リレーショナルデータベースマシンにおけるデータステージング機構の考察

SOME CONSIDERATIONS ON DATA STAGING MECHANISM FOR RELATIONAL DATABASE MACHINES

加藤 洋，瀬尾 和男，上林 憲行，相磯 秀夫
Hiroshi KATO, Kazuo SEO, Noriyuki KAMIBAYASHI, Hideo AISO
慶應義塾大学 工学部
Faculty of Engineering, Keio University

1. はじめに

近年，データベース管理システム（DBMS）の重要性が高まってきている。しかしDBMSを従来の計算機システム上でソフトウェアレベルでサポートするには，処理効率，システムの信頼性，ソフトウェアの生産性等に関して問題点があることからいわゆるデータベースマシンの研究開発が各所で精力的に行なわれてきている。しかし現時点では，まだまだ試行錯誤的な状況を脱していない。それは提案されているシステムが現行の計算機システム技術の延長線上にあることを仮定している場合が少なく非常にラディカルな変化を要求していることと，専用化のレベルがいろいろ考えられたが基本データモデルとの関連もあって普遍的な方式やメカニズムとして定着しにくいということが主なる理由である。

現行の計算機システムの基本的な体系と方式を踏まえた上でデータベース処理に特有なメカニズムを付加して処理効率を改善しようというアプローチは現行の計算機システム技術との整合性や現実性という面から重要であるとされている。こうしたアプローチは，特定のデータモデルに依存しない汎用的なデータベース処理の特質を強力にサポートするため一般にデータベース処理の最大のボトルネックであるデータ供給能力に関連した記憶システムや記憶管理技法の改善を目指したものが多い。そうした流れにあるシステム例としては，IBM社が発表したOSインタフェース（チャネルコマンド）との整合性を維持しディスクコントローラに1トラック分のバッファを用意しそこで物理レコードから目的レコードを選択する機構を付加したDASDプロセッサ[1]，同様に記憶管理技法としては単一レベル記憶とリレーショナルに準じたデータベースのプリミティブをファームウェアレベルでサポートしているIBMシステム／38，記憶階層技法としてはIBM 3850システムやCCD等の電子ディスクを採用したディスク・キャシュシステムの提案と実用化，日電中研で開発が進められているGDS[2]も記憶システムをデータベース処理という観点から再構成したものである。また，MITのMadnickの提案している記憶の多階層構造と各レベルにおける最適な論理の付加を組み合せた記憶システム（Infoplex）なども注目されている。

一方，特定のデータモデルを基本としたデータベースマシンにおいては，データベース格納媒体に対して直接，連想処理機構を付加するいわゆるデータベース格納媒体依存型アーキテクチャを採用しているものとデータベース格納媒体とは独立した演算システムをHost計算機との間に設定し，データベース処理の負荷分散と専用化による処理効率向上を目指すアプローチに大きく分けて考えられる。特にリレーショナルデータベースマシンを完全にサポートするには前者の固定ヘッドディスク等の回転周期の長い回転型記憶媒体に直接的に論理を付加しても集合論的な演算系を効率良く実行しえない（Original RAP）[3]。むしろ現実的な方法としては現行の特に記憶密度の進歩によって大容量化が著しい移動ヘッド型ディスク（IBM 3370では実に

500MB以上の容量）を前提としたシステムを考えることが肝要であり，さらにデータベース向きの記憶システムの管理技法，ディスクスケジューリング方式，データ格納方式，読み出し方式等の工夫により，データステージング能力を最大限に高める記憶システムを設定し，その上で演算システム（RAP-2[4]，DIRECT[5]）を構築するというアプローチが有望視されている。（RAP-2[4]，DIRECT[5]のようなシステムはデータベースストア独立型アーキテクチャである。にもかかわらずデータステージング能力の向上に対する配慮がないデータベースマシンには自づからその能力に限界がある。）

本稿では現在，筆者らが検討を進めているリレーショナルデータベースマシンSPIRIT[7][9]で，リレーショナルデータベースを効果的にサポートするため次のようなストラテジを導入している。①強力なデータステージング機構を実現する先回りステージング技法，すなわちデータ駆動型による演算系とデータステージング系との体系的な制御方式，②この制御方式と整合のとれたディスクスケジューリング方式と記憶階層方式，③ディスクの並列読み出し機構の付加，格納単位の最適化及びデータ圧縮技法等の導入（転送量・処理量の削減によるスループット向上を目指す。）

次章以下では，これらの方法論の説明とそれらの方式の効果に対する基本的考察及びその定性的評価とシミュレーションによる定量評価を行なう。

2. リレーショナルデータベース向き記憶システムの基本的考察

この章では従来の汎用計算機の記憶システム，汎用データベースマシンの記憶システム及びリレーショナルデータベース向き記憶システムについて比較・検討し，議論を進めていく。

2.1 汎用計算機の記憶システム

汎用計算機の記憶システムの機構の特徴は次の通りである。（Fig.2-1(a)）①プログラム（コード）とデータの混在　②デマンドページングによる記憶管理技法　③ワーキングセットやLRUに基づくリプレースメント　④チャネルによる，演算と入出力の同時動作とマルチプログラミング技法。

