データベースシステム 109－28
（1996. 7. 25）

時制オブジェクトデータベースシステムにおける履歴情報アクセスの研究

鈴木 孝幸† 北川 博之††

†筑波大学 工学研究科
††筑波大学 電子・情報工学系
E-mail : {suzu, kitagawa}@dblab.is.tsukuba.ac.jp

概要

データベース応用分野の高度化に伴い、複雑なデータ構造や手続きを伴うオブジェクトを扱うことが可能なODBSの利用が進みつつある。一方、多様なデータベース応用の中には、時間情報やオブジェクトの更新履歴管理を必要とするものも多い。我々の研究グループでは、こうした要求に対応するため時制オブジェクトデータベースシステムの開発を行ない、その基本性能についても測定してきた。本稿では、オブジェクトの過去の状態を表す履歴オブジェクトのアクセスの高速化を目的として、履歴ファイルの構成法の改良、履歴オブジェクトのポインタ書換え法の改善について、その性能も含めて検討する。

Study on Access Schemes to Historical Objects in a Temporal Object Database System

Takayuki Suzuki† Hiroyuki Kitagawa††

†Doctoral Degree Program in Engineering, University of Tsukuba
††Institute of Information Sciences and Electronics, University of Tsukuba
E-mail : {suzu, kitagawa}@dblab.is.tsukuba.ac.jp

Abstract

ODBSs are used in advanced database applications to manage complex objects and objects that have methods. Some of the applications also require temporal and historical information management in the context of such object management. We have developed a temporal persistent object management system, named POST/C++, and studied its basic performance. In this paper, we describe improvements of the organization of the history file, and application of the lazy pointer swizzling scheme to historical objects. We discuss the performance issues on our design and experiments.

1 はじめに

近年、工学などデータベース応用分野の高度化に伴い、複雑なデータ構造と手続きをもったオブジェクトを扱うオブジェクトデータベースシステム(ODBS)の研究・開発が進んでいる。また、データベースに対する応用要求として、時間情報管理やオブジェクトの更新履歴管理を行ないたいという要求

もある [1–4]。我々の研究グループでは、こうした要求に対応するための時制オブジェクトデータベースシステム POST/C++ の研究を進めている。 [5–7] では、 POST/C++ の設計・実装ならびに基本性能測定について報告した。その結果、オブジェクトの現在の状態を表す現在オブジェクトに対するアクセスに比べて、過去の状態を表す履歴オブジェクトに対するアクセスは、3倍から10倍ものオーバヘッドがかかることが測定されている。そこで、履歴オブジェクトアクセスの高速化の手法の検討が必要になる。そこで、 [8] で次の4つの高速化手法を提案した。

1. 履歴ファイルの構成法の改良
2. オブジェクト単位でのポインタ書換え
3. 履歴オブジェクト中の参照の解釈の拡張
4. 履歴ページの解放のオプション

本論文では、履歴オブジェクトアクセスの高速化の手法のうち、履歴ファイルの構成法の改良と履歴オブジェクトに対するオブジェクト単位でのポインタ書換え法の適用について、その性能も含めて検討する。

2章では、現在の POST/C++ の設計・実装の概要を述べる。3章では、新しい履歴ファイルの構成法について検討し、4章では、履歴オブジェクトに対するポインタ書換え法について検討し、5章では、今回の両手法を複合して用いた場合について検討する。6章では本研究のまとめを行なう。

2 POST/C++ システムの概要

本章では、時制オブジェクトデータベースシステム POST/C++ の概要について説明する。

2.1 時制オブジェクトモデル

本節では、 POST/C++ の基となる、時制永続オブジェクトモデルの概要について述べる。詳細は [5–7] で述べられている。本モデルでは、永続オブジェクトを対象とし、永続オブジェクトの時間による状態変化を扱うことを目的にする。

永続オブジェクトとは、永続ヒープと呼ばれる2次記憶上に領域に確保されることにより、複数のトランザクションにまたがって存在するオブジェクトのことである。各オブジェクトは、オブジェクト識別子 (OID) がふられ、一意的に識別される。

今、オブジェクトの時間による状態変化を考えると、生成されてから、更新トランザクションにより更新され、あるいは最終的には削除されるかもしれない。永続オブジェクトの生成は、 pnew オペレーションにより行なわれる。 pnew オペレーションは永続ヒープ上に確保した領域のアドレスを返り値にとる。

