D-035

スケールアップ指向のインメモリデータベースエンジン開発

In-Memory Database Engine for Scale-up System

礒田 有哉† 友田 敦† 牛嶋 一智† 田中 剛†
Yuya Isoda Atsushi Tomoda Kazutomo Ushijima Tsuyoshi Tanaka
上村 哲也† 花井 知広† 青木 英郎† 山本 祐介†
Tetsuya Uemura Tomohiro Hanai Hideo Aoki Yusuke Yamamoto

１．はじめに

近年，ハードウェア技術の進歩により，CPU のコア数増加やメモリの大規模化が進んでいる．DBMS（Database Management System）においては，メモリの大規模化により OLTP（Online Transaction Processing）処理のデータセットをメモリのみに配置することが可能となった．今後，DB のインメモリ化，SSD（Solid State Drives）の汎用化，不揮発性メモリである SCM（Storage Class Memory）の登場によって，DB 領域及びログ領域のデータ帯域が大幅に拡大するため，アーキテクチャの見直しが不可避である．特に，マルチコア化が進む CPU に対応したスケーラブルな DBMS の開発は急務である．そこで，本報告では，最新技術の動向を把握し，近年の技術潮流に沿ったインメモリ DBMS の提案および開発について報告する．

インメモリ DBMS に関する研究は，数多く報告されている[1, 2, 3, 4, 5, 6, 7, 8].

H-Store は，データパーティショニングを駆使したロックフリーなインメモリ DBMS である[7, 8]．H-Store は，CPU コア数分の論理サイトを作成し，テーブルを分割して格納する．各論理サイトでは，1 つのトランザクション処理スレッドで SQL を実行することで，ロックフリーな操作を実現している．複数の論理サイトに跨る SQL は，SQL をサイトごとに分割して実行し，実行結果を集計して処理する．また，H-Store は，登録した関数へ引数を渡して実行するストアドプロシージャで動作する．

Hekaton は，ロックフリーなマルチバージョンを利用した楽観的同時実行制御を用いたインメモリ DBMS である[1]．Hekaton の特徴は，ストアドプロシージャをネイティブコードにコンパイルするネイティブ コンパイル ストアド プロシージャにある．ネイティブコードにより，従来のストアドプロシージャで行っていたインターフェース間のやりとりを削減し高速化を実現する．また，ガベージコレクションをノンブロッキング，処理スレッドとの協調動作によって実施することによって，高いスケーラビリティを持つ．

SAP HANA は，参照効率が高い列形式で管理するインメモリ DBMS である．SAP HANA の特徴は，データを行形式でバッファし，列形式に変換するデルタマージにある．これにより，参照処理と OLTP 処理の両立を実現する．

本報告では，スケールアップ型のインメモリ DBMS として MPDB（Massive Parallel Database）を提案し，MPDB のコンセプト提案，技術開発および，評価について報告する．MPDB の特徴は，大規模な SMP（Symmetric Multi Processing）構成に起因する NUMA（Non Uniform Memory Access）を考慮したインメモリ DBMS である．大規模な SMP 構成は，NUMA によりローカルメモリとリモートメモリのアクセスレイテンシが大きく異なり，NUMA を想定した DBMS の設計により高いスケーラビリティが得られることが知られている[9, 26, 27, 28]．システムによっては，UMA（Uniform Memory Access）を模擬可能であるが，アクセスレイテンシはローカルメモリとリモートメモリを合わせた中間程度に遅くなる．このため，MPDB では，NUMA を考慮したスレッドやメモリの配置を実施しており，ログファイルや通信割り込みについても分散させている．また，CPU コア数の増加によりロック制御のスケーラビリティが低下する懸念もあり，MPDB では，ロックフリー制御と MVCC（Multi Version Concurrency Control）によって，高いスケーラビリティを実現している．このように，MPDB では，低レイテンシなメモリアクセスかつ，高スケーラビリティな実装によって，単一サーバにおける最大性能を実現する．

評価実験は，業界標準ベンチマーク（TPC Benchmark™ C）の New Order をモデルにしたワークロード，測定環境に 10 CPU コアを搭載する 8 基の Intel® Xeon® E7 8870，64 枚の 16 GB メモリ，4 枚の 8 Gb FC Dual Port HBA を搭載した 4 Blade SMP 構成のサーバと，54 GB Cache，4 枚の 1.6 TB SSD を搭載したストレージを用いて実施した[10]．実験結果から，MPDB では 85%以上の CPU コアスケーラビリティと，80 CPU コアで 189 K TPS（Transactions Per Second）の性能が得られることを確認した．

以下，2 章ではディスク型の DBMS における課題，近年の技術潮流から，最新技術が活用できない原因を整理する．3 章では，最新技術を最大限に活用するための技術方針を定め，MPDB の技術概要を述べる．4 章では，MPDB のコンポーネント構成，スレッド構成，データ構造について述べる．5 章では，評価実験について，システム構成および，ワークロードを述べ，実験結果について言及する．最後に，6 章で今後の課題について述べる．

† (株) 日立製作所 研究開発グループ
情報通信イノベーションセンタ
Hitachi, Ltd., Resarch & Development Group,
Center for Technology Innovation – Information and Telecommunications.