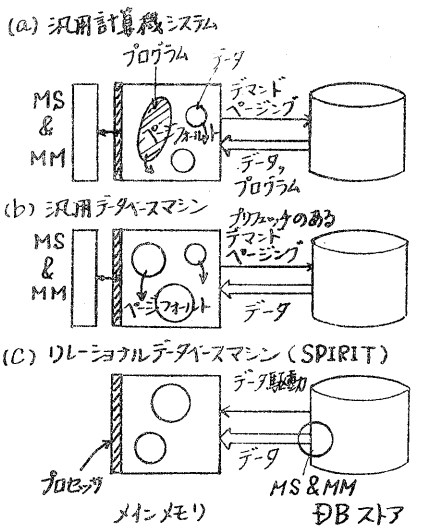


Fig. 2-1　記憶システム

2.2 データベース向きの記憶システムの特性

一般にデータベース処理においては，大量のデータに対する単純な操作（構造検索・連想検索）が代表的である。このため，汎用システムと異なり，プログラム（コード）のフォールトは起こらず，データのフォールトだけが起こる。論理セグメントのサイズは大きく，1セグメント内では定形化された処理が繰り返されるデータベースではデータの供給順序を予想することが可能である。また，データベースに対する検索はファイルを宣言するために必要なファイル群はあらかじめ知ることができる。データベース処理といっても実際の物理操作レベルでは各種のディレクトリ情報や転置ファイルを含めて構造的なデータ構造を持っている場合とリレーショナルデータベースのように均質で非構造的な場合では大きな違いもあるが，以上述べたことは両者に共通した特質である。

2.2.1 汎用データベースマシンの記憶システム

汎用データベースマシンの記憶システムの機構の特徴を列挙すると次の通りである。(Fig.2-1(b))①処理の対象はデータでプログラムは固定。②構造データに対する配慮からデータステージングの完全な予見が不可能なためデマンドページ

ングが行なわれるが，論理セグメント内の他ページのプリフェッチが可能である．③リプレースメント管理は必要である ④演算と入出力の同時動作は可能であるが，ページ駆動型のプロセススケジューリングは不可能である。

2.2.2 リレーショナル専用マシンの記憶システム

リレーショナルデータベースマシンの記憶システムを考察する。(Fig. 2-1 (C)) その特徴としては，①処理の対象はデータで，プログラムは固定 ②リレーションは一般的に，内部的に構造を持たないためリレーションを構成するページはすべて同等・均質と見なすことが可能であり，データステージングの順序を静的に決定することが可能である。すなわち，動的な挙動を追尾するデマンドページング方式ではない先回りのステージングが可能となる。③記憶システムを完全に制御してリプレースメントの動的な要求を起こさないようにすることが可能である。④データ駆動型のプロセススケジューリングを行なうことが可能である。

次章では，このデータ駆動型制御方式の定性的評価及びシミュレーションによる定量的評価を述べ，さらにこの制御方式をサポートする記憶階層方式，ディスクスケジューリングに言及し，リレーショナルデータベースマシンの記憶システムの構成について考察する。

3. リレーショナルデータベースマシンの記憶システムの構成に関する考察

3.1 従来のシステムにおけるデータベース処理に対するマルチプログラミング技法の限界

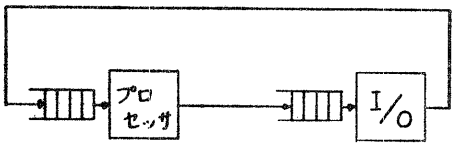
プロセッサのスループット向上に対するマルチプログラミング技法の限界を Fig. 3-1 に示す解析モデルに基づいて考察する。すべてのプロセスはプロセッサで平均 λ^{-1} 時間処理された後，I/O 要求を出し，平均 μ^{-1} 時間 I/O 処理された後，プロセッサに処理要求を出すということを繰り返す。

m ： プロセスの多重度

ρ ： μ/λ とすると，

プロセッサのスループット η は，

$$\eta = \begin{cases} \rho(1-\rho^{m})/(1-\rho^{m+1}) & [\rho<1] \\ m/(m+1) & [\rho=1] \\ \rho(\rho^{m}-1)/(\rho^{m+1}-1) & [\rho>1] \end{cases}$$



λ : プロセッサ処理率 (λ^{-1}: プロセッサ処理時間)

μ : I/O 処理率 (μ^{-1}: I/O 処理時間)

η : プロセッサのスループット

$\lambda e^{-\lambda t}$: プロセッサの処理時間分布

$\mu e^{-\lambda t}$: I/O の処理時間分布

Fig. 3-1 解析モデル

m と η の関係を ρ をパラメータとして示したグラフが Fig. 3-2 である。このグラフから，$\rho<1$ で小さいときにはプロセッサのスループット η はきわめて悪く，プロセスの多重度 m を大きくしてもほとんど効果がないことがわかる。結局，η を増加させるためには μ^{-1} を小さくするか，λ^{-1} を大きくすることが必要である。

