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

合田　和生[†a)]　　早水　悠登[†b)]　　喜連川　優[†,††c)]

Performance Test of Query Processing by Out-of-Order Database Engine (OoODE) in an Experimental Environment with 160 Magnetic Disk Drives

Kazuo GODA[†a)], Yuto HAYAMIZU[†b)], and Masaru KITSUREGAWA[†,††c)]

あらまし　本論文では，アウトオブオーダ型と称する独自のソフトウェア実行方式に基づく高速データベースエンジンを提案する．従来看過されてきた非同期入出力を本格的に導入することにより，相対的に安価になりつつある演算資源を活用し，入出力資源の利用効率を大幅に向上させることを目指す．少なくない種類の問合せにおいて，従来型の逐次的な実行方式によるエンジンと比べて，問合せ処理の高速化が期待される．本論文では，これまで著者らがオープンソースDBMSをベースとして進めてきたアウトオブオーダ型データベースエンジンの試作実装を示すとともに，24個のプロセッサコアと160台の磁気ディスクドライブを備えたミッドレンジクラスの実験環境における性能試験結果を示し，その高速性を明らかにする．

キーワード　OoODE，アウトオブオーダ型実行，問合せ処理，非同期入出力，データベースエンジン

1. まえがき

Ingres [1] 等の初期の実装以来，多くのデータベースエンジンの実装は，関係問合せを受け付けると，問合せ実行木を生成し，実行木において演算ノードを逐次的に辿って実行することにより，当該問合せを処理する．この際の演算ノードの実行において，演算対象のデータをデータベースから取得する必要がある場合には，データベースが格納されているストレージに入出力命令を発行し，その完了通知を待って当該データの演算を実行することを，対象となる全てのデータに対して処理を終えるまで繰り返す [2], [3]．本論文では，このようなデータベースエンジンの実行方式を**インオーダ型**と称することとする．

関係モデルに基づくデータベースエンジンの実装が試みられた当時においては，プロセッサ性能と主記憶容量は，今日と比べて遙かに限定的であり，実装戦略としては，当該演算資源と主記憶資源を節約するべく，コードパスとフットプリントを節約することが望ましく，上述のような同期入出力を用いる実行方式は妥当な選択肢であったと言えよう．しかしながら，今日のハードウェア環境は，当時のものとは大きく異なってきている．プロセッサ技術としては，2004年頃から周波数の向上は停滞しているものの，マルチコア化が進展し [4]，依然としてトランジスタの集積度はこれまでと同様に年率60%程度のペースで指数関数的に増大してきている [5]．他方で，ストレージ技術に目を向けると，磁気ディスクドライブの容量密度としては，1990年代に，主記憶との熾烈な競争を背景とし，巨大磁気抵抗効果 [6] 等，次々と新しい技術が投入されていた頃程の高いペースは見られないものの，今後も，年率20-30%程度の向上が維持されるものとみられている [7]．一方，入出力性能としては，シークレイテンシの改善はこの10年でほぼ停滞していると言っても過言ではなく，また転送帯域の向上も2000年以降，急激に鈍化している [8]．すなわち，ハードウェア技術としては，演算資源に比して，相対的に入出力資源が

† 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: kgoda@tkl.iis.u-tokyo.ac.jp
b) E-mail: haya@tkl.iis.u-tokyo.ac.jp
c) E-mail: kitsure@tkl.iis.u-tokyo.ac.jp

高価になってきており [9]，システムを設計する際に，演算を司るサーバではなく，入出力を司るストレージを中心に据えるという考え方が登場しているのは自然な流れと言える [10]．

新たなハードウェアの特性バランスに基づき，ソフトウェアの実装方式を再考する段階に入ったとも言える [9], [11]．圧縮技法 [12]〜[14] やスケールアウト指向のシステム [15], [16] が注目されてきているが，著者らは，より根源的なアプローチに挑戦することとし，**アウトオブオーダ型**と称する非同期入出力を用いたソフトウェア実行方式を考案した．本論文では，当該実行方式に基づくデータベースエンジンを提案する．今日一般的なインオーダ型のデータベースエンジンにおいては，入出力は従前と変わらず同期的に行われるが，著者らの提案するアウトオブオーダ型データベースエンジン (OoODE: Out-of-Order Database Engine) においては，従来看過されてきた非同期入出力を本格的に導入することにより，相対的に安価になりつつある演算資源を活用し，入出力資源の利用効率を大幅に向上させることをを目指す．問合せの種類によってその効果は異なるものの，少なくない問合せにおいて，インオーダ型のデータベースエンジンに比して，大幅な性能向上が期待される．従前の論文では，アウトオブオーダ型データベースエンジンの基本アイデアと，小規模環境における性能試験を報告してきたが [17], [18]，本論文においては，非同期入出力を用いたソフトウェア構成法を明らかにするとともに，4 基のプロセッサと 160 台の磁気ディスクドライブを備えたミッドレンジクラスの実験環境における性能試験結果を示し，提案データベースエンジンが，幅広い問合せ選択率に対して高い有効性をもつことを明らかにする．

