大規模マルチメディアを対象とした スクラップアンドビルド型DBMSの構築

田中俊顯† 中家路也†† 松田一章†† 家富誠敏††† 富井尚志†††† 有澤博††††
† 横浜国立大学大学院 環境情報学府 情報メディア環境学専攻
†† 横浜国立大学大学院 工学研究科 電子情報工学専攻
††† 横浜国立大学 エコテクノロジーシステムラボラトリー
†††† 横浜国立大学大学院 環境情報研究院
〒 240-8501 横浜市保土ヶ谷区常盤台 79-7

E-mail : {toshi,nakaie,cross,eto,tommy,arisawa}@arislab.dnj.ynu.ac.jp

あらまし
我々の研究室では、フレキシブルなシステムのオンライン拡張性を考慮したDBMSとして、共有メモリを介して協調作業を行うマルチエージェントシステムに基づいたスクラップアンドビルド型DBMSを提案している。本論文では、同システムにおいて、共有メモリに対し複数のエージェントが同時書き込みを行った場合の、効率のよい排他制御を実現する機構を設計し、実機を用いて評価を行った。

Construction of Scrap-and-Build DBMS for Large Scale Multimedia

Toshiaki TANAKA† , Michiya NAKAIE†† , Kazuaki MATSUDA†† , Masatoshi IETOMI††† , Takashi TOMII†††† , and Hiroshi ARISAWA††††
†Graduate School of Environment and Information Sciences, Yokohama National University
††Division of Electrical and Computer Engineering, Faculty of Engineering, Yokohama National University
†††Ecotechnology System Laboratory, Yokohama National University
††††Faculty of Environment and Information Sciences, Yokohama National University
79-7, Tokiwadai, Hodogaya-ku, Yokohama 240-8501 JAPAN

E-mail : {toshi,nakaie,cross,eto,tommy,arisawa}@arislab.dnj.ynu.ac.jp

Abstract
We propose Scrap-and-Build DBMS based on the multi-agent system which does cooperation work through a shared memory as DBMS in consideration of the on-line extendibility of a flexible system. In this system, the mechanism in which efficient exclusion control when two or more agents perform simultaneous writing to a shared memory was realized was designed, and this paper estimated using the system.

1 はじめに

近年、マルチメディア DB の研究が盛んに行われている。マルチメディア DB の応用分野は様々であり産業、医療、スポーツなどへの幅広い利用が期待される。それらの利用の中には、一度構築したデータベースに対して必要に応じてデータベース及びシステムの規模を拡大しつつ何年何十年と長期間の運用が求められるケースも考えられる。例えば、サッカーや野球などのスポーツでプロリーグ戦の試合を（古いデータを消さずに）数十年にわたって、全てデータベースに蓄積する場合などがそうである。しかし、データベースを長期間運用していくとなると、時代の変化と共にユーザの検索内容が変わっていくことや、時間の経過と共に蓄えられるデータ量も徐々に増え、長い年月をかけて非常に膨大な量になることが考えられる。だが、大規模なマルチメディア DB では、データベース設計の初期段階で将来を見越したシステムのハードウェア構成やスキーマ、検索オペレーションを決めることは難しい。

それゆえ、このような時々刻々変化するデータベースに対しては、その時々のデータ量、計算機の性能、ユーザのニーズ、マルチメディア情報処理技術に応じてハードウェア構成、スキーマ、検索オペレーションを必要に応じて変更することが重要となる。しかし、ただ変更を行えるだけでは不十分で、実用面を考えれば上で挙げた様々な変更を利用者へのサービスを維持しながら、すなわちシステム全体を停止することなく行なえる必要がある。

この考えに基づき、我々の研究室では、データフォーマットやデータに対するオペレータの変更、ハードウェア構成の変更を柔軟にかつ、DBMS を停止させることなく行うシステムとして、スクラップアンドビルド型 DBMS[4] を提案している。同システムはマルチエージェントシステムをベースとし、すべてのエージェントが共有メモリを介して互いの情報を参照し協調作業を行っていくシステムで、データに対する様々なオペレータをそれぞれ独立したエージェントとして扱い、エージェントの追加、削除を自由に行うことで、システムの柔軟な拡張を可能にしている。

