THE IEICE TRANSACTIONS ON INFORMATION AND SYSTEMS (JAPANESE EDITION)

電子情報通信学会 論文誌 D 情報・システム

VOL. J98-D NO. 5
MAY 2015

本PDFの扱いは、電子情報通信学会著作権規定に従うこと。
なお、本PDFは研究教育目的（非営利）に限り、著者が第三者に直接配布することができる。著者以外からの配布は禁じられている。

情報・システムソサイエティ

一般社団法人 電子情報通信学会
THE INFORMATION AND SYSTEMS SOCIETY
THE INSTITUTE OF ELECTRONICS, INFORMATION AND COMMUNICATION ENGINEERS

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*

山田　浩之[†a)]　　合田　和生[†b)]　　喜連川　優[†,††c)]

Performance Evaluation of Out-of-Order Parallel Data Processing System in 128-Node Storage Intensive Cluster*

Hiroyuki YAMADA[†a)], Kazuo GODA[†b)], and Masaru KITSUREGAWA[†,††c)]

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速並列データ処理系を開発してきた．高速アウトオブオーダ型並列データ処理系は，並列データ処理を行う各々の計算機において，入出力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並びにネットワークで接続され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．すなわち，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では，系全体の性能が入出力によって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並列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てきた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，24台の磁気ディスクドライブを備えた計算機からなる128ノード構成のクラスタシステム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，非同期入出力，並列データ処理，並列問合せ処理，ビッグデータ解析，Hadoop

1. まえがき

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低減は年率5%以下に留まっている[2]．同様に，単一ハードウェアコンポーネントにおけるレイテンシ低減の停滞は，メモリモジュール並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]．これらの単一のハードウェアコンポーネント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コンポーネントを効率的に活用することが重要となると考えられる．企業においてはハードウェアを高密度に集積した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，近年，先進的な応用が期待されつつあるビッグデータ解析においては，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]〜[10]，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．

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速並列データ処理系を開発してきた．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[11], [12]を拡張し，並列データ処理を行う各々の計

† 東京大学生産技術研究所，東京都
Institute of Industrial Science, The University of Tokyo, 4–6–1 Komaba, Meguro-ku, Tokyo, 153–8505 Japan
†† 国立情報学研究所，東京都
National Institute of Informatics, 2–1–2 Hitotsubashi, Chiyoda-ku, Tokyo, 101–8430 Japan
a) E-mail: hiroyuki@tkl.iis.u-tokyo.ac.jp
b) E-mail: kgoda@tkl.iis.u-tokyo.ac.jp
c) E-mail: kitsure@tkl.iis.u-tokyo.ac.jp
* 本論文はデータ工学研究専門委員会推薦論文である．
DOI:10.14923/transinfj.2014DET0001

 ©一般社団法人電子情報通信学会 2015

算機において，入出力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並びにネットワークで接続され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当該入出力に関連する演算を実行する．すなわち，並列データ処理における入出力を非同期化する．データインテンシブな並列データ処理においては，全体の性能が入出力に律速されることが多く，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並列データ処理において，飛躍的な高速化が期待される．

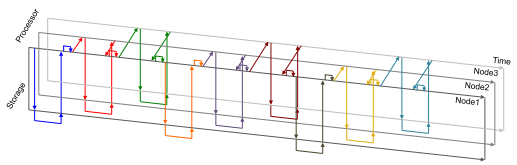
本論文では，著者らが試作を行ってきた Hadoop をベースとするアウトオブオーダ型並列データ処理系 Hadooode の構成法を示すとともに，24 台の磁気ディスクドライブを備えた計算機からなる 128 ノード構成のクラスタシステムにおいて，人工データセット及び実データを用いた当該試作による性能評価実験を示し，その有効性を明らかにする．著者らは文献 [13], [14] において，Hadooode を提案し，20 ノード構成のクラスタシステムにおける人工データセットを用いた性能評価実験により Hadooode の有効性を示してきたが，本論文では，128 ノードの大規模クラスタ構成における性能評価実験を行い，加えて，実データを用いた評価実験を行う．また，文献 [15] においては，128 ノード構成のクラスタシステムにおける TPC-H データセットを用いた性能評価結果のみを示してきたが，本論文では，MapReduce 等の性能評価で広く利用されている解析タスクベンチマークデータセットを用いた性能評価を加え，Hadooode の有効性を明らかにする．

本論文の構成は以下のとおりである．**2.** では，並列データ処理におけるアウトオブオーダ型実行方式を示し，**3.** 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 Hadooode の実装を示す．**4.** では，128 台の計算機から構成されるクラスタシステムにおいて，人工データセット及び実データを用いた Hadooode の性能評価実験を示し，その有効性を明らかにする．**5.** で関連研究を述べ，**6.** で本論文をまとめる．

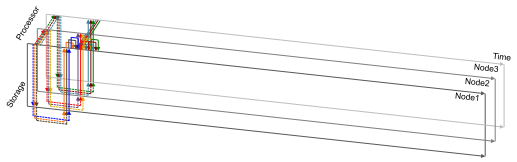
2. 並列データ処理のアウトオブオーダ型実行方式

並列データ処理は，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，あらかじめ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)，並列データ処理のジョブが与えられると，当該ジョブを分割若しくは複製して計算機に割り当て，各々の計算機において割り当てられたジョブを実行する [16]．各計算機においては，割り当てられたジョブにおいて，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いて，単一ノードを起点としたときの入出力と演算におけ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」と称する実行方式を考える．当該実行方式では，各々の計算機におけるジョブの実行時に，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，タスク分解によって，入出力が非同期的に発行されることとなり，入出力の完了に伴って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



(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.

