データベースシステム 96－9
（1993. 10. 29）

履歴管理機構をもったファイルマネジャの設計

鈴木 孝幸† 北川 博之†† 大保 信夫††
† 筑波大学理工学研究科, †† 筑波大学 電子・情報工学系

工学分野などデータベースの応用分野の拡大に伴い、データの版管理や履歴管理を行ないたいという要求が高まっている。本稿では、データベース管理システムの基本モジュールとしてページ単位の履歴管理機能を組み込んだファイルマネジャの設計について述べる。ページ単位の履歴管理には、レコード構造やレコードフォーマットに依存しないより汎用性の高い履歴管理機能を提供できるといった特徴がある。しかし一方、レコード単位で履歴を取る方式に比べて、データの格納効率やI/O量が悪化する恐れがある。本稿では、レコード単位による履歴管理とページ単位の履歴管理をシミュレーションにより比較し、これらの問題点の検討も併せて行なう。

Design of a File Manager with History Management Function

Takayuki Suzuki†, Hiroyuki Kitagawa†† and Nobuo Ohbo††
†Master's Degree Program in Science and Engineering, Univ. of Tsukuba,
††Institute of Information Sciences and Electronics, Univ. of Tsukuba

Temporal and historical information mangement is required in many advanced datebase applications. We discuss design of a file manger which features management of the datebase history at the physical page level. The file manger has advantages such as providing uniform history mangement function independent of page formats and page layouts. However, page level versioning has a few drawbacks in space efficiency of the secondary storage and in I/O performance. By simulation, we compare the page level versioning and the record level versioning and we evaluate the performance of the file manager.

1 はじめに

工学などの応用分野におけるデータベース利用の高度化に伴い[1]、DBMS (DetaBase Management System) による版管理やデータベースの更新履歴等の時間情報管理の要求[2)3)4)5)6)7)8)9)]が高まっている。特に、更新履歴管理については、DBMSのトランザクション管理機構とも深く関連するので、DBMS自身が本来的にそのための機能を備えることが必要である。これまでにも、DBMSの履歴管理についてはいくつかの研究が行なわれてきた[2)5)11)12)]。多くのアプローチでは、レコード毎に履歴を管理する方法をとっている。しかし、この方法では、複合オブジェクトの履歴管理を行ないたい場合や、長大レコードを含む場合など多様な形態のレコードに対応するには、各々のレコード構造に対応した履歴管理機構を設計してDBMSに組み込まなければならない。すなわち、今日の多様な応用要求に対応するためには、レコードフォーマットやレコードレイアウトに依存しない統一的な履歴管理の機構の開発が望まれる。また、リカバリや同時性制御などのトランザクション管理機能は、履歴管理機能と密接な関係にあるが、そこではファイルのI/Oの単位であるページが基本的な管理単位とする場合が多い。そこで本研究では、ページを単位にとして履歴管理を行なうファイルマネジャの設計と実現について検討する。

ページ単位で履歴管理を行なう方式には、以下のような利点がある。

- レコードフォーマットやレコードレイアウトに依存しない履歴管理機能を提供できる。
- 同じページ単位で管理するので、トランザクション管理と履歴管理の融合が図りやすい。
- ページサーバ[10)]として実装した場合、サーバ側に履歴管理機能を集中できる。また、これによりサーバとクライアント間での通信量を減少させることができる。
- ある時刻でのページイメージそのものを復元できるので、例えば、永続プログラミング言語におけるデータ構造のように従来のレコード構造のみでは捉えにくいオブジェクトの履歴管理にも対応しやすい。

一方、ページ単位で履歴管理を行なう方式には、レコード単位で履歴管理する方式に比べて、以下の問題点がある。

- レコード単位で更新が行なわれた場合でもその単位を認識できないため、履歴情報が膨大になる可能性がある。差分管理を行なった場合でも、レコードレベルの差分管理に比べて格納効率が悪くなる恐れがある。
- 履歴情報の格納に多くの領域が必要なため、必要なデータを取り出すための2次記憶へのI/Oが増加する恐れがある。

本稿では、DBMSの基本モジュールとして、ページ単位での履歴管理機構をもつファイルマネジャの設計について述べる。また、上記のような問題点について検討するため、ページ単位の履歴管理方式とレコード単位の履歴管理方式の格納効率とI/O回数を比較したシミュレーションを行ない、その結果について報告する。

本稿の第2章では、本ファイルマネジャの概要とその構成について述べる。第3章では、ページ単位の履歴管理方式とレコード単位の履歴管理方式を比較するためのシミュレーションモデルについて述べる。第4章では、シミュレーション結果について報告する。第5章は、まとめである。

2 ファイルマネジャの設計

本節では、ページ単位の履歴管理機構をもったファイルマネジャの概要、設計の方針、構成要素、ならびにその動作について述べる。

2.1 ファイルマネジャの概要

ファイルマネジャはDBMSの構成において、2次記憶を管理し、2次記憶上のデータにアクセスする機能を提供する。2次記憶上のファイルを物理ファイル、物理ファイル上のページを物理ページと呼ぶ。物理ファイルには物理ファイルID(pfid)、物理ページには物理ページID(ppid)が付けられる。これに対して、本ファイルマネジャがアプリケーションに提供するファイルを論理ファイル、論理ファイル上のページを論理ページと呼ぶ。各論理ファイルには論理ファイルID(lfid)が、論理ページには論理ページID(lpid)が付けられる。2次記憶上のファイル構成や実際のページイメージ(ページの内容)の管理、データをアクセスする際のバッファリング、トランザクション管理などはファイルマネージャ内で行なう。したがって、ファイルマネジャを利用するアプリケーションなどでは、これらの機能について考慮する必要がない。

履歴管理を行なうためには、論理ページがトランザクションにより更新された時、更新前のイメージとその更新時刻をファイルマネジャ中に保持する必要がある。また、ファイルマネジャに論理ファイル、論理ページ、時刻が与えられた時に、その与えられた時刻でのページイメージを取り出せるようにファイルマネージャを設計する。

