データベースシステム　97－8
（１９９４．　３．　１８）

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

掛下 哲郎　　高橋 正行
佐賀大学理工学部情報科学科

トランザクション処理を行なう際には並行処理制御が不可欠である。しかし、並行処理スケジューラは、すべてのデータ操作を処理するためボトルネックになり易い。また、確定処理に伴うディスクアクセスも効率低下の主要な要因である。本稿では、このようなトランザクション処理システムのボトルネックを解消するための基本アーキテクチャを提案する。本アーキテクチャは、処理単位モジュールとデータモジュールからなる。並行処理制御をデータモジュール間で分散し、バスを用いたブロードキャストによってモジュール相互間の通信量を低減する。さらに、データモジュール内でのパイプライン処理によって確定操作のスループット向上を図る。本アーキテクチャのボトルネック解析やブロードキャストを利用したデータの動的な再配置についても論じる。

A Basic Architecture using Broadcasting for High Performance Transaction Processing Machine

Tetsuro Kakeshita　Masayuki Takahashi
Department of Information Science, Saga University,
Sage 840, Japan

Concurrency control is necessary for transaction processing systems. However the concurrency control scheduler tends to be a bottleneck since it handles every data access operation. Commit processing is also a major performance degrading factor because it requires disk access. We propose a new hardware architecture in order to avoid such bottlenecks within transaction processing systems. The architecture is composed of transaction modules and data modules. Concurrency control facility is distributed among data modules. Communications between the modules are significantly reduced using broadcasting. Furthermore commit operation is optimized by pipelining within data modules. We also discuss the bottleneck analysis of the architecture and dynamic data migration using broadcasting.

1 まえがき

定型的な処理単位 (Transaction) を高速に実行するトランザクション処理システム (以下 TPS と略記) は社会的なニーズも高く、将来的にも一層の性能向上が要求されている。TPS 内で処理単位は共有データをアクセスしながら並行処理されるため、データの矛盾を防ぐために並行処理制御が不可欠になる。性能向上のための効果的な手法として並列処理が挙げられるが、通常の TPS では処理単位の全てのデータ操作を並行処理スケジューラが直列に処理しているので、スケジューラ自体が並列動作のボトルネックになりやすい。また、確定操作も二次記憶アクセスを必要とするためボトルネックになる可能性が高い。

このような TPS のボトルネックを解決するために、TP 専用マシンのアーキテクチャを提案する。本アーキテクチャはマルチプロセッサのハードウエアモデルを用いて負荷分散を行ない、バスによるブロードキャストを利用して通信回数を低減する。また、確定操作にもパイプライン処理やデータの再配置等を導入することによって負荷の高い処理の効率化を目指している。

従来、TPS の効率化は TP モニタによってソフトウエア的な手法で行なわれることが多かった [1]。[2] では、マイクロプログラミングを用いて並行処理スケジューラのハードウエア化を試みているが、ボトルネック問題には触れていない。TP 用に特別なアーキテクチャを設計している研究としては、図 1に示す DataCycle アーキテクチャ[3] がある。しかし、このシステムはデータベースの全データが数秒程度で転送できるような高速のブロードキャストネットワークを必要とする。また、読み出し処理単位に関するボトルネックは生じないが、データ変更を行なう処理単位に関するボトルネックは依然として残る。

商用のシステムでは、ORACLE が 1073TPS の性能を持つ TP 専用マシンを提案している [4]。このシステムは、Sun Workstation のバックエンドとして動作し、ハードウエアは 113 個のノードを持つ。このうち 64 個のノードはアプリケーションの実行に使用され、各ノードが 16MB の RAM を持つ。また、48 個のノードは IO 処理に使用され、16 ノードが 1MB の、32 ノードが 4MB の RAM をそれぞれ持つ。残りの 1 ノードはホストとのインタフェースを司る。

