データベースシステム 109－31
（1996. 7. 25）

高速バスを活用した トランザクション処理アーキテクチャの性能評価

久保 進　　掛下 哲郎
佐賀大学 理工学部 情報科学科

〒840 佐賀市本庄町1番地,Tel. (0952) 28-8565,E-mail: kubo@cs.is.saga-u.ac.jp

データベース技術の高度化が急速に進行する中で、データベースシステムの中で最も低速なディスクシステムの高速化と並列化は不可欠である。我々はTrinityアーキテクチャを設計し、並列ディスク間でデータアクセス分布の変動に対して、データを動的に再配置することによって、効率的な負荷分散が行えることが分かっている。本論文では、Trinityアーキテクチャにおいてディスクアクセスを伴いボトルネックになり易い確定操作とアドレスを管理するモジュールの負荷分散の評価を行った。シミュレーション結果より、確定操作にかかる時間はディスク数に依存しないこと、またデータディクショナリを管理するモジュールの負荷分散ができることが分かった。

キーワード：並列処理アーキテクチャ、トランザクション処理、ブロードキャストバス、負荷分散

Performance Evaluation of Transaction Processing Architecture Utilizing High-Speed Bus

Susumu Kubo　　Tetsuro Kakeshita
Department of Information Science, Saga University

Saga 840, Japan,Tel. +81-952-28-8565,E-mail: kubo@cs.is.saga-u.ac.jp

Performance enhancement and parallel processing of the disk system which is the slowest component in the database system is essential while the advancement of database technology is rapidly proceeding. We have designed the Trinity architecture and have shown that the dynamic data migration against the change of the data access distribution can keep efficient load balancing. We evaluated the commit processing and the module managing data dictionary which tend to be a bottleneck due to disk access. By means of simulation experiments, we show that the time spent during commit processing does not depend upon the number of disks and the load balancing is effectively performed for the module managing data dictionary.

Keywords : Parallel Architecture, Transaction Processing, Broadcast Bus, Load Balancing

1 まえがき

コンピュータシステムの効果的な高速化手法として並列処理が挙げられるようになって久しい。データベースの高度化が急速に進行する中で、データベースシステム中で最も低速なディスクシステムの並列化は不可欠である。しかし、並列処理を行う場合、要素間の負荷分散を図らなければ台数効果による速度向上は望めない。並列ディスクに対する最適な負荷分散は、データアクセス分布に依存して決定されるため、アクセス分布の変動に伴ってデータをディスク間で動的に再配置する必要がある。そこで我々は高効率トランザクション処理用マシンのための Trinity (三位一体) アーキテクチャ[1, 2, 3] を設計してきた。これまでに、動的データ再配置のシミュレーション結果よりディスク間の負荷をほぼ均等に分散できることが分かっている [4]。

ここで Trinity アーキテクチャの特徴を簡潔に示す。(1) 各モジュール間での並列処理により高い処理能力を実現する。(2) ブロードキャスト通信により、複数モジュールへの同時通信を行うことで、通信コストを低減する。(3) 通信オーバーヘッドなしに動的データ再配置を行い、ディスク間の負荷を均一に保つ。

今回は以上で述べた動的データ再配置による結果を踏まえて、他にディスクアクセスを伴いボトルネックになり易い部分、確定操作とアドレスを管理するモジュールの評価を行った。ブロードキャストバスにより各モジュールに同時通信できるため、データベースやデータディクショナリへの書き込みがディスク間で並列処理できるため、確定操作にかかる時間はモジュール数に影響されない。またデータディクショナリを管理するモジュールは 1 台ではボトルネックになり易い。そこで複数台のモジュールにそれぞれ全データディクショナリを保持し、アドレス変換は各モジュールに均等に、アドレス更新は全モジュールで行うようにしてモジュール間で負荷分散できる。さらにグループコミット技術を応用して、アドレス変換に伴うディスク書き込みを減らす方策を提案する。

本論文の構成は以下のようになっている。2 節で Trinity アーキテクチャの概要を示し、3 節で確定操作の動作、4 節で動的データ再配置の動作を説明する。そして、3 節、4 節のアイデアや性質に基づくシミュレーション結果をそれぞれ 5、6 節で示す。

2 Trinity アーキテクチャ

図 1に Trinity アーキテクチャを示す。モジュール間のメッセージはメインバスまたはアドレスバスへブロードキャストされるため、すべてのモジュールは接続されたバスを常時モニタリングしており、当該モジュールの物理番号が付加されたメッセージだけを受け取る。以下、各構成要素について概要を述べる。