各論理ページの最新のページイメージをカレントページイメージと呼ぶ。カレントページイメージは更新操作に伴い時間とともに、書き換わって行く。このとき、書き換わる前のカレントページイメージは履歴ページイメージとなり、書き換わった後のイメージが新たにカレントイメージとなる(図1)。履歴ページイメージは、過去の経歴としてのページイメージなので更新対象にされることはなく、カレントページのみが更新対象となる。一般に、データベースの最新の状態であるカレントページへのアクセスの要求が多いのと比較して、履歴ページイメージへの高速での読み出しの要求は少ない。そこで、ページイメージの性質の違いに従い、履歴ページイメージは、履歴ファイルという別のファイルに保存する。また、カレントページイメージが格納されるファイルをカレントファイルと呼ぶ。こうして、カレントページイメージと履歴ページイメージを格納するファイルを分けることによって、カレントページイメージへのアクセスの機構は履歴ページのアクセスの機構の負荷をうけず、ページイメージの性質に合わせた2次記憶装置の利用[11)]が可能になる。

履歴ページイメージの量は膨大なものになるので、そのまま2次記憶上に格納するのは困難である。本ファイルマネジャでは、その解決策として履歴ファイルでは、各履歴ページイメージは直前の履歴ページイメージとの差分が取られ、圧縮して格納される。履歴ファイルの各ページには同じ論理ページに関する差分情報が格納される。また、カレントページには、対応する履歴ファイルページのppidが格納され、履歴ページイメージを取り出す場合に利用される。

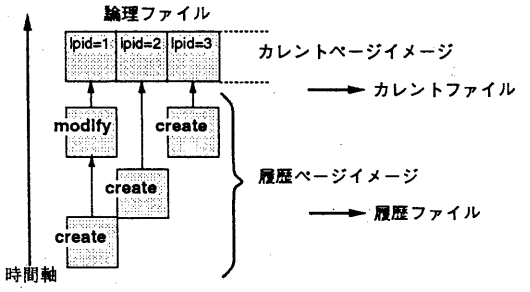

図 1. 論理ファイルのページイメージ

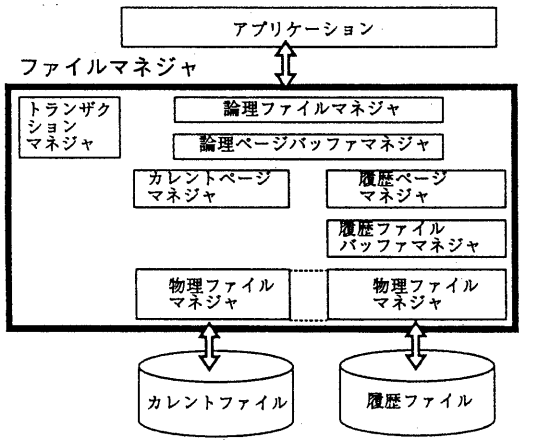

図 2. ファイルマネジャ

2.2 ファイルマネジャの構成

ファイルマネージャは図 2のような構成からなる。カレントページイメージへのアクセスは、論理ページマネジャ ⟶ 論理ページバッファマネジャ ⟶ カレントページマネジャ ⟶ 物理ファイルマネジャ ⟶ 物理ページ (カレントページイメージ) となる。それに対して、履歴ページイメージへのアクセスは、論理ページマネジャ ⟶ 論理ページバッファマネジャ ⟶ 履歴ページマネジャ ⟶ 履歴ファイルバッファマネジャ ⟶ 物理ファイルマネジャ ⟶ 物理ファイル (履歴ページイメージ) となる。以下、各マネジャについて説明する。

2.2.1 論理ファイルマネジャ

論理ファイルマネジャは論理ファイルと論理ページの管理を行なう。ページイメージへのアクセスは論理ページバッファマネジャを介して行なわれる。

トランザクションが書き換え操作を行なう場合には、カレントページイメージを保存するため、新たな物理ページであるワーキングページと論理ページバッファを確保して、そのワーキングページに書き込むことにする。トランザクションがアボートする場合には、ワーキングページイメージは捨てられることになる。また、同時性制御により各論理ページの書き換え操作を行なえるトランザクションは同時には1つのみに制約する。また、論理ページの書き換え時に、ワーキングページが既に作成されているかどうかのチェックや、作成されていない場合の作成の許可は論理ページマネジャがトランザクションマネジャに問い合わせることによって行なう。ファイルマネジャに論理ページへの読み出しの要求が来た時には、与えられた時刻と論理ページ ID とそのトランザクションにおける要求に従って、カレントページイメージ、ワーキングページイメージまたは履歴ページイメージにアクセスすることになる。ここで、論理ページイメージの更新された場合の時刻としてどの時刻を用いるかが重要になる。ページイメージに付される更新が行なわれた時刻のタイムスタンプには、トランザクションマネジャで決定するトランザクションのコミット時刻を用いることにする。各ページイメージには、そのページイメージが作成された時刻と更新操作によってカレントページイメージでなくなった時刻の2つのタイムスタンプを付す。この2つの時刻の間をそのページイメージの生存期間と呼ぶ。ただし、カレントページイメージについては後者のタイムスタンプとしては、未定を表す特殊な値を付す。ページ ID の他に、ページにアクセスする時、ファイルマネジャには時刻が与えられた場合は、その時刻が生存期間に入っている履歴ページイメージへのアクセスを行なう。また、“現在”を示す時刻には、“now” という特殊な値を取ることにする。“now” が与えられた場合は、そのトランザクションがその論理ページに対して更新を行なっていたらワーキングページイメージへ、更新を行なっていなければ、カレントページイメージへアクセスすることになる。

2.2.2 論理ページバッファマネジャ

カレントページイメージ、ワーキングページイメージおよび履歴ページイメージをバッファに読み込み、管理するマネジャである。各バッファページは LRU で管理される。また、更新トランザクションがコミットしてカレントページイメージが新たに履歴ページイメージになった場合は、その直前のカレントページイメージとワーキングページイメージを履歴ページマネジャに渡し、管理させる。また、履歴ページイメージを読み出したい場合は、カレントページイメージをバッファに読み出し、それを履歴ページマネジャに渡して差分情報を戻させる。