(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．

る二次記憶に対してのみならず，他の計算機が管理する二次記憶に対して行う必要があり，この場合，計算機間のネットワークを介した通信（ネットワーク入出力）も同様にタスク上で実行される．すなわち，アウトオブオーダ型実行方式に基づいた並列データ処理においては，系全体にわたって入出力が非同期的に行われる．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，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当該タスクから発行される入出力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，近年，データインテンシブ処理用途のクラスタシステムで広く用いられている 24 ディスクドライブ程度を 2U きょう体に具備す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きている．アウトオブオーダ型の並列データ処理実行方式は，当該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．

なお，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，

（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

の二つの方法に大別できると考えられる．

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用いて，アウトオブオーダ型並列データ処理系の実現を試みるものと言え，例えば，既存研究である HadoopDB [17] 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アウトオブオーダ型実行方式によって一定程度の高速化が期待されるものの，その効果は各計算機が管理する二次記憶に対する入出力のみに留まる．すなわち，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．

一方，本論文は，より根源的と言えるアプローチ（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，並列データ処理の入出力全体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，若しくは，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，比較実験により，アプローチ（2）の優位性を明らかにする．

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 の実装を紹介する．Hadooode は，主にジョブ実行器，ストレージ管理器，ジョブ受付器，問合せ計画器から構成される．

ジョブ実行器は，ジョブにおける並列データ処理をアウトオブオーダ型で実行するものである．例えば，通常の Hadoop のジョブ実行器における Mapper では，入出力を管理する RecordReader 機構が備える Next() なる手続きを呼び出すことにより，計算機自身の二次記憶若しくはネットワークで接続された他の計算機の二次記憶からデータを読出し，読み出したデータを用いて Mapper の Map() 手続きを実行し，このような手順を対象データの全体を読み出すまで繰り返す．Next() の実行と Map() の実行は，逐次的に繰り返され，すなわち，入出力は同期的に行われる．これに対して，Hadooode のジョブ実行器は，Next() 手続きを呼び出す都度にタスクを生成し，当該タスク上において計算機自身の二次記憶若しくはネットワークで接続された他の計算機の二次記憶からデータを読出し，読み出したデータを用いて Map() 手続きの実行を行う．すなわち，Next() の段階で，入出力の非同期化を実現する．また，Next() 手続きの内部においては，多くの

場合，複数のレコード入出力や複数のネットワーク入出力を行うが，Hadooode のジョブ実行器は，当該入出力処理においても同様にタスクを生成し，非同期に実行する．当該ジョブ実行器は，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．ジョブ実行器の更なる詳細については，文献 [14] を参照されたい．

一方，ストレージ管理器は，レコードの集合からなるファイルを管理し，ジョブ実行器からの要求に応じてレコードの取得を行うものである．ファイルにおいては，HDFS によるラウンドラビン分割，若しくは，ハッシュ分割により複数の計算機に分配して管理する．加えて，当該ファイルに対して索引を構成する機構を有し，必要に応じて索引を管理する．ファイルに対するアクセスパスとしては，全走査または索引を用いた選択的アクセスを行う．

また，ジョブ受付器は，MapReduce ジョブまたは Hadoop の SQL 処理系である Hive からのジョブ実行プランを受け取り，それをジョブ実行器へと受け渡すものである．すなわち，Hadooode においては，Map, Reduce の手続きからなるジョブのみならず，Hive からの SQL 問合せにおいても，アウトオブオーダ型で実行することが可能である．

問合せ計画器は，SQL からアウトオブオーダ型のジョブ実行プランを導出するものである．この際，当該計画器は Left-Deep 木を探索対象とし，問合せの選択率等に基づいたコストベースの最適化を行う．当該計画器により，例えば，ファイルの走査方法（全走査または索引走査），結合方法，及び結合順序等が決定される．Hadooode においては，当該機構を Hive の問合せ計画器に実装した(注2)．問合せ最適化については，本論文ではここまでの説明に留め，詳細は別稿に譲りたい．

4. 128 ノードクラスタを用いた評価実験

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．

(注2)：Hive の問合せ計画器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 × 2 (2p/16c)
Memory	64GB
HDD (OS)	2 × 15,000rpm 300GB SAS HDDs
HDD (Data)	24 × 10,000rpm 900GB SAS HDDs
RAID Controller	Dell PERC H710P
OS	CentOS 5.8 (Linux kernel 2.6.18)

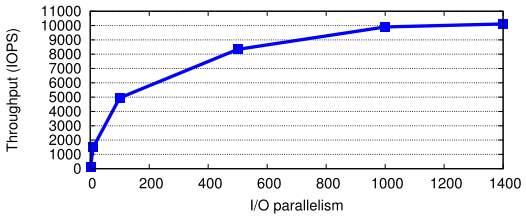


図 3　論理ユニットのランダム読み出し性能
Fig. 3　Random read performance of the logical unit.

4.2 基本性能試験

Hadooode の処理性能を測定をする前に，予備的な試験として，実験環境における入出力性能を確認した．

はじめに，24 台のデータ格納用の磁気ディスクドライブから構成される論理ユニットの性能を測定した．I/O サイズは 4KB とし，当該論理ユニットの約 1TB の領域をローデバイスとしてアクセスした．図 3 に，入出力の多重度を増やした場合のランダム読み出し性

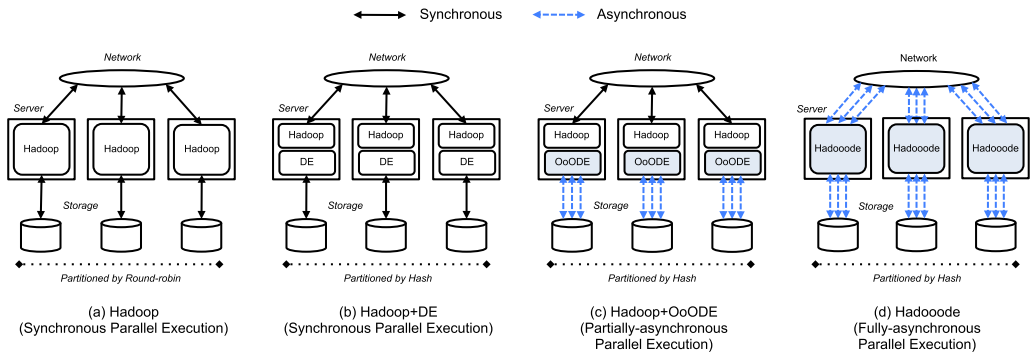


図 5　実験に用いた Hadoop ベースの並列データ処理系
Fig. 5　Hadoop-based parallel data processing system used in experiments.