2．課題

2 章では，技術課題と技術潮流を述べ，MPDB の開発方針について言及する．2.1 節でディスク型の DBMS の課題，2.2 節で近年の技術潮流を述べ，課題を明確にする．

2.1 従来課題

Harizopoulos らは，ディスク型の DBMS における OLTP 処理の課題を整理するために，TPC Benchmark™ C（TPC-C）の New Order，測定環境（Intel® Pentium® 4 3.2GHz 1GB RAM Linux® 2.6），DBMS（Shore）を用いて，OLTP 処理の内訳について調査した[11, 12]．この調査結果から，バッファ処理（30 %），ログ処理（21 %），ロック・ラッチ処理（29 %）で処理全体の 80%を占めることが判明した．バッファ処理は，サーバがストレージから取得したデータを一時的に管理するための仕組みであり，ディスク型の DBMS で最もボトルネックとなり易いストレージアクセス回数を削減するために用いられる．ログ処理は，ログをストレージへ書き出す処理であり，トランザクション処理の結果を永続化するための障害対策である．ロック処理は，データベースの整合性を維持するために用いられ，トランザクション処理でインデックス，テーブル，レコードに対して用いられる．

2.2 技術潮流

近年のシステム構成は，CPU コア数の増加，メモリの大規模化，NVMe（Non Volatile Memory Express），SCM（Storage Class Memory）といった新しい技術の登場によって目覚ましい進歩を遂げている[13, 14]．ソケットあたりの CPU コア数は急激に増加している一方，CPU の周波数は 3 GHz 程度で収束しつつある[15]．このことから，技術潮流に沿った性能向上には，ソフトウェアの並列化，高スケーラビリティ化は益々重要といえる．メモリ容量は，DB の規模を上回る勢いで大規模化しており，OLTP 処理のデータセットは数 TB 以下が多く，ストレージからデータを取得しないインメモリ処理が可能となった．NVMe は，従来と比べてアクセスレイテンシが小さく，帯域の大きい PCIe®（PCI Express®）接続の SSD（Solid State Drives）を活用するための仕組みであり，アクセスの高速化や I/O キューの分散を実現する．NVMe の運用例として，Oracle Database の DSFC（Database Smart Flash Cache）がある[16]．DSFC は，データベースのバッファを拡張するために NVMe を活用しており，アクセス頻度が高いデータのキャッシュ・アウトを防ぐ．これにより，ストレージへのアクセス回数を削減し，OLTP 処理の性能低下を抑止する．一方，SCM は，DRAM（Dynamic Random Access Memory）より遅く，SSD より速く，不揮発の特性を持つメモリである．インメモリ処理における SCM は，不揮発な特性を永続性確保に活用できると期待されている．

このようにハードウェア技術は進歩しているが，これらの恩恵をディスク型の DBMS では得られていない．ディスク型の DBMS は，2000 年以前に開発されたものが多いため， CPU コア数の増加や，データセットをメモリに常駐させることを前提としたアーキテクチャに見直す必要がある．ログ処理においても，ネットワーク性能以上に CPU 性能が向上したため，通信の割り込み処理がボトルネックに陥っている．

3．MPDB の技術概要

3 章では，低レイテンシかつ高スケーラビリティな DBMS を実現するための技術について述べる．3.1 節では，近年の技術潮流に沿った課題の解決方針を提示する．3.2 節では，インメモリ処理におけるバッファ処理について述べる．3.3 節では，整合性制御の課題と対策について述べ，3.4 節では，ログ処理における課題と対策について述べる．

3.1 開発方針

2 章の課題や技術潮流を踏まえて，MPDB では，データベース規模が数 TB 以下の OLTP 処理向けのインメモリ DBMS を開発する．また，開発するインメモリ DBMS は，性能効率が高いスケールアップ指向とする．理由は，今後もメモリの大規模化が進み，格納可能なデータベース規模の拡大が見込まれること，SCM による高速な障害対策を実現できることに起因する．また，スケールアウト構成のアプローチを取った場合においても，コストパフォーマンス向上の観点から 1 サーバ当たりの性能向上は必須である．

MPDB における課題対策を表1 に記載する．MPDB では，インメモリ処理によりバッファ処理を廃止し，ロック処理をロックフリー処理に変更し，ログ処理を並列化と投機実行可能とし，スレッドやデータ配置の最適化を実施する．これにより，メモリアクセスレイテンシの削減，スケーラビリティの向上によって，高性能なインメモリ DBMS を開発する．

表1：MPDB の開発方針

#	課 題	開発方針
1	BUFFER	インメモリDBによるバッファ廃止
2	LOCK / LATCH	ロックフリー処理
3	LOG	並列化，投機実行
4	INDEX	インメモリ最適化
5	OTHER	NUMA対応，MVCC

3.2 インメモリ処理

ディスク型の DBMS と MPDB のバッファ処理の違いを図 1に記す．ディスク型の DBMS は，低速なストレージへのアクセス回数を削減するためにバッファ処理を実装していた．しかし，メモリ搭載容量の大規模化に伴い，バッファ制御範囲が拡大し，バッファ処理の負荷が増大した[17]．更に，メモリ搭載容量は，OLTP 処理のデータセットが格納できるまでに大規模化した．このことから，MPDB では，バッファ処理を廃止した完全なインメモリ DBMS とする．ただし，依然として永続化媒体としてのストレージは，トランザクション処理のログ，スナップショットや，システム情報を保持するために必要となる．