このような複数のエージェントが共有メモリを参照し、協調作業を行う場合、複数のエージェントが同時に同じデータに変更を行う可能性がある。そのような場合どちらかのデータが勝手に上書きされ、無効となるといったことが考えられ、排他制御の機構が重要となる。

そこで本研究では共有メモリにおいて、複数のエージェントによる同時書き込み時の、効率のよい排他制御を実現する機構を提案する。これにより、共有メモリ内のデータの整合性が保たれ、複数エージェントによる協調作業が可能となる。以下、2 章でスクラップアンドビルド型 DBMS の概念、3 章で同システムの実現手法について述べ、4 章で大規模共有メモリにおける排他制御の方法を提案する。５章では、提案する方法を並列計算機上に実装し、評価を行い、最後にまとめを述べる。

2 スクラップアンドビルド型 DBMS の概念

本章ではスクラップアンドビルド型 DBMS の概念について述べる。ソフトウェアアーキテクチャ的観点から見た、スクラップアンドビルドの意味は、「システム全体が独立性の高い構成部品の集まりとしてできていた場合、その中のいらない要素を自由に外せて、新たに必要になった要素を自由に付け加えることができるもの」であると言える。この考え方を DBMS に適応したものが我々の提案するスクラップアンドビルド型 DBMS の概念である。概念図を図 1 に示す。

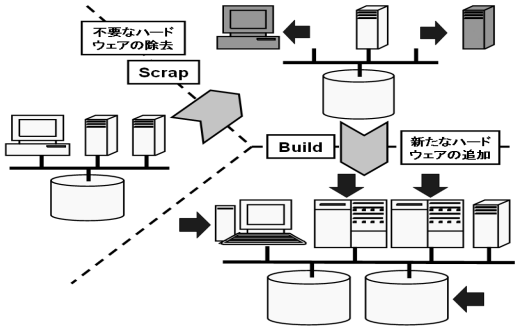

図 1: スクラップアンドビルド型システム

特に、大規模なマルチメディア DB では、長期間の運用の中で変化するデータ量、計算機の性能、ユーザのニーズ、マルチメディア情報処理技術に応じてハードウェア構成、オペレータを柔軟に拡張する必要があることから、スクラップアンドビルド型 DBMS においてスクラップ (削除) 及びビルド (追加) することが必要なものは大きく分けて次の 2 つであると考えられる。

- 計算機資源のスクラップアンドビルド

  大規模なマルチメディア DB では、データベース運用開始後も時間の経過と共に蓄積されるデータが増大していくことから、システムのハードウェア構成をデータベース設計の初期段階で決めることは難しく、運用開始後のハードウェア構成の拡張が必要となる。

- オペレータのスクラップアンドビルド

  ユーザの新たなニーズに対応するためや、マルチメディア情報処理技術の発展により可能となる新たな検索をサポートするため、必要に応じて既存のオペレータを拡張（追加、削除）することが必要となる。

また、以上を行う場合、システムのオンライン拡張性を保持する必要がある。オンライン拡張性とは「部分的な拡張、修正、追加があったとしても、他の部分の稼動を停止することなく、かつ、それらを変更しなくてよいこと」[1] と定義されている。データベース運用中のシステムの変更や修正は、システム全体を停止することなく行うことが望ましく、スクラップやビルドをシステム全体を停止することなく行うことができれば、データベースの運用方針やシステムの変更を自由に行うことが可能になると考えられる。

関連研究として、拡張可能 DBMS である、Earth[3] や COMMON[2] が挙げられる。COMMON では、ダイナミックリンクライブラリを利用した動的な機能拡張を行っているが、ハードウェア構成の拡張にまでは触れていない。また、Earth では、システム構成の異なる複数の拡張可能 DBMS を用意し、個々のシステム・アーキテクチャに適応させている。しかし、運用中にシステム・アーキテクチャが変更される場合の対応については考えられてはいない。

これに対し、スクラップアンドビルド型 DBMS では、DBMS の機能のみを拡張するのではなく、システムの各ハードウェア構成要素までも、柔軟な拡張を行うことを可能にしている。これにより、より多様な DBMS の拡張が可能となると考えられる。