以上を踏まえて，演算処理系（プロセッサ）とデータ供給系（I/O 処理系）を体系的に統合したデータ駆動型先回りステージング方式を考案し，アーキテクチャ的に μ^{-1} を小さくすることを目指した。

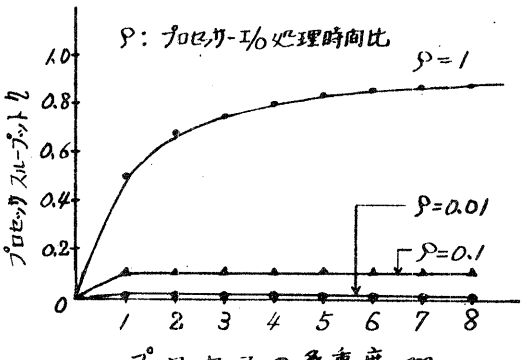
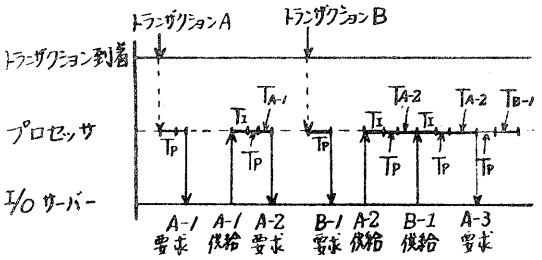


Fig. 3-2 プロセッサスループットとプロセスの多重度の関係

3.2 データ駆動型制御方式による先回りステージング

デマンドページング方式とデータ駆動型制御方式の違いを示すために Fig. 3-3 にそれぞれ



T_P : プロセス スケジューリング時間

T_I : 割り込み処理時間

T_{A-i} : トランザクションA(B)の i 番目のセグメントの処理時間
$_{(B)}$

Fig. 3-3(a) デマンドページング方式

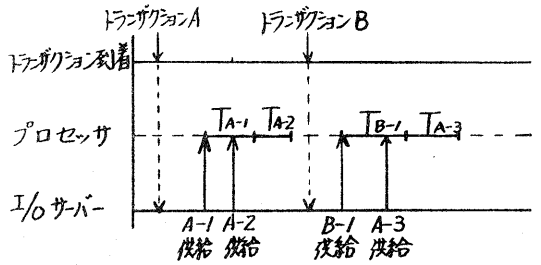


Fig. 3-3(b) データ駆動型先回りステージング方式

の方式の挙動を示す。デマンドページング方式においては、データが必要になったところでデータフォルトによるI/O要求が発生し、そのデータがプロセッサメモリ(記憶階層における最上位のメモリ)にステージアップされるまで、そのI/Oを出したプロセスは止まるか、まだ処理可能な部分があれば、その処理が行なわれる。マルチプログラミングの環境においては、そのプロセスがそこで実行不可能になった場合、他のプロセスに制御を移し、転送完了の割り込み処理、プロセススケジューリングを経て、再び初めにI/Oを出したプロセスに制御が戻り、処理が続行される。(Fig. 2-1(a) 参照) 一方、データ駆動型制御による先回りステージング方式はデータの処理、スケジューリング、記憶管理、データ転送制御が機能分散された複数のプロセッサの間で非同期に処理され、データ処理系とデータ供給系の間の並列処理の可能性を高める。この方式は①データベース専用プロセッサ指向(データ専用メモリの用意)②リレーショナルデータベースの採用 を前提としている。デマンドページング方式では命令シーケンスがスケジューリングのマスターであるのに対し、データ駆動型制御方式ではデータ(ページ)のシーケンスがスケジュールを決定する。

データ駆動型制御方式がデマンドページング方式に比べてまさっている点は大きく次の2つに集約されると思われる。①演算プロセッサにおけるデータ操作とそこへのデータ転送処理の非同期化によるI/O, 割り込み処理オーバーヘッドの除去 ②各トランザクションで使われるファイル群がトランザクションのシステム到着時にわかるため先回りステージングによってディスクスケジューリングの対象となるI/O要求数が増し、スケジューリングの効果が高まる。

次節において、これらの点を評価するための、ディスクシステムを中心とした、データ処理系及びデータ供給系を含めた全体の系をモデル化し、解析する。

3.3 ディスクシステムのモデル化及び解析

ディスクシステムを中心としたデータ処理・供給系のモデルをFig. 3-4に示す。このモデルにおける仮定は以下のようなものである。①ディスクスケジューリング方式としては、先回りの効果をうまく反映でき、かつスケジューリングの公平さという観点からCSCAN方式を採用する。(Table 3-1 参照)[8] FCFS方式ではデータの局所性を無視した場合、I/O要求が

