論　文

アウトオブオーダ型データベースエンジン OoODE における優先度に基づく複数クエリ実行間の動的資源調停

早水　悠登[†a)]　　合田　和生[†]　　喜連川　優[†,††]

Priority-based Dynamic Resource Arbitration among Concurrent Queries for Out-of-Order Database Engine (OoODE)

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

あらまし　著者らは，アウトオブオーダ型データベースエンジンと称する独自のデータベースエンジンの開発を進めてきた．当該データベースエンジンは、高多重の非同期入出力とスレッドを用いることにより，大規模データベースに対して選択性を有するクエリの処理を大幅に高速化する点に特長を有する．本論文では，アウトオブオーダ型エンジンにおいて複数のクエリが実行される際に，優先度に基づき各クエリに割り当て可能な資源量の動的な調整を行う手法を提案するともに、24 プロセッサコアと 160 ディスクドライブを備えた実験環境における評価実験を示すことにより，提案手法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，動的資源調停，データベースエンジン

1. はじめに

データベースシステムにおけるエンジン実装は，Ingres [1] 等初期の実装から今日のオープンソース実装 [2], [3] や商用実装 [4] に至るまで，関係代数演算をノードとし，データ流をエッジとするクエリ実行プラン木を逐次的にたどり，演算を実行する際においてストレージに格納されているデータを要する都度に，入出力を要求し，その完了を待ってから演算を実行するという処理方式を基本としてきた．本論文では，このような処理方式を**インオーダ型**と称することとする．今日のデータベースシステムは多数のプロセッサコアや高密度に実装されたストレージアレイから構成され，豊富な演算・入出力資源を有しており，パーティション並列性やパイプライン並列性に基づく並列クエリ処理やベクトル演算命令の活用等によって，選択性の乏しい，即ち全件走査を選択することに相対的な利得が認められるクエリにおいては，インオーダ型の実行方式によって，十分に高効率に資源を利用することにより高速なクエリ処理を実現するに至っている．一方，近年，データベースに格納されるデータ量は年々増加してきており，即ち，クエリが都度に全データを走査して処理することは徐々に現実的でなくなりつつあり，自ずと対象を限定した選択性のあるクエリを繰り返す機会が増えると想定され，このような選択性を有するクエリの処理においては，索引走査時における入出力のランダム性が高く，上述のようなインオーダ型の実行方式では，入出力の遅延によって全体の性能が律速されることとなり，資源利用の効率性は限定的とならざるをえない．

著者らが開発を進めている**アウトオブオーダ型データベースエンジン** OoODE は，クエリ処理の動的タスク分解によって実行並列性を最大限に抽出することにより，高多重の非同期入出力の発行とスレッドの実行によって，入出力帯域とマルチコアプロセッサの利用効率を高めることにより，特に選択性を有するクエリ処理を大幅に高速化する点に特徴を有する [5], [6].

本論文では，アウトオブオーダ型データベースエンジンにおいて複数のクエリ処理を同時に実行している際に，各クエリ処理に割り当て可能な資源量を優先度

† 東京大学生産技術研究所　〒 157-8505　目黒区駒場 4-6-1
Institute of Industrial Science, the University of Tokyo 4-6-1 Komaba, Meguro-ku, Tokyo, 157-8505 Japan
†† 国立情報学研究所　〒 101-8430　東京都千代田区一ツ橋 2-1-2
National Institute of Informatics 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430 Japan
a) E-mail: haya@tkl.iis.u-tokyo.ac.jp

に応じて実行時に調整する動的資源調停手法を提案する．従前のインオーダ型データベースエンジンでは，クエリ実行において入出力帯域を十分に活用することが困難である場合が散見されるのに対し，アウトオブオーダ型データベースエンジンは積極的な非同期入出力発行によって入出力帯域を最大限に活用することが可能である．即ち，従前のインオーダ型データベースエンジンとは異なり，クエリ実行へ割当てられる入出力帯域によって処理性能が律速される可能性が高く，特に複数クエリ実行時にはクエリ間で入出力帯域の利用が競合することが想定され，入出力帯域割当量の調整が各クエリの実行性能に与える影響は大きい．本論文の提案手法は，動的な資源調停を行うことでアウトオブオーダ型データベースエンジンが優先度に応じた柔軟なクエリ処理性能の調整を可能とする新規な取り組みである．実際のデータベースシステムにおいては，応用からの多様な性能要求に応じてクエリ処理を行うことが求められ，提案手法はアウトオブオーダ型データベースエンジンを当該システムに適用する際に，不可欠の技術と位置付けられる．本論文では，アウトオブオーダ型データベースエンジンの詳細な実行モデルを新たに構築してその実行時挙動と処理性能について考察し，実行中のクエリがそれぞれ優先度に応じた入出力性能を確保することが可能となる動的資源調停アルゴリズムの設計を示す．著者らはオープンソースのデータベースエンジン PostgreSQL を元に開発したアウトオブオーダ型データベースエンジンの試作器において，提案手法を実装し，24 プロセッサコアと 160 ディスクドライブを備えたミッドレンジ級データベースシステム環境において，当該実装を用いた評価実験を行った．本論文では，当該実験を示し，提案手法の有効性を明らかにする．