各バッファには LRU リストとは別に、論理ページバッファに当該ページが読み込まれているかどうかを判定するために、論理ページ ID でインデックスが張られ、同じ論理ページ ID のページイメージは生成時刻順にチェーンして管理を行なう。また、ワーキングページについては別のインデックスが張られている。

2.2.3 カレントページマネジャ

ある論理ページ ID に対して、カレントページイメージとワーキングページイメージが存在する場合、それらを格納する物理ページはそれぞれ別のものとする。カレントページマネジャは、これも含め lpid と ppid の対応表を管理する。

2.2.4 物理ファイルマネジャ

2次記憶へのアクセスは、物理ファイルマネジャを介して物理ページ単位で行なわれる。

2.2.5 トランザクションマネジャ

トランザクションマネジャは、トランザクションの同時性制御と、コミットの時刻のタイムスタンプ (ts) の決定を行な

う。機能の詳細については、現在検討中である。

2.2.6 履歴ページマネジャ

履歴ページマネジャは、履歴ページイメージの差分管理や履歴ファイルの管理を行なう。

2.2.7 履歴ファイルバッファマネジャ

履歴ファイルバッファマネジャは、履歴ファイルの各ページを履歴ファイルバッファに読み込み、そのバッファの管理を行なう。

2.3 論理ページへのアクセス

2.3.1 ページイメージの読み出し

ファイルマネジャを利用してページイメージを読み出す場合は、論理ファイルID(lfid), 論理ページID(lpid), 時刻(time), トランザクションID(xid)をファイルマネジャに与える。timeはその時刻でのカレントページイメージを読み出すための指定であり、xidは読み出すトランザクションのIDである。これらを受けとったファイルマネジャは、以下の手順でページイメージを論理ページバッファに読み込み、その内容を返す。

1. 論理ファイルマネジャはワーキングページの有無をトランザクションマネジャに問い合わせる。
2. lfid, lpid, time, xid, ワーキングページの有無を論理ページバッファマネジャに渡す。
3. 論理ページバッファマネジャは自分が管理しているバッファの中に該当するページイメージがすでに読み込まれていないかを調べる。もし、すでに読み込まれていたら10へ行く。
4. 論理ページバッファを1ページ確保する。そのバッファをbuffとする。
5. カレントページマネジャにlfid, lpid, xid, buff、ワーキングページの有無を渡す。
6. カレントページマネジャはxidで、論理ページ物理ページの対応表を引き、カレント(またはワーキング)ページイメージに対応するpfidとppidを得る。
7. pfid, ppid, buffを物理ファイルマネージャに渡し、カレント(またはワーキング)ページイメージをbuffに読み込み、論理ページバッファマネジャに戻す。
8. 論理ページバッファマネジャは、timeを調べて、timeが"now"であった場合は、10へ行く。
9. "now"以外の場合は、lfid, lpid, time, buffを履歴ページマネジャに渡す。履歴ページマネジャはカレントページイメージに対して時間順に差分を復元し、timeの時刻でのカレントであった履歴ページイメージをバッファbuffに作る。
10. 該当する論理ページバッファ中のページイメージを返す。

上記の説明では、現在のカレントページイメージを元に差分情報の復元を行なっているために3では、該当するページイメージを読み込んでいる論理ページのみを探している。しかし、カレントページイメージを元にする場合以外に、次のように論理ページバッファに読み込まれているページイメージを差分情報の復元元として選択すると差分を戻す回数を減少させられる。つまり、論理ページバッファに読み込まれている同じ論理ページIDを持つページイメージの中で、ページイメージの作成タイムスタンプが指定された時刻以降のもので最も古いタイムスタンプを持つページイメージを差分情報の復元元として選択するのである。

2.3.2 ページイメージの書き込み

上位のモジュールがページイメージを書き込む場合は、lfid, lpid, xid, ページイメージをファイルマネジャに与える。xidは書き込みを行なうトランザクションのトランザクションIDである。これらを受けとったファイルマネジャは、以下の手順で論理ページイメージをバッファにコピーする。

1. 論理ファイルマネージャは、トランザクションマネジャにワーキングページの有無と作成の許可を問い合わせる。ワーキングページが有るか、作成の許可が得られたら以降を行なう。
2. lfid, lpid, xid, ワーキングページの存在または作成許可を論理ページバッファマネジャに渡す。
3. ワーキングページがすでに存在する時は、論理ページバッファマネジャは、自分の管理するバッファの中に該当するワーキングページイメージがあるかを探す。見つかった場合は、6へ行く。
4. 見つからなかった場合、1論理ページバッファを確保して、6へ行く。
5. ワーキングページの作成許可が渡された場合、新たにワーキングページ用の物理ページの割り当てをカレントページマネジャに求め、論理ページ/物理ページの対応表を改め、その後、論理ページバッファを確保する。
6. ページイメージを、対応する論理ページバッファにコピーする。

2.3.3 コミット処理

カレントページイメージは該当する更新トランザクションがコミットする時に履歴ページイメージになる。同時に、ワーキングページイメージがカレントページイメージになる。この時、論理ファイルマネジャはxidを渡され、次のような動作をする。

1. トランザクションマネジャにトランザクションxidのコミットの要求を出し、コミット時刻のタイムスタンプtsを要求する。
2. 論理ページバッファにある、そのトランザクションが更新したワーキングページイメージを物理ファイルにフラッシュする。
3. そのトランザクションのワーキングページイメージ、トランザクション開始前のカレントページイメージ、tsとを履歴ページマネジャに渡し、差分を取り、圧縮し、履歴ファイルに格納する。
4. カレントページマネジャが、論理ページ/物理ページの対応表を更新する。

3 シミュレーションモデル

前節でページ単位での履歴をとるファイルマネジャの設計を説明したが、履歴をとるには、レコード構造を利用する方式もある。本節では、ファイルの格納効率と検索時のページアクセスの回数について、ページ単位で履歴を取る方式とレコード単位で履歴を取る方式をシミュレーションにより比較検討する。