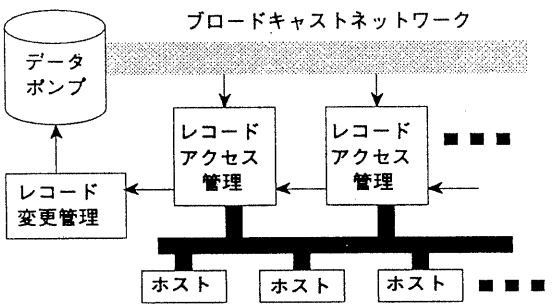

図 1: DataCycle アーキテクチャ

本研究では、TP マシンのアーキテクチャを工夫することによって、ORACLE の TPS を上回るスループットが可能になることを示す。

本稿では、以下の順序で説明を行なう。2節では、設計したアーキテクチャとその基本的な性質を説明する。3節では、ブロードキャストを用いたモジュール間通信のプロトコルやブロードキャストを用いる利点について説明する。4節では、設計した TP アーキテクチャにおける並行処理方式について考察する。5節では、パイプライン処理とデータの再配置を用いた確定操作の効率化について説明する。6節では、設計した TP アーキテクチャの各モジュールについてボトルネック解析を行なう。

2 TP アーキテクチャの特徴

図 2は、TP 専用マシンのアーキテクチャである。$T_1, \cdots, T_n$ は処理単位または処理単位集合を実行するためのモジュールであり、処理単位モジュール (以下 T モジュールと略記) と呼ぶ。$D_1, \cdots, D_m$ はデータまたはデータブロックに対するアクセスを処理するためのものであり、データモジュール (以下 D モジュールと略記) と呼ぶ。D モジュールについては 2.1節でその内部構成を説明する。図中の太線はバスを表しており、バス管理モジュールはバス調停を行なう。また、確定管理モジュールは確定操作の管理を行なう。確定用ラインは D モジュールをデージーチェーン結合しており確定操作の際に用いる。このアーキテクチャには、以下のような性質がある。

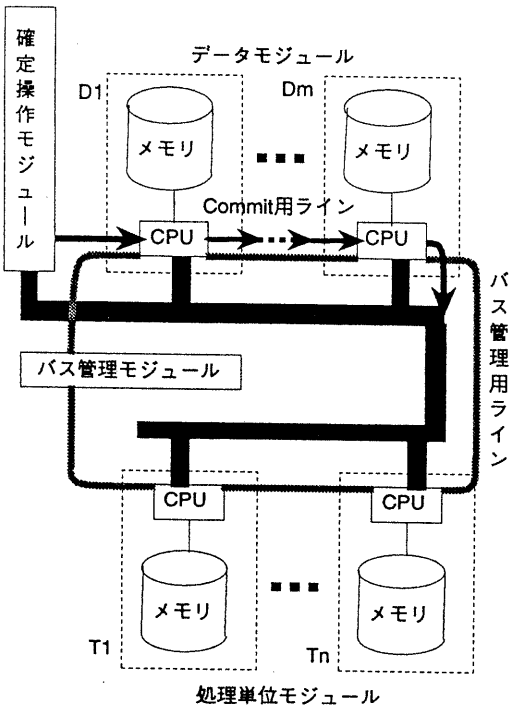

図 2: TP 専用マシンのアーキテクチャ

ブロードキャスト バスを通じてTモジュールとDモジュールが直接通信可能になっている。このため、1回の通信で1つのTモジュールから複数のDモジュールへデータアクセス要求を出したり、Dモジュールから複数のTモジュールに対してデータの転送を行なうことが可能になり、通信コストを削減できる。

1CPU/処理単位 処理単位毎の実行は直列に行なわれることが多いため、各処理単位にCPUを割り当て、処理単位間でデータ操作以外の処理を並列に実行する。通常、処理単位数はCPU数よりも多いため、各CPUには複数の処理単位を割り当てることによって並行処理を行なう。この際、再入可能手続きによる実行コードの共有を行ない、二次記憶アクセスを低減することもできる。

1CPU/データ 各データに対する操作は並行処理制御を行なうために直列処理しなければならないので、各データに独立したCPUを割り当てることによって、データアクセスの並列度を最大にできる。また、Dモジュール間でデータを分散しているため、モジュール当りのワーキングセットを小さくでき、二次記憶アクセスの低減を図ることができる。

