行うため，選択率が 0.001%のところでは **Hadoop** に対して性能の向上がみられるものの，他の選択率においては．**Hadoop** に対して大幅に性能が低くなっていることがわかる．**OoO Hadoop** は，選択率 0.1%以下では結合処理がアウトオブオーダ型実行により行われ，他のシステムに対して大幅に性能が向上していることがわかる．**OoO Hadoop** は，選択率 0.01%において, **Hadoop** に対して約 22 倍, **Hadoop w/ index** に対して約 65 倍の性能向上の達成が確認された．

図 8 に，**OoO Hadoop** が利用する問合せ処理の実行多重度を変化させた場合の，問合せの実行時間を示す．結合処理においても，活用する問合せ処理の実行多重度が向上するにつれて，アウトオブオーダ型実行の効果が増加し，問合せの実行時間が大幅に減少していることがわかる．

4.4　TPC-H ベンチマークによる性能評価

本実験で使用する TPC-H データセットのスケールファクタ（SF）は 20K（約 20TB）とした．本実験では，代表的な問合せとして Q3，Q8 を用いて，各システムの比較を行う．

図 9，10 にそれぞれの問合せの実行結果を示す．横軸は問合せにおける最外表の選択率を表し，縦軸は実行時間（対数）を表す．また，実行時間は選択・結合処理と，それ以外の処理（集約・整列処理等）にブレークダウンして示す．**Hadoop** は

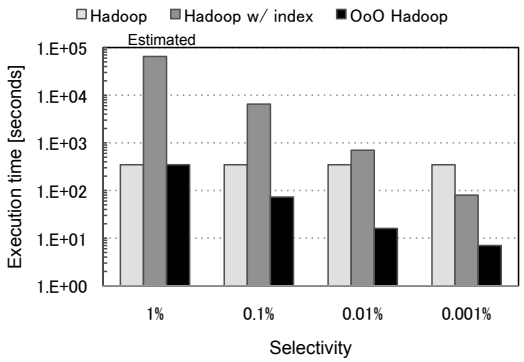


図 7　Join タスクにおける選択率ごとの実行時間の比較

図 8　Join タスクにおける実行多重度による実行時間の変化（選択率 0.1%）

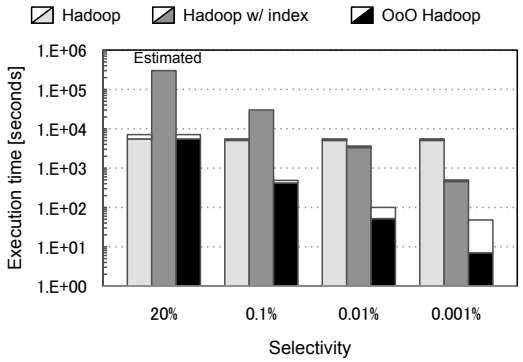


図 9　TPC-H Q3 おける選択率ごとの実行時間の比較

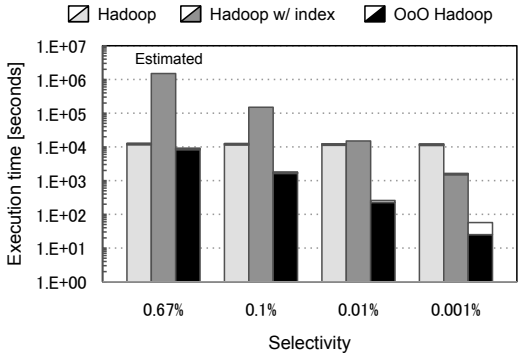


図 10　TPC-H Q8 おける選択率ごとの実行時間の比較

問合せの選択率に大きく依らず長い実行時間を要していることがわかる．**Hadoop w/ index** は索引を利用した結合を行うため，選択率 0.01%以下では，**Hadoop** に対して多少の性能の向上がみられるものの，**Hadoop** に対する高速化の効果は小さいことがわかる．**OoO Hadoop** は Q3 では選択率 0.1%以下で，Q8 ではすべての選択率でアウトオブオーダ型実行の結合処理が行われ，他のシステムに対して大幅に性能が向上していることがわかる．Q3（選択率 0.01%）においては，**Hadoop** に対して約 52 倍（選択処理と結合処理のみにおいては約 104 倍），**Hadoop w/ index** に対して約 36 倍（選択処理と結合処理のみでは約 65 倍）の性能向上の達成が確認され，Q8（選択率 0.01%）においては，**Hadoop** に対して約 46 倍（選択処理と結合処理のみにおいては約 55 倍），**Hadoop w/ index** に対して約 46 倍（選択処理と結合処理のみでは約 65 倍）の性能向上の達成が確認された．