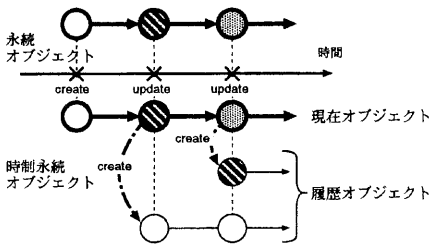

図 1: 時制オブジェクトの時間変化

図1 はオブジェクトの更新を示す。更新トランザクションはコミット時にオブジェクトの状態を変更する。本モデルでは、オブジェクトの状態変化を扱うために時制オブジェクトという概念を導入する。時制オブジェクトの更新履歴は、現在オブジェクトと履歴オブジェクトにより管理される。現在オブジェクトは時制オブジェクトの最新の状態を表し、履歴オブジェクトは時制オブジェクトの過去の状態を表す。現在オブジェクトのみが更新を受け付ける。時制オブジェクトの更新は“論理的な”更新となり、以下のように実行される。すなわち、指定された時制オブジェクトの現在オブジェクトが更新の内容を反映するように状態を変える。また同時に、更新が起こる直前の状態を保持する新たな履歴オブジェクトが生成される。各履歴オブジェクトは、その内容が時制永続オブジェクトの最新の状態を表していた時間帯を示す現在インターバルをもつ。時制オブジェクトの履歴を時間順番にアクセスするために、 iterator クラスを用意する。

また、時制オブジェクトの参照は、ディフォルトでは現在オブジェクトへの参照として解釈されるが、 snapshot オペレータによって時刻を与えられた場合は、その与えられた時刻で現在オブジェクトであった履歴オブジェクトへの参照として解釈される。

2.2 POST/C++ の設計と実装

前節で述べた時制オブジェクトを管理するためのシステム POST/C++ の設計と実装について述べる。 POST/ C++ では、 ODBMS の実装方法の1つとして知られているメモリマップ方式アーキテク

チャを採用した。

メモリマップ方式では、通常主記憶上と2次記憶上でOIDによる他のオブジェクトの参照の形式が異なるために、ポインタ書換え(pointer swizzling)を行なわなければならない。主記憶上でのOIDは、オブジェクトが主記憶のどこに格納されているかを示すポインタであるのに対して、2次記憶上のOIDは、ディスク上のどの場所に格納されているかを示すものであるからである。

POST/C++のアーキテクチャは、図2のようになる。より詳細については、[5-7]を参照されたい。以下、この基本となるシステムをBASICと呼ぶ。

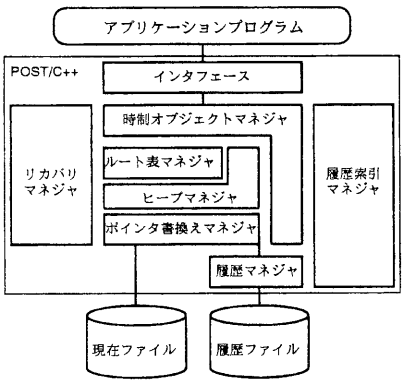

図2: POST/C++の基本アーキテクチャ

履歴オブジェクトと現在オブジェクトは、それぞれ履歴ファイル、現在ファイルと別のファイルに格納する。ある時刻での履歴オブジェクトを決定するためには、履歴索引を用意する。

2.2.1 履歴オブジェクトアクセスの手順

履歴オブジェクトにアクセスするには、`snapshot (coid, ts)`として、対象とするオブジェクトの主記憶上のアドレス`coid`と時刻`ts`が指定する。システムの内部では、以下のような手順で対象となる履歴オブジェクトが格納された履歴ファイルのページが主記憶にマップされ、その履歴オブジェクトを格納したアドレスが`snapshot`オペレータの返り値となる。

1. `coid`から現在オブジェクトのOIDを調べる。
2. 履歴索引を検索して、指定された時刻`ts`での履歴オブジェクトのOID(`HOID`)を調べる
3. 仮想記憶マッピング表を調べて、その履歴ページがマッピングされているかを調べる。マッピングされていたら5へ
4. `HOID`で示されるアドレスを含む履歴ページを主記憶に読み込み、そのページ中の他のオブジェクトへの参照のポインタ書換えをする
5. マップした主記憶上のアドレスを返す