3 スクラップアンドビルド型 DBMS の実現手法

2 章で述べたような自由な拡張を実現するため、マルチエージェントシステム [5] をベースとした DBMS を構成した。本システムのアーキテクチャを図 2 に示す。検索の処理は、まずユーザからの検索文をコンパイルして実行コードに落とし、それを実行器のプリミティブであるエージェントが協調しながら検索結果を生成する。そして最後に最終検索結果をユーザに渡すという流れである。実行器によって実行される個々の演算は、当然 DBMS がよって立つところのデータモデルやデータベースの物理構造、そして提供すべき ADT オペレータ等によって異なるが、ここでは関係 DB などでよく用いられ、かつ我々が開発中の DBMS プロトタイプとの相性も良いことから、「集合 (群) へのポインタを引数としてもらい、数値演算や集約演算、画像の加工などの演算を行い、集合を変更し、結果についての値を返す」という形の操作が同時並行的に実行されることを仮定している。実行器を構成するエージェントは、大きく、DB アクセスエージェント、メディア依存処理エージェントに分けられ、それぞれに様々

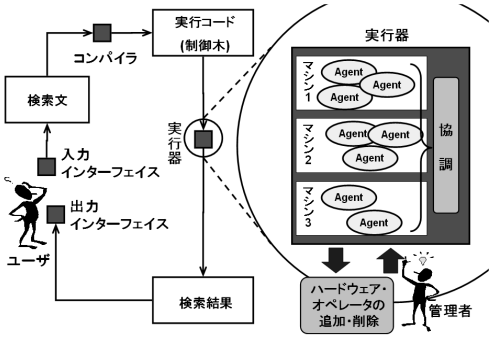

図 2: スクラップアンドビルド型 DBMS のアーキテクチャ

なエージェントが存在する。DB アクセスエージェントは、データベースにアクセスするエージェントで、R-tree やハッシュなど、参照方法の違いにより複数のエージェントが存在する。メディア依存処理エージェントは、各種メディア依存処理を行うエージェントで四則演算、画像の合成のエージェントなどが挙げられる。

これより、我々が提案、実装するスクラップアンドビルド型 DBMS の構造と演算処理の実現手法について詳細に述べていく。マルチメディア DBMS の実現手法にはいろいろな方式が考えられるが、ここではエンティティ(あるいはオブジェクト) 集合と、集合間の構造・リンク情報のすべてをメモリ空間上におき、メディアデータのみをディスクなどの外部記憶上に置く、いわゆるオンメモリデータベースを想定する。このようなアーキテクチャは現在必ずしも一般的ではないが、計算機資源の価格性能比の変化によって、マルチメディア DB のように、参照構造と原データを分離できる中規模以下のデータベースにあっては、むしろ一般的になると我々は考えている。

この場合のシステム構成図を図 3 に示す。図 3 の

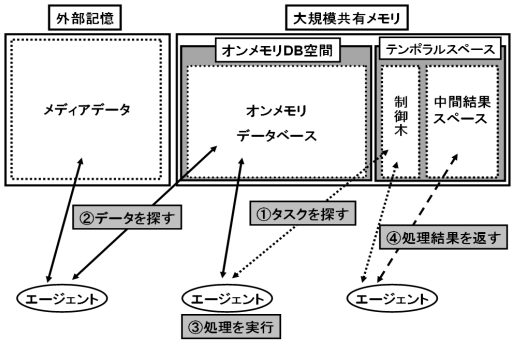

図 3: スクラップアンドビルド型 DBMS の構成

ように本システムでは、大規模共有メモリ上にオンメモリ DB 空間と、エージェントが一時的に作業を行うスペースであるテンポラルスペースを構築し、そこを各エージェントが参照することで、大規模共有メモリを介した協調作業を行う。以下、システムの各構成要素、エージェントについて詳しく述べる。

3.1 大規模共有メモリ

大規模共有メモリは、大きくオンメモリ DB 空間と、エージェントのテンポラルスペースである制御木及び中間結果スペースから構成される。それぞれの空間の役割を以下に述べる。