本論文の構成は以下のとおりである．**2.** では，アウトオブオーダ型データベースエンジンにおける非同期入出力を用いた問合せ処理方式を明らかにする．**3.** では，著者らが進めているアウトオブオーダ型データベースエンジンの試作実装を示す．**4.** では，当該試作機においてミッドレンジクラスの実験環境を用いて行った TPC-H ベンチマークデータセットによる性能評価を報告し，データベースエンジンの有効性を明らかにする．**5.** においては関連研究を紹介し，**6.** において本論文を纏める．

2. アウトオブオーダ型のクエリ処理実行方式

関係データベースにおける問合せ処理は，主に，リレーションに対する集約演算から構成される．従前のインオーダ型のデータベースエンジンでは，このような集約演算を，データベースからのタプルの取得と，取得したタプルに対する演算とを繰り返すことによって，実行する．例えば，二つのリレーション R と S に対する結合演算 $R \bowtie_A S$ を考える．この際，R を外表，S を内表とするネステッドループ結合を想定すると，データベースエンジンは，リレーション R からタプル r を取得する繰り返しを行い，この繰り返しの中で，取得したタプル r のそれぞれに対して，更に，リレーション S から s を取得して，r と s の組み合わせに対して結合条件 A を判定することを繰り返す．すなわち，多段にネストした繰り返し手続きによって，データベースからのタプルの取得と，取得したタプルに対する演算を行う(注1)．

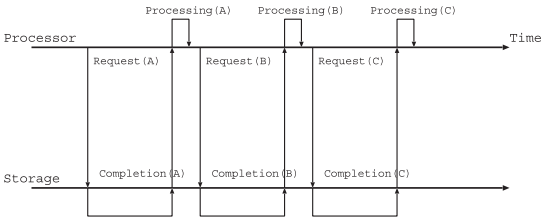
上述のインオーダ型のデータベースエンジンに対して，著者らの提案するアウトオブオーダ型のデータベースエンジンは，問合せ処理における集合演算を実行する際に，逐次的な繰り返しを行うのではなく，繰り返し手続きのループ展開 (loop unrolling) を行い，データベースからのタプルの取得とそれに対する演算を非同期的に実行する点に特色を有する．再び，結合演算 $R \bowtie_A S$ を例にすると，データベースエンジンは，まず，外表であるリレーション R からタプル r を取得する要求を発行するものの，この際，データベースエンジンは，タプル r の取得待ちによって中断されることはなく，即座に別の手続きを実行可能な状態に移行する．その後，タプル r の取得の完了が通知されると，データベースエンジンは，取得したタプル r に対して，更に，内表であるリレーション S からのタプル s の取得を要求する．この際も，データベースエンジンはいったん，別の手続きを実行可能な状態に移行し，タプル s の取得が完了してから，取得したタプル s に対して，r との結合性判定のための演算を実行する．一般に，結合演算には，ファンアウトが見られることが多く，すなわち，一つのリレーションのレコー

(注1)：ここでは簡単のため，索引については述べていない．一般には，内表であるリレーション S からのタプルの取得は，索引を用いて，結合条件 A によって選択的に行われることが多い．この場合においても，本論文の議論はそのまま適用することが可能である．

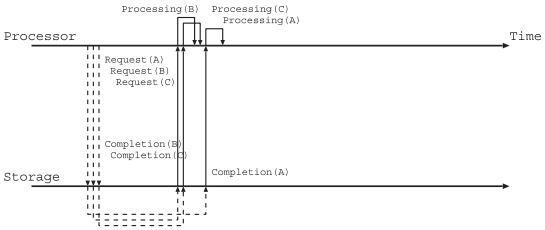
ドに対して結合対象のリレーションでは複数のレコードが対応することがあり，このようなケースにおいては，上述のようなループ展開によって，タプルの取得とそれに対する演算が重層的に実行されることとなる．