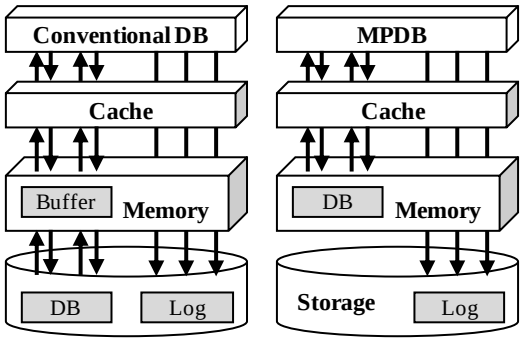

図1：DBMS のバッファ処理

3.3 整合性制御

整合性制御には，シングルバージョン（SV）とマルチバージョン（MV）があり，これらの概要を図 2 に示す．SV は，更新前データを更新後データで上書きすることによって更新し，MV は，更新後データを別領域に作成し，更新前データと関連付けることによって更新する．SV では，トランザクション処理の参照と更新を同時に実行出来なかったが，MV によって同時に実行可能となり性能が向上する．MPDB では，MV をロックフリーで処理可能なリンクドリスト形式で用いる．MPDB の MV 管理を図 3 に示す．MPDB では，最新のレコードはリンクドリストを辿った先にあり，レコードを追加するように更新する．

MV の整合性制御では，タイムスタンプ処理を用いることによって，整合性制御やガベージコレクト（GC）の簡易化が実現できる．整合性制御は，ロック処理を用いることによって実現できるが，CPU コア数が多い大規模な SMP 構成では，ロック処理がスケーラビリティ低下の懸念となる．このため，MPDB では，ロック処理を必要としないタイムスタンプ処理を用いる．MPDB の実装を図 3 に示す．タイムスタンプ処理は，データベースを論理時間で管理する仕組みであり，トランザクション処理をコミットする際に更新する．このタイムスタンプを時系列で管理したいデータに付与することによって，トランザクション処理があるスナップショットのデータのみを操作可能となり整合性を維持することができる．タイムスタンプ処理による整合性は，ANSI が定義する分離レベルと乖離があるため正確ではないとされてきたが，近年の研究では ANSI が定義する SERIALIZABLE を実現可能であることが証明された[18, 19, 20, 21]．また，タイムスタンプ処理による SERIALIZABLE は，分散環境においても高いスケーラビリティを発揮することが報告されており，スケーラビリティを得るためにも必要な技術である[22, 23].

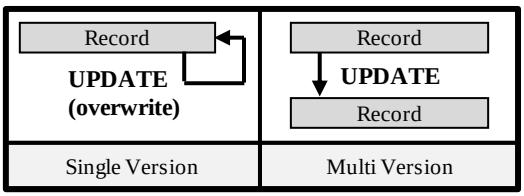

図2：シングルバージョンとマルチバージョン

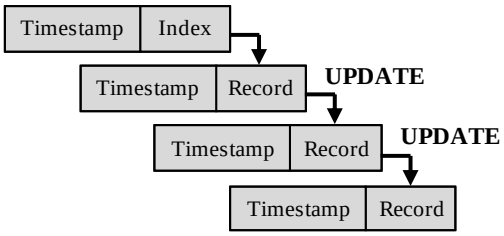

図3：MPDB のマルチバージョン管理

3.4 ログ処理

インメモリ処理では，クライアント通信やログ処理がボトルネックとなる．クライアント通信は，性能に比例して増加するため，通信の割り込み処理が増加しボトルネックとなる．また，ログ処理においても通信の割り込み処理，レイテンシ低下，スループット低下によるボトルネックが発生する．

割り込み処理は，RSS（Receive Side Scaling）や irq balance によって分散でき，割り込み処理をポーリング処理に変更することで処理回数を削減できる．ただし，複数クライアントからの通信は分散できるが，単一のログ処理は分散できない．ハードウェアでは，複数の HBA（Host Bus Adapter）や NVMe デバイスを用いることによって割り込み処理を分散できる．このため，ログ処理には，並列化や複数のハードウェア活用が求められる．従来，DBMS の信頼性を維持するために，ログを時系列に沿って一次元で管理する必要があるため，ログ処理を逐次実行していた．しかし，厳密いえば，ログファイルを物理的に時系列に沿った一次元で管理する必要はない．トランザクション処理は，タイムスタンプ処理によって，コミット時にタイムスタンプを得る．このタイムスタンプをログに書き込み，参照するときはタイムスタンプの順にアクセスすることで，時系列に沿ってログ処理したことと同等となる．これにより，タイムスタンプをログに含めることで，ログファイルを複数に分割することができ，並列にログ処理を実行することが可能となる．このことから，ログ処理においても割り込み処理を分配することができる．ログ処理の並列化を図 4 に示す．

