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

清水 晃† 徳田 晴介† 田中 美智子† 茂木 和彦¶ 合田 和生‡ 喜連川 優‡

† ㈱日立製作所 中央研究所 〒244-0817 神奈川県横浜市戸塚区吉田町 292 番地
¶㈱日立製作所 情報・通信システム社 IT プラットフォーム事業本部
〒244-0817 神奈川県横浜市戸塚区吉田町 292 番地
‡東京大学 生産技術研究所 〒153-8503 東京都目黒区駒場 4-6-1
E-mail: †{akira.shimizu.wv, seisuke.tokuda.gk, michiko.tanaka.ry}@hitachi.com,
¶kazuhiko.mogi.uv@hitachi.com, ‡{kgoda, kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし 本論文では，アウトオブオーダ型データベースエンジン（OoODE）のタスク管理機構の一実装方式の評価について報告する．当該データベースエンジンは，問合せ処理をアンフォールドすることにより多数のプロセッサコアを活用し，また，複数の非同期 I/O を同時に発行することで多数のディスクドライブを活用し，これにより性能向上を目指すものである．アンフォールドされた処理であるタスクの数は膨大となるため，性能向上実現のためにタスク管理機構が重要となる．本論文では，OoODE のタスク管理機構をプロトタイピングし，評価した結果を報告する．

キーワード OoODE，アウトオブオーダ型実行，データベースエンジン，問合せ処理，タスク管理

Akira SHIMIZU†, Seisuke TOKUDA†, Michiko TANAKA†, Kazuhiko MOGI¶,
Kazuo GODA‡ and Masaru KITSUREGAWA‡

† Central Research Laboratory, Hitachi, Ltd.
292, Yoshida-cho, Totsuka-ku, Yokohama-shi, Kanagawa-ken, 244-0817 Japan
¶ IT Platform Division Group, Information & Telecommunication Systems Company, Hitachi, Ltd.
292, Yoshida-cho, Totsuka-ku, Yokohama-shi, Kanagawa-ken, 244-0817 Japan
‡ Institute of Industrial Science, The University of Tokyo Komaba 4-6-1, Meguro-ku, Tokyo, 153-8505 Japan
E-mail: †{akira.shimizu.wv, seisuke.tokuda.gk, michiko.tanaka.ry}@hitachi.com,
¶kazuhiko.mogi.uv@hitachi.com, ‡{kgoda, kitsure}@tkl.iis.u-tokyo.ac.jp

1. はじめに

近年，検索されうるデジタルデータ量が増加しており，このデジタルデータに対する分析のニーズが高まっている．IT 専門調査会社 IDC の報告[1]によると，1 年間における生成および複製されるデータ量 (デジタルユニバース) は，2012 年において 2.8ZB (ゼッタバイト) にのぼると予想されている．その中で分析に活用できるデータ量 (タグ付けされたデータの量) は，643 EB (エクサバイト) にものぼると予想されている．今後，デジタルユニバースは，2 年ごとに倍増し，2020 年には 40ZB に到達すると予想されている．さらに，このようなデータに対する分析ニーズが高まっていると報告されている．

このような大規模なデータに対する分析のニーズが高まっている状況の中，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) という，新しい実行原理により問合せ処理を行う DB エンジンの構想を提案している[2]．従来型の DB エンジンは，レコードのフェッチから当該レコードの処理に至る一連の動作をプログラムされた決定的な順序に基づき問合せ処理を進めていく．これに対し，OoODE は，互いに独立な一連の処理を複数駆動し，実行可能な順に処理していくことで問合せ処理を進めていく．以下，問合せで行われる処理のうち互いに独立な処理をタスクと呼ぶ. OoODE は問合せ処理を実行時に複数のタスクを駆動し，複数の非同期 I/O をストレージ装置に発行する．タスクの単位は最も細かい単位でレコード単位の処理となる．このため，問合せで指定される条件にもよるが，多数のタスクを生成することが可能である．この結果，ストレージ装置へは大量の非同期 I/O が発行され，ストレージ装置の性能を引き出すことができると考える．

OoODE は, プロセッサのマルチコア化と外部ストレージシステムの高性能化といったハードウェアの変革の中，このようなハードウェア資源を効率的に活用することを目指した DB エンジンである．近年，マルチコアプロセッサが一般的になり，大規模なサーバは複数のプロセッサを搭載することで合計が 100 コア以上となるモデルが存在する．一方，ストレージシステムも大規模で高性能なシステムが存在する．このため，OoODE では，100 コアレベルのサーバにおいて，多数のタスクを複数のコアで動作させることが必要となる．