本論文の構成は次の通りである．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クエリ処理に掛かる入出力ならびに演算のスループットの定量的なモデルを議論する．3. では，当該モデルをベースとして，優先度に基づく複数クエリ実行間の資源調停方法を提案する．4. では，提案手法の評価実験結果について述べ，その有効性を示す．5. では，本研究に関連する研究を紹介し，6. において本論文をまとめる．

2. アウトオブオーダ型データベースエンジンの実行モデル

従前のインオーダ型データベースエンジンでは，クエリ実行プランに示された実行論理に従い逐次的にクエリ処理が行われる．一般にクエリ実行プランは木構造で表され，各ノードはクエリ処理に必要な演算を表す．演算を実行する際に，ストレージに格納されるデータが必要とされる場合には，インオーダ型データベースエンジンは当該データを取得するために入出力を要求し，その完了を待ってから取得されたデータに対して演算を実行する．ここで，入出力とその入出力の結果のデータに対する演算の実行との対を**演算インスタンス**と称することとする．インオーダ型データベースエンジンにおけるクエリ実行は，演算インスタンスの逐次的な実行の繰り返しとして捉えることができる．

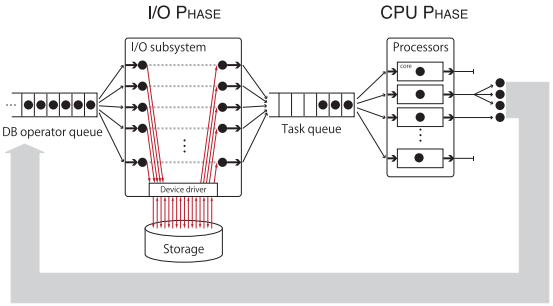


図 1　アウトオブオーダ型データベースエンジンの実行モデル
Fig. 1　Execution model of Out-of-Order Database Engine.

これに対して，アウトオブオーダ型データベースエンジンは動的なタスク分解によって，クエリ処理における演算インスタンスの実行並列性を抽出することにより，システムの資源が許す範囲において多数の演算インスタンスを重層的に実行する [6]．演算インスタンスを単位とするアウトオブオーダ型データベースエンジンの詳細な実行モデルは従前にはなく，著者らは実行モデルを構築し，それに基づき動的資源調停手法の設計を行った．以降は当該実行モデルならびに動的資源調停手法を示す．演算インスタンスの実行は，簡単には， 図 1 に示すように入出力を行うフェーズと演算を実行するフェーズから構成され，演算インスタンスの実行においては，まず入出力フェーズにおいて演算の対象となるデータを取得するための入出力を行う．この際，入出力サブシステムが同時に受付可能な入出力要求の数は有限であり，よって，この受付可能な入出力数を D とすると，入出力サブシステム内の滞留入出力要求数が D を下回った場合にのみ，アウト

オブオーダ型データベースエンジンは新たな演算インスタンスの実行を開始することができる．入出力が完了すると，次に取得されたデータに対する演算フェーズへと進む．演算フェーズにおいては，演算インスタンスにかかるデータを以って演算を実行するためのスレッドが生成され，当該スレッドがオペレーティングシステムのタスクスケジューラによりプロセッサコアに割り当てられることにより実行が駆動される．演算フェーズにおいてタプル選択，結合，集約演算等の処理が行われた結果，新たに実行すべき演算インスタンスが生じた場合にはこれらは演算インスタンスキューへと格納され，上述と同様に，随時実行される．このように，アウトオブオーダ型データベースエンジンにおける演算インスタンス実行は 図 1 に示す図面によってモデル化することができる．

1 つの演算インスタンスの入出力要求処理に要する平均時間を $1/\mu_{\mathrm{io}}$，演算に要する平均時間を $1/\mu_{\mathrm{cpu}}$，またプロセッサコア数を M とすると，各フェーズにおける最大の演算インスタンス処理スループットは次のようになる．

- 入出力フェーズスループット：$\mu_{\mathrm{io}}D$
- 演算フェーズスループット：$\mu_{\mathrm{cpu}}M$

よってアウトオブオーダ型データベースエンジンにおける最大スループット $T_{\max} = \min(\mu_{\mathrm{io}}D, \mu_{\mathrm{cpu}}M)$ である．