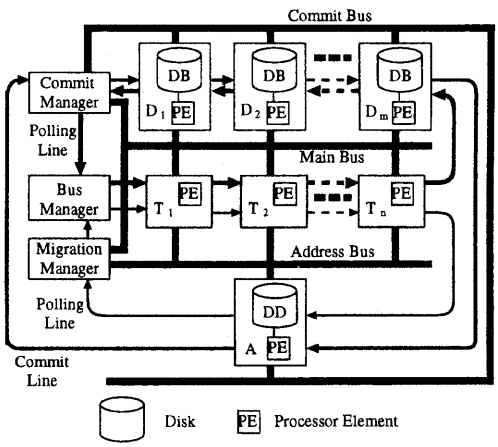

図 1: Trinity アーキテクチャ

トランザクションモジュール　トランザクションモジュール T_i は処理単位を実行する。**read/write** 要求を実行するために、T_i はアドレス管理モジュール A に依頼して、論理アドレスを物理アドレスに変換する。T_i は変換された物理アドレスを用いて、データモジュール D_j に **read/write** 命令を送ってデータアクセスを要求する。D_j から応答を受け取ると、当該処理単位の実行を再開する。

データモジュール　データモジュール D_j には物理データ (DB) が割り当てられている。D_j が保持するデータに対するアクセス要求が到着すると、それに対するデータアクセスを行って、応答メッセージを T_i モジュールに送る。**write** 操作を行うと、書き換えられたデータがログに保持される。ログに保持されたデータは、確定管理モジュールの指示に従って二相コミットされる。

アドレス管理モジュール　アドレス管理モジュール A はデータディクショナリ (DD) を保持しており、T_i からアドレスバスを経由して到着した論理アドレスを物理アドレスに変換する。

バス管理モジュール　本アーキテクチャでは、すべてのモジュールがバスマスターになれるため、メインバスおよびアドレスバス上でのバス調停が不可

欠である。バス管理モジュールはこのバス調停をバス利用キーを用いたポーリングによって行う。

再配置管理モジュール　再配置管理モジュールは、メインバスを常時モニタリングすることによって各データモジュールの負荷状況を調べ、必要に応じて再配置開始命令や終了命令をアドレスバスに出力する。

確定管理モジュール　確定管理モジュールは、T_iからの確定要求が到着すると、全てのD_j及びAモジュールを制御して二相コミットを実行する。本モジュールの動作は3節で詳述する。

3　確定操作

本アーキテクチャではDBとDDをAモジュールとDモジュールで分散しているため二相コミットが必要である。この二相コミットの動作を図2と表1に示す。二相コミットは処理単位が変更した内容をログに書き込むまでのPrepare Phaseとログからデータベースあるいは DD へ反映するまでのCommit Phaseに分かれる。

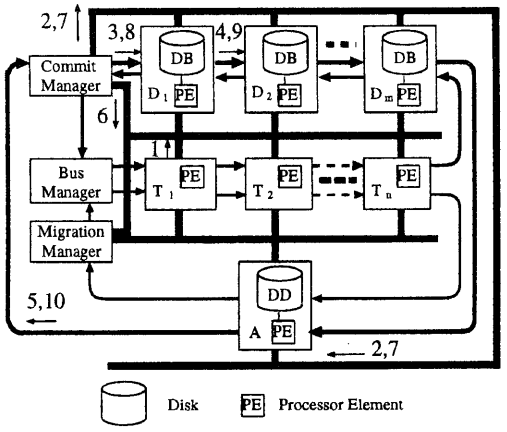

図 2: 確定操作の動作

本アーキテクチャの二相コミットのアイデアを以下に示す。図2と表1の2と7の命令はブロードキャストバスにより全モジュールに同時通信され、同時にそれぞれのモジュールで書き込みを開始する。次に、3と8のメッセージが各モジュールに書き込みの終了確認のために流れ、各モジュールでの書き込みを順次チェックする。最終的に、5と10のメッセージを確定管理モジュールが受け取るとそれぞれ

	Prepare Phase
1	確定管理モジュールへのコミット開始命令
2	変更データのログへの書き込み開始命令
3	A、Dモジュールの書き込み確認
4	自モジュールでの確認後、次のモジュールへ確認メッセージを送信する
5	全モジュールの書き込み終了確認
	Commit Phase
6	論理的な意味での処理単位の確定操作終了
7	ログからディスクへの書き込み開始命令
8	A、Dモジュールの書き込み確認
9	自モジュールでの確認後、次のモジュールへ確認メッセージを送信する
10	全モジュールの書き込み終了確認