3.1.1 オンメモリ DB 空間

本システムではメディアデータを除くデータベースを大規模共有メモリ上にオンメモリで構築している。これがオンメモリ DB 空間である。エージェントのデータベースへのアクセスはこのオンメモリ DB 空間に対して行い、必要に応じて外部記憶からメディアデータを取り出す。

3.1.2 制御木

ここでは、マルチメディア DB に対する検索をストリームデータ（映像、音声など）に対するストリーム処理と考え、検索手順をストリーム処理の流れを示す有向グラフで記述したものを制御木と呼ぶ。制御木の例を図 4 に示す。図 4 では、小さい四

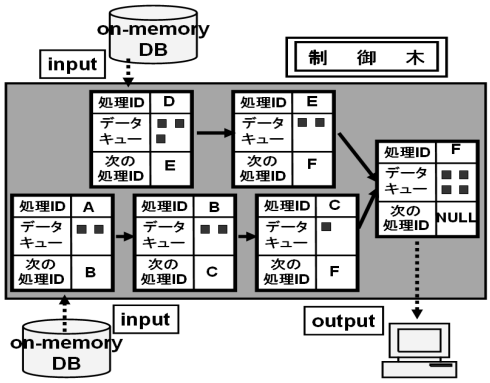

図 4: 制御木の例

角が、処理されるデータのデータ ID とそのデータの処理結果に付加される ID(返り値 ID) の組を、各ノードがそれぞれ処理の種類を、矢印が処理の流れを表している。各ノードは、自分の処理 ID と、次にデータが流れる先の処理 ID(次処理 ID)、処理対象のデータの情報を管理するためのデータキューを保持している。

各ノード間にはデータ ID のみが流れ、それぞれのノードでは、流れてきたデータ ID をデータキューに登録する際、そのデータ ID に対する返り値 ID を一意に決め、データ ID と返り値 ID を組みにし、一タプルとしてデータキューに登録する。また、エージェントの制御木に対するアクセスは処理 ID を用いて行われる。

3.1.3 中間結果スペース

中間結果スペースとは、検索処理の途中で生じる中間結果が書かれる空間である。エージェントは、処理結果を返り値 ID と共に中間結果スペースに書き込み、ここに書かれたデータにはデータ ID を用いてアクセスする。なお、中間結果スペースと制御木は、検索の度に生まれる一時的なものである。

3.2 エージェント

検索時におけるエージェントの処理の流れを示す。

1. エージェントは、処理 ID をもとに制御木を参照し、該当するノードのデータキューにキューがあるか確認する。(read アクセス)
2. あれば、先頭のキューに対する処理を行うために予約を入れ、予約が認められればキューを取り出す。(write アクセス)
3. データ ID を指定することで、対象となるデータをオンメモリ DB 空間又は中間結果スペースから読み出し、処理を行う。
4. 処理が終われば、処理結果を返り値 ID と共に中間結果スペースに書き込む。
5. また、制御木に対しては、次処理 ID を持つノードにアクセスし、返り値 ID を渡す。(write アクセス)

以上の繰り返しで検索処理が行われる。このうち、1,2,5 は制御木に対するアクセスである。

3.3 並列計算機でのシステム構築

エージェントが大規模共有メモリを介した協調作業を行うためには、大規模共有メモリに対する高速な読み書きのアクセスが可能であることが求められる。

一方、並列計算機の中にはこのような共有メモリ機構をハードウェアレベルでサポートしているものもある。並列計算機は、必ずしも計算機資源の拡張性に優れているとは言い難いが、将来ネットワークが高速化されれば、計算機資源の拡張を柔軟に行える、ネットワーク分散したコンピュータクラスタにも容易に移行できることから、今回、本システムを実現するものとして並列計算機上でシステム構築を行った。並列計算機上でのシステム構成を図 5 に示す。図 5 のように並列計算機の各プロセッサをまたいだ形で大規模共有メモリが存在し、各プロセッサにエージェントが分散して配置されている。