ここで課題となるのが，実行時に動的に生成されるタスクをどのように複数のコアで実行するかということである．タスクをスレッドで実現すると, OS のスレッドスケジューリングや管理オーバヘッドの影響が懸念される．

そこで，我々は，スレッドスケジューリングの影響や管理オーバヘッドを小さくするために，コア数程度のスレッドを生成し，スレッド内で複数のタスクを動かすことで，全体として多数のタスクを実現するスレッド内複数タスク方式を検討した．そして，スレッド内複数タスク方式のタスク管理機構のプロトタイプ実装し，その評価を行った．

以下，第 2 章ではプロトタイプ実装したスレッド内で複数のタスクを動かすタスク管理機構について述べる．第 3 章では, 該タスク管理機構の評価結果を示す．最後に第 4 章，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タスク管理機構

OoODE は問合せ処理を実行時に複数のタスクを駆動する．タスクの処理単位は様々あるが，最も細い単位はレコード処理である．OoODE が特に高い性能を発揮するのは，アウトオブオーダ実行により高速化されたネストループ結合がハッシュ結合よりも有利となる選択率の範囲である[2]．近年の大規模なデータベースであればレコードの件数が数億件となることは珍しくなく, OoODE の適用範囲となる選択率における問合せのタスク数は膨大となると考える．

OoODE におけるタスクは, 実行時に動的に生成されるといった特徴がある．また，最も細かいものでレコード処理の単位であるため，処理が小さく，すぐに終了する．つまり, OoODE におけるタスクは高い頻度で生成と消滅を繰り返すといった特徴がある．

ここで課題となるのが，実行時に動的に生成される膨大なタスクを，どのように複数のコアで実行するかという点である．

タスクを実行する手段としてスレッドがある．スレッドを複数生成し，スレッドが 1 つのタスクを実行することで，複数のコアを使うことが可能になる．しかし，タスクは生成と消滅を高頻度で繰り返すため，スレッドを使った場合には, OS のスレッドスケジューリングの影響や管理オーバヘッドの増大が懸念される．

そこで, OS のスレッドスケジューリングや管理オーバヘッドの影響を小さくするために，スレッドはコア数程度を想定し，スレッド内に複数タスクを動かす方式を考える．本方式をスレッド内複数タスク方式と呼ぶ. スレッド内複数タスク方式では, あるタスクが I/O 完了を待っている間に別のタスクが処理できるようにするため, タスクが行う I/O には非同期 I/O を用いる．プロセスが I/O の完了を検知した場合, I/O に対応するタスクを再開させる．これにより，スレッド内で複数のタスクが並列に動作することを可能にする．

新たなタスクを別のスレッドで開始させる際に，スレッド間でデータの受け渡しが必要となる．タスクの生成は I/O 処理に伴い発生するため, I/O 性能に比例してタスクは生成されると考えられる．その際，データの受け渡しに伴うロック競合も頻発する可能性があり，それに伴う性能低下が懸念される．このロック競合を回避するために，送信側スレッドと受信側スレッドが 1 対 1 となるよう管理データ構造を複数作成することとした．

3. アウトオブオーダ型データベースエンジンのタスク管理機構プロトタイプと評価

スレッド内複数タスク方式のタスク管理機構の性能を調査するために，スレッド内複数タスク方式のタスク管理機構を実装したアウトオブオーダ型データベースエンジンのプロトタイプを作成し，スレッド数またはスレッド内タスク数を変化させ，評価を行った．

スレッド内複数タスク方式は，前述したようにロック競合の回避を狙って設計している．ロック競合の発生頻度が十分小さければ，I/O あたりの CPU 処理量は一定のままで，I/O 性能と CPU 利用率は線形の関係となる．一方，ロック競合が発生するとロック解消まで CPU サイクルを消費するため，発生頻度が上昇するにつれ CPU 利用率が I/O 性能に対して線形以上に上昇してしまう．このため本評価では， I/O 性能と CPU 利用率が線形であることを確認することで，I/O 性能が上昇してもロック競合の発生頻度が上昇していないことを確認する．