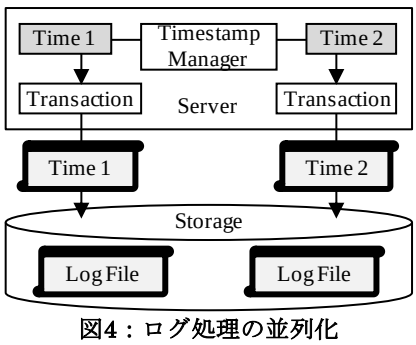

図4：ログ処理の並列化

次に，ログ処理のレイテンシやスループットは，ネットワークやストレージの技術進歩によって支えられている．これらを最大限活用するための処理方式として，グループコミットや投機実行がある[24]．グループコミットは，トランザクション処理ごとにログを発行せず，複数のトランザクション処理のログをまとめてから発行する．

この方式は， IOPS（I/O Per Second）やランダムライトが課題となるとき，スループットやシーケンシャルライトで解決する技術である．ログ処理の投機実行は，ログ発行が失敗する確率は極めて低いことを前提にした方式であり，ログ発行完了前にコミット結果（予定）を操作可能とする．投機実行の方式を図 5に，投機実行の効果を図 6に記載する．図 6に示すように，ログ処理を高多重に実行でき，性能向上を実現できる．

これらのことから，MPDB では，ログ処理の並列化と投機実行を実施する．グループコミットを用いない理由としては，ランダムライト性能が高い SSD や不揮発性メモリの登場により，ボトルネックとならないと考えたからである．また，グループコミットは，ログをまとめるためにログ管理が必要となり，スケーラビリティを阻害する懸念がある．このことから，MPDB では，グループコミットは利用しないこととした．

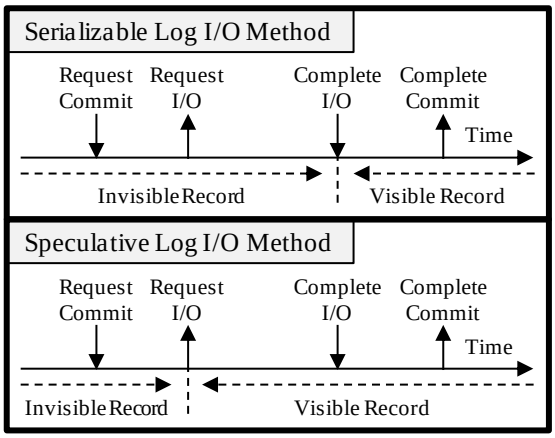

図5：非投機実行と投機実行

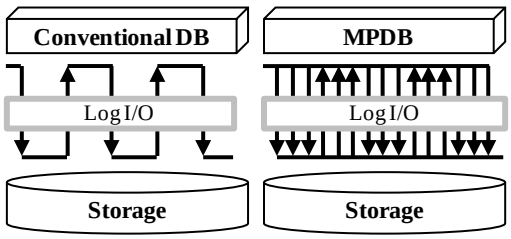

図6：投機実行による効果

4．MPDB 構成

4 章では，MPDB の処理やデータの構成について述べる．4.1 節では，MPDB のコンポーネント構成を述べ，4.2 節では，MPDB エンジンのスレッド構成について述べる．4.3 節では，データ構造を述べ，4.4 節では，領域管理について述べる．

4.1　コンポーネント構成

MPDB は，パーサ，オプティマイザ，エグゼキュータ，エンジンのコンポーネントから構成され，テーブルやインデックスの構成情報を持つディクショナリ，テーブル，インデックスを管理する．これらの関係を図 7に示す．パーサでは，クライアントからの処理命令（クエリ）の構文解析を実施する．オプティマイザでは，構文解析結果から最適な実行プラン（SQL）を作成する．オプティマイザは，インデックスの有無，データの分布などから処理コストを見積もり，最も低コストで実行可能なプランを生成する．エグゼキュータでは，実行プランに基づきエンジンで処理命令を実行する．エンジンは，SELECT や UPDATE といった簡単な SQL 文を受付け，実行結果をエグゼキュータに返し，エグゼキュータは，エンジンから結果を受け取り，実行結果のソートやジョインを実行し，クライアントへ結果を返す．本報告では，エグゼキュータとエンジンを新たに開発した．