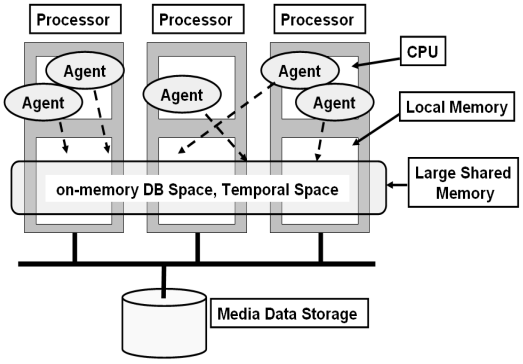

図 5: 並列計算機上でのシステムの構成

本章で述べてきた考え方で DBMS を構築するにあたって、議論すべき点は多々あるが、中でも「複数プロセッサにまたがる大規模共有メモリと複数プロセッサ上に存在するエージェントからの対等で効率の良いアクセスをどう実現したらよいか」は重要な課題である。そこで本論文では、以下この点に絞り、大規模共有メモリの構成技法や並列計算機による評価について述べていく。

4 大規模共有メモリにおける排他制御

3.1 節で述べたように、制御木と中間結果スペース、オンメモリ DB 空間では、その役割の違いから参照の頻度やそこに書き込まれるデータ量に違いが

ある。特に制御木に関してはその特徴として、他の二つと比べデータ量が圧倒的に小さいことや多くのエージェントから参照され、頻繁に更新が繰り返されることが挙げられる。

その際同時に複数のエージェントが制御木の同一の部分に変更を加えることも考えられ、特に制御木のような頻繁に更新されるデータの場合、同時に変更要求が起きたときの排他制御機構が重要となる。

4.1 制御木の管理方法

制御木の排他制御を効率良く行うことができれば、たとえハードウェア的にサポートされる共有メモリを用いたとしても、物理的に異なるプロセッサのメモリ上に、制御木を無秩序に分散させ管理することは望ましくなく、データの特徴に応じて最適な管理方法を用いることが望まれる。そこで本システムではデータベースと中間結果データに関しては通常の管理方法に任せるが、制御木に関しては独自の管理方法を二通り設計した。

以下で、スクラップアンドビルド型 DBMS のプロトタイプで用いられている方法とあわせて、三通りの管理方法について詳しく述べる。

4.2 server-client type

第一の方法が server-client type である。server-client type の概念図を図 6 に示す。この方法は、ただ一つ server となるプロセッサのローカルメモリ上で制御木を管理し、エージェントの制御木に対するアクセスは、server との通信を通して行うというものである。これは、スクラップアンドビルド型 DBMS のプロトタイプでも用いられている方法で、この場合、制御木は一ヶ所で一括管理するため、制御木に対する排他制御を容易に行うことができる。また、エージェントの把握もこの server のみが行うため、client 側のプロセッサでは、エージェントのオペレータ処理のみを行えばよい。

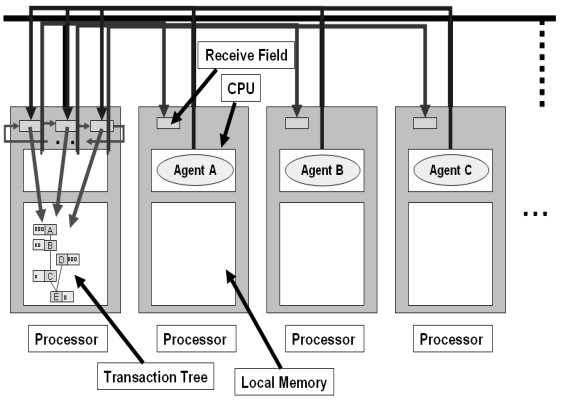

図 6: server-client type

しかし、server 側で制御木やエージェントの管理を行うことや、制御木にアクセスしようとする client からの通信が集中することから、server となるプロセッサに負荷が集中し、制御木に対する処理に時間がかかってしまうことが問題となる。また、何らかの障害により、server 機能がダウンしてしまうと、システム全体が機能しなくなるといった問題がある。

4.3 server-client type 改良型

4.1 節で述べた管理方法を改良し、処理時間の短縮を図ったものが、server-client type 改良型である。server-client type 改良型の概念図を図 7 に示す。この方法の、server-client type との大きな違いは、制御木を client 側の各プロセッサで、一つずつ保持・管理することである。