バス調停 Tモジュールからのデータアクセス要求が公平にサービスされるように、TモジュールとDデータモジュールを以下に示す順序でリング上に結合し、その上でキーを順次送付することでバス利用権を与える (polling)。

$$T_1 \quad T_2 \cdots T_n \quad T_1 \cdots$$
$$D_1 \quad D_2 \cdots D_m \quad D_1$$

また、確定管理モジュールのバス利用は最優先であり、TモジュールとDモジュールの順序に関係なくバス利用権を与える。これを実現するために、確定管理モジュールのバス利用権要求はバスへのブロードキャストによって行なう。

確定用ライン Dモジュールを確定用ラインでデージーチェーン結合することによって、確定操作のパイプライン処理が可能となる。これによって、確定操作のスループット向上を図る。

2.1 Dモジュールの内部構成

Dモジュールの内部構成を図3に示す。Dモジュールは以下に示す処理を行なう。

1. Bus Monitorによって、ブロードキャストをモニタリングし、要求されたデータに関する入出力を行なう。(3.1節参照)
2. 各Dモジュールの負荷を均等にするために、データの動的な再配置を行なう。(3.2節参照)
3. データアクセスに関する並行処理制御を分担する。(4.1節参照)
4. 確定された処理単位が変更したデータをログに転送し、その後DBに反映する。(5.2節参照)

3 ブロードキャストを利用したモジュール間通信

本節では、ブロードキャストを用いたモジュール間通信について説明する。以下、TモジュールとD

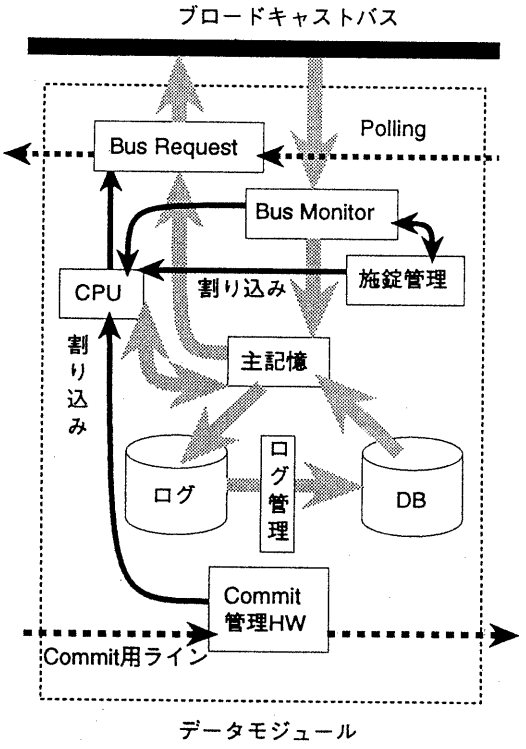

図 3: D モジュールの内部構成

モジュール間の通信の具体的動作について説明し、データの再配置を用いた D モジュール負荷の均一化について説明する。

3.1 T モジュールと D モジュールの通信

Read, Write 操作を実行する際の T モジュールと D モジュール間の通信の具体的動作について説明する。各モジュールは表 1に示す情報をバスに出力する。

以下に Read 操作の処理手順を示す。Write 操作に関する処理は、データを T モジュールから D モジュールに送ることと、操作終了を Acknowledge で通知することが異なるだけで、Read 操作と同様に定義される。

【 Read 操作の処理 】

1. T モジュールは、バス利用権を待つ。
2. T モジュールは、バスに処理単位識別子 T_ID とデータアドレスを出力する。

Read 操作	
T モジュール	
⇓ T_ID, アドレス ⇓	⇑ T_ID, DATA ⇑
D モジュール	

Write 操作	
T モジュール	
⇓ T_ID, DATA, アドレス ⇓	⇑ T_ID, ACK ⇑
D モジュール	

表 1: データ操作に対応するバス情報

3. T モジュールは、バスをモニタしながら D モジュールの送信メッセージを待つ。(バスを専有しない)
4. D モジュールは、バス上のアドレスを識別して、T_ID のデータ操作を操作待ち行列に追加する。
5. D モジュールは、T_ID のデータ操作を実行した後にバス利用権を待つ。
6. D モジュールは、バスに T_ID とデータ値を出力する。読み出し要求が複数の処理単位から要求されている場合には、複数の T_ID とデータ値をバスに出力する。
7. T モジュールは、バス上の識別子を判断して、次のデータ操作を実行する。

