128 ノード規模のストレージインテンシブクラスタ環境における アウトオブオーダ型並列データ処理系の性能評価と 実データを用いた有効性の検証

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
††† 国立情報学研究所　〒 101–8430 東京都千代田区一ツ橋 2-1-2
E-mail: †{hiroyuki,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　著者らは，アウトオブオーダ型並列データ処理系と称する高性能並列データ処理系を開発している．アウトオブオーダ型並列データ処理系では，各々の計算機において，並列データ処理の実行時にタスク分解を行い，分解されたタスクにおいて自らの二次記憶ならびにネットワークを介した他の計算機の二次記憶への入出力を行い，入出力の完了に伴い関連する演算を実行する．即ち，並列データ処理における入出力全体を非同期化する．データインテンシブな並列データ処理においては，入出力に性能が律速されることが多く，当該入出力を非同期化することにより，従来型の処理系に比して，特にデータセット空間の一部のデータを対象とするデータ処理において，飛躍的な高速化が期待される．本論文では，著者らが試作を行った Hadoop をベースとするアウトオブオーダ型並列データ処理系 Hadooode の構成法を示すとともに，128 台の計算機からなるクラスタマシンにおいて当該試作を用いて行った性能評価実験を示し，その有効性を明らかにする．

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

1. はじめに

計算機システムを構成するハードウェア技術の潮流を見ると，プロセッサコアの動作周波数の向上は 2008 年からほぼ停滞し [1]，また，磁気ディスクドライブのレイテンシの削減は年率 5%以下に留まっており [2]，当該傾向はメモリモジュールならびにネットワーク装置を構成するハードウェアにおいても見られる [3]．即ち，これらの単一のハードウェアから構成される計算機システムにおいては，今後著しい性能向上は期待できないと考えられ，計算機システムの高性能化の実現には，複数のハードウェアを集約することが必須となりつつある．企業においてはハードウェアを高密度に集積した単一の大型システムが広く利用されている一方で，複数のコモディティサーバを高速なネットワークで接続したクラスタシステムが利用されるケースも少なからず見られ，昨今のビッグデータ解析と称される超大規模データの解析においては，価格性能比の点から，後者が進展しつつあることが伺える．クラスタシステム上で高速にデータ処理を行う並列データ処理系が産業界・学術界から次々と発表・提案されるに至っており [4]〜[9]，並列データ処理系の高速化はビッグデータ解析における一つの鍵になると言えるだろう．

著者らは，アウトオブオーダ型並列データ処理系と称する高性能並列データ処理系を開発している．アウトオブオーダ型並列データ処理系は，関係データベースエンジンで試みられているアウトオブオーダ型実行方式 [10], [11] を拡張し，並列データ処理系の各々の計算機において，並列データ処理の実行時にタスク分解を行い，分解されたタスクにおいて自らの二次記憶ならびにネットワークを介した他の計算機の二次記憶への入出力を行い，入出力の完了に伴い関連する演算を実行する．即ち，並列データ処理における入出力全体を非同期化する．データインテンシブな並列データ処理においては，入出力に性能が律速されることが多く，当該入出力を非同期化することにより，従来型の処理系に比して，特にデータセット空間の一部のデータを対象とするデータ処理において，飛躍的な高速化が期待される．本論文では，著者らが試作を行った Hadoop をベースとするアウトオブオーダ型並列データ処理系 Hadooode の構成法を示すとともに，128 台の計算機からなるクラスタマシンにおいて TPC-H データセットならびに実データを用いた当該試作による性能評価実験を示し，その有効性を明らかにする．著者らは文献 [12], [13] において，20 ノードのクラスタ構成において人工データセットを用いた評価実験により Hadooode の有効性を示してきたが，本論文では，128 ノードの大規模クラスタ構成における性能評価実験を行い，また，新たに提案する最適化方式の評価ならび実データを用いた評価実験を行うことにより，Hadooode の有効性，ならびに有用性を明らかにするものである．

本論文の構成は以下の通りである．第 2 節では，アウトオブオーダ型並列データ処理系の概要およびその問合せ最適化方式を紹介し，第 3 節では，著者らが開発を進めているアウトオブオーダ型並列データ処理系の実装を示す．第 4 節では，128 台

の計算機から構成されるクラスタマシンにおいて，TPC-H ベンチマークデータセットを用いた当該実装の性能評価実験を行い，第 5 節では，実データを用いた性能評価実験を行う．第 6 節で関連研究を述べ，第 7 節で本論文を纏める．