ソフトウェアとしての実装は多様な方式が考えられるが，各種のスレッドやプロセス等の同時実行可能なタスク機構と，非同期入出力命令を組み合わせて用いることにより，上述の実行方式を実現することが考えられる．これにより，問合せ処理は実行時に，実行論理の許す限りにおいて，動的に多数のタスクに分解されることとなり，多数の非同期入出力がストレージに発行されることとなる．現実には，利用可能な資源は有限であり，同時実行可能なタスクや同時発行可能な入出力は，利用可能な主記憶容量やストレージや入出力パスを構成するコンポーネントの処理能力によって制約される．しかしながら，最近のサーバはマルチコア化が著しく，ハードウェアレベルで数十程度のスレッドを同時実行する機能を備えているほか，OS レベルでは数千程度のスレッドの同時実行が可能であり，また，最近のストレージにおいても高集積化が進んでおり，ミッドレンジクラスにおいて百台以上の磁気ディスドライブを搭載したサブシステムも珍しくない．従来に比して，多数のタスクと入出力を同時処理することが可能となってきている．殊に，入出力パスは，高度なスケジューリング機能を具備するに到ってきており，論理的な発行順序とは異なる順序で入出力を処理することにより，スループットを高める能力が備えられている [19], [20]．従来のインオーダ型のデータベースエンジンにおいては，コードパスとフットプリントを抑制するべく，同期入出力命令を用いて，逐次的に問合せを処理することにより，極めて限られた数の入出力が同時発行されていた（図 1 (a)）．これに対して，著者らの提案するアウトオブオーダ型のデータベースエンジンは，非同期入出力を用いることにより，実行論理の許す限りにおいて，多数の入出力を発行することを実現し（図 1 (b)），これによって，潤沢なプロセッサ資源を有効活用するとともに，多数の磁気ディスクドライブの同時駆動によって，従来比での入出力スループットの飛躍的な向上を目指す．

ここでは，二つのリレーションの結合演算を例に，アウトオブオーダ型の問合せ処理実行方式を述べたが，当該方式は単一表の索引走査や多段の索引結合に同様に適用することができ，これらを含む多くの問合せにおいて，類似の効果が期待される．



(a) In-order execution.



(b) Out-of-order execution.

図 1　データベースエンジンの実行挙動の比較
Fig. 1　Comparison of execution behavior in database engines.

なお，リレーションを事前に複数のパーティションに分割し，パーティションごとに問合せを分割して実行する技法も提案され，用いられるに到っているが，これは，パーティションごとにインオーダ型の問合せ処理を行っているにすぎず，本論文で提案するアウトオブオーダ型の実行方式とは根源的に異なる．

問合せ処理におけるアクセスパスとしては，従前より，問合せのリレーションに対する選択率に応じて，大きく分けて，索引を用いてリレーションの一部をアクセスするか，若しくは，リレーションの全体を走査するか，の選択が行われてきた．問合せがリレーションのごく一部のデータのみを必要とする場合には，索引を用いて選択的にリレーションにアクセスすることの利得が大きいが，逆に，問合せがリレーションの多くの部分のデータのみを必要とする場合には，ディスクシークのオーバヘッドを低減するべくリレーション全体をアクセスすることの利得が大きい．リレーション規模がギガバイトからテラバイト，更にはペタバイト級への巨大化するにつれ [21]，単純な走査が可能な機会はより限られてくるはずであり，相対的に選択的な問合せの重要性が高まる可能性が高い．本論文で提案する非同期入出力を用いたアウトオブオーダ型データベースエンジンによって，このような問合せに対して，飛躍的な性能向上が期待される．

3. アウトオブオーダ型データベースエンジンの試作実装

著者らは，これまでオープンソースのデータベースソフトウェアをベースとして，アウトオブオーダ型データベースエンジンの試作機を複数実装してきた．本論文では，開発を進めている一実装である MySQL [22] ベースの試作実装を紹介する．

当該実装は，主に問合せ処理器とストレージエンジンから構成される．問合せ処理器は，前節に述べた非同期入出力を用いた問合せ処理の実行方式に基づき，受け付けた問合せの処理を実行する．すなわち，問合せ実行木を生成した後に，当該実行木を辿って演算ノードを実行するが，この際，新たにデータをデータベースから取得する必要がある場合には，ストレージエンジンに対して，非同期命令を用いてレコード取得を要求する．当該非同期命令によって，問合せ処理器の一連の手続きはブロックされることはなく，即座に別の処理を実行することが可能となる．レコード取得が完了すると，コールバック手続きが呼び出され，当該コールバック手続きによって，取得されたレコードに対する演算が実行される．現状の実装は，Linux オペレーティングシステムを対象としており，コールバック手続きは，二つのスレッド機構を組み合わせて実現することにより，同時実行可能なものとしている．すなわち，プロセッサコアごとに一つの pthread 機構によるカーネルスレッドを割り当て，各々のカーネルスレッド上において，独自実装によるユーザスレッドを複数配置し，レコード取得が完了すると，ユーザスレッドにコールバック手続きを割り当てて，並行して実行できるようにした．なお，ユーザスレッドの管理に必要なフットプリントは，カーネルスレッドの管理に必要なフットプリントと比べると極めて小さいものの，利用可能な資源は有限であることから，カーネルスレッドあたりのユーザスレッドの数には上限を設け，上限を超えるユーザスレッドの駆動が要求された場合には，問合せ処理器の手続きをブロックすることとした．著者らの経験では，上述のようにカーネルスレッドとユーザスレッドを併用することにより，1,000 を上回る同時実行を行ったとしても，大きなオーバヘッドは確認されていない [23]．