プロトタイプの評価には、以下に記すハードウェアおよび問合せを用いた．

サーバには，Intel Xeon X7560 (8 物理コア/ソケット) を 8 ソケットと 1024GB のメモリを搭載した日立製サーバ BladeSymphony BS2000 1 台を用いた．コア数が多い環境下での測定を行うため, Xeon の Simultaneous

Multi-Threading 機能を ON にし，論理コア数を 128 とした．OS には RedHat Enterprise Linux 6.1 を用いた．

ストレージには，48 台の SAS HDD (15krpm, 600GB) を利用する日立製ストレージ AMS2500 を 4 サブシステム用い，合計 192 台の HDD を利用した．ストレージの HDD は RAID5 を組み，RAID から切り出された Logical Unit (LU) は Logical Volume Manager (LVM) によりストライピングした．これにより，4KB の DB ページを 192 台の HDD に均一に配置した．なお，サーバとストレージは，16 本の Fibre Channel (FC) にて接続した．

データベースには TPC-H ベンチマーク[3]で利用されている表定義およびレコードを用いた．測定には，図 1 に示す外表側が索引を使う二表のネステッドループ結合とグループ化を行う問合せを用いた．なお，データベースサイズは Scale Factor 1000 のデータを用いた．問合せで用いる customer 表は 1.5 億件で，orders 表は 15 億件のデータである．

```
SELECT c_nationkey, sum(o_totalprice), count(*)
FROM customer, orders
WHERE c_custkey = o_custkey
      and c_mktsegment = 'AUTOMOBILE'
      and c_acctbal > 9980
GROUP BY c_nationkey;
```

図 1 測定で用いた SQL 文(NL 結合(CxO))

測定は，プロトタイプ起動直後の状態から問合せを処理する実行を 3 回行った．以下，計測値を以下のように定義する．問合せの 1 件目の結果取得要求を送ってから全件結果取得完了までの経過時間を処理時間とする．1 秒あたりの I/O 処理数を示す iostat コマンドの r/s を DB ページを配置した全 LU 分合計し，プロトタイプ動作中の時間平均値を IOPS 性能とする． 128 コアで 100%となる mpstat コマンド ALL 出力の%usr と%sys，%irq，%soft を合計し，プロトタイプ動作中の時間平均値を CPU 利用率とする．なお，mpstat コマンドおよび iostat コマンドの表示間隔は 2 秒とする．また，処理時間は，従来型の DB エンジンと同等の処理を行う順序実行の処理時間を分母とした相対値 (相対処理時間) とする．IOPS 性能も同様に順序実行における IOPS 性能を 1 とする相対値 (相対 IOPS 性能) とする．

3.1. スレッド数の変化による性能評価

スレッド数の変化による性能を評価するために，スレッド内タスク数を 16 に固定して，スレッド数を 1 から論理コア数である 128 まで変化させた測定を実施した．図 2 は相対処理時間を示す．図 3 は相対 IOPS 性能を示す．図 4 は CPU 利用率を示す．

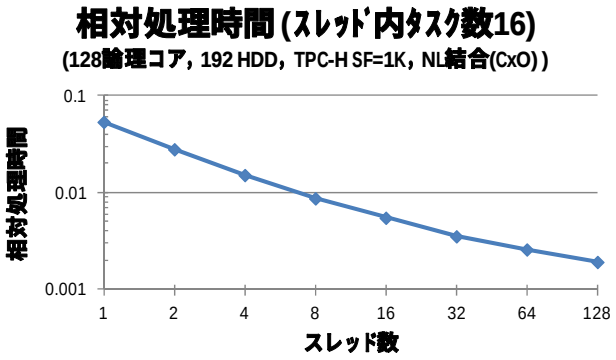

図 2 相対処理時間(ｽﾚｯﾄﾞ内ﾀｽｸ数 16)

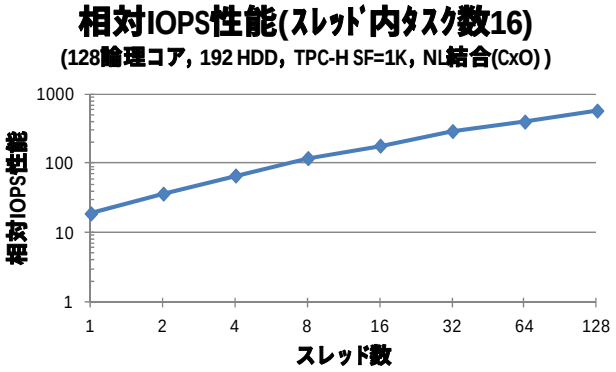

図 3 相対 IOPS 性能(ｽﾚｯﾄﾞ内ﾀｽｸ数 16)