2.3 基本性能測定

また、基本性能の測定は、以下のように時制オブジェクトに対するオブジェクトのアクセスのパタンとしては、次の2通りのパタンが多いものと考える[9,10]。

1. ある時刻を指定して、その時刻でのオブジェクト間の参照を辿る
2. ある一つの時制オブジェクトの履歴を辿る

この2つのパタンに対して、OO1ベンチマーク[11]を拡張したベンチマークを用いて、基本性能を測定した[5-7]。測定は、オブジェクトの更新を行なうUpdateを行ない、その後、ランダムにオブジェクトを読み出すLookup、オブジェクト間の参照を辿るTraversal, 履歴を辿るHistory Traversalの3通りを行なった。更新は、オブジェクトの属性値のみで、オブジェクト間の参照は更新しない。LookupとTraversalは、現在オブジェクトについてと、ある時刻を決めて過去のある時点での履歴オブジェクトについて読み出しを行なった。以下の章では、1のある時刻でのオブジェクト間の参照を辿る場合について検討する。その結果が表1である。以下の表では、結果はトランザクションの平均実行時間を秒を単位に表示している。また、[8]における、履歴ページの解放/非解放のオプションは、非解放としている。そのため、Hotの時の時間は、ほとんどが履歴索引を検索する時間である。したがって、履歴オブジェクトにアクセスする時間の内、履歴索引を引く以外の時間は、Cold - Hotの時間になる。以下の章での比較は、この値で行なう。

表1: BASICのベンチマークの結果[秒]

	Cold	Warm	Hot
Lookup	76.66	51.83 – 45.90	45.00
Traversal	100.51	89.09 – 80.76	79.66

3 履歴ファイルの構成法の改良

[8] で提案した履歴オブジェクトアクセスの高速化手法のうち、この章では、履歴ファイルの構成法の改良について述べる。

3.1 設計

POST/C++ BASIC システムでは、履歴オブジェクトは単純に生成された順番に履歴ファイルに格納していた。これでは、更新の元となる現在ファイル中で現在オブジェクト間に局所性がある場合に、履歴ファイルではその局所性が生かされない。一般に、オブジェクトをクラスタリングする手法はいくつも提案されているが、オブジェクトの履歴まで考えると、クラスタリングは容易ではなくなる。そこで、本研究では、現在ファイルのページを単位にして、履歴オブジェクトの格納位置を決めることにする。つまり、現在ファイルで同じページに入っていた履歴オブジェクトは、履歴ファイル中でも出来るだけ同じページに格納されるようにする。

3.2 実装

実装は、現在ファイルのページを単位に履歴オブジェクトの格納場所を決めるためのテーブルを新たに用意することで行なった。また、履歴オブジェクトの格納は、更新トランザクションのコミット時に、更新のあった現在オブジェクトの格納されているページ毎に処理を行なうので、今回の実装と非常に適合性が良かった。

3.3 性能評価

こうして、履歴ファイルの構成法を変更したシステムを HISTORY と呼び、ベンチマークを行ない、BASIC と比較を行なった。その結果が、表 2である。1つのトランザクションでランダムな 100 オブジェクトを更新する Update は、BASIC、HISTORY でほとんど差が出なかった。また、Lookup、History Traversal も差が出なかった。しかし、現在オブジェクトの参照の局所性があるので Traversal については、HISTORY は BASIC に対して、15.8% の性能の向上が測定された。これは、現在オブジェクトの参照の局所性が履歴オブジェクトの格納の時に有効に利用されて格納されているために、読み出す履歴ファイルのページ数が減少して性能の向上になっている。

つまり、現在オブジェクトがクラスタリングされていたデータベースの履歴を HISTORY の方法で履歴オブジェクトを格納した場合、ある時刻でのデータベースのスナップショットを取り、そのスナップショットの履歴オブジェクトを辿るような問合せに対しては、有効であるといえる。

表 2: HISTORY のベンチマークの結果 [秒]

	Cold	Warm	Hot
Lookup	76.45	53.11 – 45.24	45.32
Traversal	96.52	88.47 – 79.60	78.97

4 ポインタ書換え方式の検討

この章では、2次記憶からオブジェクトを主記憶に読み出す場合のポインタ書換えの方式について検討する。

4.1 オブジェクトの性質とポインタ書き換え方式の比較

POST/C++ BASIC システムでは、[12,13] で分類するところの Eager なポインタ書換えを行なっている。Eager なポインタ書換えとは、主記憶に存在している全てのポインタはポインタ書換えが行なわれていることを保証している方法である。それに対して、同分類による Lazy なポインタ書換えとは、実際にポインタを辿る時に要求に応じてポインタ書換えを行なう方式である。実際にポインタを辿る場合、Eager なポインタ書換えでは余分なオーバヘッドがかからないが、Lazy なポインタ書換えではポインタを辿る毎にポインタが書き換えられているかいないかとチェックをしなければならないので、Eager な方式と比べてオーバヘッドがある。

