データベース・システム 84−15
（１９９１．７．１８）

オンラインログの適用手法

横田治夫、野口泰生、赤星直輝
（株）富士通研究所

戦略情報システム（SIS）の要求などから、データベースにおける複雑な問い合わせ処理や統計処理に代表される情報系処理と呼ばれる処理を、日常業務の OLTP 等と一緒に行う必要性が生じてきている。本報告では、OLTPに代表される基幹系処理と情報系処理を同時に行う方法について考察を行う。基幹系処理と情報系処理でそれぞれ別のデータ領域を持ち、基幹系処理での更新内容をログとして、あるインターバルで情報系処理用のデータ領域にその更新内容を反映することにより、それぞれの領域間でのデータの等価性をできるだけ保つことを考える。このとき、基幹系処理のログを用いた遅延更新にかかる時間をできるだけ短くすることが重要となる。このため、ディスクアクセスの最適化のシミュレーションを行い、遅延更新に要する時間を見積もる。さらに、情報系処理に割り当てる時間を長くし、情報系処理の応答時間を改善するために、情報系処理用の領域を二重化し、ログをオーバラップして適用する方法を提案する。

Applying On-Line-Transaction Logs to Decision Support Databases

Haruo Yokota, Yasuo Noguchi, and Naoki Akaboshi
FUJITSU LABORATORIES LTD.
1015 Kamikodanaka, Nakahara-ku, Kawasaki 211, Japan

Online transaction processing (OLTP) to update databases and complex query processing on the databases must be executed simultaneously in order to implement strategic information systems (SIS). An effective model supports separate storage spaces for OLTP and query processing. We examine a deferred update strategy using OLTP update logs to avoid discrepancies between the two storage spaces. Since disk access optimization is important to make deferred update practical, we investigated several disk access patterns. Our proposed method duplicates the storage space for complex query processing. Complex queries can always be executed because the update log applications are overlapped across the duplicate storages.

1．はじめに

データベースの利用分野では、オンライントランザクション処理（OLTP）に代表されるような即時性を重視しながら1回当り比較的軽い処理を同時に多数行う処理（以下基幹系処理と呼ぶ）と、基幹系処理で集められたデータ等に対して複雑な問い合わせ処理や統計処理を行って意志決定支援を行うような非常に重い処理（以下情報系処理と呼ぶ）の2つに、大きく処理を分類することができる。銀行業務を例にとると、顧客毎の現金払い出し等によるデータベースアクセス処理が基幹系処理にあたり、金融商品当たりの資金の流れの解析といったような処理が情報系処理にあたる。

従来、日常業務中は基幹系処理を行い、情報系処理は日常業務の終了した夜間などに行われるのが普通であった。これは、排他制御等の関係から、1つの情報系処理を行うことにより、多くの基幹系処理の即時性が損なわれるため日常業務中に情報系処理を行えなかったことによる[1)、2)]。このため、情報系処理によって得られるデータについては、多少の時間的遅れは仕方がないと諦められていた。しかし近年、戦略的情報システム（SIS）等の必要性から、日常業務を行いながら、即時性の高いデータを抽出する情報系処理に対する要求が高まっている。

基幹系処理と情報系処理を混在して実行させるために、これまでは複数の版のデータベースを管理する多版並行処理制御の研究[2)、3)、4)]が行なわれてきた。多版並行処理制御では、基本的には更新の度に新たな版を生成することにより、排他制御による待ちを減少させる手段が取られる。しかし、多数の版が同時に存在するため版の管理が複雑になる上、大規模なデータベースにおいては新たな版を頻繁に作ることにも問題がある。

一方、このような研究とは独立に、基幹系処理を同時に多数行なうための専用OLTPマシン[5)]や、情報系処理を並列に高速に行なうための専用データベースマシン[6)]が数多く実用化されている。専用マシンを有効に利用する上でも、基幹系処理と情報系処理を同時に実行可能とすることが重要である。このような場合には、データベースを多版にするのではなく、基幹系処理用と情報系処理用で別のデータ領域を持つことを前提とする必要が出てくる。

本報告では、専用マシンを利用することも考慮して、基幹系処理と情報系処理でそれぞれ別のデータ領域を持ち、基幹系処理での更新内容を情報系処理の対象データに反映させるための方法について考察する。ただし、ここで検討するモデルは必ずしも別々の専用マシンを前提とするものではない。