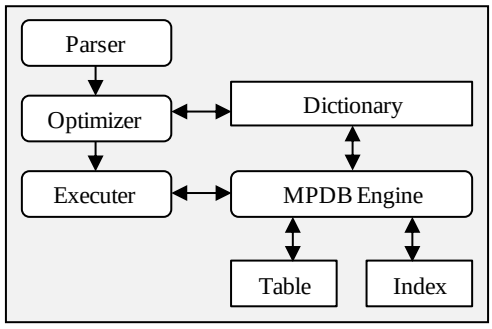

図7：MPDB のコンポーネント

4.2　スレッド構成

MPDB エンジンは，複数の処理スレッドから構成される．処理スレッドは，エグゼキュータから SQL 文を受付け，SQL 文の実行結果をエグゼキュータへ返す．

MPDB では，クライアントから受付けたクエリを複数の SQL 文に分解し，処理スレッドで実行する．このとき，あるクエリから分解された複数の SQL 文は，クエリごとの作業領域を用いて実行する．作業領域には，SQL 文を実行した更新や参照の履歴が記載されており，トランザクション処理の整合性制御に用いられる．MPDB では，NUMA を考慮した最適なメモリアクセスを実現するために，クエリを処理スレッドにバインドし，処理スレッドを CPU コアにバインドし，作業領域を CPU コアから直接アクセス可能な NUMA ローカルなメモリにバインドした．これにより，高頻度にアクセスされる作業領域を低レイテンシでアクセス可能となる．この処理を実現するために，エグゼキュータは，エンジンへトランザクション ID（Tx.ID），スレッド ID（Th.ID），SQL 文を通知する．これらの概要を図 8に記載する．

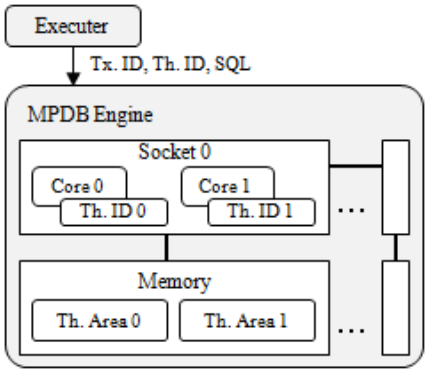

図8：MPDB エンジン

4.3 データ構造

MPDB のテーブルやインデックスの構造について述べる．MPDB では，テーブルの行をレコード，列をカラムと呼ぶ．レコードは，固定長のヘッダー部と可変長のデータ部に分かれる．固定長と可変長は，固定長で配列アクセスやシーケンシャルスキャンを容易に実施できるように使い分けている．これらの関係を図 9に示す．また，MPDB では，ロックフリーを実現するために，レコードの更新前後をリンクドリストで管理しており，レコードを DELETE する際には，データへのポインタを持たないヘッダー部のみのレコードを追加することによって，レコードの削除を表す．レコードの構造と操作を図 10に示す．この構造により，レコード操作のロックフリー化を実現している．

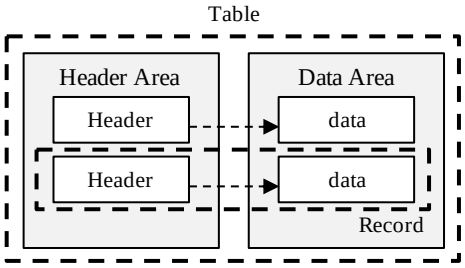

図9：テーブル構成

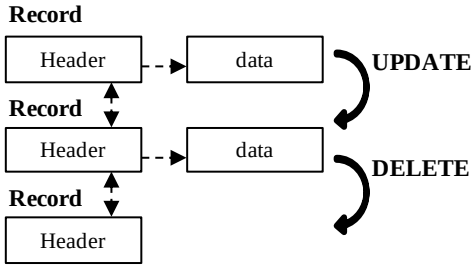

図10：レコードのリンクドリスト構造

また，MPDB では，B-tree インデックスをリンクドリストで実装している．MPDB で実装している B-tree の構成を図 11に示す．B-tree の頂点をルートノードと呼び，ルートノードからブランチノードへのポインタをルートエッジと呼ぶ．同様に，ブランチノードはブランチエッジ，リーフノードはリーフエッジを持つ．また，リーフエッジは，ロウへのポインタを示す．ロウとは，レコードの更新前後の全ての履歴を示す．各ノードは，複数のエッジを保持しており，エッジは昇順で並び，ノードは降順で並ぶ．この関係を図 12に示す．エッジとノードで異なる順序関係によって，ロックフリーな更新・削除や処理の簡素化による高速化を実現する．また，擬似双方向なデータ構造を持たせることで，インデックスのメンテナンス性を高める意味もある．

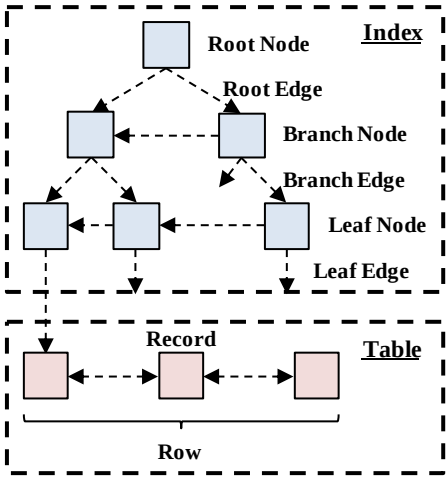

図11：MPDB の B-tree インデックス

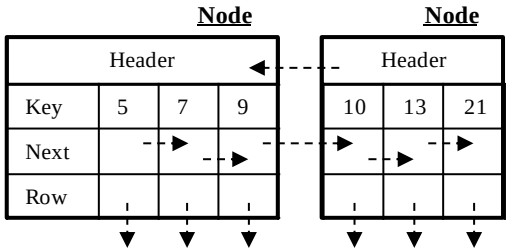

図12：ノードとエッジ

4.4 領域管理

MPDB では，複数のメモリ空間を区別して利用する．メモリ空間は，処理スレッドごとで扱うスレッド領域，ソケットごとで扱うソケット領域，システムで扱うグローバル領域に分かれる．メモリ空間の概要を図 13に示す．