3.1 シミュレーションモデルの概要

シミュレーションのモデルとしては、任意の時刻でのレコードのイメージを取り出せることを前提として、次の3つのモデルを考える。

モデルA 全てのレコードイメージをチェーンして管理する。

モデルB レコードの差分をチェーンして管理する。

モデルC ページを差分管理する。

3.1.1 ファイル

基本となるファイルはレコードの集合である。各レコードは、属性の並びであり、レコードID(4bytes)が付けられている。ページのサイズは4096bytesであり、レコードは、ページに跨ることはないものとする。レコードを格納するページは図 3のようにヘッダ部、スロット部、レコード部からなり、レコードIDから、ページIDとスロット番号に変換され、レコードにアクセスする。ページヘッダ部の大きさは32bytesであり、管理情報が納めれている。スロットは、レコードのページ中での位置を示すオフセット(2bytes)とレコードサイズ(2bytes)の情報からなる。

各レコードの履歴を以下では版と呼ぶ。また、同じレコードIDのレコードの版の集合を版集合と呼ぶ。

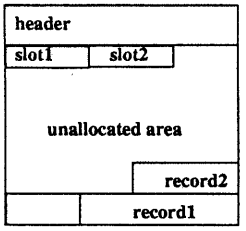

図 3. ページのレイアウト

以下では、次のことを仮定する。

- レコードは固定長であり、ページ内でのレコードの位置は変化しない。
- 作成トランザクションで一括して、データベース中のすべてのレコードの最初の版の作成が行なわれる。する。その後、更新トランザクションが各レコードの更新を行なう。全ての更新が終了した後に検索を行ない、その時のページアクセス数を比較する。
- 作成トランザクションでは、1ページにある割合(ロードファクタ)でレコードを挿入していくことにする。
- 各更新トランザクションでは1レコードの更新のみを行なうものとする。
- 更新トランザクションによるレコードの更新は、全てのレコードに対して偏りな同一の頻度で行なわれる。

3.1.2 モデルA

各版集合について全てのレコードイメージをそのままカレントの版から逆時間順にポインタでチェーンしてファイルに格納する。更新時に元のレコードが格納されたページに適当な空き領域がある場合は、直前の版はそのページに格納される。空き領域がない場合には、溢れページと呼ぶ他のページが割り当てられる。

ある時刻でのレコードの値が取り出したい場合は、レコードIDからカレントの版のレコードを探し、チェーンをたどって、該当するレコードを取り出す。

3.1.3 モデルB

Postgress[12]などで取られている方法の修正版である。各レコードの版間の差分を取り、その差分を現在の版から逆時間順にポインタでチェーンする。レコードの版の差分としては、更新された属性値のみを格納する。ある時刻でのレコードの値を取り出したい場合は、レコードIDから、カレントのレコードを取りだし、取り出したい時刻までの差分を順に戻していき該当するレコードの版を得る。モデルAと同様に必要に応じて溢れページに版間の差分を格納していく。

3.1.4 モデルC

基本となるファイルをページ履歴管理機能をもつ本ファイルマネジャで履歴管理した場合のモデルである。この場合、レコードが更新された場合でもレコードID、すなわち、ページID、スロット番号は保たれることとする。

また、ページの差分はXOR(eXclusive OR)でとり、差分の圧縮方法としては、WBS(White Block Skipping) コーディング法[13]を用いる。

3.2 パラメタ

以下パラメタでシュミレーションを行なう。

3.2.1 共通のパラメタ

PS ページサイズ。(4096byte = 4Kbytes)

HS ページヘッダサイズ。(32bytes)

TS タイムスタンプのバイト数。(4bytes)

Rid レコードidのバイト数。(4bytes)

SlS スロットのサイズ。SlS = offset + size

offset オフセットを格納するのに必要なバイト数。(2bytes)

size スロットに入れるデータのサイズを格納するのに必要なバイト数。(2bytes)

AS 属性のサイズ。(8bytes)

AN 1レコード中の属性の数。(16)

RS 1レコードのサイズ。

RN データベース中のレコードの数。(50,000)

MaxRN 1ページ中に入る最大のレコード数。
MaxRN = $\lfloor$(PS − HS)/(RS + SlS)$\rfloor$

UR 1レコードの更新回数。(50)

UA 1レコード中の更新される属性数。

US 1レコード中で更新されるバイト数。
US = AS × UA

VN1 元のレコードと同じページに格納される場合の平均バージョン数。

VN2 元のレコードとは別のページの格納される場合の各ページの平均バージョンの数。

LF データベース作成時に1ページ中に作成するレコードの割合 (%)。

FRN データベース作成時に1ページ中に作成するレコードの数。
FRN = ⌈LF × MaxRN/100⌉

FPN データベース作成時に作られるページ数。
FPN = ⌈RN/FRN⌉

PUS 最初のレコードの挿入後の1ページ中の未使用のバイト数。
PUS = PS − HS − FRN × (RS + SIS + TS + Rid)

TPN データベースを格納するのに必要な総ページ数。

3.2.2 モデルAに固有のパラメタ

RS RS = AS × AN + Rid + TS × 2

FRN データベース作成後に1ページの残りに入るレコード数。
FRN = MaxRN − FRN

VN1 VN1 = ⌊FRN/FRN⌋

VN2 VN2 = ⌊MaxRN/RN⌋

TPN TPN = ⌈RN/FRN⌉ + ⌈(UR − VN1) × RN/MaxRN⌉

3.2.3 モデルBに固有のパラメタ

RS RS = AS × AN + Rid + TS × 2

ABS 1レコード中で更新された属性値がどの属性であるかをしめすビットマップのサイズ。
ABS = ⌈AN/8⌉

RDS 版間の差分のバイト数。
RDS = AS × UA + TS

RDRS 版間の差分レコードのサイズ。
RDRS = RDS + ABS + TS × 2 + Rid

FDRN データベース作成後に1ページに入る差分レコードの数。
FDRN = ⌊PUS/(RDRS + SIS)⌋

MaxDRN 1ページ中に入る最大の差分レコード数。
MaxDRN = ⌊(PS − HS)/(RDRS + SIS)⌋

VN1 VN1 = ⌊FDRN/FRN⌋

VN2 VN2 = ⌊MaxDRN/RN⌋

TPN TPN = ⌈RN/FRN⌉ + ⌈(UR − VN1) × RN/MaxDRN⌉

3.2.4 モデルCに固有のパラメタ

RS RS = AS × AN + TS × 2

PHDS 1ページの履歴間の差分のバイト数
PHDS = US + TS

HRS 履歴ページに格納されるレコードのサイズ
HRS = CPS + SIS + TS

MaxHN 1履歴ページに入る履歴ページイメージの差分の数
MaxHN = ⌊(PS − HS)/HRS⌋

TPN TPN = ⌈RN/FRN⌉ × (1 + ⌈RN × UR/MaxHN⌉)