ブロックストレージデバイスにタプルを格納する際には，4KB から 64KB 程度のページに複数のタプルをまとめ，ページ単位で入出力を行うことが一般的である．選択性を有する分析的クエリにおいては，データベース全体に分散するわずかな割合のタプルが選択される．即ち，1 回の入出力で取得される正味のタプル数は極小数であることが想定される (注1)．今日の計算機システムにおいては，1 プロセッサコアのみでも毎秒 1000 万タプル以上の処理が実現されるに至っている一方 [7]，ストレージ帯域はエンタープライズ級のものでも最大で 100 万 IOPS 程度であることから [8], [9]，多くの場合において入出力帯域が最大スループットを律速する状況にあると考えられる．

入出力フェーズによってスループットが律速される場合，クエリ実行における演算インスタンスのスループットは $\mu_{\mathrm{io}}D$ である．また定常状態においては入出力サブシステムを飽和させるに足る数の演算インスタンスが演算インスタンスキューに充填された状態となり，入出力完了を契機として即座に次なる演算インスタンスの実行が駆動され，入出力要求が発行される．つまり，入出力サブシステム内にはほとんど常に D 個の入出力要求が滞留した状態となる．

3. 優先度に基づく複数クエリ実行間の動的資源調停

本論文の想定する入出力性能がボトルネックとなる状況において，複数のクエリが同時実行される際に，各クエリに優先度に応じて資源を割り当てる問題は，次のように規定できる (注2)．

> クエリ $Q_1, Q_2, \cdots, Q_N$ に優先度 $p_1, p_2, \cdots, p_N$ $(p_k > 0)$ が付与されている時，クエリ Q_k の入出力スループット U_k が次の条件を満たすよう，クエリ実行間の資源調停を行う．

$$\forall k. \left(U_k = \mu_{\mathrm{io}} D \frac{p_k}{\sum_{i=1}^{N} p_i} \right) \tag{1}$$

クエリ Q_k 実行における入出力スループット U_k は，クエリ Q_k に関して入出力サブシステム内に滞留する入出力要求を D_k とすると $U_k = \mu_{\mathrm{io}} D_k$ であるため

$$\forall k. \left(D_k = D \frac{p_k}{\sum_{i=1}^{N} p_i} \right) \tag{2}$$

を満たすことができればよい．

式 (2) の条件を制御目標として，各クエリ実行に割当てられる入出力資源の調停を行うアルゴリズムを図 2 に示す．ここでは，同時実行可能なクエリ数は最大 N_{max} であり，N_{max} 個のクエリ実行スロットが存在するものと仮定している．Q_k は k 番目のクエリ実行スロットを表し，当該スロットにおいて実行されるクエリの優先度を p_k とする．ただし実行中のクエリが存在しないクエリ実行スロット Q_k に関しては $p_k = 0$ とする．アウトオブオーダ型データベースエンジンは，入出力サブシステムが新たに入出力要求を受付可能となる度に (注3)，各クエリ実行スロット Q_k に対して

(注1)：ただし選択軸によってタプルがクラスタ化されている場合を除く．

(注2)：演算性能がボトルネックとなる状況においても，同様にして問題設定を行うことが可能である．

(注3)：このイベントは，オペレーティングシステムのシグナルによる入出力完了通知や，入出力管理用のポーリングスレッドを用いるなどの手段によって検出可能である．

```
Initialization
for i ← 1, ··· , N_max do
    D_i ← 0
end for

k : an index of a target query slot Q_k
procedure QueryTryIO(k)
    if IsQueueEmpty() or D_k ≧ D×p_k/∑_{i=1}^{N_max} p_i then
        return
    else
        o ← PopOperatorInstance(k)
        IssueAsyncIO(o)              ▷ Issue I/O request of o
        D_k ← D_k + 1
    end if
end procedure

k : an index of a target query slot Q_k
o : an operator instance whose I/O is finished
procedure QueryFinishIO(k, o)
    D_k ← D_k − 1
    StartThread(o)
end procedure
```

図 2 優先度に基づく入出力帯域の動的調停アルゴリズム
Fig. 2 Priority-based dynamic I/O bandwidth arbitration algorithm.