表 1: 確定操作の流れ

のPhaseは終了する。2と7の命令によって同時に書き込みを行うので、確認に要する時間は全A、Dモジュール中の最大アクセス時間とほぼ一致し、A、Dモジュール数にほとんど依存しない。

4　動的データ再配置

動的データ再配置は再配置管理モジュールの指示に従って動作する。Dモジュールの中で最も負荷の大きいDモジュールをD_{max}、最も負荷の小さいDモジュールをD_{min}とする。動的データ再配置の動作を図3と表2に示す。本方式は一つのメッセージを複数モジュールが受け取って、異なった動作を行う点に特徴がある。

動的データ再配置方式は次に列挙する性質を持つ.

- データアクセスとデータ再配置がブロードキャストバス上で一度で行えるために、メインバス上での余分なデータ転送が必要ない。
- アクセス頻度の高いデータほど再配置されやすくなるため、少ない回数の再配置で負荷分散を行える。
- 動的再配置に伴うオーバーヘッドは、最も負荷の軽いDモジュールへの書き込みコストとして発生するため、性能への影響が少ない。

次に、今回のシミュレーションによる評価で関連のある動的データ再配置に伴うAモジュールの動作について説明する。AモジュールはDモジュー

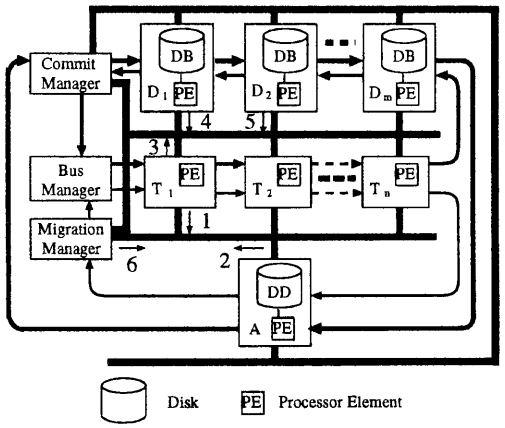

図 3: 動的データ再配置の動作

ルのデータベースを検索するために必要なアドレスを全て保持している。従って、D モジュールのデータベースの読み出し/書き込みのアクセスに対して、必ず A モジュールでアドレス変換を行わなければならない。これでは A モジュール 1 台ではボトルネックになり易い。また再配置中のデータに対して再びアクセス要求があった場合には、再配置された後のデータのアドレスを確認する前にそのデータをアクセスすることはできない。従って、再配置中のデータに対する要求は、再配置中の処理単位が終了するまで待たされることになる。このため処理単位の各読み出し/書き込み要求のアドレス変換から要求終了までの平均処理時間を知ることは非常に重要である。

5 確定操作の高速化

本アーキテクチャの確定操作のアイデアと性質は 3節で述べたが、ここではその理論通りにシミュレーション結果が表れることを確かめる。結果を図 4に示す。縦軸はコミット時間、横軸はアクセス要求の理論的限界値に対する割合を示す。そして 5 Modules と 10 Modules は A モジュール数 1 個に対して D モジュール数がそれぞれ 5 個と 10 個の場合である。図 4より確定操作にかかる時間は、理論的限界値の 70%までは D モジュール数によってはほとんど影響がないことが分かる。これはより多くのデータ量を持つトランザクションシステムにおいて、ディスクが増大しても、確定操作にかかる時間はボトルネックにならないことを示している。

読み出し要求時の動的データ再配置	
1	A モジュールへのアドレス変換命令
2	T_iへのアドレス変換応答
3	D_{max}への読み出し命令
4	D_{min}へのデータ書き込み命令 T_iへの読み出し応答
5	D_{max}へのデータ無効化命令 再配置管理モジュールへのアドレス更新命令
6	A モジュールへのアドレス更新命令
書き込み要求時の動的データ再配置	
1	A モジュールへのアドレス変換命令
2	T_iへのアドレス変換応答
3	D_{min}への書き込み命令
4	(該当する命令なし)
5	D_{max}へのデータ無効化命令 T_iへの書き込み応答 再配置管理モジュールへのアドレス更新命令
6	A モジュールへのアドレス更新命令

表 2: 動的データ再配置の流れ

6 アドレス管理モジュールの負荷分散

6.1 シミュレーション結果

A モジュールのデータも D モジュールと同様にディスク上に保存されている。従って、要求数が増えると、4節で明らかにしたように、A モジュールがボトルネックになり易い。そこで A モジュールでも負荷分散する必要がある。しかし、A モジュールのデータは再配置することができない。それは D モジュールのデータを再配置するために、データのアドレスを知っている A モジュールが必要であったように、A モジュールを再配置するならば A モジュールのデータのアドレスを知っている何らかのモジュールが必要になるからである。そこで以下のような方法を考えた。