2. アウトオブオーダ型並列データ処理系

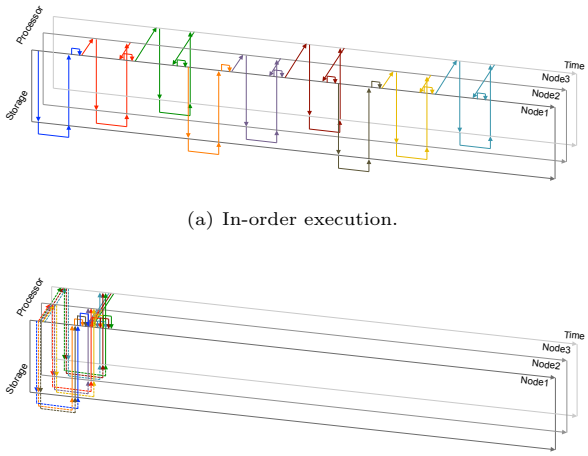
2.1 並列データ処理のアウトオブオーダ型実行

並列データ処理は，高速なネットワークによって接続された複数の計算機を用いて，与えられたデータ処理を並列化して実行することにより，その高速化を目指すものである．これまで多様なシステムアーキテクチャが検討されてきたが，多くの並列データ処理系では，事前にデータ処理の対象となるデータセットを分割して計算機の備える二次記憶に格納しておき (注1)，データ処理のジョブが与えられると，当該ジョブを分割もしくは複製して計算機に割り当て，各計算機において割り当てられたジョブを実行する．各々の計算機では，割り当てられた各ジョブにおいて，データの入出力命令を発行して，当該入出力の完了を待って演算を実行することを，対象の全てのデータを処理し終えるまで繰り返す．入出力と演算は事前にプログラムされた順序に基づき行われることから，当該方式をインオーダ型の実行方式と称する．並列データ処理における単一ノードを起点とした入出力と演算に対する当該実行方式の挙動を模式的に図 1(a) に示す．

これに対して，本論文では，並列データ処理系におけるアウトオブオーダ型実行方式を提案する．当該実行方式では，各計算機においてジョブの実行時に，新たな入出力を発行する必要が生じると，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当該入出力とそれにかかる演算を実行する．すなわち，ジョブにおいては，タスク分解によって，その入出力が非同期的に発行されることとなり，入出力の完了と共に演算が駆動されることとなる．並列データ処理系のストレージアーキテクチャに依存するが，入出力は発行元の計算機の管理する二次記憶に対してのみならず，他の計算機のノードが管理する二次記憶に対して行う必要がある場合があり，この場合，計算機間のネットワークを介した通信も同様にタスク上で実行される．入出力と演算が多段で行われるような場合，このような手順に従って，タスク分解が再帰的に行われることとなり，実行時に，データセットとジョブの実行論理が許す限りにおいて，多数の入出力が並行して発行され，また，多数の演算が並行して実行されることとなる．当該挙動を模式的に図 1(b) に示す．

実際には，同時に実行可能なタスクの数や同時に実行可能な入出力や通信の数は計算機が有する資源によって制約されるものの，例えば，後述する実験環境における 2U 程度の筐体に収まるサーバ型計算機においては，少なくとも 1,000 個程度のタスクの同時実行が可能である．マイクロプロセッサ技術の潮流としては，プロセッサあたりのコア数は着実に増加する傾向にあり，ビッグデータブームに牽引され，サーバ型計算機が備える二次記憶装置の集積密度は従来に比して高まっている．アウトオブオーダ型のソフトウェア実行方式は，これらの資源を効率的に活用することにより，インオーダ型の実行方式に比して，データ処理のスループットの大幅な向上を目指すものである．

(注1)：提案方式はストレージネットワーク等による共有型の二次記憶へも適用可能である．



(a) In-order execution.

(b) Out-of-order execution.

図 1　並列データ処理のアウトオブオーダ型実行．
Fig. 1　Out-of-order execution in parallel data processing.

尚，並列データ処理系にアウトオブオーダ型実行方式を適用するアプローチとしては，次の二つの方法に大別できると考えられる．

（1）　並列データ処理系における各計算機のストレージエンジンにアウトオブオーダ型実行方式を適用

（2）　並列データ処理系全体にアウトオブオーダ型実行方式を適用

アプローチ（1）は，既存のアウトオブオーダ型データベースエンジンを活用し，アウトオブオーダ型並列データ処理系を実現しようとするものである．例えば，既存研究である HadoopDB [14] のアイデアを応用し，Hadoop における各計算機のストレージとして OoODE 等のアウトオブオーダ型のデータベースエンジンを用いる実装が考えられる．当該アプローチを適用した並列データ処理系は，既存システムを大幅に変更することなく既存システムの組合せにより実現可能であると考えられるが，各計算機が管理する二次記憶に対する入出力のみがアウトオブオーダ型データベースエンジンにより非同期化されるものである．即ち，当該並列データ処理系は，アウトオブオーダ型実行方式が部分的に適用されたシステムであり，例えば，他の計算機が管理する二次記憶に対するネットワークを介した入出力は同期的に行われることとなる．

一方，本論文は，より根源的であるアプローチ（2）を提案し，その有効性を明らかにするものである．当該アプローチを適用した並列データ処理系は，実現には既存システムの大幅な改変または拡張を要する可能性があるが，並列データ処理全体の入出力が非同期化されるという特徴を有する．並列データ処理系においては多様な問合せが想定され，アプローチ（1）によりアウトオブオーダ型実行の効果が充分に発揮される問合せも多く存在すると考えられる一方で，アプローチ（2）により

初めて，もしくは，更にその効果が発揮される問合せも少なくない．本論文では，上記のアプローチの効果の違いを比較実験により明らかにする．

2.2 アウトオブオーダ型並列データ処理を考慮した問合せ最適化

本節ではアウトオブオーダ型並列データ処理の問合せ最適化方式を提案する．提案する最適化方式は，問合せの実行時間をコストとして見積もり，多様な問合せ処理の中から最小コストのものを選択するコストベースの最適化方式である．当該最適化方式においては，事前に構築したインオーダ型実行ならびにアウトオブオーダ型実行の並列データ処理のコストモデルを用い，対象データセットのサイズ，タプル数，属性ごとのカーディナリティ，最大値，最小値等の統計情報を利用し，問合せ時にコストの計算を行う．また結合順序においては，二表の結合を繰り返すことで複数表の結合を実現するレフトディープ（Left-Deep）実行木に探索空間を絞り，当該探索空間において，動的計画法を用いて決定する．この際，データセットの複数ノードへのパーティション構成を考慮し，結合処理の各段階において，パーティションごとに最適プランを保持することにより，動的計画法における最適性の原理を保証する．本論文では，紙面の制約上，最適化方式の概略を述べるに留め，詳細は別稿に譲りたい．