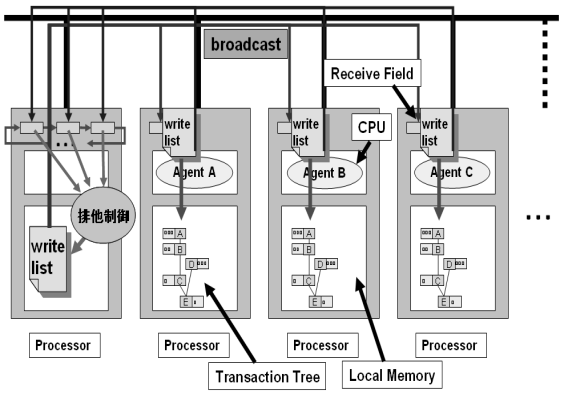

図 7: server-client type 改良型

クライアント側で制御木を保持することにより、制御木に対する read 処理は、それぞれのローカル

メモリにある制御木を参照すればよく、処理時間の大幅な短縮が見込める。一方、write 処理は、各プロセッサ上にある制御木の、プロセス間の整合性を保つため、まず各エージェントは書き込み命令を一度 server に送り、そこで同一データに対する書き込み命令があった場合、命令の到着順による排他制御を行い、有効な書き込み命令のみを server が client にブロードキャストする。そして各 client で、送られてきた書き込み命令を制御木に対して行うという方法をとる。この方法により、各プロセッサにある複数の制御木を同一の状態に保つことができる。

この改良により、read、write 処理時間の大幅な短縮が見込めるが、依然 server 機能の停止が、システム全体の停止につながるといった問題は残る。また、各 client に制御木が置かれる為、4.1 節の方法と比べ、制御木の管理に使用されるメモリの領域が何倍にもなってしまうことや、client 側の負荷の増加が欠点として挙げられる。

4.4 broadcast type

4.2,4.3 節で述べた方法に比べ、各プロセッサの自立性の高い管理方式が broadcast type である。broadcast type の概念図を図 8 に示す。この方法

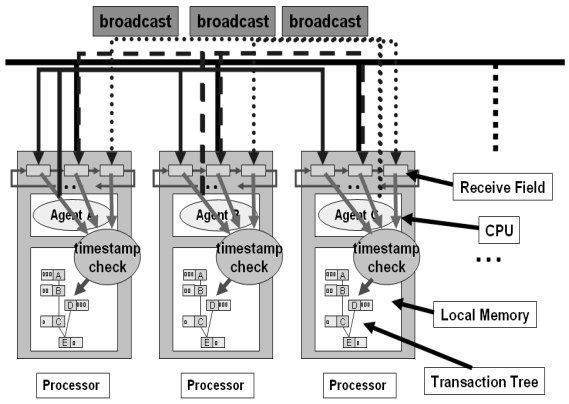

図 8: broadcast type

は、4.2、4.3 節の方法と異なり、全てのプロセッサが対等の関係にある。各プロセッサはそれぞれのローカルメモリに制御木を持ち、管理する。read アクセスは自メモリの制御木に対して行い、write アクセスは全てのプロセッサにブロードキャストを行うことで実現する。また、server となるプロセッサが存在しないため、排他制御やエージェントの管理もそれぞれのプロセッサで行う必要があるが、プロセッサごとに write 処理命令の到着順が異なる可能性があるため、到着順による排他制御では、各制御木を同一の状態に保持できない。そこで、broadcast type では、タイムスタンプを用い、同一データへの書き換え命令があった場合、それぞれの命令のタイムスタンプをチェックし、タイムスタンプの小さい命令を有効にするといったた排他制御を行い、各制御木の同期をとる。

新しくエージェントを登録する際は、エージェントの ID、処理名、引数と返り値のデータ型といった情報を全てのプロセッサにブロードキャストし、到着した情報を各プロセッサ上で管理する。検索文をコンパイルするエージェントは、自メモリ上に管理されるエージェントの情報を参照し、コンパイルの際に、検索に必要なエージェントが全て揃っているかチェックする。