SgS WBSコーディング法におけるセグメントのサイズ。(128bytes)

SgN 1ページ中のセグメントの総数。(32)
SgN = PS/SgS

NWSgN whiteでないセグメントの数。
NWSgN = ⌈(TS + AS × UA)/SgS⌉

CPS 圧縮後の履歴ページイメージ間の差分のサイズ。
CPS = ⌈(SgN + SgS × 8 × NWSgN)/8⌉

3.3 シミュレーションでのコスト式

シミュレーションに際し、次の2通りの場合を考える。すなわち、ある1レコードの履歴を辿る場合と、ある時刻においてファイルに含まれていた全レコードをスキャンする場合である。上に述べた3つのモデルについて、この2通りの場合のページアクセスの回数を比較することにする。とくに、ページアクセスの回数に注目するのは、差分を取ったり、戻したりするのは、メモリ上での操作であるので、それよりもファイルのI/Oの方が処理時間に与える影響が大きいと考えたからである。

レコード単位で履歴を取る方式では、溢れページに履歴情報を格納するときにどのように格納して行くかは重要な要素となる。本稿では、溢れページには、元のページが同じレコードに関する履歴のみをグルーピングして格納する場合と、グルーピングしないで格納する場合を考える。グルーピングした場合は、ある溢れページには、データベース作成時に同じページに格納されたレコードに関する履歴情報のみが格納されることになる。

従って、溢れページの格納方針によって、モデルを以下のように分ける。

モデルA-1 レコードをチェーンする。履歴情報のグルーピングはしない。

モデルA-2 レコードをチェーンする。履歴情報のグルーピングを行なう。

モデルB-1 レコードの差分をチェーンする。履歴情報のグルーピングはしない。

モデルB-2 レコードの差分をチェーンする。履歴情報のレコード、ページ単位でグルーピングを行なう。

モデルC ページ単位で履歴を取る。

これらの5つのモデルについて、以下ようにの条件を変えて検討を行なう。

- バッファサイズはレコードを読み出す際のページアクセスに大きく影響するが、本稿では、バッファサイズが1の場合と、無限大の場合のみを考える。バッファサイズが有限の場合は、この2つの間に入る。
- ロードファクタは、50%と100%について考える。
- 更新される属性の数は1属性と4属性の2つの場合を想定する。

この条件で、各モデルについて、次の量を計算することにする。

ROR(AsOf) あるレコードのAsOf個前の版を読み出すのに必要なページアクセス数。

SAR(AsOf) 全てのレコードについてAsOf個前の版が存在していた時点でのファイル中の全レコードをスキャンする時の総ページアクセス数。

各々のモデルについて、計算式は以下のようになる。

モデルA

モデルA-1

$$\mathbf{ROR(AsOf)} = 1 + \lfloor (\mathrm{AsOf} - \mathrm{VN1}) \rfloor$$

$$\mathbf{SAR(AsOf)} = \begin{cases} \mathrm{FPN}, & \mathrm{AsOf} \leq \mathrm{VN1} \\ \mathrm{ROR(AsOf)} \times \mathrm{RN}, & \text{バッファサイズが1} \\ \mathrm{FPN} + \lceil (\mathrm{AsOf} - \mathrm{VN1}) \times \mathrm{RN}/\mathrm{MaxRN} \rceil, & \text{バッファサイズが無限大} \end{cases}$$

モデルA-2

$$\mathbf{ROR(AsOf)} = 1 + \lfloor (\mathrm{AsOf} - \mathrm{VN1})/\mathrm{VN2} \rfloor$$

$$\mathbf{SAR(AsOf)} = \begin{cases} \mathrm{FPN}, & \mathrm{AsOf} \leq \mathrm{VN1} \\ \mathrm{ROR(AsOf)} \times \mathrm{RN}, & \text{バッファサイズが1} \\ \mathrm{FPN} \times (1 + \lceil (\mathrm{AsOf} - \mathrm{VN1})/\mathrm{VN2} \rceil), & \text{バッファサイズが無限大} \end{cases}$$

モデルB

モデルB-1

$$\mathbf{ROR(AsOf)} = 1 + \lfloor (\mathrm{AsOf} - \mathrm{VN1}) \rfloor$$

$$\mathbf{SAR(AsOf)} = \begin{cases} \mathrm{FPN}, & \mathrm{AsOf} \leq \mathrm{VN1} \\ \mathrm{ROR(AsOf)} \times \mathrm{RN}, & \text{バッファサイズが1} \\ \mathrm{FPN} + \lceil (\mathrm{AsOf} - \mathrm{VN1}) \times \mathrm{RN}/\mathrm{MaxDN} \rceil, & \text{バッファサイズが無限大} \end{cases}$$

モデルB-2

$$\mathbf{ROR(AsOf)} = 1 + \lfloor (\mathrm{AsOf} - \mathrm{VN1})/\mathrm{VN2} \rfloor$$

$$\mathbf{SAR(AsOf)} = \begin{cases} \mathrm{FPN}, & \mathrm{AsOf} \leq \mathrm{VN1} \\ \mathrm{ROR(AsOf)} \times \mathrm{RN}, & \text{バッファサイズが1} \\ \mathrm{FPN} \times (1 + \lceil (\mathrm{AsOf} - \mathrm{VN1})/\mathrm{VN2} \rceil), & \text{バッファサイズが無限大} \end{cases}$$