3. アウトオブオーダ型並列データ処理系 Hadooode

本論文では，オープンソースの並列データ処理系である Hadoop をベースとするアウトオブオーダ型並列データ処理系 Hadooode を紹介する．Hadooode は，主に問合せ実行器，問合せ受付器，問合せ最適化器から構成される．問合せ実行器は，問合せにおける並列データ処理をインオーダ型実行，もしくは前述のアウトオブオーダ型実行により行うものである．アウトオブオーダ型の実行エンジンは拡張 RecordReader とそれから参照されるモジュールとして実装されており，Hadoop 本体の改変は一切行っていない．問合せ受付器は，MapReduce ジョブまたは Hadoop の SQL 処理系である Hive からの問合せプランを受け取り，それを問合せ実行器へと受け渡すものである．本論文の評価実験では，Hive からの問合せのみを扱うが，Hadooode は Map，Reduce の手続きからなるジョブにおいても，アウトオブオーダ型で実行することが可能となっている．また，問合せ最適化器は，前述のアウトオブオーダ型並列データ処理を考慮したコストベースの問合せ最適化方式を Hive に実装したものである (注2)．

4. 128 ノードクラスタを用いた評価実験

著者らは，Hadooode を用い，アウトオブオーダ型並列データ処理の評価実験を行った．本節では，128 ノードクラスタにおける評価実験の結果を示し，提案方式の有効性を明らかにする．

(注2)：元来の Hive はクエリツリーの作成と当該ツリーに対するルールベースの最適化のみを行うものである．

図 2　実験システム：128 ノードクラスタ．
Fig. 2　Experimental system : 128-node cluster.

表 1　実験システムの諸元．
Table 1　Experimental system specification.

Cluster Configuration	
# of nodes	128
Interconnect	Dell Force10 Z9000 (10Gbps)
Node Configuration	
Processor	Intel Xeon E5-2680 2.70GHz x 2 (2p/16c)
Memory	64GB
HDD (OS)	2 x 15,000rpm 300GB SAS HDDs
HDD (Data)	24 x 10,000rpm 900GB SAS HDDs
RAID Controller	Dell PERC H710P
OS	CentOS 5.8 (Linux kernel 2.6.18)

4.1 実験環境

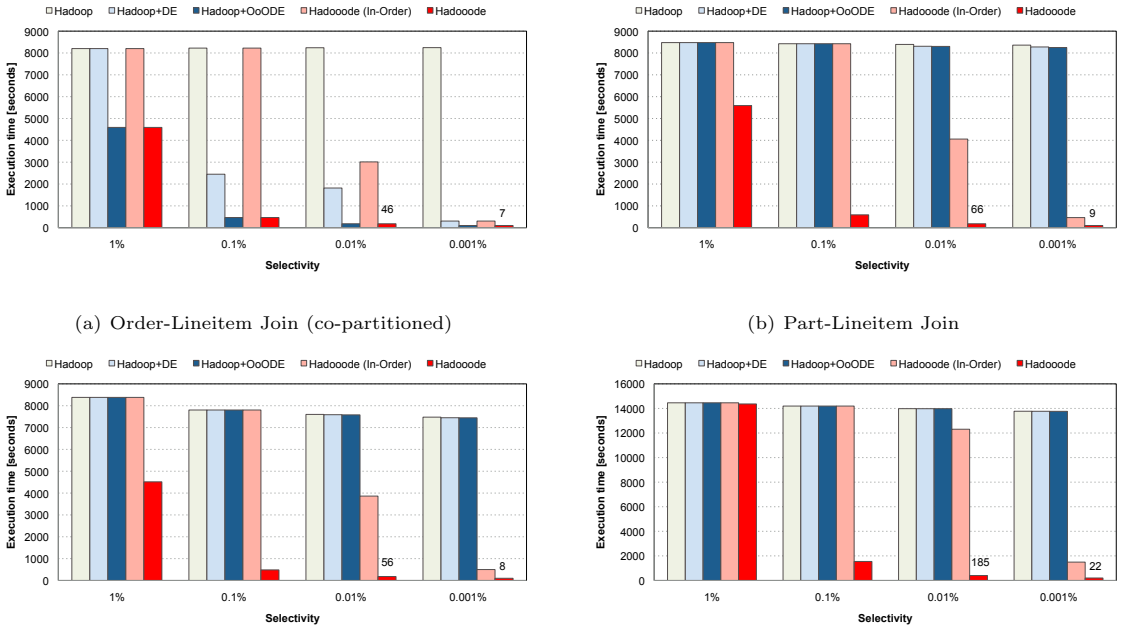
著者らが構築した実験システムを図 2 に示し，その諸元を表 1 に示す．24 台のデータ格納用の磁気ディスクドライブから，SAS ホストバスアダプタが備える RAID 機構によって，セグメントサイズ 512KB の RAID-6 編成（22D+2P）の論理ユニットを構成し，当該論理ユニット上に ext4 ファイルシステムボリュームを構築した．以下の実験では，当該ファイルボリュームに対象データセットを格納して，実験を行った．

評価実験においては，データベース分野における標準的なベンチマークである TPC-H のデータセットを用い，当該データセットに対して解析的なクエリを実行し，その実行に要する時間を計測した．