一方，ストレージエンジンは，問合せ処理器からの要求に応じて，レコード取得を行う．問合せ処理器と同様に，現状の実装は Linux オペレーティングシステムを想定しており，レコード取得のための非同期命令を受け付けると，`libaio` なる機構を用いて非同期入出力システムコールを呼び出し，これによってストレージへの非同期入出力を発行する．非同期入出力が完了すると，取得したデータからレコードを抽出して，問合せ処理器にレコード取得の完了を通知する．この際，レコードを取得した後に，更に取得できるレコードが存在する際には，これを取得して，問合せ処理器に通知する．なお，ストレージエンジン内では，データ構造の並列性を活用して，エントリやレコードの取得手続きを展開して並行実行する工夫を施した．現状の実装においては，性能試験を円滑に進められるように，MySQL の主要なストレージエンジンである InnoDB のストレージフォーマットを採用しており，InnoDB によって構成されたデータベースをマウントすることができることとしてある．

4. ミッドレンジクラスの実験環境における性能試験

著者らは，4 基のプロセッサと 160 台の磁気ディスクドライブを備えたミッドレンジクラスの実験環境を構築し，当該環境においてアウトオブオーダ型データベースエンジンの性能試験を行った．

4.1 実験環境

表 1 に，著者らが構築した実験環境の諸元を示す．磁気ディスクアレイは，160 台の SAS 型磁気ディスクドライブを備えている．これらの磁気ディスクドライブからは，8 台ごとにパリティストライピング (RAID-5) によって論理ユニットが編成されており，総じて 20 個の論理ユニットが構成されている．磁気ディスクアレ

表 1 実験環境
Table 1 Experimental setup.

Server (IBM SystemX X3850M2)	
Processors	4x Intel Xeon X7460 2.66GHz (4p/24c/24t)
Memory	32GB
HBA	4x Emulex 4Gbps FibreChannel dual-port HBAs
OS	Redhat Enterprise Linux 5 x86_64
DBMS	OoODE prototype MySQL 5.5 PostgreSQL 9.2
Storage (IBM System Storage DS5300)	
Controller	Dual controller
HBA	8x 4Gbps FibreChannel ports
Cache	2GB
Disk	160x SAS 15Krpm 450GB HDDs

イは二つのコントローラを備えており，各々10 個の論理ユニットが割り当てられている．サーバと磁気ディスクは 8 本の 4Gbps ファイバチャネルで接続されており，磁気ディスクアレイが備える二つのコントローラの各々に 4 本ずつ接続されている．20 個の論理ユニットは，いずれかのファイバチャネルに割り当てるよう LUN マスキングを施した．すなわち，ファイバチャネルあたり 2 若しくは 3 個の論理ユニットが割り当てられている(注2)．サーバは，24 個のプロセッサコアと 32GB の主記憶を備えており，Linux オペレーティングシステムが動作する．サーバにおいては，Linux の備える論理ボリュームマネージャを用いて，磁気ディスクアレイよりエクスポートされた 20 個の論理ユニットのそれぞれの先頭 512GB の区画から，パリティなしストライピング (RAID-0) によって，論理ボリュームを構成した．

4.2 非同期入出力の基本性能試験

アウトオブオーダ型データベースエンジンによる問合せ処理性能を測定する前に，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．入出力性能測定のためのマイクロベンチマークを作成し，当該マイクロベンチマークを用いて，磁気ディスクアレイに対して構成した論理ボリュームへの入出力性能を測定した．標準的な InnoDB におけるページサイズは 16KB であり，論理ボリュームを raw デバイスとして直接マウント可能である．これを模すため，論理ボリュームに対してダイレクト入出力による非同期入出力を用いた 16KB のランダム読込みを行い，そのスループットを測定した．

図 2 に，非同期入出力の多重度に対する入出力スループットの変化を示す．実験環境においては，多重度が 1，すなわち，同期入出力に相当する場合，90 IOPS 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見られた．1,024 多重度までは多重度の増加に対してスループットの向上が著しく，1,024 以降ではスループットの向上は急激に鈍化した．4,096 多重度においては，56,200 IOPS のスループットを確認することができた．このことから，当該ランダム読込みに対して，実験環境は 55,000 IOPS を超える入出力スループットを備えているものの，そのような入出力スループットを引き出すためには，1,024 程度以上の高多重性を以って非同期入出力を発行することが必要であることが確認された．

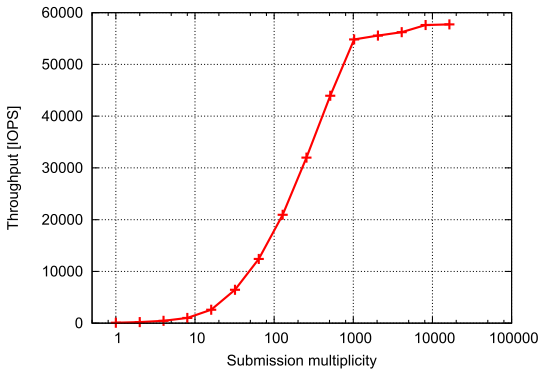


図 2 入出力基本性能
Fig. 2 Basic inputs/outputs performance.