QueryTryIO を呼び出し，新たな演算インスタンスの実行を開始することで，入出力要求の発行を試みる．この際，Q_k が既に発行中で未だ完了に至っていない入出力数は D_k によって捕捉されており，当該アルゴリズムでは D_k が $D \times p_k / \sum_{i=1}^{N} p_i$ を下回る場合のみ新たな演算インスタンス o に関する入出力要求を発行する．入出力要求の完了が検出されると，アウトオブオーダ型データベースエンジンは QueryFinishIO を呼び出し，D_k を更新した上で，入出力が完了した演算インスタンス o の演算を行うためのスレッドを生成する．

実行中の全てのクエリが定常状態にあり，各々の演算インスタンスキューが十分多くの演算インスタンスを格納している場合， 図 2 に示すアルゴリズムが式 (2) に示す条件を満たすよう動作することは自明である．一方，クエリ Q_k のみが実行開始直後や終了直前などの過渡状態にあるときには，演算インスタンスキューに十分な数の演算インスタンスが格納されておらず，$D_k < p_k / \sum_{i=1}^{N} p_i$ となる可能性がある．即ち，$\sum_i D_i < D$ となり一時的にシステム全体の入出力帯域の利用率低下が発生する．クエリ Q_k 以外は定常状態にあり入出力帯域を最大まで利用できていたとすると，時刻 t_0 から t_1 に至る間の損失である入出力量 $L(p_k|t_0, t_1)$ は次のようになる．

表 1 実験システム諸元
Table 1 Specifications of the experimental system.

Server: IBM x3850 M2	
Processor	4x Intel Xeon X7460 (4p24c)
Memory	32GB DDR2 DIMMs
HBA	8x Emulex Fibre Channel HBA
OS	RedHat Enterprise Linux 5.8 (Linux kernel 2.6.18)
Storage: IBM DS5300	
RAID controller	Dual active
Cache memory	2x 4GB
Host interface	8x 4Gbps Fibre Channel
Disk drive	160x 450GB 15,000rpm FC HDD
RAID volume	20x RAID5(7D+1P) volumes

$$
\begin{aligned}
L(p_k|t_0, t_1) &= \mu_{\mathrm{io}} \int_{t_0}^{t_1} \left(D \frac{p_k}{\sum_{i=1}^{N} p_i} - D_k(t) \right) \mathrm{d}t \\
&= \frac{(\mu_{\mathrm{io}} D \Delta t) p_k}{p_k + \overline{p_k}} - \int_{t_0}^{t_1} D_k(t) \mathrm{d}t \quad (3) \\
&\left(\text{ただし } \overline{p_k} \equiv \sum_{i \neq k}^{N} p_i, \Delta t = t_1 - t_0 \right)
\end{aligned}
$$

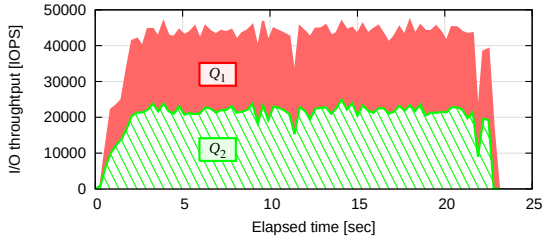
p_k が大きいほど $L(p_k|t_0, t_1)$ の第一項は大きくなるため，優先度の高いクエリほどその過渡状態における損失が大きくなることがわかる．

前述の考察は，システムの入出力スループットを最大限に活用するという観点からすると，クエリ Q_k が過渡状態にあるときにはクエリ $Q_l (l \neq k)$ は $D \times p_l / \sum_{i=1}^{N} p_i$ より多くの入出力を発行したほうが良いことを示唆する．しかしながら，実際に計算機システムで用いられるストレージの多くは系内に滞留する入出力要求数が多いほど平均応答時間が長くなる傾向にあるため，このような戦略はクエリ Q_k が定常状態に至るまでに要する時間を増大させることが懸念される．特に本方式では応答性要求の高い (優先度の高い) クエリほどその実行時間を優先的に短縮することを旨とする．そのため，クエリ Q_k が定常状態に至るための時間が，他のクエリ Q_l の干渉によって長くなることを抑制する代償として損失 $L(p_k|t_0, t_1)$ が生じると考えることができる．

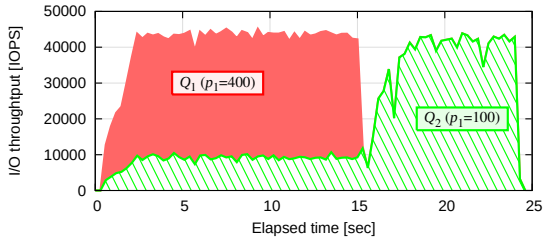
4. 評 価 実 験

本論文の提案する動的資源調停手法を評価するため，オープンソースデータベース管理システム PostgreSQL [3] をベースとするアウトオブオーダ型データベースエンジン試作機 [10] に当該手法を実装し，評価実験を実施した．