この際，著者らの提案する Hadooode の実装アプローチの有効性を検証するために，ネイティブの Hadoop 実装と比較することに加えて，Hadoop のストレージ層にアウトオブオーダ型データベースエンジンを用いる実装アプローチとの比較を行った．比較対象のシステムを以下に示す．

Hadoop： CDH4.5 に含まれる標準的な Hadoop (MRv1)．Hadoop においては，HDFS のブロックサイズを 128MB とし，ブロックのレプリケーションは行わない設定とした．また，Map 数，Reduce 数はコア数と同数の 16 とした．

Hadoop+DE： HadoopDB に相当する，ストレージにデー



(a) Order-Lineitem Join (co-partitioned)

(b) Part-Lineitem Join

(c) TPC-H Q3'

(d) TPC-H Q8'

図 3　128 ノードクラスタ構成における性能比較.
Fig. 3　Performance comparison in a 128-node cluster.

タベースエンジンを用いる並列データ処理系．HadoopDB は，Hadoop のストレージに PostgreSQL データベースエンジンを用いることにより，索引アクセス等のデータベース技法を Hadoop において活用することを目指すものである．論文 [14] や過去の HadoopDB の公開 ソースコード (注3) を参考に，著者らにおいて CDH4.5 をベースとした実装を行い，これを用い計測を行った．

Hadoop+OoODE： Hadoop+DE においてデータベースエンジンに代えてアウトオブオーダ型データベースエンジン（OoODE）を用いる並列データ処理系．Hadoop+OoODE においては，各計算機が管理する二次記憶に対する入出力のみが OoODE により非同期化される．

Hadooode： 著者らが CDH4.5 をベースに試作実装を行った Hadooode．Hadooode においては，並列データ処理系全体の入出力が非同期化されることとなり，例えば，他の計算機が管理する二次記憶へのネットワークを介した入出力も非同期的に行われる．

Hadooode（In-Order）： 動的なタスク分解を行わない Hadooode

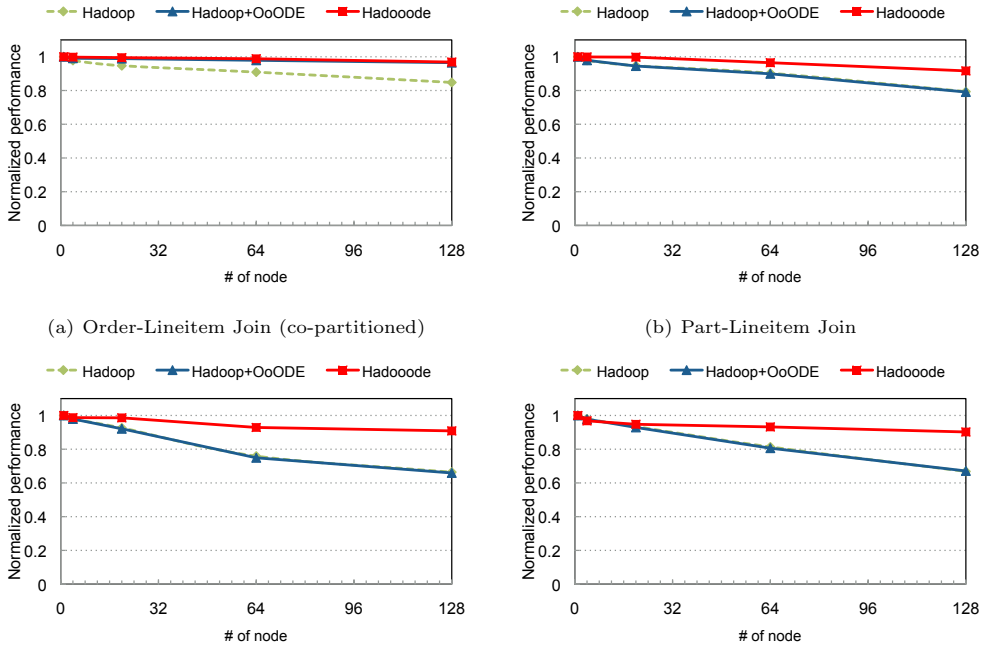
実験用の問合せは，二つの表の結合を行う 2 つのシンプルな結合問合せ（OL 結合，PL 結合）と，TPC-H 規定の問合せ Q3，Q8 をベースとした Q3' および Q8' からなる (注4)．当該問合せの選択率を 1%，0.1%，0.01%，0.001% と変化させて，Hive を介して実行した．

実験に際しては，SF=128K（合計約 128TB）のデータセットを作成して，これを用いた．Hadoop においては，当該データセットを HDFS 機構のラウンドロビン分割によって各計算機に分配し，残りの 4 つのケースにおいては，ハッシュ分割によって各計算機に分配するとともに，索引を構成した．この際のハッシュ分割は，各表の主キーを基に行った．索引としては，各表の主キーと外部キーに対する二次索引に加えて，Orders 表の o_orderdate と o_totalprice，Part 表の p_retailprice，Customer 表の c_acctbal に対して二次索引を構成した．

(注3)：https://hadoopdb.svn.sourceforge.net/svnroot/hadoopdb/
(注4)：集約処理を削除し，選択率の調整を行った．