スレッド領域には，トランザクション処理の作業領域，トランザクション処理の情報領域，ログバッファ領域，テーブルのヘッダー部やデータ部の領域が含まれる．これらのスレッド領域は，全スレッドの参照では共有するが，領域取得の競合を削減するためにスレッドごとに配置している．また，スレッド領域には，処理スレッドが高頻度に操作する領域が多く含まれるため，NUMA を考慮して直接アクセス可能なメモリに配置する．

ソケット領域には，特定ソケットで動作する複数スレッドが共有するデータを配置する．例えば，MPDB では，ソケットごとにログファイルを扱うため，ログファイル情報をソケット領域に持つ．この他に，ソケット領域は，スレッド領域を拡張する際にも利用される．

グローバル領域には，局所性が存在しないデータを配置する．例えば，複数スレッドが操作するインデックスやシステム情報などは，グローバル領域に配置する．SQL 処理において，参照回数が多いインデックスは，メモリアクセス負荷を分散させるために，グローバル領域に配置している．

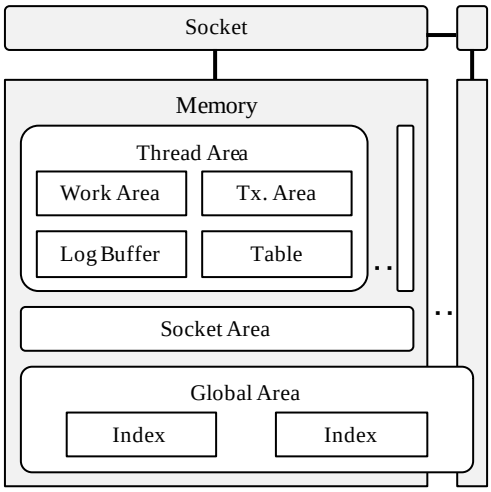

図13：MPDB の領域管理

5．評価実験

5 章では，MPDB エンジンの性能測定結果を述べる．実験を実施するにあたり，5.1 節でシステム環境，5.2 節でワークロードを述べ，5.3 節で実験結果を述べる．本報告の実験では，クライアントからの通信で SQL 文を投入するのではなく，エグゼキュータが登録済の関数（ストアドプロシージャ）を繰り返した場合の MPDB エンジンに対する高負荷実験の結果である．

5.1 システム環境

システム環境の概要を図 14に示す．サーバは，BS2000 で，4 ブレードの SMP 構成とし，ブレードは 2 CPU（Xeon® E7 8870），256 GB Memory（16 GB x 16），2 Port HBA である．ストレージは，Hitachi Unified Storage VM （HUS-VM） で， 54 GB Cache， 6.4 TB Hitachi Accelerated Flash（HAF）（1.6 TB x 4，RAID5：3D + 1P）である．サーバとストレージの FC 通信は，1 段の FC スイッチを持つ．

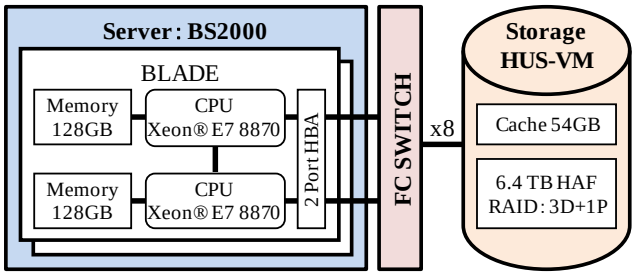

図14：システム環境

システムの設定は，OS は CentOS 6.4 を用い，1 CPU あたりに 1 FC Port を割当て，FC Port の割り込み処理は各 CPU の Core 0 に分配した．MPDB エンジンの挙動を詳細に把握したいため，ハイパースレッディングはオフとしている．

MPDB エンジンの設定は，1 処理スレッドを 1 CPU コアに割り当てる．また，1 CPU（10 CPU コア）あたりに 1 ログファイルを割り当てる．

5.2 ワークロード

ワークロードは，業界標準ベンチマークである TPC Benchmark™ C（TPC-C）の New Order をモデルにして作成した．ワークロードの概要を表 2に示す．ワークロードは，New Order で繰り返し実行される箇所を模擬しており，1 回の実行で表 2を 5～15 回繰り返し実行し，平均繰り返し回数は 10 回とした．テーブルは，TPC-C の item 表，stock 表，order_line 表を用いた．インデックスは，item 表では i_id，stock 表では s_w_id と s_i_id の複合インデックス，order_line 表では ol_o_id，ol_w_id の複合インデックスを作成した．

表2：ワークロード

1	SELECT	i_price, i_name, i_data
	INTO	:i_price, :i_name, :i_data
	FROM	item
	WHERE	i_id = :ol_i_id
2	SELECT	s_quantity, s_data, s_dist_...
	INTO	:s_quantity, :s_data, :s_dist_...
	FROM	stock
	WHERE	s_i_id = :ol_i_id AND s_w_id = :ol_supply_w_id
3	UPDATE	stock
	SET	s_quantity = :s_quantity
	WHERE	s_i_id = :ol_i_id AND s_w_id = :ol_supply_w_id
4	INSERT	
	INTO	order_line („„„)
	VALUES	(,,,,,,)