4.2、4.3 節の方法では、server 機能を持つプロセッサが落ちてしまうことが、システム全体の停止につながっていたが、broadcast type では、たとえ一つのプロセッサが落ちたとしても、そのプロセッサで動いていたエージェントの機能が停止するだけで、システム全体に与える影響は小さい。

broadcast type は、複雑な排他制御を行うことや、ブロードキャスト通信が同時に多発した場合の通信のオーバヘッドを考えると、4.2 節と比べ、制御木に対する処理に時間がかかることが予測される。しかし、システムの信頼性においては、三つの方法の中で最も優れていることから、スクラップアンドビルド型 DBMS における制御木の管理方法としては、broadcast type が最も適していると考えられる。そこで、broadcast type の妥当性を検証するため次章で評価を行う。

5 評価

4 章で述べた制御木の管理方法をそれぞれ、並列計算機アーキテクチャ上で構築し、制御木に対する read 処理にかかる時間、write 処理にかかる時間を測定し、性能評価を行った。

なお、今回の測定に用いた並列計算機は、HITACHI SR2201(プロセッサ数32、メモリ512MB×9、256MB×23、計10.5GB) で、OS は HI-UX/MPP、通信方式はリモート DMA である。

5.1 実験結果

各実験はそれぞれの処理を 10 回ずつ測定し、その処理にかかった最小時間と最大時間を除き、平均をとったものである。また、write の命令を 4 種類用意し、同一データに対する同時書き込みが、頻繁に起こる環境を設定した。なお、実験結果の図の横軸 ”同時アクセス数 ”は同じタイミングで起こる制御木に対する処理要求の数を表し、縦軸 ”time[us] ” は同時に起こる処理要求を全て行うためにかかる時間 [us] を表している。

5.1.1 制御木に対する read 処理時間

各管理方法の read 処理時間を図 9 に示す。図 9 から、server-client type の read 処理時間は同時アクセス数に比例して増加していることがわかる。これは、read 処理の際も、server との通信を行うことや server 側で全ての read 処理を行うことが要因として挙げられる。一方、4.3、4.4 節で示した方法のように、自分のローカルメモリにアクセスする場合、同時アクセス数によらず、また他のプロセッサとの通信を行わないことから、高速にアクセスできることがわかる。

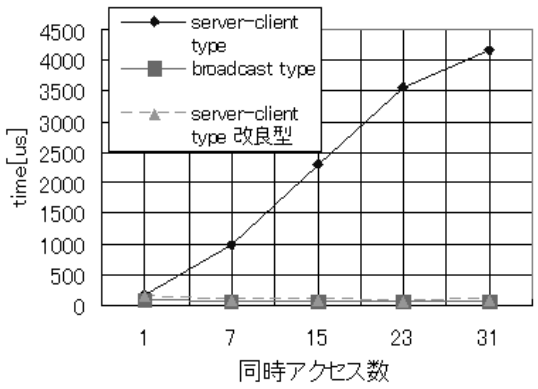

図 9: 同時アクセス数と read 処理時間の関係

5.1.2 制御木に対する write 処理時間

制御木に対する write 処理時間は、大きく 3 つに分類できる。

- プロセス間通信にかかる時間
- 排他制御にかかる時間
- 制御木にアクセスして書き込む時間

各管理方法の全 write 処理時間を図 10 に示す。また、write 処理時間の内訳のうち、同時アクセス数と通信時間の関係を図 11 に、同時アクセス数と排他制御にかかる時間の関係を図 12 に示す。