まず、基幹系処理の更新内容を情報系処理用データに反映させるために基幹系処理の更新ログを用いる遅延更新のモデルを示す。次に、遅延更新に要する時間を見積もるために、ディスクアクセス方法のシミュレーションを行ない、遅延更新を行なうための条件について考察する。単純な遅延更新では、情報系処理のために十分な時間を割り当てることが困難であり、処理を開始するまでに待ち時間も必要である。そこで、情報系処理の待ち時間をなくし、実行可能な時間を増やすため、情報系処理用データ領域を二重化し、ログをオーバーラップして適用する方法を提案する。

2．ログを用いた遅延更新による等価性維持

ここではまず、基幹系処理と情報系処理を同時に行うため、基幹系処理と情報系処理でそれぞれ別のデータ領域を持ち、それぞれのデータ領域の内容をできるだけ等価に保つことを考える。以下では、基幹系処理用のデータ領域をA領域、情報系処理用のデータ領域をB領域と呼ぶことにする。

A領域とB領域の内容の等価性を維持するため、A領域への更新内容をログに保持し、後でそのログをまとめてB領域に適用する方法が考えられる。つまり、基幹系処理ではA領域の内容を常に更新し続けると共に、その更新した内容をログに記録しておく。あるインターバルでそのログの内容をB領域に適用し、適用後のB領域内のデータを使って情報系処理（検索処理）を行うことにする。ここでは、このようなログによるB領域の更新を遅延更新と呼ぶことにする。その場合のシステムのモデルを図1に示す。

このようにすることによって、情報系処理が基幹

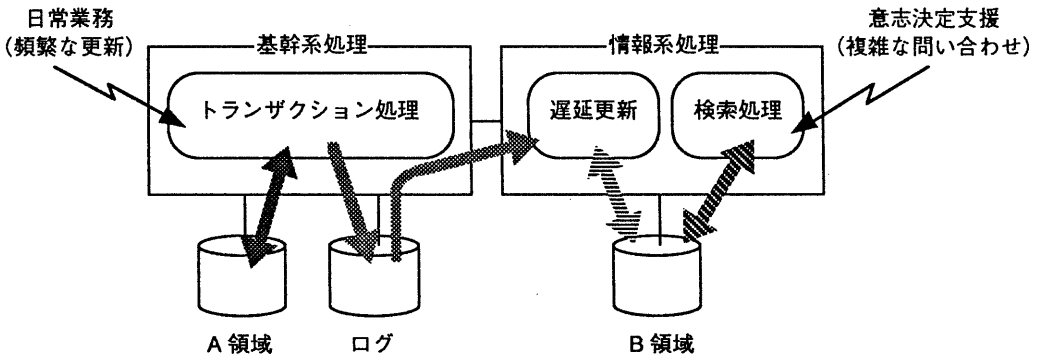

図1　遅延更新のモデル

系処理に影響を与えることが無くなるため、基幹系処理のレスポンス時間の制限を守ることができる。また、多版並行処理制御のような、複雑な版管理や、更新の度のデータベースのコピーを必要としない。また、これは専用 OLTP マシンや情報系処理専用データベースマシンの利用にも適する。実際、これらの全ての処理を1つの汎用マシン上で実現することも、別々の専用マシンで実行することも可能であるが、以下ではマシンの構成とは独立に考察を進める。

図1のモデルの処理の進み方を図2に示す。T_{i-1} から T_i までのインターバルで A 領域に対して行なわれた更新の内容を、T_i の時点で B 領域に対して反映させる。その遅延更新が終了した後で、B領域に対して検索処理を行なう。