While (Repeats 5 ~ 15 times)

5.3 実験結果および考察

評価実験では，性能および CPU 増加に伴うスケーラビリティを測定する．1 CPU あたり 10 Core を持つ CPU を使用し，最大構成では 8 CPU，80 Core の性能測定を実施する．

図 15にシステム性能を示し，X 軸を CPU コア数，Y 軸をシステムあたりの TPS（Transactions Per Second）としている．図 16に 10 CPU コアを基点としたスケーラビリティを示し，X 軸を CPU コア数，Y 軸を 10 CPU コアでの 1 スレッド性能を 100 としたときの性能比率としている．方式比較は，ログ処理について評価した．MPDB 方式は，1 CPU あたりに 1 ログファイルを割り当てる．一方，従来方式は，常に 1 ログファイルとした．実験結果から，従来方式では，20 CPU コアまでは高い性能を得られるが，CPU コア数の増加に伴い性能が低下することが判明した．一方，MPDB 方式では，CPU コア数の増加に伴い性能が向上しており，80 CPU コアにおいても 85%以上のスケーラビリティを維持できることが判明した．従来方式では，Linux の mpstat による解析から，割り込み処理が特定の

CPU コアに集中し，性能がスケールしないことを観測した．このことから，3.3 節に示すようにログ処理の並列化が必要であるといえる．

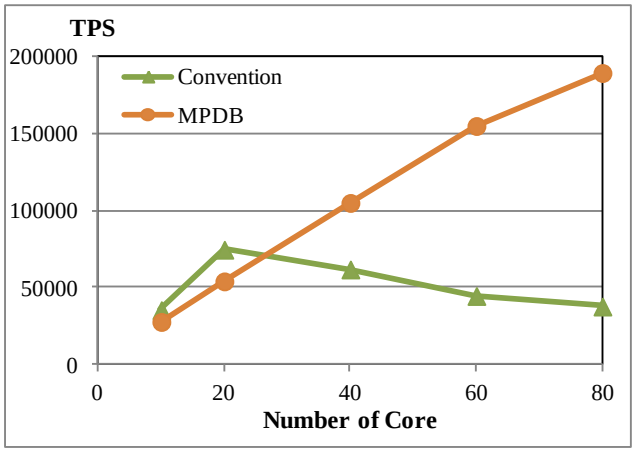

図15：システム性能

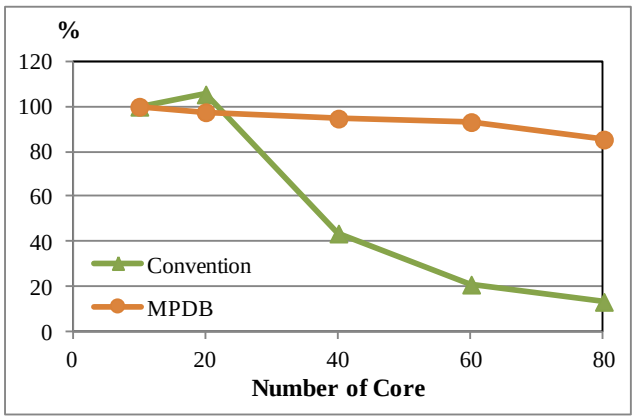

図16：スケーラビリティ

6．今後の方針

実験結果から，MPDB 方式では 1 CPU（10 CPU コア）から 8 CPU（80 CPU コア）において，85%以上のスケーラビリティを得られることが判明した．今後は，スケーラビリティを維持しつつ，性能を向上させることが課題となる．現状の処理内容の内訳を Intel® VTune™ によって解析した結果を図 17に記載する[25]．図 17は，X 軸を CPU コア数，Y 軸を CPU 時間で表している．この結果から，上位 2 つの関数が突出して CPU 時間を消費していることが分かる．最上位の『hop_ix_make_leaf_link』は，INSERT に起因した B-tree インデックスのリーフエッジを追加する関数である．また，『hin_cm_btree_search_leaf_node』は，SELECT に起因した B-tree インデックスのリーフノードを検索する関数である．この他にもインデックス関連の関数が上位にランクインしていることから，インメモリ DB に適したチューニングやアルゴリズム最適化を実施していく必要がある．