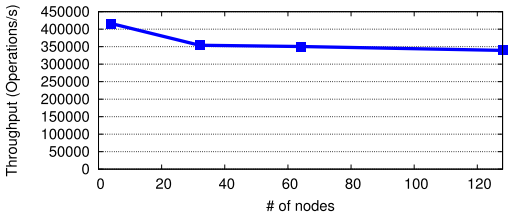


図 4　N 対 N 通信におけるノード間の送受信回数
Fig. 4　The number of operations between two nodes in N-to-N communication.

能を示す．多重度が 1，すなわち，同期入出力に相当する場合は，130IOPS 程度のスループットが得られたのに対して，多重度を向上することにより，スループットの向上が確認でき，多重度が 1400 のときに最大 10000IOPS 程度のスループットが得られた．

次に，ネットワークスイッチを介した通信性能を測定した．転送データを 1 バイトとし，TCP/IP を用いて当該データの送受信を行った．図 4 に，同時に通信を行うノード数 N を変化させたときの，N 対 N 通信における秒間あたりのノード間の平均送受信回数を示す．N の増加に従い，若干の性能低下は見られるが，秒間 30 万回程度の送受信がノード間で実行可能であることがわかる．

4.3　評価システム

著者らの提案する Hadooode の有効性を検証するために，Hadoop のストレージ層にデータベースエンジンを用いるシステム（Hadoop+DE）と，当該データベースエンジンにアウトオブオーダ型データベースエンジンを用いるシステム（Hadoop+OoODE）との比較を行った．また，ジョブ実行器において動的なタスク分解を行わない Hadooode（Hadooode（In-Order））との比較を行った．更に，参考としてネイティブの Hadoop との比較も行った．比較対象のシステムを以下にまとめる．

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，Mapper 数，Reducer 数はコア数と同数の 16 とした．なお，Hadoop は，ファイルに対してラウンドロビン分割を行い，また，分割ファイルに対して全走査を行うことを前提としており，その他のシステムとは実験の条件が異なるため，実験結果は参考として示すものである．図 5 (a) に当該システムを模式的に示す．

Hadoop+DE：　HadoopDB に相当する，ストレージにデー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，ハッシュ分割や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 [17] 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，これを用いた．図 5 (b) に当該システムを模式的に示す．

Hadoop+OoODE：　Hadoop+DE において通常の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 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，当該システムにおいては，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

(注3)：https://hadoopdb.svn.sourceforge.net/svnroot/hadoopdb/

る．図 5 (c) に当該システムを模式的に示す．

Hadooode： 著者らが CDH4.5 をベースに，本論文で提案する実装アプローチに基づき試作実装を行ったアウトオブオーダ型並列データ処理系．Hadooode においては，並列データ処理系全体にわたって入出力が非同期化される．図 5 (d) に当該システムを模式的に示す．

Hadooode（In-Order）： ジョブ実行器において動的なタスク分解を行わない Hadooode．

本実験を行うにあたっては，著者らは，**3.** に述べた問合せ計画器を実装し，当該計画器により生成された実行プランを用いて，問合せの性能評価を行った．また，比較のため，Hadoop を除く他のシステムにおいても，同様の問合せ計画器を実装し，同様に実験を実施した．各実験において選択された具体的な問合せの処理方式は，実験結果とともに後述する．

4.4 解析タスクベンチマークデータセットを用いた性能評価

性能評価においては，まず，MapReduce プログラムのベンチマークに広く用いられている解析タスクベンチマーク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ジョブを実行し，その実行に要する時間を計測した．

解析タスクベンチマークデータセットは，Web サーバのアクセスログを模擬したデータセットであり，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 Documents 表，Documents 表に対して付与されたメタデータを保持する Rankings 表からなる．このデータセットのスキーマ情報の詳細は文献 [18] を参照されたい．

実験に際しては，約 1 兆件の UserVisits 表レコード（約 128TB）並びに約 128 億件の Rankings 表レコード（約 6.4TB）を作成して，これを用いた(注4)．この際，Hadoop においては，当該データセットを HDFS 機構によるラウンドロビン分割によって各計算機に分配してファイルに格納した．対して，残りの四つのケースにおいては，論文 [18] のケースを参考に，当該データセットをハッシュ分割によって各計算機に分配してファイルに格納するとともに，索引を構成した．この際のハッシュ分割としては，UserVisits 表に関しては，比較のため，destURL をキーとする場合と sourceIP をキーとする場合の 2 通りのハッシュ分割を設けることとし，Rankings 表は pageURL をキーとしてハッシュ分割を行った．また，索引としては UserVisits 表の destURL 並びに adRevenue，Rankings 表の pageRank に対して構成した．

(注4)：Documents 表は本実験におけるクエリでは使用しないため，作成しなかった．

```
SELECT sourceIPAddr, destinationURL
FROM UserVisits
WHERE adRevenue BETWEEN X AND Y
```

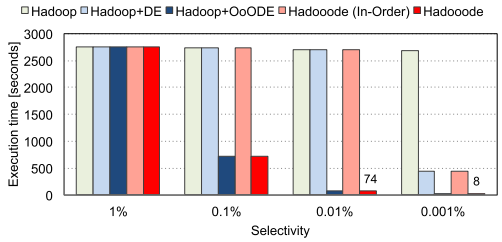
Listing. 1 Selection Job

本実験における当該データセットに対するジョブは，UserVisits 表の選択を行う Selection ジョブと，Rankings 表と UserVisits 表の結合を行う Join ジョブからなる．選択率ごとの性能特性を見るため，ジョブの選択率を 1%，0.1%，0.01%，0.001%と変化させて実験を実施した．