実験環境に用いたサーバ・ストレージの諸元を 表 1



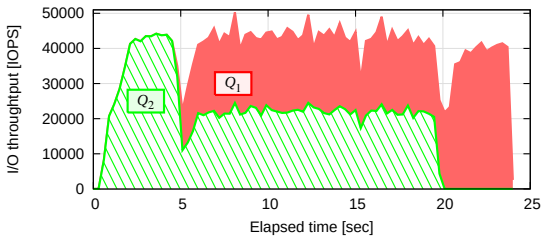
(a) 動的資源調停なし



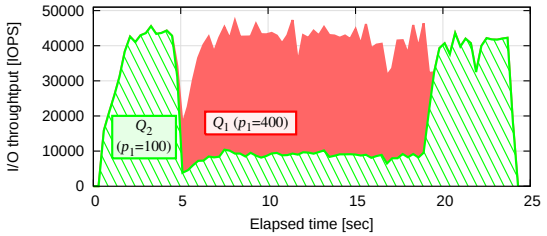
(b) 動的資源調停あり (優先度 $p_1 = 400, p_2 = 100$)

図 3　2 つのクエリを同時に実行開始した際の入出力スループット

Fig. 3　I/O throughput behaviour when two queries starts at the same time.



(a) 動的資源調停なし



(b) 動的資源調停あり (優先度 $p_1 = 400, p_2 = 100$)

図 4　2 つのクエリを異なる時刻に実行開始した際の入出力スループット

Fig. 4　I/O throughput behaviour when two queries starts at the different time.

に示す．IBM DS5300 においては，8 HDD ごとに 7D+1P の RAID グループが編成され，1 つの RAID グループあたり 1LU，合計 20LU を構成した．20LU はソフトウェア RAID0 によってストライピングされており，当該領域を ext4 ファイルシステムによってフォーマットしてデータベース領域として使用した．

評価実験用データベースとしては，TPC-H ベンチマーク [11] の dbgen により Scale Factor = 1000 でデータ生成を行い，PostgreSQL へデータロードしたものを使用した．事前の基本入出力性能測定より本実験環境では $D = 480$ として入出力発行数の抑制を行った．

4.1　動的資源調停の挙動検証

本実験では，複数のクエリ処理が同時実行される際の，各クエリの入出力スループット測定を実施した．同時実行されるクエリはいずれも TPC-H Q.3 の選択率が 1.8×10^{-3}%となるよう調整されたものであり，それぞれ相異なるタプルを取得するよう選択条件が設定されている．

2 つのクエリ Q_1, Q_2 を同時に実行開始した際の振舞いを 図 3 に示す．優先度に基づく動的資源調停が行われない場合，図 3 (a) に示すように各クエリ実行はほぼ同じ入出力スループットを示し，ほぼ同時刻に実行を完了した．一方，図 3 (b) に示すように Q_1, Q_2 の優先度をそれぞれ $p_1 = 400, p_2 = 100$ として動的資源調停を行った場合には，定常状態において Q_1 の入出力スループットは Q_2 の約 4 倍となり，Q_1 が先に実行を完了した．Q_1 の実行完了後，Q_2 の入出力スループットが増加し，実行開始から 25 秒後に実行を完了した．この結果より，動的資源調停により優先度の高い Q_1 に対して入出力資源がより多く割り当てられたことで，その実行時間が短縮されることが確認できる．

次に，クエリ Q_2 実行開始の約 5 秒後にクエリ Q_1 の実行を開始した際の，各クエリの入出力スループットを 図 4 に示す．動的資源調停を行わない場合，クエリ Q_1 の実行が開始され定常状態に至ると，Q_1, Q_2 それぞれの入出力スループットはほぼ同程度であり，Q_2 が実行を完了した後に Q_1 が実行を完了した．一方，優先度を $p_1 = 400, p_2 = 100$ として動的資源調停を行った場合，図 4 (b) に示すように定常状態における Q_1 の入出力スループットは Q_2 に対して約 4 倍となり，Q_1 が先に実行を完了し，その後に Q_2 が実行を完了した．このように，異なる時刻に開始される複数クエリに関しても動的な資源調停が行われ，優先度に応じた入出力資源が割り当てられていることがわかる．

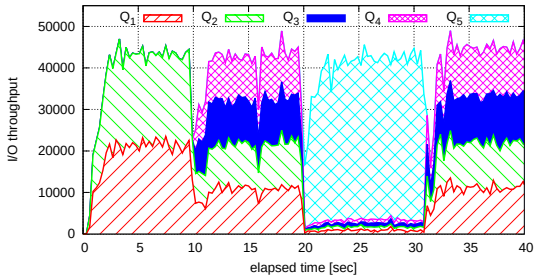


図 5　$Q_1, \cdots, Q_5$ の実行における入出力スループット
Fig. 5　I/O throughput behaviour with five queries: $Q_1, \cdots, Q_5$.