図2から分かるように、基幹系処理にかかる時間と遅延更新にかかる時間の差が情報系の検索処理に割り当てられることになる。ここで、遅延更新の時間を X（秒）、検索の時間を Y（秒）、T_{i-1} から T_i までのインターバルを I（秒）とすると、

$$I = X + Y \tag{1}$$

となる。ここで、X をインターバル I 当たりの更新数 U に対する一次の関数で近似できるとして、

$$X = kU \tag{2}$$

さらに U は、基幹系処理における1秒当たりのトランザクションの平均到着件数を a TPS (Transaction Per Second)、1トランザクション中の平均更新数を u とすると、

$$U = auI \tag{3}$$

となる。以上から、情報系処理の検索に割り当てられることのできる時間 Y は、

$$Y = (1 - kau)I \tag{4}$$

となる。

この検索に割り当てる時間を長くするためには、k の値を小さくする、つまりB 領域に対する遅延更新を効率よく行うことが重要となる。そのため、ロ

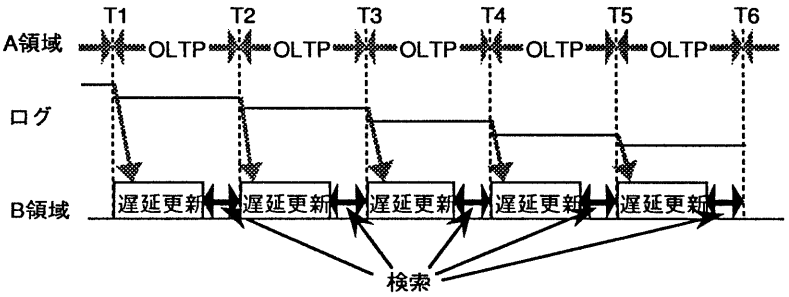

図2　遅延更新の処理の進み方

グの内容をまとめてB領域にできるだけ速く適用するためのディスクアクセスの方法を考える。そこで次に、いくつかのディスクアクセスパターンを用意しシミュレーションを行い、kの値を見積もることにする。

3. ディスクアクセスのシミュレーション

更新対象のセクタがディスク上にランダムに分布しているとした場合の、ログに基づいてその更新内容を反映するためのアクセスパターンを考える。既に一部のOS で行なわれている[7]ように、セクタのアクセス順序を制御する。

(a) ランダムアクセス

シリンダ、トラックの切り替えを考慮せず、セクタ単位でランダムアクセスする。

(b) シリンダ単位ソート

シリンダの切り替え（シーク）を極力抑えるように、アクセスパターンをシンダ単位でソートする。

(c) ヘッド単位ソート

(b) の中で、さらにヘッドの切り替えを抑えるために、(b) のパターンをヘッド単位でソートする（このとき、同一セクタに対する read と write は別コマンドとみなすため、あるセクタをアクセスするには余分に1回転することになる）。

(d) ヘッド単位スキャッタリング

(c) の同一ヘッド内を、単なるソートでなく、コマンド切り替えに要する適当な間隔が開くように整列する。

(e) シリンダ単位スキャッタリング

(d) において、ヘッドの切り替えを使って、シリンダ内でセクタの切り替えを最適化する。この時、同一ヘッドの時の間隔と異なるヘッドの時の間隔を変える。

以上のアクセスパターンの例を図3に示す。斜線の1つの箱がアクセス対象のセクタを表し、横一列がトラックを表現している。更に、トラックのいくつかの集まり（図では3トラッ