モデルC

$$\mathbf{ROR(AsOf)} = 1 + \lceil \mathrm{AsOf} \times \mathrm{UR} \times \mathrm{FRN}/\mathrm{MaxHN} \rceil$$

$$\mathbf{SAR(AsOf)} = \mathrm{FPN} \times (1 + \lceil \mathrm{AsOf} \times \mathrm{FRN}/\mathrm{MaxHN} \rceil)$$

4 シミュレーション結果

4.1 総ページ数

各モデルについて、履歴情報を含めて格納するのに必要な総ページ数は次のようになる。

モデル	ロードファクタ50%		ロードファクタ100%	
	4属性更新	1属性更新	4属性更新	1属性更新
A-1	91072	91072	91072	91072
A-2	92872	92872	91086	91086
B-1	33202	17996	32651	17812
B-2	35720	17860	46436	19646
C	60012	40008	56925	37950

総格納ページ数で比較すると、モデルB-1が、一番ページ数が小さくなっている。グルーピングを行なうとフラグメントが多数生じることになるので、グルーピングをしない場合に比べて格納効率が悪化する。また、ロードファクタが100%で変更属性の数が多くなる場合は、レコードの差分の大きさも大きくなるので、モデルB-2の格納ページ数とモデルCの格納ページ数の差は減少していく。

4.2 1レコードの履歴を辿る

以下1レコードの履歴を辿る場合のページアクセス数を比較する。図の縦軸はページアクセス数、横軸はAsOfすなわち何版前の版までを読み出しの対象にするかを示す。

1レコードを辿る場合、ロードファクタ50%、4属性更新、各モデルを比較すると、図4のようになる。同じ条件でロードファクタを100%にすると図5のようになる。図4と比較すると、モデルA-1以外では、ロードファクタ100%の方がアクセス数が多くなっている。また、いずれのロードファクタにおいてもレコード単位で履歴を取っているモデルA、Bの場合、グルーピングをするとアクセスページ数が減っている。ロードファクタが50%のときも100%の時も、モデルB-2、つまり、レコードイメージの差分のチェーンで、グルーピングした場合がアクセス数が1番少なく、次にモデルC、モデルA-2、モデルB-1、モデルA-1の順になっている。

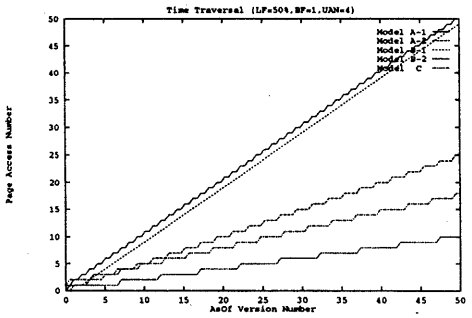

図4. 1レコードを辿る場合(ロードファクタ50%, 4属性更新)

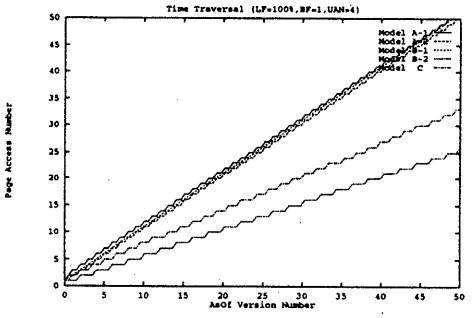

図 5. 1 レコードを辿る場合 (ロードファクタ 100%, 4 属性更新)

次に、1 レコードを辿る場合で、更新する属性数を変えてみる。ロードファクタ 50% で、1 属性更新の場合には、図 6 のようになる。また、4 属性更新の場合は図 7 のようになる。この場合、レコードイメージのチェーンであるモデル A では、属性更新数が変化しても影響を受けない。また、レコード単位で履歴管理をしている場合グルーピングの効果によって、モデル A-2、モデル B-2 とも、それぞれモデル A-1、モデル B-1 よりもアクセス数が少なくなっている。更新属性数が少なければ、レコードの履歴間の差分も、ページの履歴間の差分も減少するので、更新属性が 1 属性の場合は、4 属性の場合に比べてモデル B、モデル C ともにページアクセス数は減少している。更新属性数が 1 属性、4 属性のどちらの場合も、ページアクセスの少ない順では、モデル B-2、モデル C、モデル B-1、モデル A-2、モデル A-1 となっている。

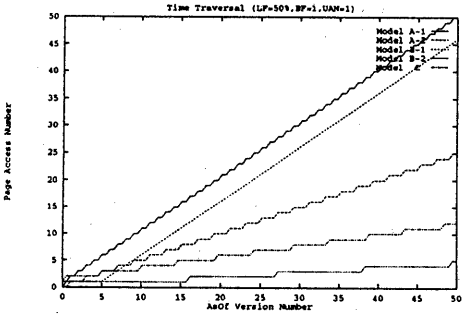

図 6. 1 レコードを辿る場合 (ロードファクタ 50%, 1 属性更新)

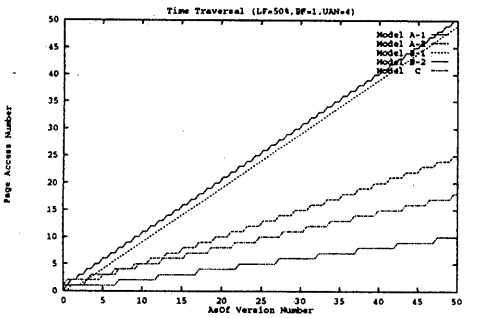

図 7. 1 レコードを辿る場合 (ロードファクタ 50%, 4 属性更新)

4.3 全レコードのスキャン

全レコードをスキャンする場合を考えてみる。図の縦軸と横軸は 1 レコードの履歴を辿る場合と同じである。

属性更新数が 4 属性、ロードファクタが 50%、バッファサイズが 1 の場合は、図 8 のようになる。バッファサイズが 1 の時は、モデル C がページアクセス数が一番少ない。次に、モデル B-2, モデル A-2 とページアクセス数が増えていていき、グルーピングしていないモデル B-1、A-1 の順になる。