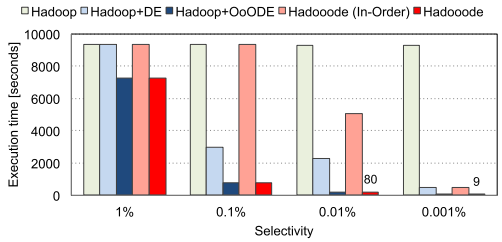
4.4.1 Selection ジョブ

Selection ジョブは，ウェブアクセスログにおいて広告収入が指定した範囲内にあるウェブページアクセスを抽出するためのジョブである．リスト 1 にはそのクエリを SQL を以って示すが，実験においては相当する MapReduce 型のデータ処理プログラムを構成して，実行した．

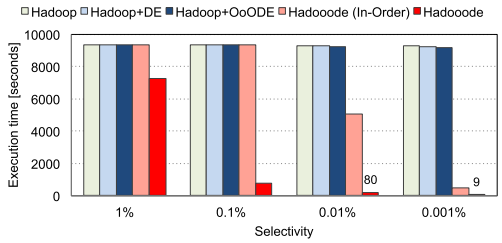
図 6 (a) に Selection ジョブの実験結果を示す．横軸はジョブの選択率を表し，縦軸は実行時間を表す．Hadoop では，表の全走査を行うため，全走査の入出力時間に性能が律速され，ジョブの選択率に大きくよらず長い実行時間を要していることがわかる．また，Hadoop+DE においては，選択率 1%から 0.01%では，アクセスパスとして全走査が選択され，Hadoop と同程度の実行時間を要したが，選択率 0.001%においては，索引を用いた選択処理が選択され，Hadoop に対して性能向上が見られた．Hadoop+OoODE においては，選択率 1%では，アクセスパスとして全走査が選択されたことから，Hadoop 及び Hadoop+DE と同程度の実行時間を要したが，選択率 0.1%から 0.001%においては，索引を用いた選択処理が選択され，アウトオブオーダ型実行の効果により，Hadoop+DE に対して，大幅な高速化が見られた．これらに対して，Hadooode（In-Order）並びに Hadooode においては，Hadoop+DE 並びに Hadoop+OoODE とそれぞれ同様の実行時間を要した．Hadooode は，選択率 0.01%の評価実験において，Hadoop+DE，Hadooode（In-Order）に対してそれぞれ約 36 倍の性能向上を達成し，Hadooode



(a) Selection-job



(b) Join-job (UserVisits is partitioned by destURL)



(c) Join-job (UserVisits is partitioned by sourceIP)

図 6　128 ノードクラスタ構成における解析タスクベンチマークデータセットを用いた性能比較
Fig. 6　Performance comparison with analytical task dataset in a 128-node cluster.

における索引アクセス機構並びにアウトオブオーダ型実行の高い有効性が確認された．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，Selection ジョブにおいては，入出力は各計算機内に留まるため，Hadoop+OoODE の実装アプローチに対する Hadooode の実装アプローチがもつ優位性は確認されなかった．なお，直接の比較は難しいが，Hadoop に対しては約 36 倍の性能向上が得られた．

4.4.2　Join ジョブ

Join ジョブは，ページランクが指定した範囲内にあるウェブページに対するアクセスのうち，特定の期間内のものを抽出するジョブである．リスト 2 の SQL で記載された Hive 問合せを実行し，この際，UserVisits 表については destURL をキーとしてハッシュ分割し

```
SELECT sourceIPAddr, destinationURL
FROM Rankings r JOIN UserVisits u ON r.pageURL = u.
     destinationURL
WHERE r.pageRank BETWEEN X AND Y AND u.visitDate BETWEEN
      "2000-01-15" AND "2000-01-22"
```

Listing. 2　Join Job

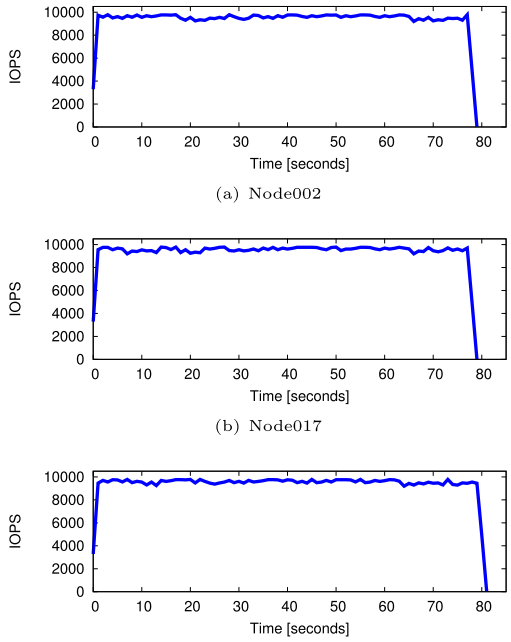
た場合と，sourceIP をキーとしてハッシュ分割した場合の二つのケースについて，測定を行った．