これらの Read/Write 操作の処理において、T モジュールはデータアクセス要求を出した後、バスを専有した状態で応答を待つことがない。従って、バスの利用時間を短縮できる。ポーリングによってバス利用権がモジュールに与えられた際、T モジュールは複数のデータアクセス要求を出しても良い。また、D モジュールは複数のデータアクセス要求に応答しても良い。

3.2 データの動的な再配置

特定の D モジュールにアクセスが集中すると、そのモジュールがボトルネックになる。これを避ける

ために、データを動的に再配置して、Dモジュールの負荷を均一化する。

データの動的再配置は、Tモジュールの書き込み要求の際、またはDモジュールの読み込み応答の際に行なう。Tモジュールはバスのブロードキャストをモニタリングすることによって、最も負荷の小さいDモジュールD_{min}を調べておく。Tモジュールがデータを書き込む際に、そのデータを保持するDモジュールD_{org}の負荷が、D_{min}の負荷に対して一定基準を超えた場合には、データをD_{min}に書き込み、D_{org}には当該データの無効化メッセージを送る。また、Dモジュールも読み込み応答の際に、当該データをD_{min}にも送信することによってもデータの動的再配置を実現できる。

この方式は、すべてのモジュールがバスをモニタリングすることによって各モジュールの負荷を把握するので、モジュール間の通信は必要なく、再配置に伴う余分な負荷がバスにかかることもない。データアクセス要求は負荷の高いDモジュールに集中するため、そのようなモジュールからのデータ再配置は容易になる。ただし、バスを常時モニタリングして負荷を調べるためのハードウエアが必要になるが、各モジュールで同一のハードウエアを利用できるため、コストを抑えられると期待している。

4 並行処理制御

本節では、設計したアーキテクチャに適する並行処理方式について考察する。また、処理単位の実行制御における各モジュールの動作について説明する。

4.1 並行処理方式

並行処理方式には様々あるが、直列可能スケジュールを判定する並行処理方式では、並行処理スケジューラ自体がボトルネックになる可能性が高い。そこで、本アーキテクチャでは、分散処理が可能な並行処理方式を採用する必要がある。その代表として時刻印方式 [5] と二相施錠方式 [6] について考察する。

時刻印方式の実現は、時刻印割り当て用のカウンタを要するだけで良いため、容易に行なえる。しかし、処理単位開始時に割り当てられた時刻印と処理単位の終了順序の不一致により、不必要な後退復帰が多発する可能性があるので、本稿のアーキテクチャに適した並行処理方式とはいえない。

二相施錠方式を用いた場合、ロックの管理は各Dモジュールが分担することができる。しかし、デッドロック検出に Wait for Graph (WFG) を用いた場合、WFG へのアクセスがボトルネックになる。これを避けるためには、各Dモジュール毎に時間切れによるデッドロック検出または Wait-Die 方式等の採用が必要である。Wait-Die 方式は、時間切れによるデッドロック検出の特別な場合 (ロック待ち時間=0) なので、本アーキテクチャでは、各Dモジュール毎に時間切れによるデッドロック検出を採用する。

この方式では、各Dモジュールは、各データ操作のロック操作の管理を行ない、バス上に Pre_Commit 命令の応答を確認すると、対応する処理単位のロックをすべて解除する。ロック待ち時間を経過してもロックが許可されない場合には、デッドロックに陥ったとみなして対応する処理単位に対する Abort 命令をバスに出力する。ロック待ち時間は、後退復帰が多発しない程度に長く、デッドロック検出が遅れない程度に短く設定しなければならない。

4.2 処理単位の実行制御

実際のシステムではデッドロック等の問題が発生する確率は低いが、問題が生じた場合には Abort 命令によって処理単位を中止 (もしくは再実行) する。Abort 命令は、表 2に示す場合に出力される。