4.3 TPC-H データセットを用いた問合せ処理性能試験

実験環境において，アウトオブオーダ型データベースエンジンの試作機を用いた性能試験を行った．この際，試作機においては最大 12 個のカーネルスレッドを使用することとし(注3)，カーネルスレッドあたりのユーザスレッドの上限並びに非同期入出力の最大発行多重度は 1,024 とした．すなわち，システム全体での実行多重度は 12,288 とした．

データセットとしては，意思決定支援系の業界標準ベンチマークである TPC-H [24] の `dbgen` を用い，スケールファクタ 100 のデータセットを生成し，先述の論理ボリューム上に，InnoDB によるデータベースを構築した．データセットのロード後のデータベースの物理サイズはおよそ 250GB 程度であった．当該データベースに対して，図 3 に示す 3 リレーションを結合する問合せを実行し，その実行時間を測定した．この際，問合せの選択率が実行時間に与える影響を計測するため，最小リレーションである `CUSTOMER` に与える選択条件を調整し，選択率が 1%から 0.0001%まで変動させ，それぞれの選択率に対して計測を行った．

アウトオブオーダ型データベースエンジンの試作機においては，図 4 (a) に示す索引結合を用いた問合せ実行計画により問合せ処理を行い，その実行時間を

(注2)：ファイバチャネルあたりの論理ユニットの個数に非対称性があるが，精緻な性能計測を行ったところ，本論文で述べる性能試験においては，これによる性能低下は観測されなかった．

(注3)：サーバは 24 個のプロセッサコアを備えているが，カーネルスレッドを 24 個配置した場合，性能低下が見られたため，12 個のカーネルスレッドを用いることとした．

```
SELECT COUNT(*), SUM(L_QUANTITY)
  FROM CUSTOMER C
    INNER JOIN ORDERS O
      ON C.C_CUSTKEY = O.O_CUSTKEY
    INNER JOIN LINEITEM L
      ON O.O_DERKEY = L.L_ORDERKEY
  WHERE [selection condition on CUSTOMER]
```

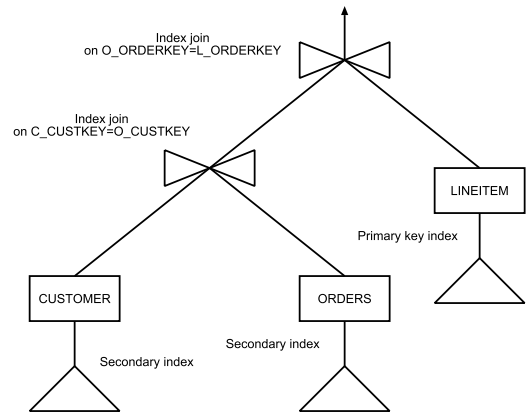
図 3　試験問合せ
Fig. 3　Test query.

計測した．この際，アウトオブオーダ型実行による高速化効果を確認するため，同実装において，非同期入出力を用いず同期入出力によって問合せ処理を行う制限を課した上で，すなわち，インオーダ型実行によって，同じ問合せ処理を行い，その実行時間を計測した．更に，これに加えて，当該試作機による問合せ処理の妥当性を確認するため，代表的なオープンソースデータベースソフトウェアである MySQL [22] 並びに PostgreSQL [25] を用いて，同じ問合せの処理を行い，その実行時間を計測した．これらのデータベースソフトウェアはインオーダ型の問合せ処理を行う．MySQL は索引結合によってリレーション結合を行うため，アウトオブオーダ型データベースエンジンの試作機と同様に図 4 (a) に示す問合せ実行計画により，問合せ処理を行った．これに対して，PostgreSQL は更にハッシュ結合 [26], [27] を備えていることから，図 4 (a) に示す索引結合による問合せ実行計画に加えて，図 4 (b) に示すレフトディープ型のハッシュ結合(注4)による問合せ実行計画によって問合せ処理を行い，それぞれの場合について実行時間を計測した．なお，問合せ処理の実行の都度には，データベースエンジンを再起動するとともに，磁気ディスクアレイにマイクロベンチマークを用いて十分な量の入出力を発行することにより，データベースバッファ並びに磁気ディスクアレイコントローラのキャッシュをクリアし，測定に影響がないようにした．