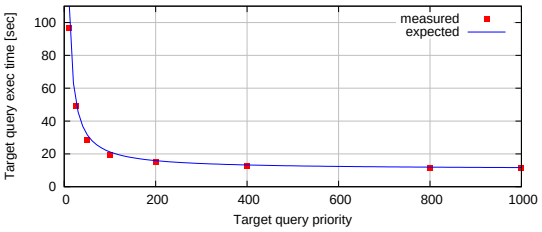


図 6　対象クエリ TPC-H Q.3，背景クエリ TPC-H Q.3（優先度 100）のときの対象クエリ実行時間
Fig. 6　Execution time of TPC-H Q.3 with another execution of TPC-H Q.3 (priority: 100) as a background query.

さらに，より多数のクエリが実行される状況において，各クエリの入出力スループット測定を実施した．測定に際しては，前述の実験と同様に，TPC-H Q.3 の選択率が 1.8×10^{-3}%となるよう調整されたクエリ Q_1, Q_2, Q_3, Q_4, Q_5 を用意し，次のような時系列で実行を開始した．

- 時刻 $t = 0$ にクエリ Q_1, Q_2 を実行開始
- 時刻 $t = 10$ にクエリ Q_3, Q_4 を実行開始
- 時刻 $t = 20$ にクエリ Q_5 を実行開始

ただし，Q_1, Q_2, Q_3, Q_4 の優先度はそれぞれ 100，Q_5 の優先度は 4,000 とした．

各クエリの入出力スループットを 図 5 に示す．優先度が等しいクエリ Q_1, Q_2, Q_3, Q_4 に関しては，定常状態において各クエリの入出力スループットはほぼ同程度であった．優先度 4,000 のクエリ Q_5 の実行が開始されると，定常状態においてその入出力スループットは全体の約 90%を占めた．この結果から，多数のクエリが随時実行を開始する状況においても，動的資源調停により優先度に応じて各クエリの入出力スループットが調整されていることが確認できる．

4.2　クエリ優先度と実行時間

本実験では，クエリの優先度を変化させた際の，実行時間の変動について測定を行った．測定に際しては，背景クエリとして TPC-H Q.3 類似クエリを優先度 100 で予め実行を開始し，当該クエリの入出力スループットが定常状態に達した後に，測定対象のクエリ実行を開始し，そのクエリ実行時間を測定した．測定対象のクエリとしては，(1) TPC-H Q.3 類似クエリ（選択率 1.6×10^{-3}%），(2) TPC-H Q.8 類似クエリ（選択率 0.8×10^{-3}%）を用いて，それぞれについて優先度を 10 から 1000 まで変化させ，実行時間を測定した．ただし，測定対象クエリが実行を完了した後に背景クエリが実行を完了するよう，優先度に応じて適宜背景クエリの選択率調整を行った．

まず TPC-H Q.3 を測定対象クエリとした際の，優先度ごとのクエリ実行時間を 図 6 に示す．凡例の measured は測定値を，expected は期待されるクエリ実行時間を表す．ただし，期待されるクエリ実行時間は，対象クエリの単独実行時の実行時間を t_{single}，優先度を p としたとき

$$t_{expected} = \frac{p + 100}{p} t_{single} \tag{4}$$

とした． 図 6 からわかるように，対象クエリ優先度が 100 以下の場合には，実行時間の実測値が期待される実行時間を 10%程度下回る結果となった．背景クエリと対象クエリが選択するタプルは論理的に重複の無いよう選択条件が設定されているが，物理的には異なるタプルが同一ページに含まれている場合が存在することや，両クエリ間で共通する索引ページの取得を行う場合等，入出力対象データには共有領域が存在するため，これによって期待されるよりも対象クエリの実行が高速になったものと推察される．優先度が 200 以上の場合には，ほぼ期待値と同等の実行時間であった．

次に，対象クエリを TPC-H Q.8 としたときの，優先度ごとのクエリ実行時間を 図 7 に示す．対象クエリ優先度 100 以下においては，実測値が期待される実行時間を 15%から 20%程度下回る結果となった．TPC-H Q.8 は Q.3 と比べて結合段数が多く，入出力全体のうち索引に対する入出力の割合が高い．そのため Q.3 と比較して入出力がバッファヒットしやすく，入出力資源の制限による性能低下の程度が小さいため，低優先度の場合における実行時間が期待値を下回ったものと考えられる．優先度が 200 以上の場合には，実行時間の実測値は期待される実行時間と同等であった．

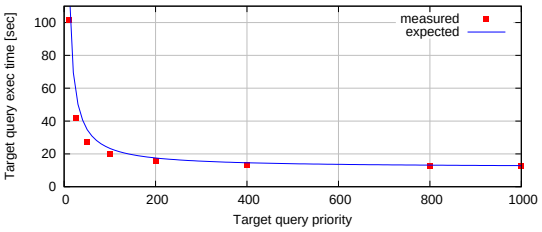


図 7　対象クエリ TPC-H Q.8，背景クエリ TPC-H Q.3（優先度 100）のときの対象クエリ実行時間
Fig. 7　Execution time of TPC-H Q.8 with TPC-H Q.3 (priority: 100) as a background query.