モジュール	後退復帰要因
処理単位	利用者による強制終了
データ	ロック待ちの時間切れ
	確定操作の失敗
確定管理	後退復帰処理単位の確定操作

表 2: Abort 命令を出すモジュール

図 4のように T_ID と Abort 命令がブロードキャストされると、各Dモジュールは T_ID の処理単位が変更したデータを復旧する。また、各Tモジュールは、T_ID を判断して対応する処理単位を中断または再実行する。これらの操作は、各モジュール単位で並列に実行できる。なお、Abort 命令は確定管理モジュールに記憶されるため、後退復帰された処理単位が確定されることはない。

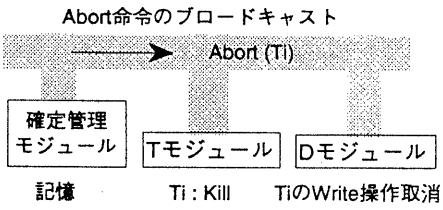

図 4: 処理単位の中止

5 確定操作の効率化

確定 (Commit) 操作は、二次記憶アクセスを必要とするため、データベース操作の中でも処理コストのかかる操作の１つである。また、本稿のアーキテクチャは D モジュールを分散しているため、二相コミットが必要である。そこで、確定操作を高速化するため Commit 用ラインを設け、D モジュールの間でパイプライン処理を行なうことによってスループットの向上を図る。本節では、確定操作のパイプライン処理の基本動作を説明する。また、二次記憶アクセスの効率化やグループコミットの導入についても考察する。

5.1 二相コミットのパイプライン処理

処理単位が終了すると、T モジュールは Pre_Commit 命令をバスに出力する。これに対するバスへの応答が Yes であれば、T モジュールは Commit 命令をバスに出力することによって二相コミットを実行する。この際に、T モジュール、各 D モジュールおよび確定管理モジュールは以下の動作を行なう。Commit 命令に対応する動作は、Pre_Commit 命令に対応する動作と本質的に同一なので省略する。

【 Pre_Commit 命令の処理 】

1. T モジュールがバスに T_ID と Pre_Commit 命令を出力する。

2. 確定管理モジュールは T_ID の Pre_Commit を認めるかどうかを判断する。

   - T_ID に対して Abort 命令が出ていなければ、Commit 用ラインに Pre_Commit 命令の応答 Yes と T_ID を出力する。
   - Abort 命令が出ていればバスに Abort 命令を出力する。

3. D_k モジュール ($1 \leq k < m$) は、Commit 用ラインからの入力が Yes の場合に限り、T_ID 処理単位の Pre_Commit を認めるかどうかを判断する。

   - Pre_Commit を認める時は Commit 用ラインに応答 Yes と T_ID を出力する。
   - Pre_Commit を認めない場合にはバスに T_ID と Abort 命令を出力する。

4. T モジュールは、D_m モジュールの応答が Yes であれば、バスに T_ID と Commit 命令を出力する。

Commit 用ラインによって D モジュールをデージーチェーン結合しているため、確定操作の実行に当たっては図 5に示すようにパイプライン処理が可能となり、スループットが向上する。また、ハードウェア量も少ないため D モジュールの追加も容易にできる。

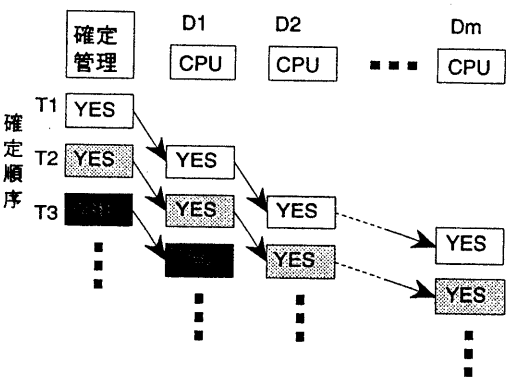

図 5: 確定操作のパイプライン処理

5.2 ディスクスケジューリング

D モジュールは二次記憶アクセスを必要とするので、ボトルネックになる可能性がある。そこで、メモリを増やしてページフォルトを少なくする等の工夫が必要となる。また、確定処理を効率化するために、処理単位毎にトラック (またはシリンダ) を割り当てる。これによってディスクヘッドのシークが不要になるため、ログへのデータ転送が高速化できる。