- T モジュールに対して A モジュールを 1 対 1 に対応させ、T_iからのアドレス変換要求は A_iで処理する。
- 各 A モジュールに全 DD を保持する。
- アドレス更新は全 A モジュールで同時に行うようにする。

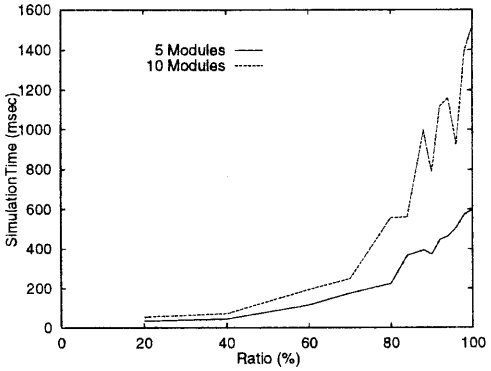

図 4: 確定操作の時間による性能評価

この方法には以下のような特徴がある。

- T モジュールに要求が均等に発生するようにすると、T モジュールと A モジュールは 1 対 1 に対応しているので、アドレス変換は各 A モジュールで均等に行われるようになる。またアドレス更新は全 A モジュールで行われる。これによって負荷が均等に分散される。
- アドレス更新数がアドレス変換数 (要求数) に対して多くなれば多くなるほど A モジュールの負荷は大きくなりボトルネックになり易い。

A モジュールの負荷分散を考えた Trinity アーキテクチャを図 5に示す。

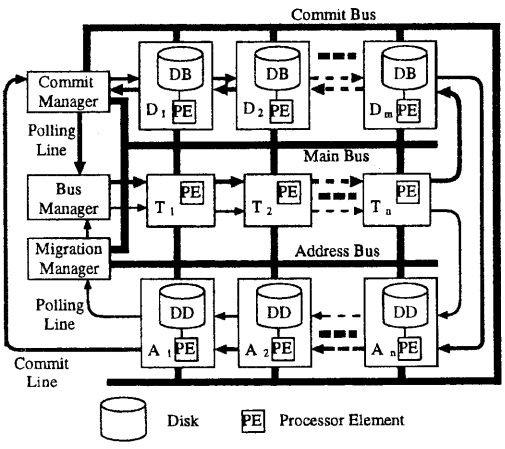

図 5: アドレス管理モジュールの負荷分散を考えた Trinity アーキテクチャ

A モジュールの負荷分散の効果を調べるシミュレーション結果を図 6 と図 7に示す。図 6と図 7はそれぞれ A、D、T モジュール数 5 個と A、D、T モジュール数 10 個の場合である。Transaction は処理単位が終了するまでの時間、CommitDone は理論的な意味での処理単位が終了するまでの時間、Write は書き込み要求が終了するまでの時間、Read は読み出し要求が終了するまでの時間を示す。横軸の Request Arrival Rate は 1 秒間に発生するトランザクション数を示す。各処理単位 (トランザクション) は読み出し要求、書き込み要求そして確定操作を順に実行する。