4.2　性 能 評 価

Orders 表と Lineitem 表の結合問合せ（OL 結合）の実験結果を図 3(a) に示す．OL 結合は，注文の合計金額が指定した範囲内にある注文明細を抽出するクエリである．Hadoop においては，両表の全走査を伴う Reduce 側結合（並列ソートマージ結合）が実行され，クエリの選択率に大きく依らず長い実行時間を要している．また，Hadoop+DE においては，選択率 1%では結合方法として Reduce 側結合が選択され，Hadoop と同程度の実行時間を要したが，選択率 0.1%，0.01%でノード内に閉じたハッシュ結合が，選択率 0.001%でノード内に閉じたネステッドループ結合が実行され，Hadoop に対して性能向上が見られた．Hadoop+OoODE においては，すべての選択率においてノード内に閉じたネステッドループ結合が実行され，アウトオブオーダ型実行の効果により，Hadoop+DE に対して大幅は高速化が見られた．Hadooode（In-Order）は選択率 1%，0.1%においては，結合方法として Reduce 側結合が選択され，Hadoop と同程度の実行時間を要したが，選択率 0.01%，選択率 0.001%においてはネステッドループ結合が実行され，Hadoop に対して性能向上が見られた．これに対して，Hadooode においては，すべての選択率においてネステッドループ結合が選択され，アウトオブオーダ型実行の効果により，他の Hadoop 処



(a) Order-Lineitem Join (co-partitioned)

(b) Part-Lineitem Join

(c) TPC-H Q3'

(d) TPC-H Q8'

図 4　128 ノードクラスタ構成におけるスケーラビリティの比較.
Fig. 4　Comparison of scalability in a 128-node cluster.

理系と同等もしくはそれ以上の性能が得られた．一方で，この高速化率は Hadoop+OoODE のそれと同程度であったことから，結合表がともに結合キーでパーティショニングされている当該問合せにおいては，計算機間でネットワークを介した入出力は行われないため，Hadoop+OoODE の実装アプローチに対する Hadooode の実装アプローチが持つ優位性は確認されなかった．Hadooode は，選択率 0.01%の評価実験において，Hadoop に対して約 178 倍，Hadoop+DE に対して約 39 倍，Hadooode（In-Order）に対して約 65 倍の性能向上を達成している．

次に，Part 表と Lineitem 表の結合問合せ（PL 結合）の実験結果を図 3(b) に示す．PL 結合は，希望小売価格が指定した範囲にある部品の注文明細を抽出する問合せである．Hadoop においては，表の分割構成に変化はないため，OL 結合と同様の結果が見られた．Hadoop+DE においては，OL 結合と異なり，結合処理はノード内に閉じないため，すべての選択率で Reduce 側結合が実行され，Hadoop と同等の性能になっていることがわかる．Hadoop+OoODE は Hadoop+DE と同様に，すべての選択率で Reduce 側結合が実行され，Hadoop と同等の性能になっていることがわかる．Hadooode（In-Order）は選択率 1%，0.1%においては，結合方法として Reduce 側結合が選択され，Hadoop と同程度の実行時間を要したが，選択率 0.01%，選択率 0.001%においては，ネステッドループ結合が実行され，Hadoop に対して性能向上が見られた．これに対して，Hadooode はすべての選択率でネステッドループ結合が選択され，アウトオブオーダ型実行の効果により，他の Hadoop 処理系と比べて高い性能が得られた．即ち，結合表がともに結合キーでハッシュ分割されていない当該問合せにおいては，計算機間でネットワークを介した入出力が行われるため，Hadoop+OoODE の実装アプローチに対して Hadooode の実装アプローチが高い優位性を持つことが確認された．選択率 0.01%の評価実験において，Hadoop に対して約 127 倍，Hadoop+DE および Hadoop+OoODE に対して約 125 倍，Hadooode（In-Order）に対して約 61 倍の性能向上を達成している．

一般に，並列データベースシステム等においては，ノード間のデータ交換がなるべく生じないように予めデータを分配しておくことが多いが，意志決定支援システム等においては，多様なクエリが発行され、これを事前に見定めることは容易ではない．即ち，結合処理において計算機の中で常に演算が閉じるように事前に計算機間でデータの分配を行うことは一般には困難であり，通常は計算機間でネットワークを介して入出力を行わざるを得ない場合がある．Hadoop+OoODE においては，各計算機が管理する二次記憶に対する入出力は非同期的に行われるものの，他の計算機が管理する二次記憶に対するネットワークを介した入出力は Hadoop により同期的に行われる．即ち，並列データ処理においてネットワークを介する入出力の割合が全体の入出力に対して高い場合においては，Hadoop+OoODE による当該並列データ処理の著しい高速化は望めない．対して，Hadooode においては，並列データ処理系全体の入出力が非同期化され，即ち，入出力が各計算機が管理する二次記憶に対してであるか他の計算機が管理する二次記憶に対してであるかに関係なく，全ての入出力が非同期的に行われる．ゆえに，Hadooode は，当該並列データ処理においてもアウトオブオーダ型実行による著しい高速化を実現し，Hadoop+OoODE に対して高い優位性を有する．

並列データ処理において 3 段以上の結合を行う場合，通

常，ネットワークを介した入出力を行わざるを得ないため，Q3’ および Q8’ の実験では，図 3(c)，図 3(d) に示すように，Hadooode におけるアウトオブオーダ型実行方式の効果が高くなり，他の Hadoop 処理系に対して大幅な性能向上が確認された．Hadooode は，Q3’ の選択率 0.01%の評価実験において，Hadoop に対して約 135 倍，Hadoop+DE および Hadoop+OoODE に対して約 134 倍，Hadooode（In-Order）に対して約 68 倍の性能向上を達成し，Q8’ の選択率 0.01%の評価実験においては，Hadoop，Hadoop+DE，Hadoop+OoODE に対して約 76 倍，Hadooode（In-Order）に対してそれぞれ約 69 倍の性能向上を達成した．