図 6 (b) に，UserVisits 表が destURL 属性によってハッシュ分割された構成における Join ジョブ（Join ジョブ 1）の実験結果を示す．横軸はジョブにおける最外表（Rankings 表）の選択率を表し，縦軸は実行時間を表す．Hadoop 以外においては，両表はともに結合キーで複数の計算機へ分割されている点に注意されたい．Hadoop では，両表の全走査を伴う Reduce 側結合（並列ソートマージ結合）が選択され，全走査の入出力時間に性能が律速されたため，問合せの選択率に大きくよ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選択され，Hadoop に対して性能向上が見られた．Hadoop+OoODE においては，全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選択され，アウトオブオーダ型実行の効果により，Hadoop+DE に対して大幅な高速化が見られた．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%においては並列ネステッドループ結合が選択され，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，全ての選択率において並列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 処理系と同等若しくはそれ以上の性能が得られた．一方で，Hadooode の Hadoop に対する高速化率は 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，Selection ジョブの場合と同様に，入出力は各計算機内に留まるため，Hadoop+OoODE の実装アプローチに対する Hadooode の実装アプローチがも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+DE に対して約 28 倍，Hadooode（In-Order）に対して約 63 倍の性能向上を達成した．なお，直接の比較は難しいが，Hadoop に対しては約 115 倍の性能向上が得られた．

次に，UserVisits 表が sourceIP 属性によってハッシュ分割された構成における Join ジョブ（Join ジョブ 2）の実験結果を図 6 (c) に示す．Hadoop 以外においては，Rankings 表は結合キー（pageURL）で分割されているが，UserVisits 表は結合キーにより分割されていない点に注意されたい．Hadoop では，表はラウンドロビンで分割されているため，結合処理の挙動としては Join ジョブ 1 と同様であり，また，UserVisits 表の全走査の入出力時間に性能が律速されたため，Join ジョブ 1 と同様の結果が見られた．Hadoop+DE では，Join ジョブ 1 と異なり，結合処理はノード内に閉じないため，全ての選択率において，実行時に表の再分割を行う Reduce 側結合が選択されたため，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，全ての選択率で Reduce 側結合が選択され，Hadoop 及び Hadoop+DE 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，並列ネステッドループ結合が選択され，Hadooode を除く他の Hadoop 処理系に対して性能向上が見られた．これに対して，Hadooode は全ての選択率で並列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．すなわち，結合表がともに結合キーで分割されていない当該問合せにおいては，入出力は各計算機内に留まらず，計算機間でネットワークを介して行われるため，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もつことが確認された．選択率 0.01%の評価実験において，Hadoop+DE 及び Hadoop+OoODE に対して約 115 倍，Hadooode（In-Order）に対して約 63 倍の性能向上を達成した．なお，参考としては，Hadoop に対しては約 116 倍の性能向上が得られた．

図 7 に Join ジョブ 2 における各ノードの IOPS トレースを示す．この際，トレース対象の 3 台は 128 台のサーバからランダムに選択した．この結果からわかるように，Hadooode における並列データ処理では，各ノードにおけるデータ処理は 10000IOPS 程度の I/O 性能を達成しており，図 3 に示す計算機の I/O 性能を十分に活用できていることがわかる．



(a) Node002

(b) Node017

(c) Node058

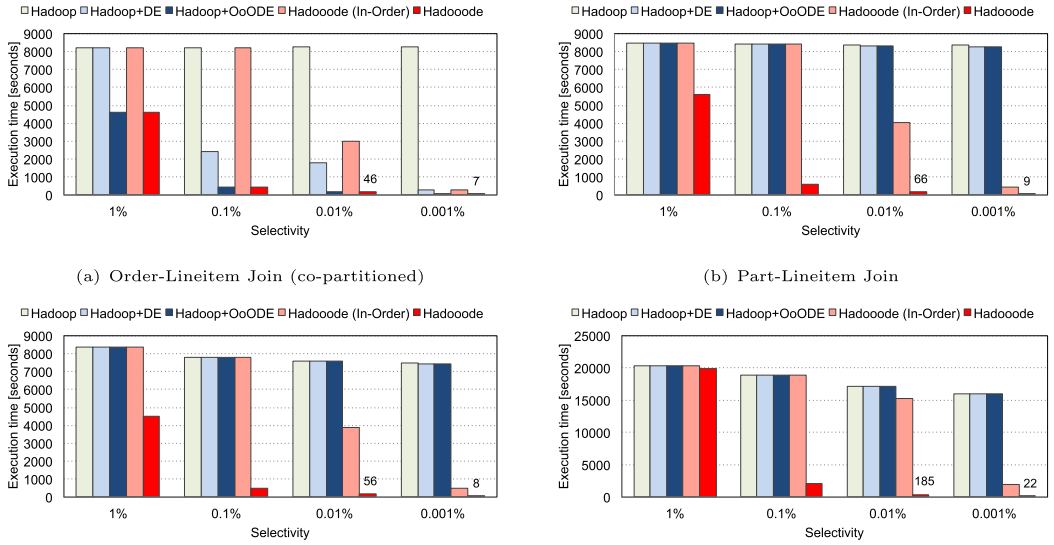
図 7　Join ジョブ 2 における I/O トレース
Fig. 7　I/O trace in Join-job2.

4.5　TPC-H データセットを用いた性能評価

次に，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．

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，残りの四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．索引においては，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．

実験用の問合せは，二つの表の結合を行う二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’ 及び Q8’ か



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

(b) Part-Lineitem Join

(c) TPC-H Q3'

(d) TPC-H Q8'

図 8　128 ノードクラスタ構成における TPC-H データセットを用いた性能比較
Fig. 8　Performance comparison with TPC-H dataset in a 128-node cluster.

```
SELECT l.l_orderkey, l.l_linenumber, o.o_totalprice
FROM orders o JOIN lineitem l ON o.o_orderkey = l.
     l_orderkey
WHERE o.o_totalprice BETWEEN X AND Y
```

Listing. 3　TPC-H OL-Join

```
SELECT l.l_orderkey, p.p_retailprice
FROM part p JOIN lineitem l ON p.p_partkey = l.l_partkey
WHERE p.p_retailprice BETWEEN X AND Y
```