同じ条件で、バッファサイズを無限大にした場合は図 9 のようになる。バッファサイズが無限大のときには、モデル A-1 と A-2 は、ほぼ同じページアクセス数で、データベースの総格納ページ数に近付いていく。モデル B-1、B-2 も同様な傾向を示す。また、バッファサイズが無限大の場合には、モデル B がモデル C よりもページアクセス数が少なくなっている。図 8 と図 9 の比較から、バッファサイズが大きくなればいずれのモデルでもページアクセス数が少なくなるが、特にモデル A、B ではその影響が大きい。

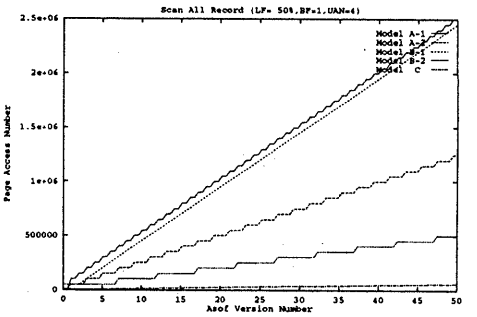

図 8. 全レコードのスキャンの場合 (ロードファクタ 50%, バッファサイズ 1, 4 属性更新)

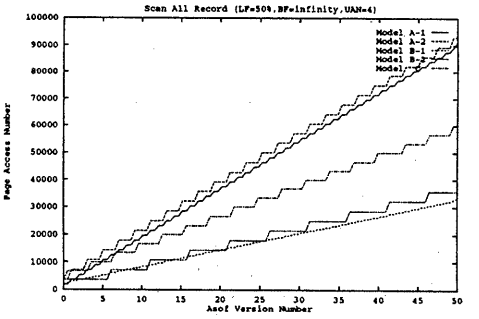

図 9. 全レコードのスキャンの場合 (ロードファクタが 50%, バッファサイズ 無限大, 4 属性更新)

同じ比較をロードファクタを変えて行なう。属性更新数が 4 属性、ロードファクタが 100%、バッファサイズが 1 の場合では、図 10 のようになる。同様に属性更新数が 4 属性ロードファクタが 100%、バッファサイズが無限大の場合には図 11 のようになる。

図 8 と図 10 を比較するとロードファクタが 100% の場合は、モデル A ではグルーピングの効果がない。これは、ロードファクタが 100% の時は、ページ中にレコードが一杯に詰まっているためである。

また、図 9 と図 11 を比較すると、バッファサイズが無限大のときは、データベースの総格納ページ数の差が、アクセス数に直接影響していることが分かる。

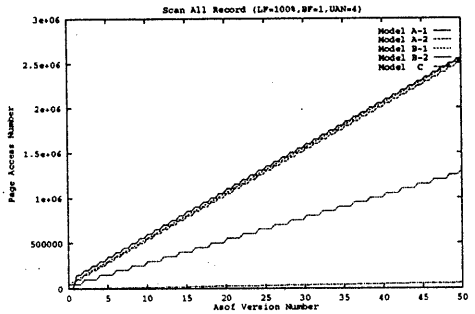

図 10. 全レコードのスキャンの場合 (ロードファクタ 100%, バッファサイズ 1, 4 属性更新)

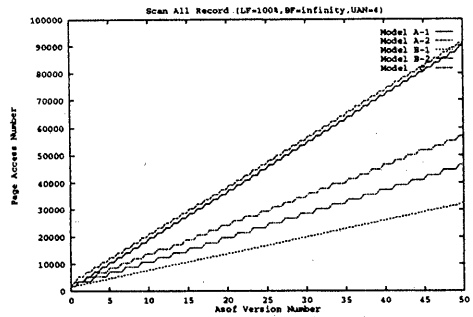

図 11. 全レコードのスキャンの場合 (ロードファクタ 100%, バッファサイズ 1, 4 属性更新)

ロードファクタが 50%、バッファサイズが 1、更新属性数が 1 の場合は、図 12 のようになる。ロードファクタ 100%、バッファサイズが 1、更新属性が 1 の場合は、図 13 のようになる。

ロードファクタが 50%、バッファサイズが無限大、属性更新数が 1 の場合は、図 14 のようになる。ロードファクタが 100%、バッファサイズが無限大、更新属性数が 1 の場合は、図 15 のようになる。

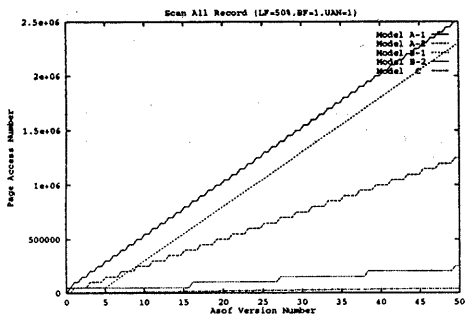

図 12. 全レコードのスキャンの場合 (ロードファクタ 50%, バッファサイズ 1, 1 属性更新)

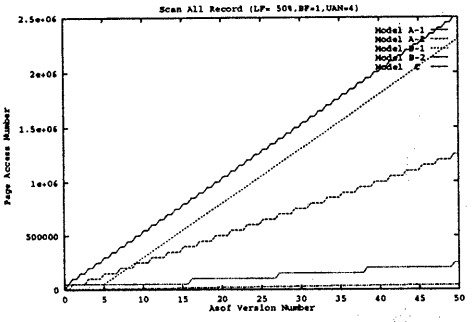

図 13. 全レコードのスキャンの場合 (ロードファクタ 100%, バッファサイズ 1, 1 属性更新)

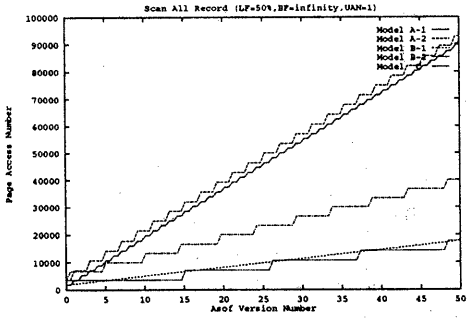

図 14. 全レコードのスキャンの場合 (ロードファクタ 50%, バッファサイズ 無限大, 1 属性更新)