ログへの確定操作時間は、平均回転待ち時間に大きく依存しているので(6.1節)、各Dモジュールで変更されたデータ量にあまり影響されず、ほぼ同一時間で処理できる。従って、確定操作のパイプライン処理が円滑に行なえる。また、グループコミットを導入して、一回毎のパイプライン処理のスループットを向上させることも可能となる。

ログからDBへのデータ転送は、図6に示すように、シーケンシャルアクセスでログからメモリに読み込み、ディスクスケジューリングを用いてDBに転送することでシーク時間を最小にできる。

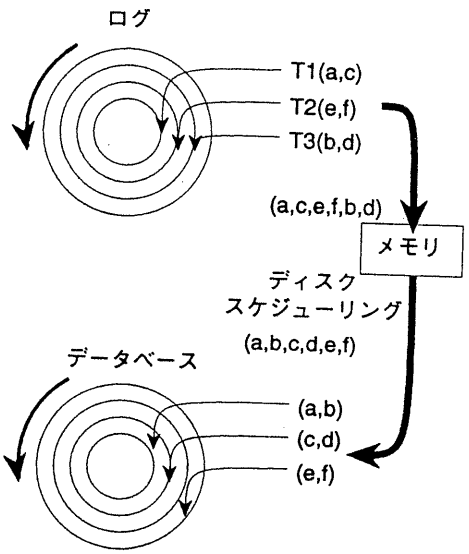

図 6: ディスクスケジューリング

6 ボトルネック解析

本稿で提案したアーキテクチャにおいては、Tモジュールのすべての処理およびDモジュールのほとんどの処理についてはモジュール毎に完全な並列動作が可能である。従って、システムの性能はモジュール数に比例する。しかし、共有バス上の通信とDモジュールによる確定処理については、上記の性質が成り立たない。本節ではこれらの処理に関するボトルネック解析を行ない、本アーキテクチャの能力を検証する。

6.1 確定操作の負荷

[1]によると、代表的なTPSの主要パラメータは、表3に示す通りとなっている。本節では、この値に基づいて確定処理の負荷解析を行なう。

表 3: TPS システムの標準値

IO/処理単位	10 回
データ量/IO	500 Byte
ディスク回転数	4000 rpm
データ転送速度	5MB/秒

各処理単位は平均10回のデータアクセスを行なうが、そのうち読み出し操作の比率を$\alpha(0 \leq \alpha \leq 1)$とする。確定操作においては、当該処理単位が変更したすべてのデータをログに転送する必要がある。このために必要なデータ転送時間は

$$\frac{10(1-\alpha) \times 500\text{Byte}}{5\text{MB/秒}} = (1.0-\alpha) \times 10^{-3}\text{秒}$$

となり、平均回転待ち時間を考慮すると、ログディスクの動作時間は$(8.5-\alpha) \times 10^{-3}$秒となる。この式から、確定操作の処理時間においては、回転待ち時間が主な決定要因になることが分かる。確定操作は8〜10ミリ秒程度を要するので、処理単位のスループットは100〜125tps程度である。この操作はパイプライン処理を行なっているため、Dモジュール毎のスループットはシステム全体のスループットと等しい。従って、処理単位毎に確定処理を行なうと、システムのボトルネックになる。これを防ぐために、グループコミット等を導入して確定操作当たりのデータ転送量を増やし、スループットを向上させる必要がある。

6.2 バス通信の負荷

バスの負荷解析は、表4に示す32bit processor用Multibus II[7]の仕様に基づいて行なう。Multibus IIは、CPU-メモリバスとしては標準的なものである。しかし、バスに対する負荷は無視できるレベルにあることが示される。

本稿のアーキテクチャでは、各モジュールが送り先の応答を待たないので、応答待ち時間を考慮する必要はない。従って、アドレスのデータ量を考慮した場合のデータ転送速度は、40MB/sとなる。この

同期バス	複数マスター可能
データバス幅	32bit
アドレスバス幅	32bit
(データ/アドレスバス共用)	
最大データ転送速度	40MB/s
(64bit データを 200ns で転送)	

表 4: Multibus II (Intel, 32bit processor 用)