4.3 ノードスケーラビリティの検証

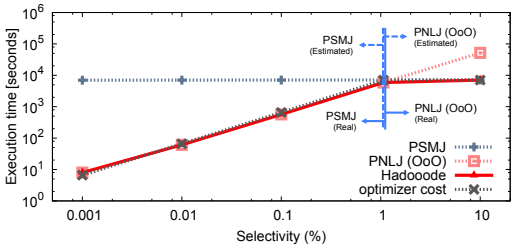
本節では，更に，ノード数を変化させた場合の性能の変化を検証する．この際，1 ノードあたりのデータサイズは一定とした．即ち，TPC-H データセットにおいて，1 ノードあたりのデータ量を約 1TB とした．実験結果は，1 ノードでの実行時間を測定対象のノード数における実行時間で割った正規化した相対性能を示す．

図 4 に，各問合せにおける選択率 0.01%のときのノードスケーラビリティを示す．いずれの問合せにおいても，Hadooode は，他の Hadoop 処理系に対して高いスケーラビリティが得られており，128 ノードで約 90%以上のスケーラビリティを達成している．Hadoop は，128 ノードで約 66-85%程度のスケーラビリティとなっていることがわかる．Hadoop+OoODE は，結合処理における入出力がノード内に閉じる場合（図 4(b)）は，99%程度のスケーラビリティが得られているものの，結合処理における入出力がノードをまたがる場合は，Hadoop と同程度のスケーラビリティとなっていることがわかる．このように，Hadooode における並列データ処理のアウトオブオーダ型実行は，Hadoop のスケーラビリティを阻害することなく，処理効率の向上を達成することができる．

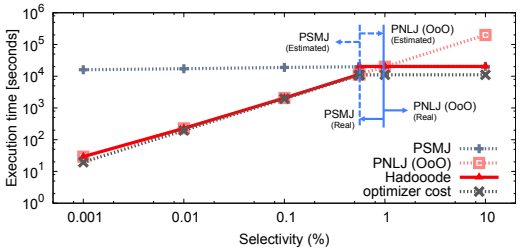
4.4 問合せ最適化方式の評価

現状の Hadooode は，並列結合アルゴズムとして，並列ソートマージ結合（PSMJ）と並列ネステッドループ結合（PNLJ）を実装している．通常，Hadooode においては，選択率中程度（1%から 0.1%）の領域においては，アウトオブオーダ型のネステッドループ結合を実行し，選択率が中程度より高い領域では，インオーダ型のソートマージ結合を実行するのが最良であると考えられる．本節では，2.2 節で提案した問合せ最適化方式がこれらのアルゴリズムならびに実行方式を適切に選択可能であるかを検証する．

図 5(a) に 128 ノードクラスタ構成における PL 結合の実験結果を示す．PSMJ は，アルゴリズムを並列ソートマージ結合に固定した場合の実験結果を示しており，PNLJ（OoO）は，アルゴリズムをアウトオブオーダ型の並列ネステッドループ結合に固定した場合の実験結果を示している．Hadoode においては，PSMJ と PNLJ（OoO）の交差ポイントで適切に実行プランを変更し，実験を行ったすべての選択率の範囲で，最良のアルゴリズムならびに実行方式が選択されていることがわかる．また，問合せ最適化器が算出する見積り時間（optimizer cost）



(a) PL-Join



(b) TPC-H Q8’

図 5 問合せ最適化方式の評価．
Fig. 5 Evaluation of query optimization.

においても，実測値とほぼ一致していることがわかる．

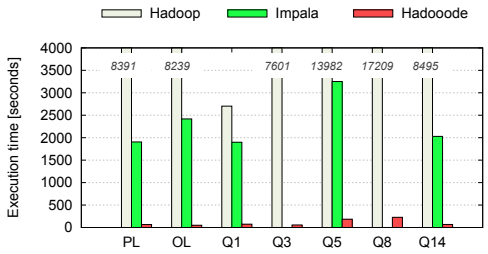
次に，図 5(b) に TPC-H Q8’ における実験結果を示す．PL 結合と同様に，Hadooode は適切にアルゴリズムおよび実行方式を選択し，また，見積り時間においても，実測値と概ね一致していることがわかる．しかしながら，PSMJ と PNLJ（OoO）の交差ポイントと，Hadooode における実行プランの切り替えポイントに若干のズレが確認できる．Hadooode の PSMJ においては，内部で Hadoop の実行エンジンを用いているが，Hadoop による Q8’ の実行は，図 4(d) に示すように，1 ノードでの性能と比較して 128 ノードでは約 35%の性能低下が起きており，問合せ最適化器が当該性能特性を正確に見積もれていないことが原因であると考えられる．

4.5 Impala との性能比較

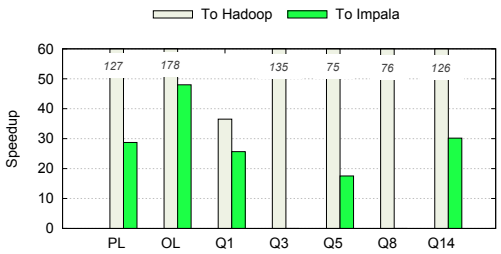
本節では，Hadoop における高速並列 SQL 処理系の一つである Impala [5] との性能比較を行う．Impala は並列データベースに類似したデータ処理方法の採用し，主記憶を用いた並列ハッシュ結合を行う点に特徴を有する．本実験では，Impala1.2.5 を用いた．当該バージョンにおいては，すべての結合処理に対してデータの全走査を基本とした並列ハッシュ結合を用い，この際，両表が再分配されるか，もしくは片表のみがブロードキャストされる方法がとられる．実験では，128 ノードクラスタ構成において TPC-H データセット（約 128TB）を用いた．