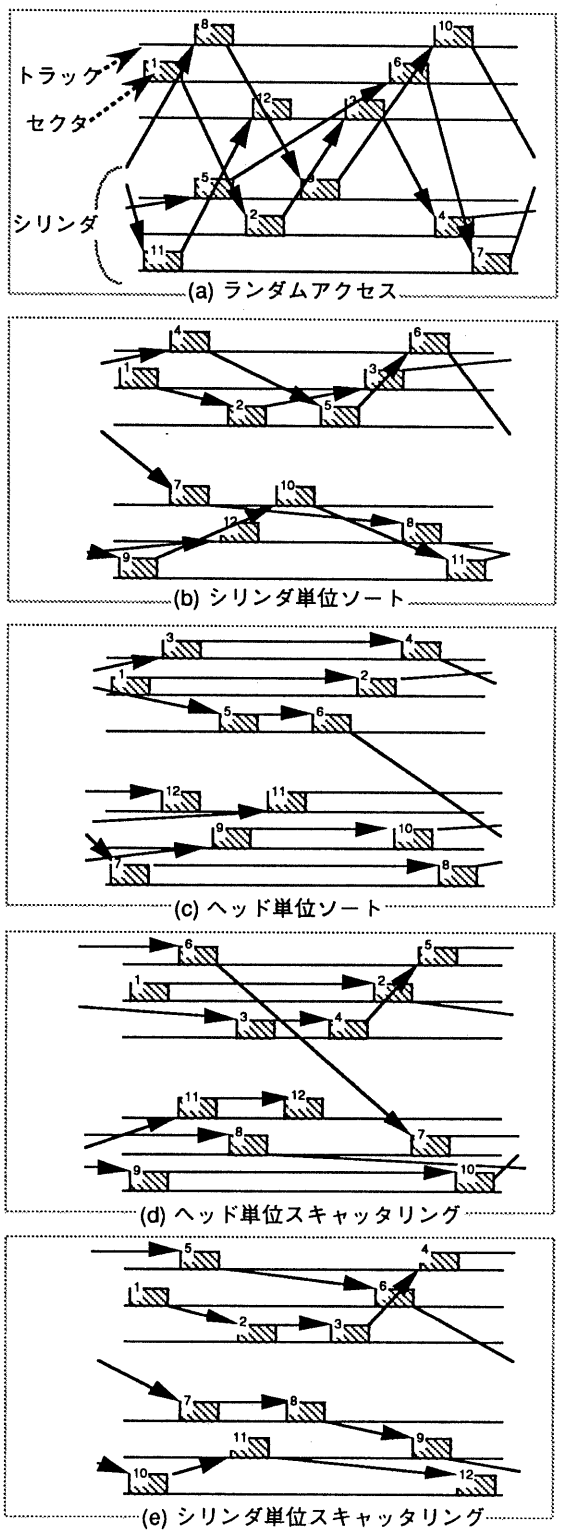

図3　アクセスパターン

ク）を１シリンダとして表している。矢印がアクセスの系列を示し、セクタの右肩の数字がその順番を示している。アクセス中でトラックの切り替えがある場合にはヘッドの切り替え、シリンダの切り替えがある場合にはシークが必要となる。

(a) で多発するシークを、(b)、(c)、(d)、(e) ではシリンダ数－１に抑えることができる（図の例では、１０回のシークが１回に減っている）。また、(c)、(d)では、ヘッドの切り替回数が全トラック数－１に減ることになる（図の例では、１０回のヘッド切り替が５回に減っている）。また、ヘッドの切り替時間よりセクタの同期待ちが時間がかかる場合には、(e) のシリンダ内での最適化の効果が期待できる。

シミュレーションでは、まず、疑似乱数を用いて、更新対象セクタを指定個数だけ分布させる。次に、その分布セクタを、前述のアクセスパターンになるように並び変える（最適化を行う）。そして、そのアクセスパターンに従ってディスクが動作するとして、表１に示したディスク諸元を基に動作をシミュレートし、更新時間を見積もる。

表１に示したディスク諸元は、最新の５インチハードディスクの一例である。なお、ヘッド切り替え時間については、コマンド転送時間、設定時間を含めて、見積もってある。また、ポジショニング時間はシリンダ間の距離によって変化するが、今回はシミュレーション簡略化するため、平均ポジショニング時間を用いることにした。さらに、ログはセクタ内の論理位置の更新をするものとし、１セクタの更新には、readとwriteを要するものとした。

表１　ディスク諸元例

容量		2 GB	
シリンダ数		1893	
ヘッド数		21	
セクタ数		10	(4KB/sector)
１回転時間		11.1 ms	(5,400 rpm)
ポジショニング時間	(min)	2 ms	
	(avg)	11 ms	
	(max)	22 ms	
ヘッド切り替え時間		3 ms	
データ転送速度		4.758 MB/s	

シミュレーション結果を図４に示す。結果を見てみると、更新数が少ない場合には、

(a) > (b) > (c) > (d) > (e)

更新数が多くなると、

(a) > (b) > (c) > (e) > (d)