以上の結果より，クエリ優先度を変化させることにより，動的資源調停によって割り当てられる資源量が優先度に応じて変化し，実行時間を調整することが可能であることが確認された．

5. 関 連 研 究

データベースシステムにおける複数のクエリ処理の同時実行に関しては，クエリ最適化の際に複数クエリ間で共通する演算を抽出し，同一演算の重複実行を削減することで効率化を図る手法が [12]〜[16] をはじめとして数多く提案されている．またクエリ処理の実行時に共通演算検出を行い，重複実行を削減する手法として，Candea らによるスタースキーマにおけるファクト表走査の共有 [17] や，Harizopoulos らによるクエリ処理パイプラインの各ステージにおける共通演算検出 [18] などが提案されている．これらの取り組みは演算実行の総量を削減することを目的としており，各クエリが優先度に応じた入出力性能が確保できることを目的とする本論文とは異なる．

優先度に基づくクエリ処理のスケジューリングに関しては，リアルタイムデータベースシステムにおけるトランザクション処理を想定した手法が提案されており，Carey らによる入出力スケジューリング・バッファ管理手法 [19] や，Brown らによる各クラスの平均応答時間目標を満たすメモリ管理手法 [20]，Pang らによるワークロードに適応したメモリ管理手法 [21] などがある．これらはいずれも数段階のトランザクション締切要求レベルを前提とし，締切に間に合わないトランザクション数を最小化することを目的とする．一方，本論文は各クエリが優先度に応じた入出力性能を確保できることを目的としており，この点について異なる．

オペレーティングシステムにおける資源管理に優先度を用いる取り組みとして，松本らによる NUMA 型並列計算機クラスタの資源管理方式 [22] が提案されている．当該方式は，各ユーザプログラムへの資源割当て公平性，およびクラスタ全体の処理効率向上を目的としたものであり，これを実現する方法として各ユーザプログラムの要求する演算・記憶資源配置の局所性制約から優先度を算出し，スケジューリングを行うものである．また一野らはプロセス優先度に基いてパケット受信処理を最適化する手法 [23] を提案している．当該手法は，ネットワークが高負荷でありパケット廃棄が頻発する状況において，宛先プロセスの優先度を考慮して廃棄パケットを選択することでシステム全体におけるパケット廃棄率の低減を狙うものである．これらの提案は，資源割当てのスケジューリングによって系全体の処理性能が改善可能である場合に，優先度に基づく制御によって性能改善を図るものである．これに対し，本論文の提案手法はユーザから与えられた優先度に従い，各クエリが優先度に応じた処理性能を確保することを目的とした資源調停を行うものであり，目的を異にするものである．

6. おわりに

本論文では，アウトオブオーダ型データベースエンジンにおいて，各クエリに設定した優先度に基づき，複数のクエリ実行間で実行時に利用可能な資源量の調停を行う手法を提案した．著者らの開発したアウトオブオーダ型データベースエンジンの試作器において当該手法を実装し，24 プロセッサコアと 160 ディスクドライブを備えたハードウェア環境において，1TB 規模の TPC-H データセットを用いた評価実験を行った結果，各クエリの入出力スループットを優先度に応じてバランスさせることができることを示した．今後はより多様なワークロード特性を考慮した資源調停方式へ拡張すると共に，アプリケーションのサービスレベルを含めた包括的な優先度設定のポリシについて研究を進めてゆきたい．

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) 24·8381 の助成により行われた．

文　献

[1] M. Stonebraker, G. Held, E. Wong, and P. Kreps, “The design and implementation of ingres,” ACM Trans. Database Syst., vol.1, no.3, pp.189–222, Sept. 1976.

[2] Oracle Corporation, “Mysql : The world’s most popular open source database,” http://www.mysql.com/, 2014.

[3] The PostgreSQL Global Development Group, “Postgresql: The world’s most advanced open source database,” http://www.postgresql.org/, 2014.

[4] M. Canim, G.A. Mihaila, B. Bhattacharjee, K.A. Ross, and C.A. Lang, “An object placement advisor for db2 using solid state storage,” Proc. VLDB Endow., vol.2, no.2, pp.1318–1329, Aug. 2009.

[5] 喜連川優，合田和生，“アウトオブオーダー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，June 2009.

[6] 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,” 電子情報通信学会論文誌 D，vol.J97-D，no.4，pp.729–737，April 2014.

[7] M.-C. Albutiu, A. Kemper, and T. Neumann, “Massively parallel sort-merge joins in main memory multi-core database systems,” Proc. VLDB Endow., vol.5, no.10, pp.1064–1075, June 2012.