図 6(a) に PL 結合，OL 結合，ならびに TPC-H 規定問合せのうち 5 つの問合せ (注5) を用いた性能比較結果を示し，図 6(b) に当該実験における Impala に対する Hadooode の高速化率を示す．また，参考値として，Hadoop の実験結果も合わせて示す．この際，問合せの選択率は 0.01%となるように調整した．Hadooode は，すべての問い合わせにおいて，Impala と比較

(注5)：集約処理を削除し，選択率の調整を行った．



(a) Execution time.



(b) Speed-up

図 6　PL 結合，OL 結合，TPC-H 規定クエリにおける Impala との性能比較.

Fig. 6　Performance comparison between Impala and Hadooode in PL-join, OL-join and TPC-H queries.

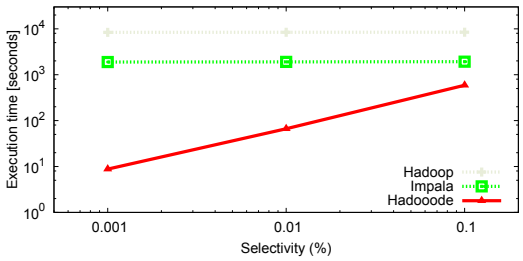


図 7　PL 結合における Impala との性能比較.

Fig. 7　Performance comparison between Impala and Hadooode in PL-join.

して高い処理性能が確認されており, 約 18 倍-48 倍の高速化を達成している．尚，Impala による Q3', Q8' の実行のおいては，結合処理におけるハッシュ表のサイズが実メモリサイズを超えたため，実行を完了することができなかった．

Impala と Hadooode の関係をより詳細に見るべく，図 7 に PL 結合における Impala と Hadooode の性能比較結果を示す．X 軸は選択率を，Y 軸は問合せ実行時間を表しており，いずれも対数軸となっていることに注意されたい．Impala は全走査の入出力時間に性能が律速され，選択率による実行時間への影響は比較的小さい．これに対し Hadooode においては，並列ネステッドループ結合が選択されたため，選択率によって性能が異なるものの，実験を行った選択率の範囲においては，Impala に対して大幅な高速化を達成していることがわかる．

5.　実データを用いた評価実験

著者らの研究グループでは，厚生労働科学研究の下，医療経済研究機構の協力を得て，Hadooode をベースとする，医療保険レセプト (注6) データの分析基盤を構築しており，当該基盤は医療保険分野の研究のための基本データセット作成等に大いに資している．本論文においては，著者らが，平成 22 年度の医療保険レセプトデータを使用し，前述の 128 ノードからなるクラスタを用いて行った試験結果を示す．試験に使用した問合せは以下の通りである．

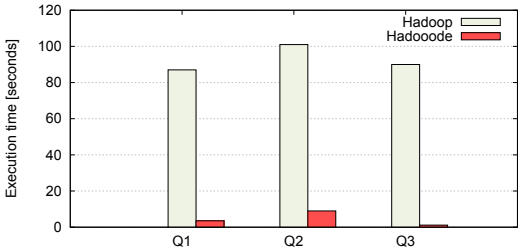


図 8　医療保険レセプトデータを用いた性能比較.

Fig. 8　Performance comparison for medical receipt data.

Q1：胃がんにおける都道府県ごとの患者数の対人口比率

Q2：脳梗塞における都道府県ごとの平均診療費

Q3：くも膜下出血における都道府県ごとの平均診療日数

なお，レセプトデータは入れ子データ構造となっているため，MapReduce 型の手続きインターフェースを用いて問合せを実行した．

図 8 に Hadoop と Hadooode の比較結果を示す．いずれの問合せにおいても，Hadooode は Hadoop に対して高い性能を示しており，最大 80 倍の高速化を達成している．このように，Hadooode が，標準的なベンチマークでの試験結果に加えて、実際のデータセットにおいても著しい高速性を備えていることが分かる．

6.　関 連 研 究

Hadoop におけるデータ処理の効率を向上させる研究は近年盛んに行われている．データの全走査を基本とする Hadoop に対して，Hadoop++ [16]，HadoopDB [14] では，索引を用いたファイル中のレコードアクセス手法，ならびに複数段の結合処理を一つの Map フェーズで実行するためのデータの配置手法を提案している．これらの研究は，既存の Hadoop における並列データ処理の効率化を図るものであり，本論文とは目的を同じとするものの，本論文は Hadoop を始めとする並列データ処理に対するアウトオブオーダ型実行方式を提案し，その有効性を明らかにするものであり，これらの研究とは異なる．

並列データベースにおいても，問合せ処理における並列度の向上を目指した研究は行われてきている．多くは，問合せ処理におけるパーティション並列性とパイプライン並列性 [17] を活用し，問合せ処理を並列に実行するものである．また，問合せ処理における演算や入出力を実行時に部分的に並列化するという取り組みも行われている．Cieslewicz ら [18] は，コンパイラ