バスを利用した処理単位のバス専有時間を計算すると以下のようになる。

【 処理単位当たりのバス専有時間 】

- 処理単位当たりのバス使用回数 B=10 回。
- 図 7のようにアドレス/データバスの bit を割り当てると、データ転送時間 T_dは、8byte につき 200ns である。(計算を簡略化するために 1byte のデータを 25ns で転送するものとする)
- Pre_Commit メッセージと Commit メッセージの送信に各々200ns 必要とする。また、Yes (または Abort) メッセージの送信に 200ns を要するため、確定操作によるバス専有時間 T_cは 800ns となる。
- データ転送要求 (Read)、Write 確認には、各々 T_o=400ns 必要である。

第 1 送信ワード				
アドレス	R/W	T_ID	空き	データ長
32bit	1bit	18bit	5bit	8bit
200ns				
第 2~n 送信ワード				
データ (64bit)				
200ns				

図 7: バスのビット割当

上記の仮定の下で、Read/Write 操作当たりの転送データ量が 500 Byte の場合のデータ転送時間は

$200\text{ns} + 25\text{ns} \times 500 = 12.7\mu\text{s}$

となり、バス専有時間は

$(12.7\mu\text{s} + 400\text{ns}) \times 10 + 800\text{ns} = 131.8\mu\text{s}$

($[\text{バス専有時間}] = ([T_d] + [T_o]) \times [\text{B}] + [T_c]$)

となる。これを tps に換算すると

$10^6 / 131.8 = 7587(\text{tps})$

となる。ディスクアクセス当たりのデータ量を D (Byte) とすると、式 1で tps を計算できる。

$$\frac{10^6}{0.25 \times D + 6.8}(\text{tps}) \qquad (1)$$

式 1を用いた計算結果をまとめると表 5が求められる。

データ量/IO	バス専有時間 (μ s)	tps
50 Byte	19.3	51,813
100 Byte	31.8	31,447
500 Byte	131.8	7,587
1 KB	262.8	3,805
5 KB	1287	777
10 KB	2567	390

表 5: バスの専有時間と tps

表 5の計算結果は、データ転送時間と確定操作だけを考慮したものであるが、従来以上の tps が期待できることが分かる。スループットはバスの速度に比例するが、本解析で用いた以上の性能を持つバスも存在する。従って、バス自体がボトルネックになる可能性は低い。

7 むすび

本稿で提案したアーキテクチャは、比較的単純でボトルネックの少ないものになっている。そのため実現も容易であることが期待される。バスのモニタを常時行なうために特殊なハードウエアを必要とするが、各モジュールともに同一のハードウエアを使用できるため、コスト的には比較的低いと予想している。

今後の課題は多いが、その一部を以下に列挙する。

- 各モジュールのハードウエア詳細設計
- 障害発生時の処理
- 実験的な性能評価

謝辞 本アーキテクチャに関して議論して頂いた情報科学科の渡辺義明 教授、松藤 信哉 助手に感謝します。

参考文献

[1] Gray, J., Reuter, R., *Transaction Processing : Concepts And Techniques*, pp.1–90, Morgan Kaufmann, 1993.

[2] Robinson, J. T., "A fast general-purpose hardware synchronization mechanism", Proc. SIGMOD, pp.122–130, 1985.

[3] Herman, G., et al., "The Datacycle architecture for very high throughput database systems", Proc. SIGMOD, pp.97–103, 1987.

[4] ORACLE, "TPC Benchmark^{TM} B Disclosure Report for the nCUBE 2 Scalar Supercomputer Model nCDB-1000 Using ORACLE V6.2", Part No. 3000097-091, Document Revision 1.2, July, 1991.

[5] Bernstein, P. A., Goodman, N., "Timestamp-based algorithms for concurrency control in distributed database systems", in Proc. Int'l. Conf. on Very Large Data Bases, pp.285–300, 1980.

[6] Eswaren, K. P., et al., "The notion of consistency and predicate locks in a database systems", Comm. ACM, Vol.19, No.11, pp.624–633, November 1976.

[7] Hennessy, J. L., Patterson, P. A., *Computer Architecture: A Quantitative Approach*, pp. 528–532, Morgan Kaufmann, 1990.