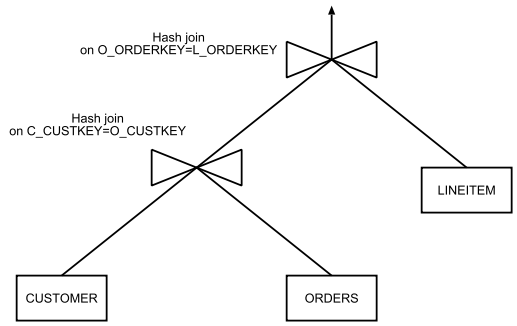
図 5 に計測結果を纏める．図中の凡例は以下のとおりである．

- **Prototype (OoO, IJ)**: アウトオブオーダ型データベースエンジンの試作機による問合せ処理
- **Prototype (IO, IJ)**: アウトオブオーダ型データベースエンジンの試作機におけるインオーダ型実行による問合せ処理
- **MySQL (IJ)**: MySQL による問合せ処理

(注4)：CUSTOMER リレーションのビルド時に索引走査を行うことも考えられるが，PostgreSQL の最適化器はそのような実行木を出力しなかった．



(a) Index join.



(b) Hash join.

図 4　問合せ実行計画
Fig. 4　Query execution plan.

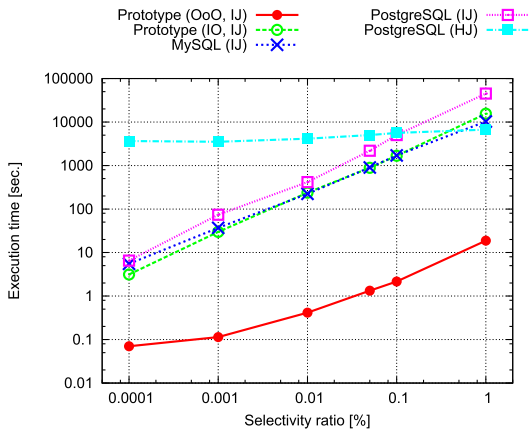


図 5　問合せ実行時間
Fig. 5　Query execution time.

- **PostgreSQL (IJ)**: PostgreSQL による問合せ処理（索引結合）
- **PostgreSQL (HJ)**: PostgreSQL による問合せ処理（ハッシュ結合）

一般に，索引結合においては，選択率が高くなると，ストレージへの入出力の数が増えることとなり，これにより実行時間が長くなる．インオーダ型実行により索引結合を行ういずれのケースにおいても，同様の傾向が確認された．著者らの試作実装による索引結合並びに MySQL による索引結合では，多くのケースにおいて実行時間に大きな差は見られなかった．対して，PostgreSQL による索引結合でも傾向は同様であったが，若干長い実行時間が観測された．著者らの実装と MySQL では同じ InnoDB ストレージエンジンフォーマットを利用したが，PostgreSQL の場合は PostgreSQL のストレージエンジンを用いたことから，実験ではこの実装差によって実行時間の差が生じたものと推察される．

これらに対して，著者らの試作実装において，アウトオブオーダ型実行により索引結合を行った場合，インオーダ型実行に対して，遙かに実行時間を短縮することができた．例えば，選択率 0.1%においては，インオーダ型の索引結合では，実行時間は 1700 秒程度（PostgreSQL を除く）であったのに対し，アウトオブオーダ型実行によっては，実行時間は 2.2 秒であった．この際の入出力スループットを計測すると，インオーダ型の索引結合では 120 IOPS 程度であり，アウトオブオーダ型の索引結合では 140,000 IOPS 程度であった(注5)．問合せ処理に必要な入出力の数は同様であると考えられ，非同期入出力による入出力スループットの向上によって，著しい高速化がもたらされたと考えられる．なお，0.001%以下の選択率においては，インオーダ型実行に比して，選択率を減じた際の実行時間の減少傾向に鈍化がみられる．当該選択率領域においては，リレーション中でアクセスするレコードの絶対数が小さくなり，問合せ処理に内在する実行並列度が低くなり，これにより相対的にアウトオブオーダ型実行の恩恵が小さくなっているものと考えられる．

一方，ハッシュ結合の場合，一般に，ストレージへの必要な入出力の量は，選択率に影響を受けず，入出力バウンドを仮定すると，実行時間は選択率に大きく影響されない．本性能試験においては，PostgreSQL によるハッシュ結合において，実際の実行時間は入出力にかかる時間に加えてプロセッサ処理にかかる時間の影響もあることから，選択率の増加に対して若干実行時間は増加するものの，索引結合程の著しい変化は見られず，実行時間は 3,500-6,700 秒程度であった．

問合せ最適化においては，索引結合かハッシュ結合かの選択が鍵となることが多い．インオーダ型実行による索引結合とハッシュ結合の交差点としては，索引結合の実装によって差はみられるものの，0.1-0.3%程度と考えられる．一方，更にアウトオブオーダ型実行を考慮すると，実験を行った範囲においては，常に，アウトオブオーダ型実行の索引結合が優位であった．全ての選択率において，アウトオブオーダ型実行の索引結合が優位であるとは考えずらいが，選択率 0.0001%から 1%という比較的広い選択率の範囲において，その著しい高速性を確認したと言える．