	処理効率	処理の公平さ	先回り環境下での処理効率向上
SSTF*	○	×	○
SCAN	○	△	○
CSCAN**	△	○	○
FCFS***	×	○	×

* Shortest-Seek-Time-First

** Circular-SCAN

*** First-Come-First-Served

Table 3-1 各ディスク・スケジューリング方式の比較

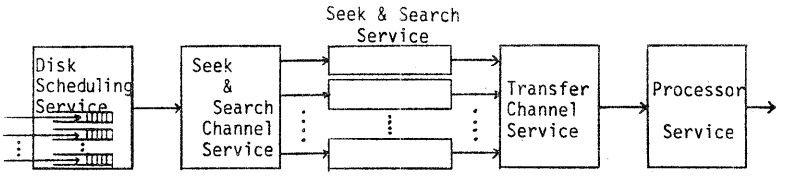


Fig.3-4 ディスクシステムを中心としたデータ処理・供給系のモデル

1つのキューを作り，I/O処理サービスの順序が決まってしまうため，先回りの効果は期待できない。②記憶階層レベルはDBストアとプロセッサメモリの2階層である。③プロセッサメモリの容量は先回りステージングを行なうのに十分なほど大きい。④複数のディスク装置が1つのチャネルを共用する。1つのディスク装置がチャネルと結合している間に他の装置がオフラインシークあるいはオフラインサーチの後，サービスを要求した場合は再結合ミスとなり，もう1回転して再びチャネルサービスを要求する。[6] チャネルの関与は，シーク動作開始時と回転待ち動作開始時及びデータ転送動作開始時からデータ転送終了時までである。⑤到着トランザクション内の各処理にはすべてシーケンスがある。デマンドページング方式においては各データのI/O要求は前のデータの処理終了後に初めて出されることになる。ところが，データ駆動型制御方式においてはプロセッサメモリの容量が無限に十分あるとの仮定により，トランザクション内の各データを同一レベルのデータとしてトランザクションのシステム到着時に同時にスケジューリングの対象とする。⑥マルチプログラミングの環境を仮定する。⑦デマンドページング方式においても割り込み処理，スケジューリング処理時間は無視する。従って，3-1節で述べた効果の①は検証できない。

シミュレーションに先だって，解析的にデータ駆動型制御方式の利点を確かめる。この解析ではチャネル，制御装置は1つのディスク装置に専用にあると仮定する。ディスクからのステージングを行なう場合，その効率はシーク時間，サーチ時間及び転送時間に依存している。ディスクスケジューリングではこれらのうち主にシーク時間を改善しようとするものであり，スケジューリングの自由度を考えた場合，ディスクへの総I/O要求数が多いほど有利であると考えられる。

今，I/O要求が各シリンダに均等に分散すると仮定する。L＝シリンダ数，t＝両い合わせ到着時間間隔，T＝応答時間，$M = T/t$ ＝多重度，N＝両い合わせあたりのI/O要求数，n＝先回りステージングをしない場合の総I/O要求数，n'＝先回りステージング導入した場合の総I/O要求数　とする。

先回りステージングをしないとき（デマンドページング方式），各両い合わせでは一時に1つのI/O要求しか出さないので，多重度Mが総I/O要求数となる。

$$n = M$$

これに対し，先回りステージングを導入したとき（データ駆動型制御方式），各両い合わせ（応答時間＝T'，多重度＝$M' = T'/t$）でのI/O要求数はある時点でそれぞれ，$\lfloor N/M' \rfloor$，$\lfloor 2N/M' \rfloor$，…，$\lfloor M'N/M' \rfloor$ となり，総I/O要求数n'は

$$n' = \sum_{i=1}^{M'} \lfloor i \cdot N/M' \rfloor$$

しかるに，総I/O要求数の差（$|n-n'|$）がそのまま応答時間の差になるわけではないので，$T' \fallingdotseq T$（$M' \fallingdotseq M$）と見なすことができる。そこで，

$$n : n' = M : \sum_{i=1}^{M} \lfloor i \cdot N/M \rfloor$$

次章で述べるシミュレーションで仮定したパラメータの値を使って，I/O要求数nと平均シーク時間の関係を Fig.3-5 に示す。これを使って

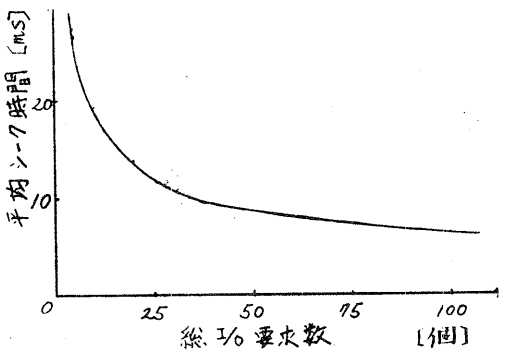


Fig.3-5 総I/O要求数と平均シーク時間

双方の平均シーク時間を比較できる。具体的な例として $x = 100$ [ms], $T = 3$ [s], $N = 5$ の場合で比較してみると $n = 30$, $n' = 90$ となり、平均シーク時間は先回りステージングをしないとき、10.7 [ms], 導入したとき、6.7 [ms] となる。