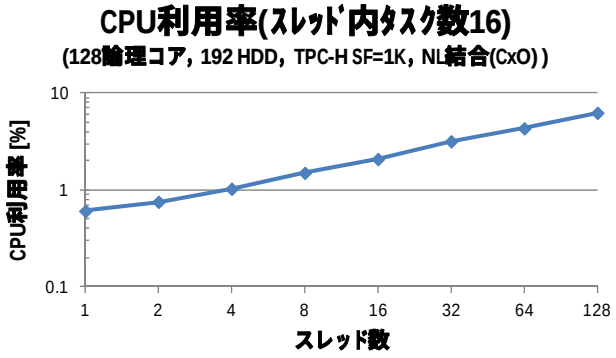

図 4 CPU 利用率(ｽﾚｯﾄﾞ内ﾀｽｸ数 16)

図 2 および図 3 より，スレッド数に応じてストレージシステムの IOPS 性能を引き出し，その結果処理時間が短縮できていることが確認できる．図 4 を見ると，IOPS 性能の向上に伴い CPU 利用率が増加しているが，2048 タスクが動作している際の CPU 利用率は 6.3%であった．

ロック競合による CPU 利用率の上昇の有無を確認するために，相対 IOPS 性能を横軸に，CPU 利用率を縦軸にプロットしたグラフを図 5 に示す．

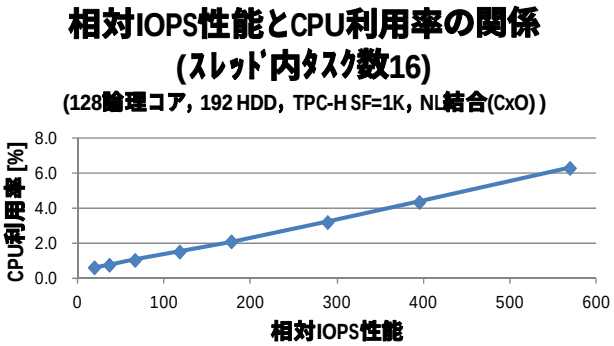

図 5 相対 IOPS 性能と CPU 利用率の関係 (ｽﾚｯﾄﾞ内ﾀｽｸ数 16)

図 5 より相対 IOPS 性能に対して CPU 利用率がほぼ線形に増加していることが確認できる．これにより，本測定においては IOPS 性能が上昇してもロック競合の発生頻度が上昇していないと考える．

また，線形近似した際の縦軸との切片は約 0.4%であった．プロトタイプ以外のプロセス (iostat コマンドや mpstat コマンド) のみを動作させた際の CPU 利用率が約 0.4%であったため，0.4%はプロトタイプ以外の CPU 利用率と考えられる．このため，本プロトタイプにおける固定オーバヘッドはほぼないものと考えられる．

3.2. スレッド内タスク数の変化による性能評価

スレッド内タスク数の変化による性能を評価するために，論理コア数と同数の 128 スレッドに固定し，スレッド内タスク数を 1 から 16 まで変化させた測定を実施した．図 6 は相対処理時間を示す．図 7 は相対 IOPS 性能を示す．図 8 は CPU 利用率を示す．

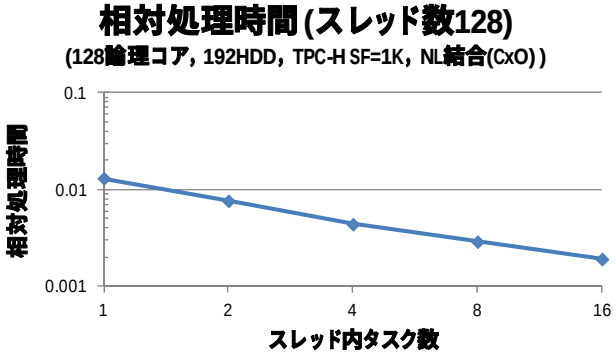

図 6 相対処理時間(スレッド数 128)