と(e) と (d) の順番が逆になっている。これは、更新数が少ない間はヘッドを切り換える時間がセクタの同期を取るための時間より少ないため (e) のシリン

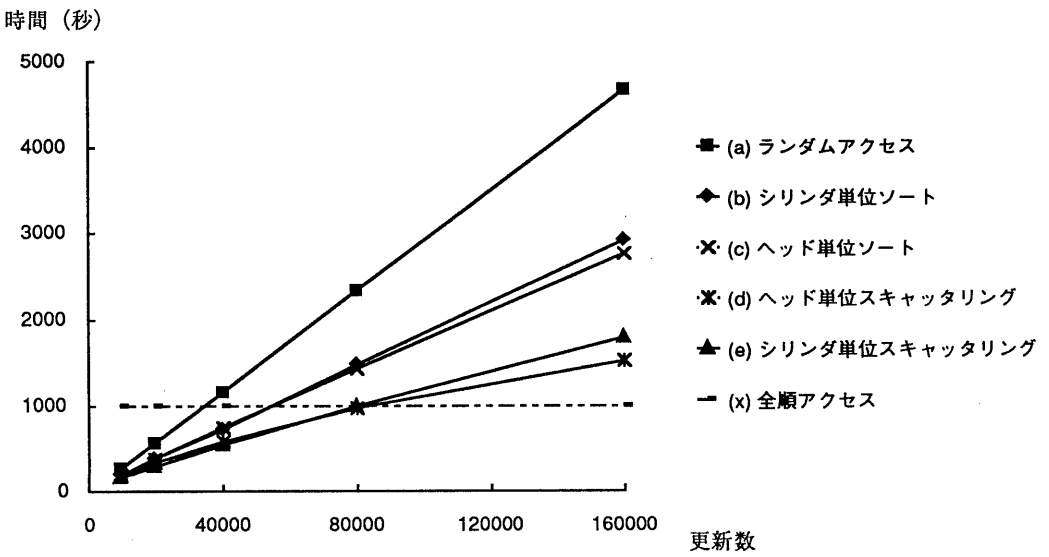

図４　シミュレーション結果

ダ単位スキャッタリングが効果があるが、更新数が増えるとヘッド切り替の時間がセクタ同期の時間より大きくなってしまうためと思われる。

さらに、全セクタをトラック単位で順アクセスするのにかかる時間 S （秒）は、表1から、

$$S = 1893\,[\text{シリンダ数}] \times ((0.0111 \times 2\,[\text{2周}] + 0.003[\text{ヘッド切り替え}]) \times 21[\text{ヘッド数}]) + 0.002[\text{隣へシーク}]) = 1{,}005.562 \quad (5)$$

この値は、更新数とは独立である。この値も(x) 線として図4に入れた。全順アクセスの(x) 線より上になる場合には、トラック単位で1つのコマンドとして全セクタを順番にアクセスしたほうが速いことになる。

以上から、更新数が少ない場合には、(e) のシリンダ単位のスキャッタリングが最も高速であり、更新数が増えるに連れて(d) のヘッド単位のスキャッタリングが有効になる。しかし、ある一定の更新数を超えた場合には全順アクセスが最も有効になると言える。

以上のシミュレーションの結果を、式 (2) の一次近似にそのまま当てはめることはできない。まず、更新数がある程度以上は延びないものとした場合を考える。この時には、(e) のシリンダ単位スキャッタリングを用いるとして、 k の値を求めると、約0.012 となる。インターバルの50% を検索に割り当てるとすれば、式 (4) から

$$au = 42 \quad (6)$$

1トランザクション中の平均更新数を1としても、平均トランザクション到着件数は高々 42 TPS が精いっぱいということになる。しかも、84 TPS の場合にはまったく検索時間に割り当てることができ無くなる。

以上の考察は1台のディスクを対象に行なったが、一般にはディスクの台数を複数にすれば、k は台数に反比例して小さくなることが期待できる。これは、遅延更新がログをまとめることにより、複数のディスクに同時にアクセスするように制御できるためである。ただし、ディスクアクセス要求が複数のディスクに平均する必要がある。

つまり、ディスクアクセス要求が平均するとした場合には、台数を N とすれば、X は、

$$X = kauI / N \quad (7)$$

これより、式 (4) は、

$$Y = (1 - kau/N)I \quad (8)$$

となる。例えば、ディスク台数を 12 にして (e) のシリンダ単位スキャッタリングを用いると、1トランザクション中の平均更新数 5、平均トランザクション到着件数 100 TPSであっても、インターバルの 50% を情報系処理に割り当てることができるようになる。