次節にシミュレーション結果を示す。

3.4 シミュレーションの結果

前節で予測したデータ駆動型先回りステージングの優位性を検証するため、Table 3-2 に示した環境のもとでシミュレーションを行なった。

プロセッサスピード		2 μs
セグメント数/トランザクション		5 個
DB ストア	移動ヘッドディスク数	2 個
	シリンダ数/装置	100 本
	トラック数/シリンダ	10 本
	トラック容量	13 kB
	シーク時間	平均 25 ms
	サーチ時間	平均 8.4 ms
セグメント容量		平均 6.5 kB

Table 3-2 シミュレーションの環境

以下にその結果を示す。Fig. 3-6 に単位時間当たりのステージアップセグメント数, Fig. 3-7 に両方式の平均シーク時間の比, Fig. 3-8 にプロセッサ稼動率を示す。これらの結果からデータ駆動型先回りステージング方式の優位性が示

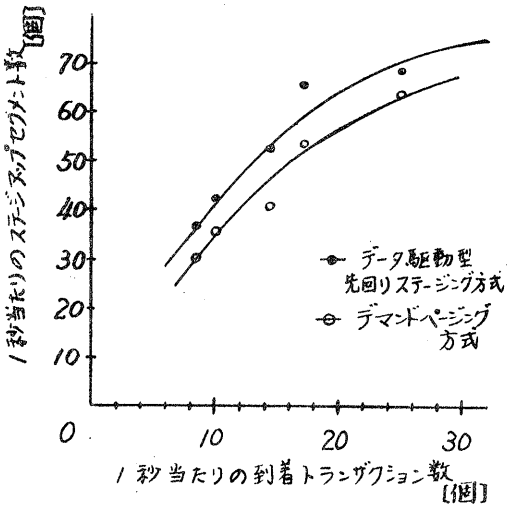


Fig. 3-6 単位時間当たりのステージアップセグメント数

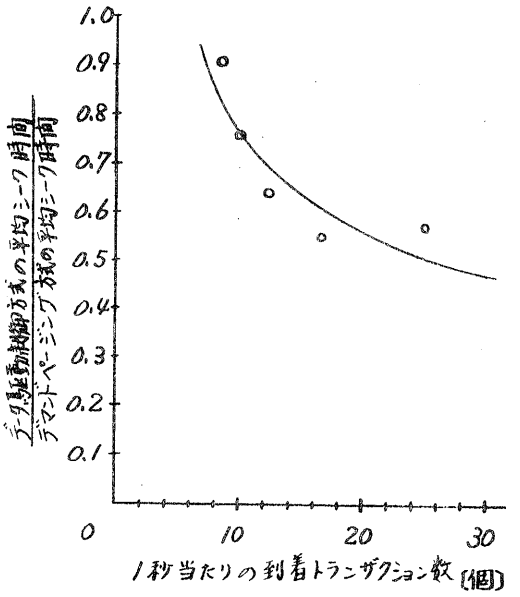


Fig. 3-7 データ駆動型制御方式導入による平均シーク時間の減少率

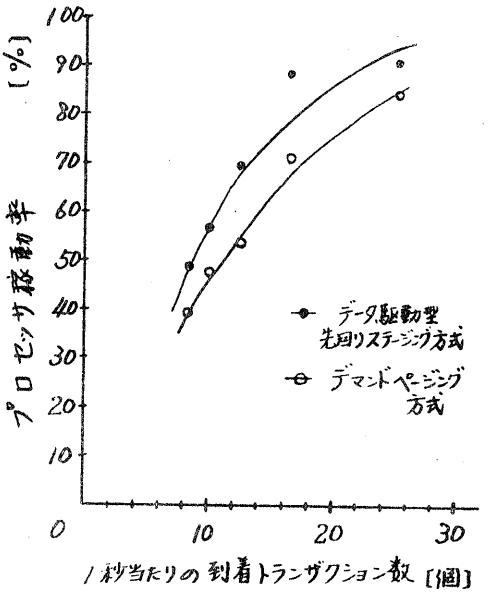


Fig. 3-8 プロセッサの稼動率

された。定性的に予測された効果とほぼ一致した。

次に、データ駆動型制御方式を具体化するうえでの実際的な技術的課題について検討する。

3. 5 データ駆動型制御方式の導入によって生じる技術的課題

[1] プロセッサメモリの容量の有限性

現実的に処理の中にあるシーケンスを無視してデータ供給系独自のストラテジに基づいてステージアップした場合、不活性データ（すぐに実行が可能でないデータ）によってメモリオーバーフローを引き起こし、デッドロックに陥る可能性がある。これを解決する手段として、①トランザクション内の並列度を高める。（活性データを増す。）②マルチプログラミングの多重度を制限する。③十分に大きなプロセッサメモリを用意する。などが挙げられるが、③はコスト的に非現実的である。そこで、もっと現実的な手段として中間バッファ導入による記憶階層方式を提案する。これについては次節で述べる。