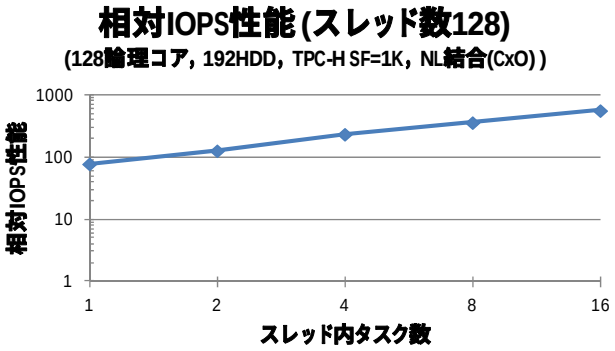

図 7 相対 IOPS 性能(スレッド数 128)

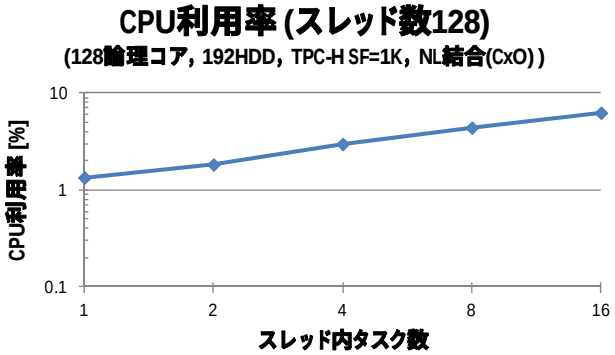

図 8 CPU 利用率(スレッド数 128)

図 6 と図 7 より，スレッド内タスク数を増加させるにつれ，ストレージシステムの IOPS 性能を引き出し，その結果処理時間が短縮できていることが確認できる．図 8 を見ると，IOPS 性能の向上に伴い CPU 利用率も増加していることが確認できる．

ロック競合による CPU 利用率の上昇の有無を確認するために，相対 IOPS 性能を横軸に，CPU 利用率を縦軸にプロットしたグラフを図 9 に示す．

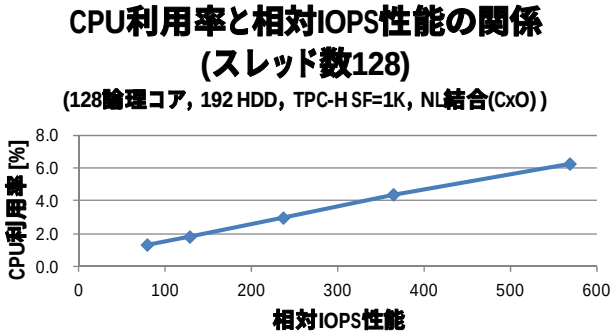

図 9 相対 IOPS 性能と CPU 利用率の関係 (スレッド数 128)

図 9 より，スレッド数と同様に，相対 IOPS 性能に対して CPU 利用率がほぼ線形に増加していることが

確認できる．これにより，本測定においては IOPS 性能が上昇してもロック競合の発生頻度が上昇していないと考える．

以上の評価結果により，スレッド内複数タスク方式を実装したアウトオブオーダ型データベースエンジンのプロトタイプでは，本実験環境でのタスク数 2048 以下の測定において，I/O 性能が上昇してもロック競合の発生頻度は上昇しないことが確認できた．

4. おわりに

本論文では，ハードウェア資源を効率的に活用することを目指したアウトオブオーダ型データベースエンジン OoODE を実現するにあたり，実行時に動的に生成される多数のタスクを複数のコアで動かすタスク管理機構のプロトタイプを実装し，その評価を行った．プロトタイプでは,OS のスレッドスケジューリングの影響や管理オーバヘッドを小さくするために，スレッドはコア数程度を想定し，スレッド内に複数タスクを動かすスレッド内複数タスク方式を実装した．プロトタイプを評価した結果，評価環境でのタスク数 2048 以下の実行においては，I/O 性能が上昇しても性能低下につながるロック競合発生頻度の上昇はないことが確認できた．

謝　　辞

本研究は，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．

参　考　文　献

[1] Digital Universe, http://www.emc.com/leadership/digital-universe/index.htm, 2012.

[2] 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，pp.131-163 (2009.06)

[3] Transaction Processing Performance Council. TPC-H, an ad-hoc, decision support benchmark. http://www.tpc.org/tpch/