また，これらの実行トレースにおいて，提案処理系はストレージのランダムアクセス帯域をほぼ最大限に活用していることが確認された(注7)．Q3，Q8 における結合では，表の分割状況から，多くのレコードアクセスはリモートノードへのアクセスとなる．通常，リモートノードへのレコードアクセスは，レコードのストレージ入出力時間に加えて，ネットワークアクセス時間を余計に要するため，ローカルアクセスに比べて多くの時間を要する．提案処理系は，多くのリモートレコードアクセスを行うものの，リモートレコードアクセスは非同期的に実行されるため，実行時間はストレージのランダムアクセス帯域により大きく決定され，すなわち，レコードアクセスにおけるレイテンシの増加の影響をほとんど受けていないことがわかる．

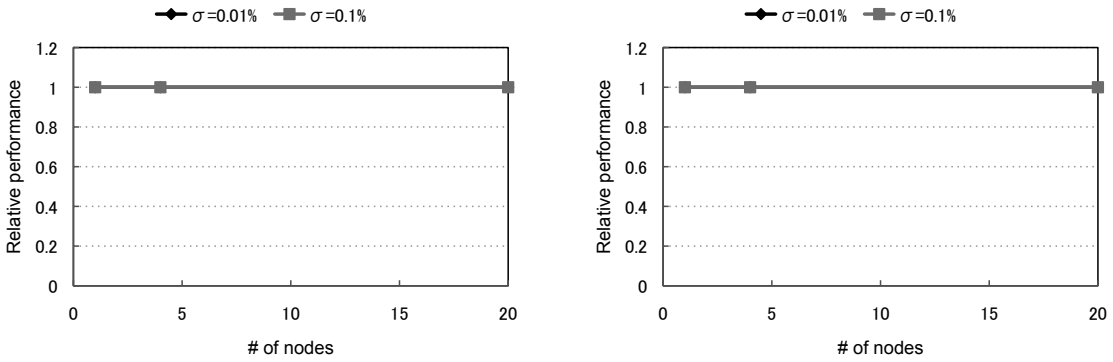
これらの実験から，TPC-H のような複雑な問合せにおいても，アウトオブオーダ型実行の並列問合せ処理が有効に機能することがわかる．

4.5　スケーラビリティの評価

本項では，スレーブノード数と処理対象データサイズを変化させ，提案処理系のスケーラビリティを検証する．スケーラビリティの検証における指標はスケールアップとする．つまり，ノード数に対するデータサイズの割合を固定し，性能評価を行う．ノード数は 1，4，20 において実験を行う．本実験では，アウトオブオーダ型実行による並列問合せ処理のスケーラビリティを検証することが目的のため，Hadoop 実行の部分は問合せ処理から除外する．実験結果は，1 ノードでの実行時間をスケーラビリティの測定対象のノード数での実行時間で割った相対的な性能を示す．

図 11(a)，11(b) に Selection タスクと Join タスクの実験結果を，図 11(c)，11(d) に TPC-H の各種問合せの実験結果を示す．Selection タスクにおける実験では線形なスケーラビリティが得られている．Selection タスクは，ローカル索引を利用した実表アクセスを行うため，リモートノードへのアクセスは発