今回の実装は、ページマップ方式ということで、ページ単位でオブジェクトを主記憶に読み込むので、このページを読む時にポインタ書換えを行なわなければならない。つまり、あるページに存在するあるオブジェクトをアクセスすると、同じページに格納されているオブジェクトも主記憶に読み込まれてポインタ書換えされることになる。この方式では、ポインタ書換えのオーバヘッドを先に支払っていることになり、同じページに格納されているオブジェクトに対してアクセスがない場合、余分なポインタ書換えを行なったことになる。

今、現在オブジェクトと履歴オブジェクトの性質を比較してみると

- 現在オブジェクト

- アクセスは頻繁で、高速なアクセスが要求される

- 履歴オブジェクト
  - 一般には、現在オブジェクトほどアクセスの頻度も高くなく、現在オブジェクトほど高速性も要求されない

となり、同じページに格納されているオブジェクトに対するアクセスの頻度という点からみると、現在オブジェクトについては頻度が高いが、履歴オブジェクトについては頻度が低いということが言える。したがって、履歴オブジェクトについては、Eager な方式でなく、Lazy な方式の方が良いと言える。また、ポインタ書換えを行なうコストを考えると、履歴オブジェクトでは現在オブジェクトに比べて、履歴索引を引くと言うオーバヘッドがあるために、要求に応じた Lazy なポインタ書換えの方が向いていると考えられる。

そこで、現在オブジェクトと履歴オブジェクトについて、別々のポインタ書換えの方式を採用する。つまり、現在オブジェクトに対しては Eager なポインタ書換えを、履歴オブジェクトに対しては Lazy なポインタ書換えを行なうのが良いと考えられる。

4.2 実装

POST/C++ では、ページマップ方式を採用している、そのため主記憶への読込みはページ単位で行なわれる。Lazy なポインタ書換えを実装するため、オブジェクトの参照のための特殊なクラスを用意して、そのポインタが書き換えられているかいないかをチェックするポインタを辿る場合の処理すれば、基本設計指針と反してしまう。ところで、現在の POST/C++ の BASIC システムでは、履歴オブジェクトに対する直接のポインタを実装していないため、履歴オブジェクトへのポインタは snapshot オペレータか iterator を使って履歴を辿る場合の 2 通りしかない。そこで、今回の実装では、特殊なクラスを用意するのではなく、snapshot オペレータと iterator クラスを改変するにした。この実装を EAGER と呼ぶ。

4.3 性能評価

この EAGER に対して、ベンチマークを行なった結果は表 3のようになる。Lookup、Traversal ともに、約 8% の向上がある。今回の実装では、履歴オブジェクトを指す直接の参照がないため、ポインタはディフォルトで現在オブジェクトへと書き換えられるだけである。そのため、BASIC と EAGER では、読み込むページ数も mprotect() で予約するページ数も変わりがなく、ページ内でポインタ書換えを行なうオブジェクトの数の差でしかない。したがって、ページの I/O に比べて、ポインタ書換えの CPU の計算コストは少ないと測定された。仮に、履歴オブジェクトに対する直接のポインタを許した場合には、EAGER の有効性は、もう少し高くなると思われる。

表 3: EAGER のベンチマークの結果 [秒]

	Cold	Warm	Hot
Lookup	74.42	54.18 – 45.81	45.33
Traversal	99.95	91.44 – 80.87	80.73

5 両手法を複合して用いた総合評価

3 章で提案した手法は、履歴ファイルを構成する方法、4 章で提案した手法は、履歴オブジェクトを辿る場合の手法であるので、両者を組み合わせることが可能である。つまり、3 章で作成した履歴ファイルに対して、4 章の方法でアクセスする。そのシステムを COMPOSITE と呼ぶ。この COMPOSITE に対して、ベンチマークを行なった結果は表 4のようになる。Lookup の場合、BASIC と比べて 7.8% の向上で、ポインタ書換えの方法を変えた効果が出ている。それに対して、Traversal の場合は、14.8% の向上で、ほとんど履歴ファイルの編成法の影響のみであると思われる。これは、ベンチマークのデータベースが現在オブジェクトの参照の局所性が高いため、アクセスする履歴オブジェクトがほぼ同じ履歴ファイルのページに集まっているためであると考えられる。したがって、その履歴ファイルのページにある履歴オブジェクトはほとんどアクセスされることになり、Eager の場合も Lazy の場合もほぼ同じ数の履歴オブジェクトのポインタ書換えが行なわれたためであると思われる。