(注6)：医療保険レセプトは，患者が受けた診療について，医療機関が保険者に請求する医療費の明細書を意味する．2000 年頃から電子化が開始され，2013 年においては約 9 割以上が電子化されている．

のループ展開を活用した，ハードウェアマルチスレッドによる演算の並列化方法を議論している．Roh ら [19] は，SSD のハードウェアレベルの並列性を活用する，B+木の並列探索方法を提案している．

これらの研究では，問合せ処理におけるオペレータもしくは特定の手続きを，事前に決定した並列度で実行し，問合せ処理の効率化を狙うものである．対して，本論文は，並列データ処理（並列問合せ処理）全体の入出力ならびに演算を実行論理が許す限りに高多重並列に実行すべく，並列データ処理の基本的な実行方式そのものを見直し，入出力の非同期化方式を提案するものである．

また，本論文で提案する問合せ最適化方式は，多くの関係データベースシステムで用いられている System-R 最適化方式 [15] に類するが，当該方式は単一計算機におけるインオーダ型のデータ処理を対象としており，複数の計算機から構成されるクラスタシステムにおけるアウトオブオーダ型並列データ処理を対象とする提案方式とは異なる．

7. おわりに

本論文は，並列データ処理におけるアウトオブオーダ型実行方式，およびアウトオブオーダ型並列データ処理系 Hadooode を提案した．Hadooode では，各々の計算機において，並列データ処理の実行時に動的にタスク分解を行い，分解されたタスクにおいて二次記憶ならびにネットワークを介した他の計算機の二次記憶への入出力を行い，入出力の完了に伴い関連する演算を駆動する．並列データ処理全体の入出力が非同期化される点に新たな特徴があり，特にデータセット空間の一部のデータを対象とするデータ処理において，入出力スループットが飛躍的に向上し，データ処理の大幅な高速化が実現される．128 ノードのクラスタマシン（合計 2048CPU コア，3072 ディスクドライブ）により Hadooode の性能評価実験を行い，実験の結果，選択率 0.01%の問合せにおいて，Hadoop に対して最大 178 倍，Hadoop において二次記憶管理にデータベースシステムを活用する処理系に対して最大約 134 倍，Hadoop において二次記憶管理にアウトオブオーダ型のデータベースエンジンを活用する処理系に対して最大約 134 倍の性能向上が確認された．また，ノードスケーラビリティの評価実験を行い，他の Hadoop 処理系と比して高いスケーラビリティがあることが示された．さらに，並列データ処理のアウトオブオーダ型実行を考慮した最適化方式を提案し，当該最適化方式が適切に最適プランを導出できていることが確認された．加えて，Hadoop における高速 SQL 処理系である Impala との性能比較実験を行った結果，選択率 0.01%の問合せにおいて，最大 48 倍の性能向上が確認された．実データを用いた評価実験においても Hadooode の高い性能は顕著であり，Hadooode の高い有効性が確認された．今後は並列データ処理のアウトオブオーダ型実行を考慮した負荷分散機構ならびに耐障害機構の検討を進めていきたい．

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．医療保険レセプトを用いた試験については，厚生労働科学研究補助金政策科学総合研究事業「汎用性の高いレセプト基本データセット作成に関する研究」の下，医療経済研究機構の満武巨裕先生に丁寧なご指導を頂いた．

文　献

[1] C. Kozyrakis, A. Kansal, S. Sankar, K. Vaid. “Server Engineering Insights for Large-Scale Online Services”, IEEE Micro, Volume 30, pp.8–19, 2010.
[2] E. Eleftheriou, R. Haas, J. Jelitto, M. Lantz and H. Pozidis. “Trends in Storage Technologies”, IEEE TCDE, 2010.
[3] D. A. Patterson. “Latency lags bandwith”, Commun. ACM 47, Issue 10, pp.71–75, 2004.
[4] Hadoop, http://hadoop.apache.org/
[5] “Cloudera Impala: Open Source, Real-time Query for Hadoop”, http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html
[6] “Pivotal HD: The World’s Most Powerful Distribution of Apache Hadoop”, http://www.greenplum.com/products/pivotal-hd
[7] “The Stinger Initiative: Making Apache Hive 100 Times Faster”, http://hortonworks.com/blog/100x-faster-hive/
[8] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis, “Dremel: interactive analysis of web-scale datasets” Proc. VLDB, pp.330-339, 2011.
[9] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica. “Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing”, Proc. NSDI, 2012.
[10] 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.
[11] 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.
[12] 山田浩之，合田和生，喜連川優. “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”, 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.
[13] 山田浩之，合田和生，喜連川優. “Hadoop をはじめとする並列データ処理系へのアウトオブオーダ型実行方式の適用とその有効性の検証”, 電子情報通信学会論文誌, Vol.J97-D，No.4, 2014. (to appear)
[14] 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.
[15] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, T. G. Price. “Access Path Selection in a Relational Database Management System”, Proc. SIGMOD, pp.23–34, 1979.
[16] 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.
[17] D. DeWitt, J. Gray, “Parallel database systems: the future of high performance database systems”, Commun. ACM 35, 6, pp.85–98, 1992.
[18] J. Cieslewicz, J. Berry, B. Hendrickson, K. A. Ross, “Realizing parallelism in database operations: insights from a massively multithreaded architecture”, Proc. DaMoN, 2006.
[19] H. Roh, S. Park, S. Kim, M. Shin, S. Lee, “B+-tree index optimization by exploiting internal parallelism of flash-based solid state drives”, Proc. VLDB, pp.286–297, 2011.