一方、インターバルをある程度以上長く取って更新数が多くなった場合には、全順アクセス(x) が有効となる。ただし、全順アクセスの場合には、ディスクの台数を増やしても、遅延更新時間に対する影響はない。このため、この切り替の条件は、

$$kauI / N > S \quad (9)$$

となる。

シミュレーションの対象となったディスク諸元では (e) のシリンダ単位スキャッタリングと全順アクセスを切り換える条件は、S を約 1000 秒、k を 0.012 とすると、

$$I > 83333N / (au) \quad (10)$$

となる。例えば、1トランザクション中の平均更新数 1、平均トランザクション到着件数 100 TPS でディスクが 1 台の場合、インターバルが 833 秒を越えると、その越えた分は全て情報系処理に割り当てられることになる。この値は、ディスク台数が増えれば大きく、トランザクション中の更新数やトランザクションの到着件数が増えれば小さくなる。

4.B 領域の二重化

以上から、遅延更新を効率化することによりある程度は情報系の検索に時間を割り当てることは可能となることが分かった。しかし、十分な時間を情報系処理に割り当てるためには、長いインターバルを取る必要があり、その場合に遅延更新に必要な時間も長くなるため、情報系処理を受け付けられない時間帯も長くなる。つまり、各情報系処理の応答に遅延更新の時間まで含めると、応答時間が悪くなるこ

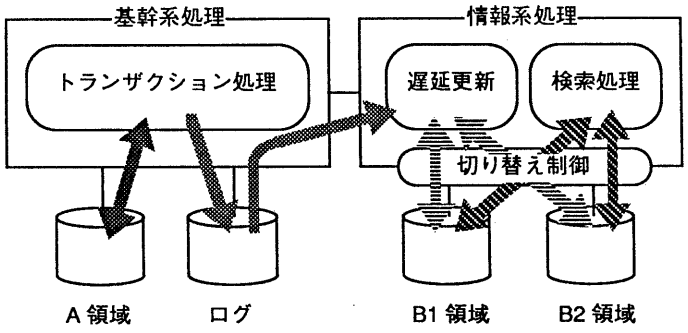

図5　B 領域を二重化したモデル

とになる。

そこで次に、B 領域を二重化し、ログの適用をオーバラップさせることにより、情報系処理を常に待ち時間なしで受け付けられるようになることを示す。B 領域を二重化した場合のシステムの構成を図5（cf 図1）に示す。ある時点で遅延更新処理の対象領域と、検索処理の対象領域とを分離し、ダブルバッファリングのようにあるインターバルでそれぞれの対象領域を切り替える。このとき、遅延更新を行うためのログの対象期間をオーバラップさせることにより、B1/B2 領域はぞれぞれ1つ前のインターバルのA 領域の内容を反映することなる（図6）。つまり、T_3 の時点で B2 領域に遅延更新されるログは、T_1 から T_3 までの内容であり、T_4 の時点で B1 領域に適用されるのは T_2 から T_4 までの内容である。そして、情報系処理の検索は、1つ前のインターバルまでの更新内容を含むデータベースに対して行なわれることになる（T_3 から開始される検索は、T_2 までの更新の内容を反映しているデータベースに対して行なわれる）。

このように、B 領域を二重化することにより、検索処理が遅延更新の処理の終ることを待つことなく、十分な時間を検索処理に割り当てることができる。さらに、図6から分かるように、検索処理は常に可能となり、応答時間は遅延更新の時間を含まれることはなくなり、本来情報系の処理にかかった時間だけとなる。また、検索処理に長時間を要するような場合も、インターバルを調節することにより対応できることになる。

このとき、条件となるのは、インターバルを基幹系処理より遅延更新が短時間で終了するように取るだけで十分であり、B領域が1系統の時のようにそれらの差の時間を気にする必要はない。つまり、基幹系の処理でA 領域を更新した時間以内で B 領域の