Listing. 4　TPC-H PL-Join

らなる(注5)．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．

Orders 表と Lineitem 表の結合問合せ（OL 結合）の実験結果を図 8 (a) に示す．OL 結合は，リスト 3 に示すように，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問合せである．Hadoop 以外においては，両表はともに結合キーで複数の計算機へ分割されている点に注意されたい．当該問合せにおいては，結合処理の挙動としては，解析タスクベンチマークデータセットにおける Join ジョブ 1 と同様であり，また，アクセス対象のデータセットのサイズにも大きな差はないため，Join ジョブ 1 と同様の結果が得られた．すなわち，Hadooode においては，アウトオブオーダ型実行の効果により，全ての選択率において他の Hadoop 処理系と同等若しくはそれ以上の性能が得られたものの，Hadoop+OoODE の実装アプローチに対する Hadooode の実装アプローチがも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+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た．なお，Hadoop に対しては約 178 倍の性能向上が得られた．

次に，Part 表と Lineitem 表の結合問合せ（PL 結合）の実験結果を図 8 (b) に示す．PL 結合は，リスト 4 に示すように，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 以外においては，Part 表は結合キー（l_partkey）で分割されているが，Lineitem 表は l_orderkey により分割されている点に注意されたい．当該問合せにおいては，結合処理の挙動としては，解析タスクベンチマークデータセットにおける Join ジョブ 2 と同様であり，また，アクセス対象のデータセットのサイズにも大きな差はないため，Join ジョブ 2 と同様の結果が得られた．すなわち，Hadooode はアウトオブオーダ型実行の効果により，全ての選択率で他の Hadoop 処理系と比べて高い性能が得られ，Join ジョブ 2 と同様に，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もつこ

(注5)：集約処理を削除し，選択率の調整を行った．

とが確認された．選択率 0.01%の評価実験において，Hadoop+DE 及び 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た．なお，Hadoop に対しては約 127 倍の性能向上が得られた．

一般に，並列データベースシステム等においては，計算機の中で入出力と演算が閉じるようにあらかじめデータを分割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るため，適切なデータ分割軸を事前に見定めることは容易ではない．すなわち，結合処理においては，データの分割キーと結合キーが同一でない限りは，通常は計算機間でネットワークを介した入出力を行わざるを得ず，例えば，スタースキーマにおけるファクト表を始めとする多数の外部キーを有する表の結合においては，常に結合キーと分割キーを同一にすることは一般には困難である．同様に，3 表以上の多段結合においても，全ての結合処理を計算機内に閉じることは困難であると考えられる．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，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 Hadoop により同期的に行われる．すなわち，問合せがデータの分割構成にうまく適合しない限りは，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，アウトオブオーダ型実行による大幅な高速化は望めない．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，すなわち，問合せにおける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，系全体にわたって入出力が非同期的に行われるため，いかなるデータの分割構成においても，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．

Q3’ 及び Q8’ は 3 表以上の多段結合問合せであるため，前述のとおり，通常はネットワークを介した入出力を行わざるを得ず，図 8 (c)，図 8 (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 処理系に対する高速性が際立っていることがわかる．Hadooode は，Q3’ の選択率 0.01%の評価実験において，Hadoop+DE 及び 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+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．なお，Hadoop に対してはそれぞれの問合せにおいて約 135 倍，約 76 倍の性能向上が得られた．

4.6 ノードスケーラビリティの評価

本節では，更に，計算機の台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．すなわち，解析タスクベンチマークデータセット並びに 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．

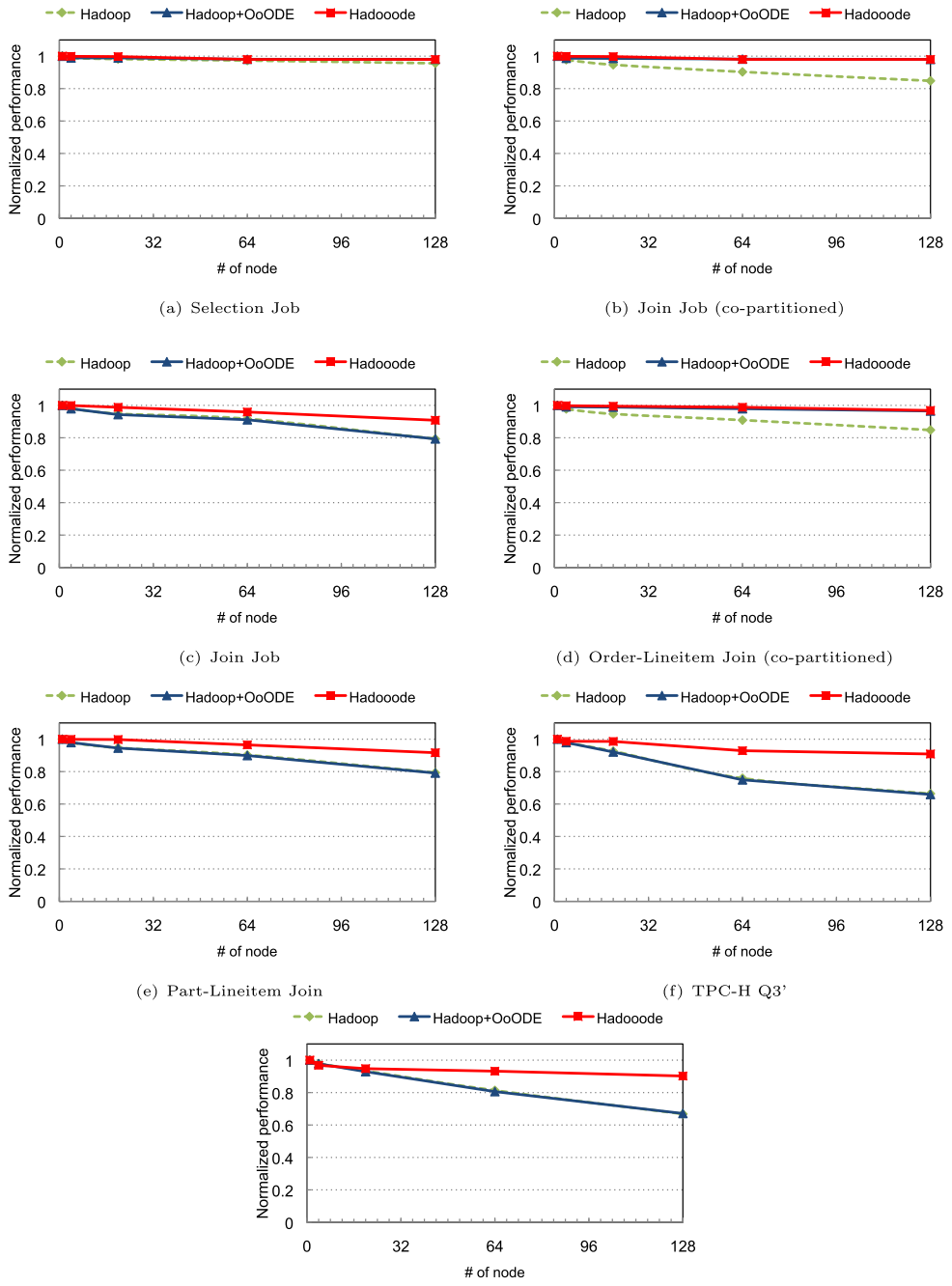
図 9 に，各解析タスク並びに各問合せにおける選択率 0.01%のときのノードスケーラビリティを示す．いずれの問合せにおいても，Hadooode は，他の Hadoop 処理系に対して高いスケーラビリティが得られており，128 ノードで約 90%以上のスケーラビリティを達成している．一方，Hadoop+OoODE は，入出力が計算機内に閉じる場合（図 9 (a)，9 (b)，9 (d)）は，99%程度のスケーラビリティが得られているものの，その他の入出力が計算機間をまたがる場合においては，約 66～85%程度のスケーラビリティとなっていることがわかる．なお，Hadoop においては，128 ノードで約 66～85%程度のスケーラビリティが得られた．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，大規模なクラスタ環境においても，高いスケーラビリティを維持しつつ，処理効率の向上を達成することができる．