表 4: COMPOSITE のベンチマークの結果 [秒]

	Cold	Warm	Hot
Lookup	76.39	53.20 – 46.03	45.73
Traversal	97.49	88.46 – 79.65	79.73

6 まとめ

本論文では、時制データベースシステムにおける履歴オブジェクトアクセス機構について、次の2点の改善法を検討し、実装・評価を行なった。

1. 現在オブジェクトの局所性に基づく履歴ファイルの構成法

2. 現在オブジェクトにはEagerなポインタ書換え、履歴オブジェクトにはLazyなポインタ書換えを適応するポインタ書換え法

上記、2方式について、ベンチマークを用いて、ある時刻でのデータベースのスナップショット中のオブジェクトを辿ってアクセスする場合は、今回の手法が有効であることを示した。

今後の課題としては、以下の課題がある。

- 今回のベンチマークでの更新は、ランダムに選んだ現在オブジェクトに対する更新のみであったが、更新に局所性のある場合についても今回の手法の有効性を実証したい。また、今回と違うアクセスパタンについては、必ずしも最適な手法とは言えないので、それについても検討をする必要がある。

- 今回のアクセスパタンのように、ある時刻でのでデータベースのスナップショットにアクセスする場合、実装している履歴索引が効率的に働いていない。そこで、Time Index [14] のような索引を適応することも検討したい。

- 現在の実装では、履歴ファイル中に、履歴オブジェクトはそのままのオブジェクトイメージで格納している。そのため、オブジェクトの更新が行なわれる毎に、履歴ファイルのサイズは増大していく。今後、履歴オブジェクトに対して、圧縮、差分管理、あるいは3次記憶への格納などを考慮しなければならない。

参考文献

[1] Soo, M. D.: *Bibliography on Temporal Databases*, ACM SIGMOD RECORD, vol. 20, no. 1, 1991.

[2] Kline, N: *An Update of the Temporal Database Bibliography*, ACM SIGMOD RECORD, vol. 22, no. 4, 1993.

[3] Tansel, A.U., Clifford, J., Gadia, S., Jajodia, S., Segev,. A. and Snodgrass, R.: *Temporal Databases – Theory, Design, and Implementations* –, The Benjamin/Cummings Publishing, 1993.

[4] Tsotras, V.J. and Kumar, A.: *Temporal Databse Bibliography Update*, ACM SIGMOD RECORD, vol. 25, no. 1, 1996.

[5] 鈴木, 北川, 林: 時制永続オブジェクト管理システム POST/C++ の設計と実装 , 情報処理学会研究会報告書, 95-DBS-102, 1995.

[6] 鈴木, 林, 北川, 柳川: 時制永続オブジェクト管理システム POST/C++ の構築と性能評価, 日本ソフトウェア科学会 第12回大会予稿集, 1995.

[7] T.Suzuki and H. Kitagawa: *Development and Performance Analysis of a Temporal Persistent Object Store POST/C++*, Proc. of 7th ADC Conf., 1996.

[8] 鈴木, 北川: 時制永続オブジェクト管理システム POST/C++ における履歴オブジェクトアクセスの高速化手法の検討, 情報処理学第52回全国大会, 1996.

[9] Ahn, I. and Snodgrass, R.: *Performance Evaluation of a Temporal Database Management System*, ACM SIGMOD RECORD, Vol. 49, No.2, 1986.

[10] Ahn, I. and Snodgrass, R.: *Performance Analysis of Temporal Queries*, Information Sciences, Vol.49, 1989.

[11] Cattell, R. G. G. and Skeen, J.: *Object Operations Benchmark*, ACM Trans. on Database Systems, vol.17, no.1,1992.

[12] J. Eliot B. Moss: *Working with Persistent Objects: To Swizzle or Not to Swizzle*, IEEE Trans. of Software Engineering, Vol. 18, No. 8, 1992.

[13] Kemper, A. and Kossmann, D.: *Adaptable Pointer Swizzling Strategies in Object Bases*, Proc. of Int. Conf. Data Engineering, 1993.

[14] Elmasri, R., Wuu, G.T.J. and Kim, Y.J.: *The Time Index: An Access Struture for Temporal Data*, Proc. of 16th VLDB, 1990.