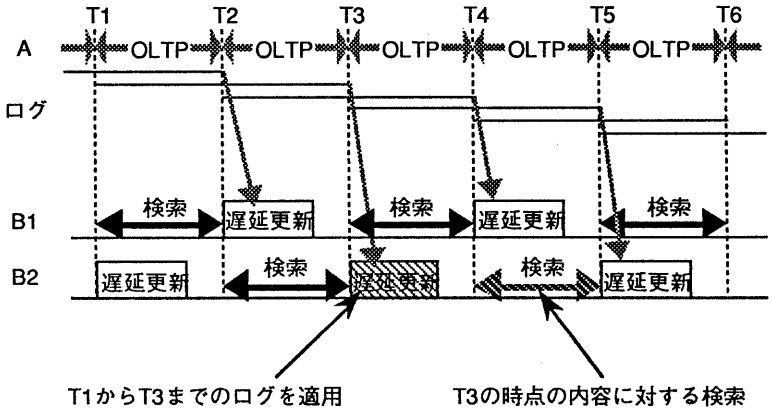

図6　B 領域を二重化した場合の処理の進み方

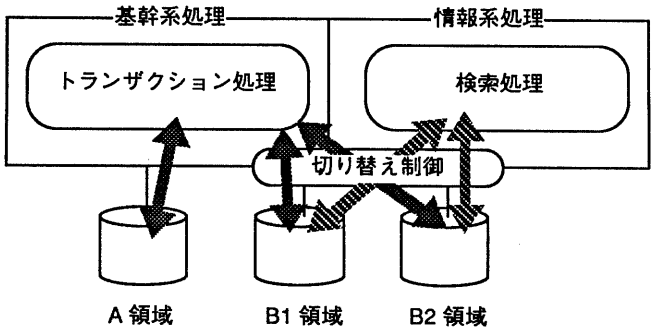

図7　ログを用いないモデル

遅延更新を行えばよいことになる。これを式で表すと、全順アクセスでない場合には、式 (8) から、

$$N > kau \qquad (11)$$

全順アクセスの場合には、

$$I > S \qquad (12)$$

となる。これから、(11) 式を満たすようなディスク台数を用意するか、(12) 式を満たすようにインターバルを取って、全順アクセスをすればよい。ただし、あまりインターバルを長く取るとセクタの順番を並べ替える処理に時間を要するようになるため、必ずしも得策でないことになる。

この二重化で重要なことは、ログをオーバラップして適用する点であることに注意する必要がある。例えば、図7のようにログを用いないモデルを考えると、あるインターバルに一方のB領域を更新した内容をもう一方の領域に反映することが困難となることが分かる。

5. おわりに

OLTP に代表される基幹系処理と、意志決定支援を行うような情報系処理を同時に行う方法について検討した。まず、基幹系処理の更新内容をログによって情報系処理用のデータ領域に反映させる遅延更新のモデルを前提に、ディスクアクセスの最適化に対するシミュレーションを行ない、遅延更新にかかる時間を見積もった。次ぎに、その結果を基に遅延更新の条件について考察を行なった。さらに、常に情報系処理を実行可能とするために、情報系処理用のデータ領域を二重化してログをオーバラップさせて適用する方法を提案し、その場合の条件について考察した。なお、ディスクアクセス最適化でセクタのアクセスの順番を並べ替えるのに要する時間については、シミュレーションを行なった更新数の範囲では十分短いことから、今回は考慮しなかった。しかし、今後より長いインターバルも対象とするためには、その時間も含めて検討する必要がある。

参考文献

1) Ohmori, T., Kitsuregawa, M., and Tanaka, H.: Concurrency Control of Bulk Access Transactions on Shared Nothing Parallel Database Machines, Proc. of 6th Int'l Conf. on Data Engineering, pp. 476-485, (1990).
2) 部分更新と全数検索の混在処理に適した多版並行処理制御方式, 信学論(D-I), Vol. J74-D-I, No. 3, pp. 224-231, (1991).
3) Papadimitriou, C. H. and Kanellakis, P. C. : On Concurrency Control by Multiple Versions, ACM Tras. on Database Systems, Vol. 9, No. 1, pp. 89-99 (1984).
4) Hadzilacos, T. and Papadimitriou, C. H.: Algorithmic Aspects of Multiversion Concurrency Control, Proc. of 4th SIGACT-SIGMOD, pp96-104, (1985).
5) 分散オンライン時代を切り開くOLTPマシン、日経コンピュータ1990.4.23, pp72-96.
6) 実用化相次ぐデータベースマシン、日経エレクトロニクス 1989.6.26, pp.177-184.
7) Mckusick, M. K., Joy, W. N., Leffler, S. J. and Fabry, R. S.: A Fast File System for UNIX, ACM Trans. on Computer Systems, Vol. 2, No. 3, pp. 181-197, (1984).