4.7 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とおりである．

Q1：疾病 A における都道府県ごとの患者数の対人口比率

（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．



(a) Selection Job

(b) Join Job (co-partitioned)

(c) Join Job

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

(e) Part-Lineitem Join

(f) TPC-H Q3'

(g) TPC-H Q8'

図 9　128 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. 9　Comparison of scalability in a 128-node cluster.

Q2：疾病 B における都道府県ごとの平均診療費
Q3：疾病 C 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．よって，SQL 問合せのみを受け付ける処理系は比較対象から除き，すなわち，

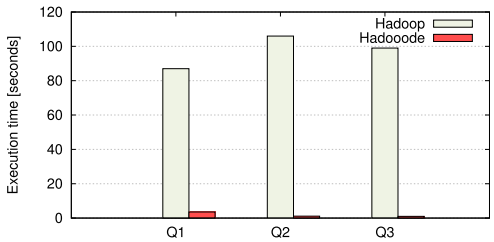


図 10　医療保険レセプトデータを用いた性能比較
Fig. 10　Performance comparison for medical receipt data.

Hadoop と Hadooode のみの比較を行った．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，Hadooode においては，当該データセットのレコード番号属性を基にハッシュ分割によって各計算機に分配した．また，Hadooode においては，疾病の属性に対して索引を構成した．

図 10 に性能比較結果を示す．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，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対しても高い性能を備えていることが分かる．

5. 関 連 研 究

並列データベースにおいては，問合せ処理の並列度の向上を目指した研究が数多く行われてきており，一つの大きな技術潮流として，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 [19], [20] を活用した，多数の並列結合アルゴリズムが提案されている [21]～[28]．

Hadoop においては，データ処理の効率を向上させる研究が近年盛んに行われてきており，データの全走査を基本とする Hadoop に対して，Hadoop++ [30]，HAIL [31]，HadoopDB [17], [29] では，索引を用いたファイルのアクセス手法を提案している．

Hadoop の普及に伴い，Hadoop が抱える問題を克服すべく新たな並列データ処理系が提案されてきている．中でも，MapReduce 処理とデータベース処理の双方の利点を活かす試みが進展しつつあり，MapReduce エンジン並びに DFS 上に関係演算を実装するシステムが広く見られる [32]～[35]．他方，Osprey [36] では，並列データベースを基盤としつつ，MapReduce に類似した耐障害機構を追加したシステムを提案している．

これらに対し，本研究は，並列データ処理（並列問合せ処理）の入出力並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り，上記においてはこれに類するアイデアは見られない．

6. む　す　び

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 Hadooode を開発してきた．Hadooode は，並列データ処理全体の入出力を非同期化する点に新たな特徴を有し，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は，入出力スループットを飛躍的に向上し，データ処理の大幅な高速化を実現する．本論文では，128 ノードの大規模クラスタシステム（合計 2048 CPU コア，3072 磁気ディスクドライブ）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 倍の性能向上を確認した．この際，直接の比較は難しいものの，Hadoop との比較では，最大 178 倍程度の高速化を確認した．また，ノードスケーラビリティの評価実験を行い，大規模なクラスタ環境においても，他の Hadoop 処理系と比して高いスケーラビリティを備えることを示した．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 の高い有効性を確認した．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方式並びに耐障害方式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」，及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，厚生労働科学研究費補助金（政策科学総合研究事業（政策科学推進研究事業））「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（H24-政策-一般-002）」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, and K. Vaid,