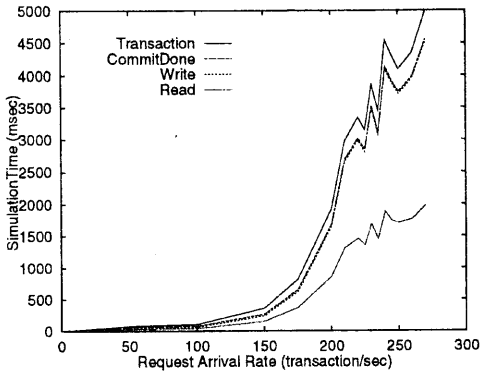

図 6: モジュール数 5 個の場合のトランザクション処理時間

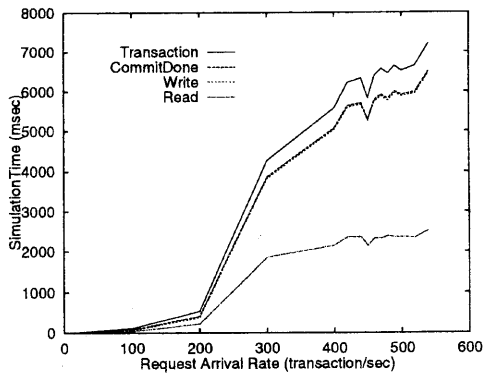

図 7: モジュール数 10 個の場合のトランザクション処理時間

A モジュール数 5 個の場合には、アクセス要求の理論的限界値 (トランザクション数 250) の約 70%までは処理時間の増加の伸びは緩やかで、処理時間と

しても問題とならない程度の値である。しかし、10個の場合には、限界値(トランザクション数500)の約40%で処理時間が急増している。この原因を調べるためにアドレス更新の発生数を調べてみた。図8がそれである。縦軸が全Aモジュールでアドレス更新された回数、5 Modulesと10 ModulesはAモジュールの数を示す。モジュール数5個に対して10個の場合では、より多くアドレス更新が行われていることが分かる。この理由は以下のように考えられる。動的データ再配置を行う場合では、アドレス変換は要求数にのみ依存し、Aモジュール数に依存しない。しかし、アドレス更新は要求数が同一であってもAモジュール数に比例して増加する。このためAモジュール数が多ければ多いほどAモジュール全体の負荷は大きくなる。このようなことから図7の結果が表れると考えられる。

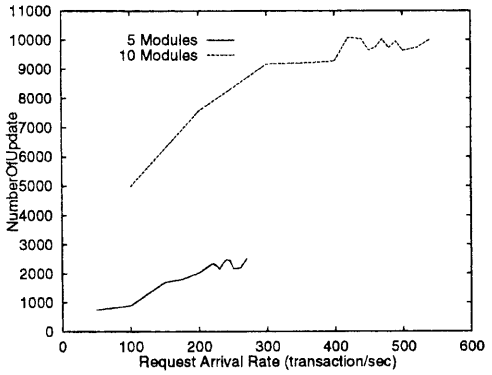

図 8: アドレス更新の発生数

6.2 考察

ここでは6.1節の結果を改善するためのアイデアを紹介する。そのためにはAモジュール数に影響されない確定操作の方法を考えなければならない。確定操作されるAモジュールのデータ長はDモジュールのそれに比べて非常に短いという特徴とグループコミット技術を活用してAモジュールの確定操作を行う。この方法を表3に示す。A_iはAモジュール(iは1〜nまでの整数)を示し、t_iは処理単位(iはトランザクションが発生した順番)を示す。アドレス更新は全Aモジュールで行うが、コミットは全てのAモジュールで行わず1つのAモジュールだけで行う。そしてそれまでコミットされずにいたデータ全てをグループコミットする。表3では、例えばA_3は処理単位t_3の時にコミットされ、t_1からt_3までに更新された内容をコミットすることを示す。これは

	A_1	A_2	A_3	⋯	A_n
t_1	1	2	3		n
t_2	n+1	2	3		n
t_3	n+1	n+2	3		n
⋯	n+1	n+2	n+3		n
t_n	n+1	n+2	n+3		n
t_{n+1}	n+1	n+2	n+3		2n
t_{n+2}	2n+1	n+2	n+3		2n

表 3: アドレス管理モジュールの確定操作

Aモジュールが保持するデータ長が非常に短いために可能である。従って、Aモジュール数がn個の場合には、n個の処理単位が終了する毎にコミットによるディスクアクセスが一回発生する。以上の改善によりAモジュール数に依存しないように負荷分散が行える。なおこの改良によるシミュレーションは現在実行中である。

7 むすび

今回は、動的データ再配置の結果を踏まえて、他にボトルネックになり易いディスク処理を伴う部分の評価を行った。確定操作にかかる時間はモジュール数にほとんど影響ないことがわかり、またアドレス管理モジュールについても負荷分散を行うことができた。以上のことから今後は、Trinityアーキテクチャの全体のバランスを考えたパフォーマンスの調整を行い、また故障発生を想定した本アーキテクチャの設計を行う。

参考文献

[1] 掛下, 高橋 : ブロードキャストを利用した高効率トランザクション処理マシンの基本アーキテクチャ, 情報処理学会研究報告, Vol. 94-DBS-97-8, 1994.

[2] 高橋, 掛下 : 高効率トランザクション処理用 Trinity アーキテクチャのフロー解析, 情報処理学会アドバンストデータベースシステムシンポジウム, 1994.

[3] 高橋, "高効率トランザクション処理用 Trinity アーキテクチャの設計", 佐賀大学情報科学専攻修士論文, 1995.

[4] 掛下, 久保 : ブロードキャストバスを活用した並列ディスク間の動的データ再配置方式, 電子情報通信学会データ工学研究会, 1995.