[2] 再接続ミスの多発

CSCAN ディスクスケジューリングは先回りステージングの効果をうまく引き出すが、複数のディスク装置が1つのチャネルを共用する場合、各ディスク装置が同じCSCANという方式をとると、統計的にチャネルに対するサービス要求に競合を起こしやすく、まして先回りステージングによってステージアップされるデータの数がふえた場合にさらに再接続ミスが多くなる。（Fig. 3-9） 再接続ミスによって生じるオーバーヘッド及びその対策については次々節で述べる。

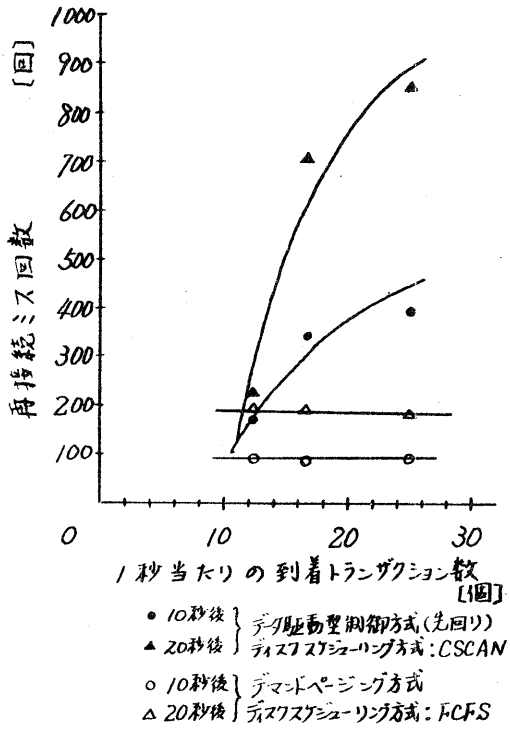


Fig. 3-9 再接続ミス回数の比較

3. 6 記憶階層方式の導入

デマンドページング方式かデータ駆動型制御方式の選択、さらに中間レベル記憶を導入するかいなかの選択で、Fig. 3-10に示すような4つの記憶システムのモデルが考えられる。

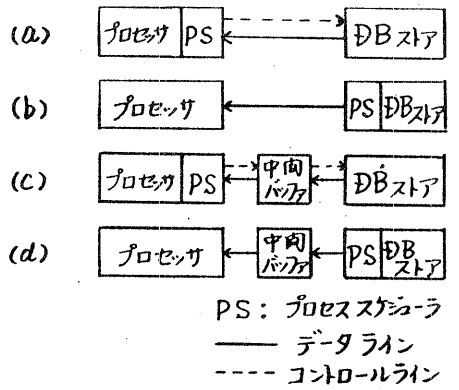


Fig. 3-10 記憶システムの各モデル

(a)は現行のディスクシステム。(b)はトランザクション内の並列性が高い場合に可能となる2レベル記憶のデータ駆動型先回り制御方式。(c)はデマンドページングで、中間バッファはいわゆるディスクキャッシュとしての効果をもつ。(d)はリレーショナルデータベースマシンの記憶システムとしては最も望ましいと考えられる形態。その理由として、以下のことが挙げられる。①中間バッファとして電子ディスクを採用すれば比較的低コストで大容量が実現できる。②プロセッサメモリと中間バッファを一体と見なしたとき、これに対するDBストアからの先回りステージングによるデータ供給力の向上が得られる。③いわゆるディスクキャッシュの効果も得られる。

3. 7 再接続ミスを削減するためのトータルスケジューリング

前々節の技術的課題の[2]で指摘したようにアクセスがビジー(busy)のときには再接続ミスが

頻発していた。このことによって生じるオーバーヘッドは再接続ミスした装置が回転待ちしている間にフリーになったチャネルがアイドルになることである。(Fig. 3-11)

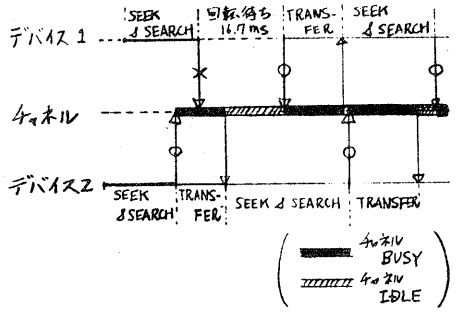


Fig. 3-11 回転待ちによるチャネルの IDLE TIMEの増加

そこで、チャネルの稼動率を高めることがアクセス状態がbusyのときのデータ供給力を向上させることになる。故にチャネルの稼動率が飽和しているならたとえ再接続ミスが多発していてもその系にとっては最高のデータ供給力が得られている状態である。もし、チャネルの稼動率が飽和していなくて、かつ再接続ミスが多発している場合、再接続ミスを少なくするようにディスクスケジューリングを行なえば、チャネルの稼動率を改善できる可能性が強い。これをステージング時間コスト(T(staging))から考察する。まず、チャネルの稼動率は前述のようにデータ供給力(C(supply))そのものである。次にT(staging)の式を示す。

$$T_{(staging)} = t_r + t_t + t_s + t_m \times P_m$$

t_r : 平均サーチ時間
t_t : 〃 転送時間
t_s : 〃 シーク時間
t_m : ディスク1回転所要時間
P_m : 再接続ミスを起こす確率

さて、C(supply)とT(staging)の間には

$$C_{(supply)} = K / T_{(staging)} \quad (K: 定数)$$

ここで、再接続ミスを考慮しないときの平均シーク時間をt_s, 考慮したときのそれをt_s', 考慮しないときの再接続ミス発生確率をP_m, 考慮したときのそれをP_m'とすれば、$t_s < t_s'$, $P_m > P_m'$となる。