5. 関 連 研 究

問合せ処理に内在する実行並列性を活用することにより，高速化を目指す研究は，古くからおこなわれてきてた．その多くはパーティション並列性とパイプライン並列性に着目したものである [28], [29]．Graefe は，exchange 命令を用いることにより並列度を向上する Volcano なるイタレータモデルを提案している [30]．Boncz らは，複数レコードをパイプライン処理し，ベクトル演算を適用することによる高速化を提案している [31]．Cieslewicz らは，コンパイラのループ展開を活用し，Cray マシンのハードウェアマルチスレッド機能を活用した問合せ処理の高速化を提案している [32]．Roh らは，フラッシュメモリに対する B+木の並列探索手法を提案している [33]．商用の実装においては，二次索引等の探索時に入出力をベクトル化して実行する技法が行われている [34], [35]．著者らの提案は，問合せ処理に内在する実行並列度を引き出すことにより高速化を目指す点においては，これらの研究と目的を同じにするが，問合せ処理において，非同期入出力を本格的に活用する点が大きく異なる．

6. む す び

本論文では，アウトオブオーダ型と称する独自のソフトウェア実行方式に基づくデータベースエンジンを提案した．従来看過されてきた非同期入出力を本格的

(注5)：**4.2** の基本性能試験における測定結果より高い入出力が観測されている．本節における問合せ処理ではストレージへの入出力に偏りがあり，ランダム読込みに比してシーク距離が短くなったことが推察される．

に用いるソフトウェア構成法を明らかにし，4 基のプロセッサと 160 台の磁気ディスクドライブを備えたミッドレンジクラスの実験環境における性能試験結果を示した．オープンソースのデータベースソフトウェア実装である MySQL 並びに PostgreSQL と比較し，選択率 0.0001%から 1%という比較的広い選択率の範囲において，当該実行方式による著しい高速性が確認された．今後は更に大規模の実験環境並びにデータセットを対象とし，多様な問合せ処理について検証を進めていくほか，問合せ最適化，更新・リカバリ，同時実行制御などの未開拓の問題について研究を進めていきたい．

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」の助成により行われた．

文　献

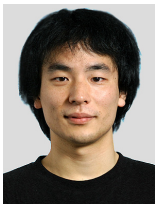
[1] M. Stonebraker, E. Wong, P. Kreps, and G. Held, “The design and implementation of INGRES,” ACM Trans. Database Syst., vol.1, no.3, pp.189–222, 1976.
[2] E. Wong and K. Youssefi, “Decomposition—A strategy for query processing,” ACM Trans. Database Syst., vol.1, no.3, pp.223–241, 1976.
[3] R. Elmasri and S. Navathe, Fundamentals of Database Systems, Addison Wesley, 1994.
[4] S. Borkar, “Thousand core chips - A technology perspective,” Proc. Annual ACM/IEEE Design Automation Conf., pp.746–749, 2007.
[5] C. Kozyrakis, A. Kansal, S. Sankar, and K. Vaid, “Server engineering insights for large-scale online services,” IEEE Micro, vol.30, no.4, pp.8–19, 2010.
[6] M.N. Baibich, J.M. Broto, A. Fert, F.N.V. Dau, F. Petroff, P. Etienne, G. Creuzet, A. Friederich, and J. Chazelas, “Giant magnetoresistance of (001)Fe/(001)Cr magnetic superlattices,” Phys. Rev. Lett., vol.61, pp.2472–2475, 1988.
[7] C. Munce, “Advancements in bit patterned media,” Presentation in IDEMA DISCON Japan 2013, 2013.
[8] W. Wilcke, R. Freitas, B.N. Kurdi, and G.W. Burr, “Storage class memory, technology and use,” Half-day tutorial on USENIX FAST 2009, 2009.
[9] I. Sutherland, “The tyranny of the clock,” Comm. ACM, vol.55, no.10, pp.35–36, 2012.
[10] U. Troppens, R. Erkens, and W. Muller, eds., Storage Networks Explained, Wiley, 2004.
[11] M. Stonebraker, D.J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E.J. O’Neil, P.E. O’Neil, A. Rasin, N. Tran, and S.B. Zdonik, “C-Store: A column-oriented DBMS,” Proc. Int’l. Conf. on Very Large Data Base, pp.553–564, 2005.
[12] D.J. Abadi, S. Madden, and M. Ferreira, “Integrating compression and execution in column-oriented database systems,” Proc. ACM SIGMOD Conf., pp.671–682, 2006.
[13] P. Francisco, The Netezza Data Appliance Architecture: A Platform for High Performance Data Warehousing and Analytics, IBM Redbooks, 2011.
[14] Oracle Corp., “Hybrid Columnar Compression (HCC) on Exadata,” An Oracle White Paper, 2012.
[15] The Apache Software Foundation, “Hadoop,” http://hadoop.apache.org/
[16] MongoDB Inc., “MongoDB,” http://www.mongodb.org/
[17] 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 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, pp.131–136, 2009.
[18] 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 の試作実装と小規模実験環境におけるソフトウェア実行挙動の観測,” 日本データベース学会論文誌，vol.12, no.1, pp.25–30, 2013.
[19] M.I. Seltzer, P.M. Chen, and J.K. Ousterout, “Disk scheduling revisited,” Proc. USENIX Tech. Conf., pp.313–323, 1990.
[20] B.L. Worthington, G.R. Ganger, and Y.N. Patt, “Scheduling algorithms for modern disk drives,” Proc. ACM SIGMETRICS Conf., pp.241–251, 1994.
[21] McKinsey Global Institute, Big data: The next frontier for innovation, competition, and productivity, 2011.
[22] Oracle Corp., “MySQL: The World’s Most Popular Open Source Database,” http://www.mysql.com/
[23] 清水　晃，徳田晴介，田中美智子，茂木和彦，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 におけるタスク管理機構の一実装方式の評価,” 電子情報通信学会／日本データベース学会データ工学と情報マネジメントに関するフォーラム，F3–5, 2013.
[24] Trans. Processing Performance Council, “TPC-H, an ad-doc, decision support benchmark,” http://www.tpc.org/tpch/
[25] PostgreSQL Global Development Group, “The world’s most advanced open source database,” http://www.postgreSQL.com/
[26] M. Kitsuregawa, H. Tanaka, and T. Moto-Oka, “Application of hash to data base machine and its architecture,” New Generation Comput., vol.1, no.1, pp.63–74, 1983.
[27] D.J. DeWitt, R.H. Gerber, G. Graefe, M.L. Heytens, K.B. Kumar, and M. Muralikrishna, “GAMMA - A high performance dataflow database machine,” Proc. Int’l. Conf. on Very Large Data Base, pp.228–237, 1986.
[28] D. DeWitt and J. Gray, “Parallel database systems: The future of high performance database systems,”