(注7)：紙面の都合上，実行トレースの結果は省略する



(a) Selection Task

(b) Join Task

(c) TPC-H Q3

(d) TPC-H Q8

図 11　解析タスク，TPC-H クエリにおけるスケーラビリティ

生せず，マルチノード化による処理内容の変化は起きないためである．また，Join タスクにおいても線形なスケーラビリティが得られている．これは，Join タスクで用いた両表は共に結合キー属性（pageURL と destURL）で分割されており，この属性で両表を結合する場合は，リモートノードへのアクセスは発生せず，Selection タスクと同様にマルチノード化による処理内容の変化は起きないためである．Q3 においては，20 ノードで，約 1.5%（選択率 0.01%）から約 6.7%（選択率 0.1%）の性能低下が起きている．これは，Customer 表から Orders 表の索引へのアクセス並びに第二段階の外表から Lineitem 表の索引へのアクセスはリモートノードへのレコードアクセスであることが起因している．多くのリモートレコードアクセスはストレージアクセスとなり，そのアクセス時間はストレージのランダムアクセス帯域に律速されるが，一部の局所性が高いリモートレコードアクセスは OS のバッファキャッシュにヒットし，リモートレコードアクセスによるレイテンシの増加の影響を受ける．リモートレコードアクセスのレイテンシは問合せ処理に参加するノード数に比例して増加することが確認されているため，ノード数の増加により若干のスケーラビリティの低下がみられると考えられる．また，選択率が高い場合に性能がより低下しているが，これは選択率の増加に比例してリモートレコードアクセスの回数が増加し，それに伴いバッファキャッシュのヒット率が増加するためであると考えられる(注8)．Q8 では，20 ノードで，約 2.9%（選択率 0.01%）から約 5.1%（選択率 0.1%）の性能低下が起きている．この性能低下は 2 つの原因からなり，1 つは Q3 と同様の原因である．もう 1 つは，Q8 がアクセスする Region 表および Nation 表の不均一な分割により，マルチノードによる実行ではノードごとの処理に不均衡が生じるためである．このため，Q3 と比較して 4 ノードで性能が低下していることが確認できる．このように，リモートレコードアクセスにより若干の性能低下が確認されているものの，アウトオブオーダ型実行の並列問合せ処理は高いスケーラビリティを有していることがわかる．

5. 関 連 研 究

索引を用いたリモートレコードアクセスに関する研究は，1990 年台の並列データベースの研究において Dewitt らにより行われてきた [9], [12]．これらは内表タプルのフェッチに同期的なリモートレコード探索を用いた方法を議論し，それを利用した結合処理（並列ネステッドループ結合処理）の性能特性を検証している．本論文はこれに対して，アウトオブオーダ型並列処理系におけるリモートレコード探索の非同期化を議論しており，議論の対象が大きく異なる．

(注8)：低い選択率の並列問合せ処理では，バッファキャッシュを完全には使い切らないため，レコードアクセス回数の増加に伴いキャッシュヒット率は向上すると考えられる．

Hadoop を始めとする近年の並列処理系においては，選択性を有する問合せを高速化すべく，並列データベース技術の応用が試みられている．Hadoop++ [4]、HAIL [13] では Hadoop におけるクラスタ化索引の適用方法と有効性を議論している．また，Hive [5] においてもパーティショニングの実装が取り入れられている．クラスタ化索引やパーティショニングは，それらが適用された属性を基点とした問合せ処理においてアクセス範囲を局所化する技術であり，我々が提案する問合せの実行方式とは直交する技術であると考えられる．すなわち，これらは相互補完的な関係にあり，例えば，クラスタ化されていない属性を基点としたアクセスを，アウトオブオーダ型実行により高速化することが可能となる．

並列処理系における結合処理の効率化についても多くの研究が存在する．Blanas ら [14] は，並列データベースにおける主要な結合方法を MapReduce で実装する方法を議論し，それらの性能特性の検証している．Afrati ら [15] は，複数段の結合を 1 つの MapReduce ジョブで実行するためのタプルの最適な分配方法を提案している．また，HadoopDB [16], [17] や Hadoop++ [4] では，結合するデータを結合キーで事前にパーティショニングしておき，結合処理を Map 側で実行する方法を提案している．これらはいずれも，Hadoop のフレームワークを利用した並列ソートマージ結合並びに並列ハッシュ結合の効率化技法を議論している．一方，我々が提案する問合せの実行方式は，これらの結合方法において必ずしも効率的に処理できない結合選択性を有する問合せを対象としている．