そこで、$t_s + t_m \times P_m > t_s' + t_m \times P_m'$が成立するような状況があれば、再接続ミスの削減をディスクスケジューリングの考慮対象に入れることがデータ供給力の向上につながる。

これを実現するためには、ディスクスケジューリングを行なうプロセッサ(チャネルの上位に位置する。)が各ディスク装置のヘッドの位置・軌跡を時間的に把握しておくことが必要である。このことによっていつ、どのディスク装置がチャネルサービスの要求を出し、どのくらいの間専有するか予測できるので、再接続ミスを起こす可能性を小さくし、かつシーク時間の短いI/O要求に対してサービスできる。

また、再接続ミスによるオーバーヘッドを軽減するための他の手段としてはバッファの導入が考えられる。再接続ミスによるオーバーヘッドは前述のようにチャネルのアイドル状態を起こすことによって生じるから、再接続ミスした装置がもう1回転待たずにそのデータをRAMなどのバッファに入れるようにすればチャネルがフリーになった直後に転送が可能となり、チャネルのアイドル状態を削減できる。バッファの容量も現行のシリアル読み出しのディスクならば、1ディスク装置に1トラック分あればいいので容量の点でも現実的である。

ここまでは主として、データ駆動型先回り制御方式を基本としてアーキテクチャ的にデータ供給力の増加、プロセッサの稼動率向上を達成し全体のスループットを向上させる ということを述べてきた。

次節では、データ転送量、処理量の削減によるスループット向上を目指して、現在、我々が実験開発中のデータベースマシンSPIRITで導入したいくつかの技法について述べる。

3.8 データ転送量・処理量削減によるスループット向上を目指したいくつかの技法

3.8.1 各技法の概要

[1] データ圧縮技法 : コード化を基礎としたデータ構造を採用することによって①演算対象データの容量圧縮による実質的な転送時間の短縮 ②格納媒体上での局所性向上によるアクセス時間の短縮 及び③集合演算時の演算コ

ストの削減を達成している。

[2] データ格納単位としての属性の採用：データの入出力及び演算の単位として、リレーショナル演算の特徴を考察して、演算の最小論理単位である属性を採用した。これにより処理時に不要な属性が転送されず、リレーション単位の格納方式と比べて細かい入出力制御が可能である。データ転送時間は当然削減される。

[3] トラック並列データ配置と並列読み出し機構：現状の移動ヘッドディスクをDBストアと仮定して、トラック並列読み出し機構を付加する。これによりトラック数に比例したデータ転送量の向上が期待できる。各トラックには属性の部分集合が均等に格納されており、並列読み出しが可能となる。

3.8.2 各技法の効果予測

効果予測のための仮定

- [1]の予測式以外は格納単位は属性
- [2] 〃 トラック並列読み出し
- [3] 〃 コード化されていない。

パラメータ

C_t ： 1トラックの容量
C_c ： 1シリンダの〃
t_s ： 平均シーク時間
t_{tl} ： 〃サーチ〃
N_t ： 1シリンダ当たりのトラック数
m ： 再持続ミスの割合と相関のある係数
C_{Ri} ： コード化していないリレーションR_iの容量
γ ： 1つのリレーションで必要な属性のそのリレーションに占める容量の割合
n ： 1つのリレーションで必要となる属性の平均数

データのアクセス・転送コストをTとして、各技法を採用した場合のT_2と採用しなかった場合のT_1を比較する。

[1] データ格納単位として[属性 対 タップル]

タップルの場合

$$T_1 = t_s + t_{tl} + \left(\frac{2t_{tl}}{C_t} \times \frac{C_{Ri}}{N_t}\right) \times m + \left(\frac{2t_{tl}}{C_t} \times \frac{C_{Ri}}{N_t}\right)$$

属性の場合

$$T_2 = t_s + t_{tl} + \left(\frac{2t_{tl}}{C_t} \times \frac{C_{Ri}}{N_t} \times \gamma\right) \times m + \frac{2t_{tl}}{C_t} \times \frac{C_{Ri}}{N_t} \times \gamma$$

[2] 読み出し方式 [トラック並列 対 シリアル]

シリアル読み出しの場合

$$T_1 = t_s + n \times t_{tl} + \left(\frac{2t_{tl}}{C_t} \times C_{Ri} \times \gamma\right) \times m + \frac{2t_{tl}}{C_t} \times C_{Ri} \times \gamma$$