[8] G. Crump, “Lab report: Designing a 2 million+ iops architecture,” Sept. 2013.

[9] R. McNelly, “A big step forward in storage,” July 2014.

[10] 早水悠登，合田和生，喜連川優，“アウトオブオーダ型クエリ実行に基づくプラグイン可能なデータベースエンジン加速機構,” 情報処理学会論文誌 データベース （TOD62），vol.7，no.2，pp.104–116，June 2014.

[11] Transaction Processing Performance Council, “http://www.tpc.org/”.

[12] S. Finkelstein, “Common expression analysis in database applications,” Proc. the 1982 ACM SIGMOD Int. Conf. on Management of Data, pp.235–245, SIGMOD ’82, 1982.

[13] T.K. Sellis, “Multiple-query optimization,” ACM Trans. Database Syst., vol.13, no.1, pp.23–52, March 1988.

[14] E.J. Shekita, H.C. Young, and K.-L. Tan, “Multi-join optimization for symmetric multiprocessors,” Proc. the 19th Int. Conf. on Very Large Data Bases, pp.479–492, VLDB ’93, 1993.

[15] H. Lu and K.-L. Tan, “Batch query processing in shared-nothing multiprocessors,” Proc. the 4th Int. Conf. on Database Systems for Advanced Applications (DASFAA), pp.238–245, 1995.

[16] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe, “Efficient and extensible algorithms for multi query optimization,” Proc. the 2000 ACM SIGMOD Int. Conf. on Management of Data, pp.249–260, SIGMOD ’00, 2000.

[17] G. Candea, N. Polyzotis, and R. Vingralek, “Predictable performance and high query concurrency for data analytics,” The VLDB Journal, vol.20, no.2, pp.227–248, 2011.

[18] S. Harizopoulos, V. Shkapenyuk, and A. Ailamaki, “Qpipe: A simultaneously pipelined relational query engine,” Proc. the 2005 ACM SIGMOD Int. Conf. on Management of Data, pp.383–394, SIGMOD ’05, 2005.

[19] M.J. Carey, R. Jauhari, and M. Livny, “Priority in dbms resource scheduling,” Proc. the 15th Int. Conf. on Very Large Data Bases, pp.397–410, VLDB ’89, 1989.

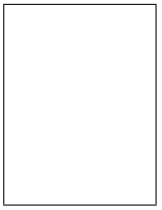
[20] K.P. Brown, M.J. Carey, and M. Livny, “Managing memory to meet multiclass workload response time goals,” Proc. the 19th Int. Conf. on Very Large Data Bases, pp.328–341, VLDB ’93, 1993.

[21] H. Pang, M.J. Carey, and M. Livny, “Multiclass query scheduling in real-time database systems,” Knowledge and Data Engineering, IEEE Transactions on, vol.7, no.4, pp.533–551, Aug. 1995.

[22] 尚　松本，敬　平木，“汎用超並列オペレーティングシステム sss-core のメモリベース通信機能,” 全国大会講演論文集，vol.53，no.1，pp.37–38，sep 1996. http://ci.nii.ac.jp/naid/110002887437/

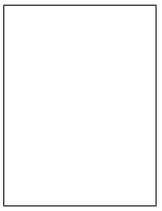
[23] 浩太朗一野，修二楢崎，“スケジューラとの協調によるプロセス優先度に基づくパケット受信処理手法,” 情報処理学会論文誌，vol.49，no.8，pp.2862–2872，aug 2008. http://ci.nii.ac.jp/naid/110007970175/

（平成 xx 年 xx 月 xx 日受付）



早水　悠登

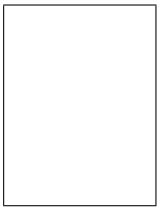
平成 21 年，東京大学工学部電子情報工学科卒業．平成 23 年，同大学院情報理工学系研究科電子情報学専攻修士課程修了．平成 26 年，同大学院情報理工学系研究科電子情報学専攻博士課程単位取得満期退学．日本学術振興会特別研究員 DC2 を経て，現在，東京大学生産技術研究所特任研究員．データベースシステムに関する研究に従事．情報処理学会会員．



合田　和生

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．



喜連川　優　（正員：フェロー）

1983 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，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．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，電子情報通信学会ならびに情報処理学会フェロー．

Abstract Out-of-Order Database Engine (OoODE), which we have developed, has a unique capability of utilizing asynchronous I/Os and threads, then being able to significantly speed up the processing of a selective query over a large-scale database. This paper proposes a priority-based dynamic resource arbitration method among concurrent queries for such a database engine. We verify the effectiveness of the proposal by showing the experiment that we conducted on a database server with 24 processor cores and 160 disk drives.

Key words OoODE, Out-of-Order Execution, dynamic resource arbitration, database engine