6. ま と め

本論文では，Hadoop をベースとしたアウトオブオーダ型並列処理系を提案した．当該処理系は，並列問合せ処理を実行時に複数の処理単位に分解し，処理単位を並行に実行することにより，高多重な非同期入出力の発行を可能とし，問合せの大幅な高速化を図るものである．提案処理系の実装を議論し，20 ノードのクラスタ環境での性能評価実験を行った．実験の結果，提案処理系は選択性を有する問合せの大幅な高速化が可能であることを示した．選択率 0.01%の TPC-H 問合せにおいて，Hadoop に対して約 46 倍から 52 倍（選択処理と結合処理のみにおいては，約 55 倍から 104 倍），索引を利用した Hadoop に対して約 36 倍から 46 倍（選択処理と結合処理のみにおいては，約 65 倍）の性能向上が確認された．さらに，スケーラビリティの検証を行った結果，提案処理系はノード数に対してほぼ線形のスケーラビリティが得られることを示した．今後は，より大規模な環境を用いた検証や負荷分散方式等の検討を順次進めていきたい．

謝辞 本研究の一部は，内閣府最先端研究開発支援プログラム「超巨大データベース時代に向けた最高速データベースエンジンの開発と当該エンジンを核とする戦略的社会サービスの実証・評価」，および日本学術振興会科学研究費補助金（特別研究員奨励費）24・7965 の助成により行われた．

文　献

[1] Oliver Ratzesberger. “Agile Enterprise Analytics”, Proc. of USENIX FAST, 2010.
[2] Vestas Wind Systems Turns to IBM Big Data Analytics for Smarter Wind Energy, 2011.
[3] J. Dean, S. Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters” Proc. OSDI, pp. 137–150, 2004.
[4] J. Dittrich, J. A. Quiane-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad. “Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing)”, Proc. VLDB, pp. 515–529, 2010.
[5] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu and R Murthy. “Hive — A Petabyte Scale Data Warehouse Using Hadoop”, Proc. ICDE, pp. 996–1005, 2010.
[6] 喜連川優，合田和生. アウトオブオーダ型データベースエンジン OoODE の構想と初期実験, 日本データベース学会論文誌, 8(1), 6 2009.
[7] 合田和生，豊田正史，喜連川優. アウトオブオーダ型データベースエンジン OoODE の試作とその実行挙動, 第 5 回データ工学と情報マネジメントに関するフォーラム, 2013.
[8] 早水悠登，合田和生，喜連川優. アウトオブオーダ型データベースエンジン OoODE によるクエリ処理性能の実験的評価, 第 5 回データ工学と情報マネジメントに関するフォーラム, 2013.
[9] D. J. DeWitt, J. F. Naughton, J Burger. “Nested Loops Revisited”, Proc. PDIS, pp. 230–242, 1993
[10] J. Liebeherr, E. R. Omiecinski, I. F. Akyildiz. “The effect of index partitioning schemes on the performance of distributed query processing”, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, pp. 510–522, 1993.
[11] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. “A comparison of approaches to large-scale data analysis”, Proc. SIGMOD, pp. 165-178, 2009.
[12] A. Brown and C. Kozyrakis. “Parallelizing the index-nested loops database join primitive on a shared-nothing cluster”, Technical Report, No. 1598, Dept. of Computer Science, Berkeley Univ., 1998.
[13] J. Dittrich, J. A QuianRuiz, S. Richter, S. Schuh, A. Jindal and J. Schad. “Only aggressive elephants are fast elephants”, Proc. VLDB, pp. 1591–1602, 2012. Only aggressive elephants are fast elephants
[14] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y Tian. “A comparison of join algorithms for log processing in MaPreduce”. Proc. SIGMOD, pp. 975–986, 2010.
[15] F. N. Afrati and J. D. Ullman. “Optimizing joins in a map-reduce environment”, Proc. EDBT, pp. 99–110, 2010.
[16] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. “HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads” Proc. VLDB, pp. 922–933, 2009.
[17] K. B. Pawlikowski, D. J. Abadi, A. Silberschatz, and E. Paulson. “Efficient processing of data warehousing queries in a split execution environment” Proc. SIGMOD, pp. 1165–1176, 2011.