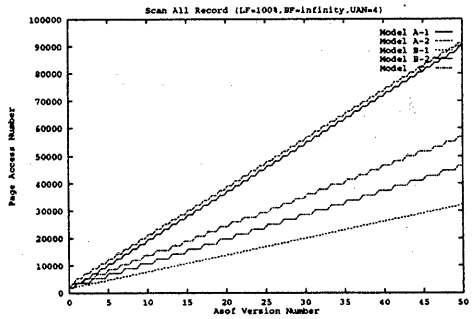

図 15. 全レコードのスキャンの場合 (ロードファクタ 100%, バッファサイズ 無限大, 1 属性更新)

5 まとめ

前節でのシュミレーション結果をまとめると以下のようなことが言える。

データベースの格納ページ数で比較すると次のようになる。レコード単位で履歴を取った場合は、履歴の格納時のグルーピングを行なう場合、行なわない場合に比べて総格納ページは多くなってしまう。また、更新属性数が多くなれば、レコードレベルでの差分を取った場合での差分の大きさが大きくなるので、ページ単位で差分を取った場合とのデータベースの総格納ページ数が近くなることが想定される。

次に、レコードの読み出しを検討する。レコード単位で履歴を取った場合は、グルーピングをした場合の方が、グルー

ビングをしない場合に比べてページアクセス数は少なくなっている。ただし、バッファサイズが無限大の場合、全レコードのスキャンはデータベースの総格納数に対応したページアクセス数になるので、全レコードのスキャンの場合は、総格納ページ数が少ない方が有理である。また、レコードの構造によっては、グルービングの機能の実現が困難な場合も想定される。

レコード単位で履歴を取った場合は、ページ単位の履歴を取った場合に比べて、設定条件によってページアクセス数が大きく変動してしまう。それに対して、ページ単位の履歴を取った場合は、ほぼ安定したページアクセス数であるといえる。従って、レコードへのアクセスパターンやデータの更新の程度によって、ページへのアクセス数は大きく変わるので、必ずしもレコードレベルでの履歴管理がページアクセス数に関して最良とは言えない。

本稿では、ページ単位での履歴管理機構を組み込んだファイルマネジャを設計し、シミュレーションにより、レコード単位で履歴を取った場合と比較した。今後の課題として以下のような点がある。

- 今回は、コスト式によるシミュレーションであったが、このファイルマネジャを実装した場合には、バッファサイズがページマネジャのI/Oのパフォーマンスに大きく影響を与えることが予想される。したがって、バッファサイズを与えてシミュレーションをすることは重要な課題である。
- 今回のシミュレーションではページサーバやレコードサーバとして実装した場合については検討していないが、サーバとして実装した場合の通信量についても検討を行なうことが必要である。
- 本稿では、ページの差分を取る方法としてXORを用いた。この方式では、ページ内のデータの一部が挿入や削除によって移動した場合などに実際に変更されたデータ量に対して、差分のサイズが大きくなるという問題がある。ページのイメージの変化に対する効果的な差分の取り方は検討の必要がある。同様に圧縮法についても検討しなければならない。
- 本稿のページマネジャの実装に向け、同時性制御やリカバリ機構について今後さらに詳細な検討が必要である。

謝辞

常日頃研究を進めるにあたり、ご助言、ご討議をいただいている筑波大学電子・情報工学系 藤原 譲教授、鈴木 功教授ならびにデータベース研究室の諸氏に感謝いたします。

参考文献

1) K. Furuse, K. Yamaguchi, H. Kitagawa, N. Ohbo, Abstract Indexing Mechanism of the Extensible DBMS MODUS, *Proc. of 3rd DASFFA*, 1993.

2) V. Lum, P. Dadam, R. Erbe, J. Guenauer, P. Pistor, G. Walch, H. Werner and J. Woodfill, Designing DBMS Support for the Temporal Dimension, *Proc. of ACM SIGMOD Conf.*, pp. 115-130, 1984.

3) R. Snodgrass and I. Ahn, A Taxonomy of Time in Database, *Proc. of ACM SIGMOD Conf.*, pp. 236-246 , 1985.

4) Gene T. J. Wuu, and Umeshwar Dayal, A Uniform Model for Temporal Object-Oriented Database, *Proc. of 8th Data Engineering*, 1992, pp.584-593.

5) D. Lomet and B. Salzberg, Access Methods for Multiversion Data, *Proc. of ACM SIGMOD Conf.*, pp. 315-324, 1989.

6) Ramez Elmasri, Gene T. J. Wuu and Yeong-Joon Kim, The Time Index:An Access Structure for Temporal Date, *Proc. of 16th VLDB Conf.*, pp. 1-12, 1990.

7) 北川 博之、田中 肇, 大保 信夫, 鈴木 功, 履歴データ型を用いた版管理データモデルの提案, 情報処理学会論文誌,Vol.34, No. 5, pp.1031-1044, 1993

8) Christian S. Jensen, Michael D. Soo and Richard T. Snodgrass, Unification of Temporal Data Models, *Proc. 9th Int. Conf. on Data Eng.*, pp. 262-271, 1993.

9) Arvola Chan, Stephen Fox, Wen-Te K. Lin, Anil Nori, and Daniel R. Rise, The Implementation of An Integrated Concurrency Control and Recovery Scheme, *Proc. ACM SIGMOD Conf.*, pp. 184-191, 1982.

10) David J. DeWitt, Philippe Futtersack, David Maier and Fernando Vélez, *Three Alternative Workstation-Server Architectures*, in "Building An Object-Oriented Database System -The Story of O_2", Morgan Kaufmann, pp 411-446, 1992.

11) Ilsoo Ahn and Richard Snodgrass, Partitioned Storage for Temporal Databases, *Inf. Sys.*, Vol.13 No. 4, pp. 369-391, 1988.

12) M. Stonebraker, The Design of The POSTGRES Storage System, *Proc. of 13th VLDB*, pp. 289-300, 1987.

13) Shi-Kuo Chang, "Principles of Pictorial Information System Design," Prentice-hall International,Inc., pp. 109-110.