トラック並列読み出しの場合

$$T_2 = t_s + t_{tl} + \left(\frac{2t_{tl}}{C_t} \times \frac{C_{Ri}}{N_t} \times \gamma\right) \times m + \frac{2t_{tl}}{C_t} \times \frac{C_{Ri}}{N_t} \times \gamma$$

[3] コード化によるデータ圧縮 [する 対 しない]

N_{Rj} ： 属性jの種別数
α ： コード化効率

$$\alpha = \sum_{j=1}^{N_{Rj}} \lceil \log_2 N_{Rj} \rceil \Big/ C_{Ri}$$

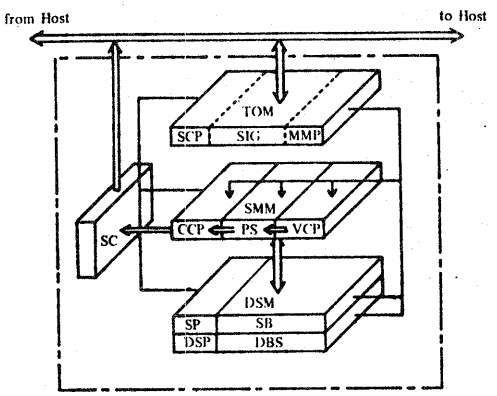
コード化しない場合

$$T_1 = t_s + t_{tl} + \left(\frac{2t_{tl}}{C_t} \times \frac{C_{Ri}}{N_t} \times \gamma\right) \times m + \frac{2t_{tl}}{C_t} \times \frac{C_{Ri}}{N_t} \times \gamma$$

コード化した場合

$$T_2 = t_s + t_{tl} + \left(\frac{2t_{tl}}{C_t} \times \frac{C_{Ri}}{N_t} \times \gamma \times \alpha\right) \times m + \frac{2t_{tl}}{C_t} \times \frac{C_{Ri}}{N_t} \times \gamma \times \alpha$$

以上、考察したことを統合してSPIRITでは Fig. 3-12 に示したような構成になっている。



TOM : Transaction Optimization Module
SCP : Security Check Processor
SIG : SPIRIT Instruction Generator
MMP : Model Mapping Processor
SC : SPIRIT Controller
SMM : Set Manipulation Module
VCP : Value Conversion Processor
PS : Pure SPIRIT
CCP : Code Conversion Processor
DSM : Data Staging Module
SP : Staging Processor
SB : Staging Buffer
DSP : Disk Scheduling Processor
DBS : Database Store

Fig. 3-12 SPIRITのシステムアーキテクチャ

4. おわりに

本稿では，リレーショナルデータベースマシンの記憶システムの制御方式としてデータ駆動型先回り制御方式を提唱し，その有効性を確かめた。現在，この制御方式を実現するための記憶階層方式，ディスクスケジューリングの方法論などを含めた種々の技法を検討・考察中である。

〔謝辞〕 本研究を始めるにあたり貴重な御教示をいただいた南部明氏（現 日本電々公社横須賀通研）に深謝いたします。また，日頃から御討議いただいている清本康，小沢裕之，田中浩一の皆様に感謝いたします。

参考文献

[1] T. Lang, et al, "An Architectural Extension for a Large Database System Incorporating a Processor for Disk Search" 3rd Int. Conp. on VLDB, pp. 204-210, 1977.

[2] K. Hakozaki, et al, "A Conceptual Design of a Generalized Database Subsystem" 3rd Int. Conp. on VLDB, pp. 246-253, 1977.

[3] E.A. Ozkarahan, et al, "RAP - An Associative Processor for Database Management" AFIPS Conf Proc. , vol.44, pp. 379-388, 1975.

[4] S.A. Schuster, et al, "RAP.2 - An Associative Processor for Database and Its Applications", IEEE Trans. Comput., vol.c-28, No.6, pp. 446-458, 1979.

[5] D.J. Dewitt, "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems", IEEE Trans. Comput. , vol.c-28, No.6, pp. 395-406, 1979.

[6] E.J. Mcbride, et al, "System Considerations for Predicting Mass Storage Subsystem Behavior", NCC Proc., pp.749-759, 1975.

[7] N. Kamibayashi, et al, "SPIRIT - A New Relational Database Computer Employing Functional-Distributed Multi-microprocessor Configuration", Proc. 1st DCS, pp. 757-771, 1979.

〔8〕 宇津宮，“循環待ち行列モデルによるディスクスケジューリング方式の評価”，信学論 (D) 59-D, 9, P636 (昭51-09)

〔9〕 上林他，“リレーショナルデータベース演算のハードウェア化アルゴリズムとそれに基づくデータベースマシンアーキテクチャ”，情報処理学会計算機アーキテクチャ研究会，(昭54-09)

〔10〕 南部，“データベース処理向き計算機の基本方式”，昭和53年度，慶應義塾大学工学研究科，修士論文

〔11〕 ハーバーマン，“オペレーティングシステムの基礎”，培風館