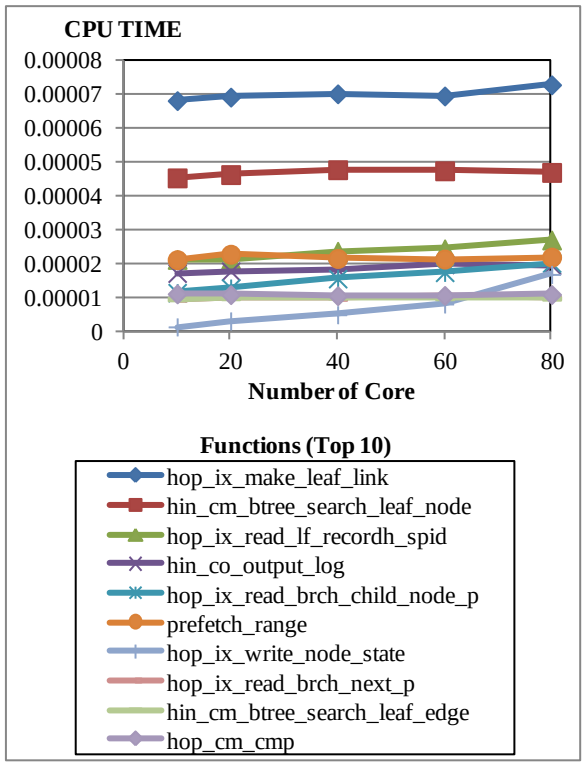

図17：VTune™ による解析結果

参　考　文　献

[1] Cristian Diaconu, et al., "Hekaton: SQL Server's Memory-Optimized OLTP Engine, " SIGMOD'13, June 22-27, 2013.

[2] Per-Ake Larson, et al., "High-Performance Concurrency Control Mechanisms for Main-Memory Databases, " VLDB, Vol.5, No.4, Pages 298-309, December 2011.

[3] ORACLE, http://docs.oracle.com/en/database/.

[4] IBM DB2, http://www-01.ibm.com/software/jp/info/db2/features.html#features_02.

[5] SAP HANA, http://hana.sap.com/abouthana.html.

[6] solidDB, http://unicomsi.com/products/soliddb/.

[7] VoltDB, http://voltdb.com/.

[8] Robert Kallman, et al., "H-store: a high-performance, distributed main memory transaction processing system, " VLDB, Vol.1, No.2, Pages 1496-1499, August 2008.

[9] David Levinthal, "Tutorial: Intel Core i7 and Intel Xeon 5500 Micro architecture, Optimization and Performance Analysis, " IEEE International Symposium on Performance Analysis of Systems and Software, March 28-30, 2010.

[10] Transaction Processing Performance Council, http://www.tpc.org/.

[11] Stavros Harizopoulos, et al., "OLTP Through the Looking Glass, and What We Found There, " SIGMOD'08, Pages 981-992, June 9-12, 2008.

[12] Ryan Johnson, et al., "Shore-MT: a scalable storage manager for the multi core era, " EDBT'09, Pages 24-35, 2009.

[13] Non-Volatile Memory Host Controller Interface Working Group, http://www.nvmexpress.org/.
[14] R. F. Freitas, W. W. Wilcke, "Storage-class memory: The next storage system technology, " IBM Journal of Research and Development, Vol.53, No.4.5, July, 2008.
[15] John L. Hennessy, David A. Patterson, "COMPUTER ARCHITECTURE A Quantitative Approach fifth edition, " Morgan Kaufmann Publishers.
[16] ORACLE, "Oracle Database Smart Flash Cache, " http://www.oracle.com/technetwork/articles/systems-hardware-architecture/index.html.
[17] 油井誠 等, "ロックフリーGCLOCK ページ置換アルゴリズム, " 情報処理学会論文誌, データベース, Vol.2, No.4, Pages 32-48, Dec. 2009.
[18] ANSI X3.135-1992, "American National Standard for Information Systems – Database Language – SQL, " Nov 1992.
[19] Hal Berenson, et al., "A Critique of ANSI SQL Isolation Levels, " SIGMOD'95, June, 1995.
[20] Alan Fekete, et al., "Making Snapshot Isolation Serializable," ACM Transaction on Database System, Vol.30, No.2, Pages 492-526, June, 2005.
[21] Michael J. Cahill, et al., "Serializable Isolation for Snapshot Databases, " ACM Transactions on Database Systems, Vol.34, No.4, December, 2009.
[22] Stephen Revilak, et al., "Precisely Serializable Snapshot Isolation (PSSI), " ICDE11, April 11-16, 2011.
[23] Hyungsoo Jung, et al., "Serializable Snapshot Isolation for Replicated Databasesin High-Update Scenarios, " VLDB, Vol.4, No.11, August 29 - September 3, 2011.
[24] Evan P. C. Jones, et al., "Low Overhead Concurrency Control for Partitioned Main Memory Databases, " SIGMOD'10, June 6–11, 2010.
[25] Intel®VTune™ Amplifier XE, http://www.xlsoft.com/jp/products/intel/vtune/.
[26] Tim Kiefer, Thomas Kissinger, Benjamin Schlegel, Dirk Habich, Daniel Molka, Wolfgang Lehner, "ERIS live: a NUMA-aware in-memory storage engine for tera-scale multiprocessor systems," SIGMOD Conference 2014, 689-692.
[27] Thomas Kissinger, Tim Kiefer, Benjamin Schlegel, Dirk Habich, Daniel Molka, Wolfgang Lehner, "ERIS: A NUMA-Aware In-Memory Storage Engine for Analytical Workload," VLDB 2014, 74-85.
[28] Tim Kiefer, Benjamin Schlegel, Wolfgang Lehner, "Experimental Evaluation of NUMA Effects on Database Management Systems," BTW 2013, 185-204.