"Server engineering insights for large-scale online services," IEEE Micro, vol.30, pp.8–19, 2010.
[2] E. Eleftheriou, R. Haas, J. Jelitto, M. Lantz, and H. Pozidis, "Trends in storage technologies," IEEE Technical Committee on Data Engineering, 2010.
[3] D.A. Patterson, "Latency lags bandwith," Commun. ACM, vol.47, no.10, pp.71–75, 2004.
[4] Hadoop, http://hadoop.apache.org/
[5] "The Stinger Initiative: Making Apache Hive 100 Times Faster," http://hortonworks.com/blog/100x-faster-hive/
[6] "Pivotal HD: The World's Most Powerful Distribution of Apache Hadoop," http://www.greenplum.com/products/pivotal-hd
[7] "Presto: Distributed SQL Query Engine for Big Data," http://prestodb.io/
[8] "Cloudera Impala: Open Source, Real-time Query for Hadoop," http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html
[9] S. Melnik, A. Gubarev, J.J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," Proc. VLDB, pp.330–339, 2011.
[10] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," Proc. NSDI, pp.15–28, 2012.
[11] 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，vol.8, no.1, p.6, 2009.
[12] 合田和生，早水悠登，喜連川優，"100 ドライブ規模のディスクストレージ環境におけるアウトオブオーダ型データベースエンジン OoODE の問合せ処理性能試験," 信学論（D），vol.J97-D, no.4, pp.729–737, April 2014.
[13] 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.
[14] 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," 信学論（D），vol.J97-D, no.4, pp.774–792, April 2014.
[15] 山田浩之，合田和生，喜連川優，"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," 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，2014.
[16] M. Stonebraker, "The case for shared nothing," Database Engineering, vol.9, pp.4–9, 1986.
[17] 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.
[18] 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.
[19] D. DeWitt and J. Gray, "Parallel database systems: The future of high performance database systems," Commun. ACM, vol.35, no.6, pp.85–98, 1992.
[20] C. Ballinger and R. Fryer, "Born to be parallel: Why parallel origins give teradata an enduring performance edge," IEEE Data Engineering Bulletin, vol.20, no.2, pp.3–12, 1997.
[21] P. Valduriez and G. Gardarin, "Join and semijoin algorithms for a multiprocessor database machine," ACM Trans. Database Systems, pp.133–161, 1984.
[22] D.A. Schneider and D.J. DeWitt, "Tradeoffs in processing complex join queries via hashing in multiprocessor database machines," Proc. VLDB, pp.469–480, 1990.
[23] M. Chen, M. Lo, P.S. Yu, and H.C. Young, "Using segmented right-deep trees for the execution of pipelined hash joins," Proc. VLDB, pp.15–26, 1992.
[24] M. Ziane, M. Zait, and P. Borla-Salamet, "Parallel query processing in DBS3," Proc. PDIS, pp.93–102, 1993.
[25] M. Ziane, M. Zait, and P. Borla-Salamet, "Parallel query processing with zigzag trees," Proc. VLDB, pp.277–302, 1993.
[26] B. Liu and E.A. Rundensteiner, "Revisiting pipelined parallelism in multi-join query processing," Proc. VLDB, pp.829–840, 2005.
[27] D.J. DeWitt, J.F. Naughton, and J. Burger, "Nested loops revisited," Proc. PDIS, pp.230–242, 1993.
[28] 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.
[29] 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.
[30] 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.
[31] J. Dittrich, J.A. Quian-Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad, "Only aggressive elephants are fast elephants," Proc. VLDB, pp.1591–1602, 2012.
[32] H. Yang, A. Dasdan, R. Hsiao, and D.S. Parker, "Map-reduce-merge: Simplified relational data processing on large clusters," Proc. SIGMOD, pp.1029–1040, 2007.
[33] T. Kaldewey, E.J. Shekita, and S. Tata, "Clydesdale: Structured data processing on MapReduce," Proc. EDBT, pp.15–25, 2012.
[34] B. Chattopadhyay, L. Lin, W. Liu, S. Mittal, P.

Aragonda, V. Lychagina, Y. Kwon, and M. Wong, “Tenzing A SQL implementation on the MapReduce framework,” Proc. VLDB, pp.1318–1327, 2011.
[35] J. Zhou, N. Bruno, M. Wu, P. Larson, R. Chaiken, and D. Shakib, “SCOPE: Parallel databases meet MapReduce,” Proc. VLDB, pp.611–636, 2012.
[36] C. Yang, C. Yen, C. Tan, and S. Madden, “Osprey: Implementing MapReduce-style fault tolerance in a shared-nothing distributed database,” Proc. ICDE, pp.657–668, 2010.

（平成 26 年 8 月 4 日受付，12 月 1 日再受付，
27 年 2 月 4 日早期公開）

山田　浩之

2003 年上智大学理工学部機械工学科卒．同年日本アイ・ビー・エム株式会社入社，2004 年ヤフー株式会社入社．分散型全文検索エンジンの研究開発に従事．2010 年慶應義塾大学大学院理工学研究科博士前期課程了．日本学術振興会特別研究員を経て，2014 年東京大学大学院情報理工学系研究科博士後期課程了．博士（情報理工学）．現在，東京大学生産技術研究所特任研究員．データベースシステムの研究に従事．情報処理学会，日本データベース学会，ACM 各会員．

合田　和生

平成 12 年東京大学工学部電気工学科卒，平成 14 年同大学大学院工学系研究科電子情報工学専攻修士課程了，平成 17 年同大学院情報理工学系研究科電子情報学専攻博士課程単位取得満期退学．同年，博士 (情報理工学)．日本学術振興会特別研究員，東京大学生産技術研究所産学官連携研究員等を経て，現在，東京大学生産技術研究所特任准教授．データベースシステム，ストレージシステムの研究に従事．情報処理学会，日本データベース学会，ACM，IEEE，USENIX 各会員．

喜連川　優　（正員：フェロー）

1983 年東京大学大学院工学系研究科情報工学専攻博士課程了，工博．東京大学生産技術研究所教授，2013 年 4 月より国立情報学研究所所長，2013 年 6 月より情報処理学会会長．データベース工学の研究に従事．本会業績賞，情報処理学会功績賞，ACM SIGMOD E.F. Codd Innovations Award 受賞．ACM，IEEE，情報処理学会フェロー．