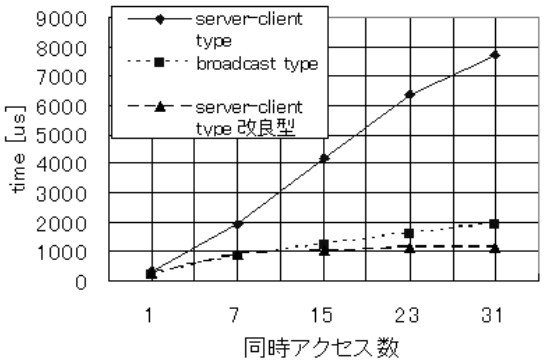

図 10: 同時アクセス数と write 処理時間の関係

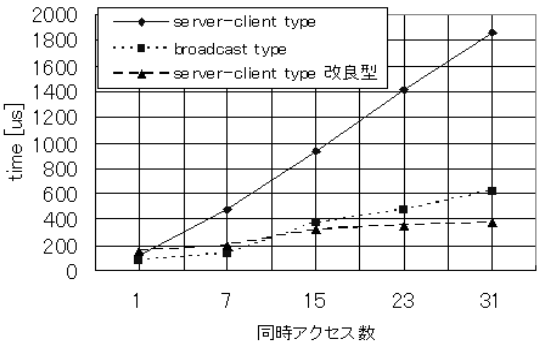

図 11: 同時アクセス数と通信時間の関係

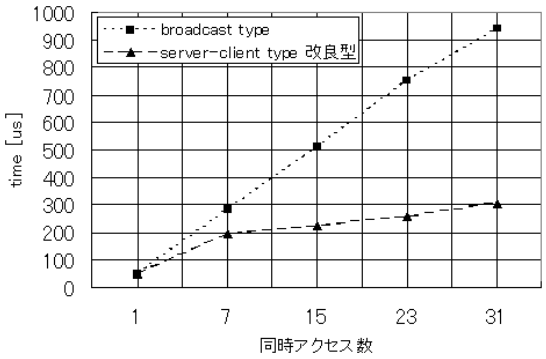

図 12: 同時アクセス数と排他制御時間の関係

図 10 から、他の二つと比べ信頼性の高い broadcast type は、write 処理による性能評価において、同時アクセス数が 7 以下で最も、また 15 以下でも、server-client type 改良型と同程度に、処理時間が短いことがわかる。

図 11 から、同時アクセス数が 7 以下では、broadcast type が最もプロセス間通信によるオーバヘッドが小さいということがわかる。また、同時アクセス数が増加してブロードキャストが同時に多数発生する場合でも、server-client type 改良型とそれほど差が開いていないことがわかる。

排他制御の方法の違いによる両者の差が図 12 よりわかる。今回は、同一データに対する同時書き込みが頻繁に起こる環境を設定したので、同時アクセス数が、増加するほど両者の差が開いているが、同一データに対しての書き込みがそれほど起こらない環境では、タイムスタンプまでチェックする必要がないため、同時アクセス数が増加しても、両者の差はそれほど開かないと予想される。

6 まとめ

本稿では、大規模マルチメディアデータを対象としたスクラップアンドビルド型 DBMS における大規模共有メモリの実現手法の提案を行った。特に、制御木の管理方法に重点を置き、二つの管理方法を設計し、従来までの方法とあわせて三つの方法を並列計算機上に構築し、実験による比較・評価を行った。

4 章で述べた管理方法の中で最も信頼性の高い broadcast type をスクラップアンドビルド型 DBMS に適した管理方法として挙げたが、write 処理についても同時アクセス数が 7 以下において、最も処理時間が短いことや、同時アクセス数が増加しても server-client type 改良型と大きく差が開かないことから、一定の実現性、妥当性を示したといえる。

今回は、制御木に対する処理時間について、同時アクセス数の関係を検証したが、処理時間には、同時にアクセスされる命令の中にどれくらいの割合で、同一データに対する命令があるかということも、関係してくると予測されるので、今後検証していきたい。

参考文献

[1] 森欣司: “自立分散システム [I]〜[V・完]”, 電子情報通信学会誌, vol.84, No.6, pp.403-408, No.7, pp.484-490, No.8, pp.611-617, No.9, pp.663-669, No.10, pp.734-740, 2001.

[2] 増永浩二、宝珍輝尚、都司達夫：“拡張可能 DBMS における部品の管理と呼び出しの一方法”, 情報処理学会論文誌, Vol.40, No.SIG6(TOD3), pp.152-161, 1999.

[3] 早田宏、渡辺美樹、田中圭、山崎伸宏：“オブジェクト指向データベース・エンジン Erath: キャッシュ共有のための設計空間”, 情報処理学会論文誌, Vol.39, No.7, pp.2240-2249, 1998.

[4] 河信司、市間健太郎、高橋賢一郎、家富誠敏、有澤博: “スクラップアンドビルド型マルチメディア DBMS の構成方式”, 情報処理学会研究報告, 情報処理学会, Vol.98-DBS-116(2), pp.273-280, 1998.

[5] 村上国男: “マルチエージェントシステムとその応用”, 電子情報通信学会誌, vol.78, No.6, pp.570-577,1995.