Comm. ACM, vol.35, no.6, pp.85–98, 1992.
[29] C. Ballinger and R. Fryer, “Born to be Parallel: Why parallel origins give teradata an eduring performance edge,” IEEE Data Eng. Bulletin, vol.20, pp.2–12, 1997.
[30] G. Graefe, “Volcano – An extensible and parallel query execution system,” IEEE Trans. Knowl. Data Eng., vol.6, no.1, pp.120–135, 1994.
[31] P. Bonca, M. Zukowski, and N. Nes, “MonetDB/X100: Hyper-pipelining query execution,” Proc. Biennial Conf. on Innovative Data Systems Research, pp.225–237, 2005.
[32] J. Cieslewicz, J. Berry, B. Hendrickson, and K.A. Ross, “Realizing parallelism in database operations: In-sights from a massively multithreaded architecture,” Proc. Int’l. Workshop on Data Management on New Hardware, 2006.
[33] H. Roh, S. Park, S. Kim, M. Shin, and S. Lee, “B+-tree index optimization by exploiting internal parallelism of flash-based solid state drives,” Proc. Int’l. Conf. on Very Large Data Base, pp.286–297, 2011.
[34] E. Ding, L. Dimino, G. Gopal, and T.K. Rengarajan, “Parallel processing capabilities of Sybase adaptive server enterprise 11.5,” IEEE Data Eng. Bulletin, vol.20, pp.35–43, 1997.
[35] Oracle Corporation, “DSS Performance in Oracle Database 11g,” White Paper, 2007.

（平成 25 年 12 月 11 日受付）

合田　和生　（正員）

平成 12 年，東京大学工学部電気工学科卒業，平成 14 年，同大学院工学系研究科電子情報工学専攻修士課程修了，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．



早水　悠登

平成 21 年，東京大学工学部電子情報工学科卒業．平成 23 年，同大学院情報理工学系研究科電子情報学専攻修士課程修了．現在，同専攻博士課程 3 年．高速データベースエンジンに関する研究に従事．日本学術振興会特別研究員 DC2．情報処理学会学生会員．

喜連川　優　（正員：フェロー）

昭和 58 年年東京大学工学系研究科情報工学専攻博士課程修了，工博．東京大学生産技術研究所教授，東京大学地球観測データ統融合連携研究機構長，平成 25 年 4 月より国立情報学研究所所長，平成 25 年 6 月より情報処理学会会長を務める．データベース工学の研究に従事．内閣府最先端研究開発支援プログラムを中心研究者として推進中．電子情報通信学会業績賞，情報処理学会功績賞，ACM SIGMOD E.F. Codd Innovations Award 受賞．ACM，IEEE，電子情報通信学会並びに情報処理学会フェロー．