時制永続オブジェクト管理システム POST/C++ の構築と基本性能評価

Implementation and Basic Performance Analysis of a Temporal Persistent Object Management System POST/C++

鈴木 孝幸[†] Takayuki SUZUKI
北川 博之[††] Hiroyuki KITAGAWA
林 恵将[†††] Yoshiyuki HAYASHI
柳川 史彦[†††] Fumihiko YANAGAWA

†筑波大学 工学研究科
Doctoral Degree Program in Engineering, University of Tsukuba
††筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†††筑波大学 理工学研究科
Master's Degree Program in Sciences and Engineering, University of Tsukuba

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 ODBMS 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。本稿では、オブジェクトの過去の状態を扱うための時制永続オブジェクトモデルを実現するための時制永続オブジェクト管理システム POST/C++ の設計と実装について述べる。POST/C++ は C++ 言語のオブジェクトに永続性を与えると共に、時制永続オブジェクトモデルに基づく、ユーザトランザクションによる更新の履歴を管理可能である。また、ベンチマークによる POST/C++ の基本性能測定についても述べる。

1 はじめに

近年、工学などの応用分野におけるデータベースの高度利用が盛んになり、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オブジェクトデータベース管理システム (ODBMS) の利用が進みつつある。また、データベースに対する応用要求の中には、時間情報管理やオブジェクトの更新履歴管理の要求がある。これまで時制データベース [4]については、主に関係データベースを基礎にした多くの研究が行なわれてきているが、ODBMS で時間情報を扱うためには、オブジェクトの状態の時間変化を管理するための時制永続オブジェクトモデルが必要となる。時制永続オブジェクトモデルの提案はこれまでにもいくつか行なわれているが、それを実現するためのシステムの設計や実装まで含めた研究は、ほとんどないのが現状である。そこで、本研究では、[3]で提案した時制永続オブジェクトモデルにを支援するデータベースシステムの設計ならびに実装を行なった。また、実装したシステムの基本性能をベンチマークを用いて測定した。

2 章では、プログラミング言語 C++ のオブジェクトに永続性を持たせた時制永続オブジェクトモデルの概要にいて述べる。3 章では、そのモデルを支援する時制オブジェクト管理システム POST/C++ の設計と実装上について述べ、4 章でベンチマークによる基本性能測定について述べる。

2 時制永続オブジェクトモデル

本章では、POST/C++ におけ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 [3]で述べられている。本モデルでは、一時オブジェクトと永続オブジェクトの 2 種類のオブジェクトを扱うことができる。一時オブジェクトとは、その寿命がオブジェクトの生成されたトランザクション中に限られているオブジェクトのことである。それに対して、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。各オブジェクトは、オブジェクト識別子 (OID) と呼ぶ識

別子により一意に識別され、オブジェクトの永続性と型は直交する。

今、オブジェクトの時間変化を考えると、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。

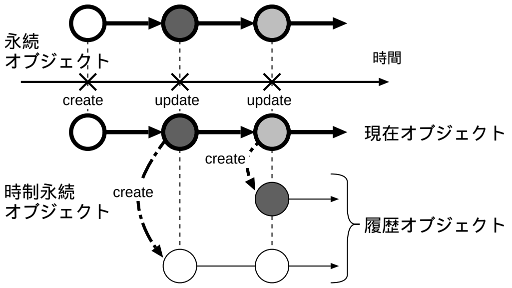

図 1　時制オブジェクトの時間変化と **OID**

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永続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永続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永続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。時制オブジェクトが削除された場合は、完全に消去されるのではなく墓石オブジェクトと呼ぶ特殊な現在オブジェクトとして存在することになる。

3　POST/C++ の設計と実装

前章で述べた時制永続オブジェクトを管理するためのシステム POST/C++ の設計について述べる。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する。メモリマップ方式の利点としては、以下の点があげられる。

- プログラミング言語処理系との親和性が高い
- 主記憶上と 2 次記憶上のページイメージのマッピングが単純である

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、*pointer swizzling* と呼ぶ変換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、[3]を参照されたい。

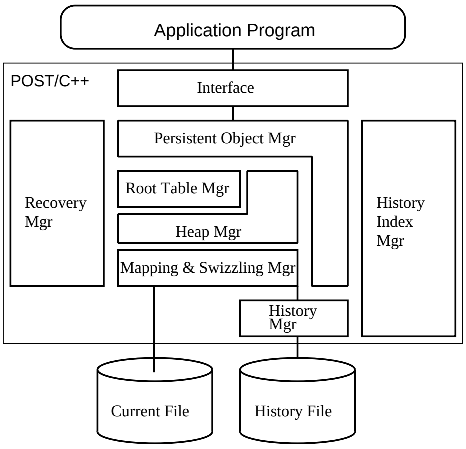

図 2　**POST/C++** の基本アーキテクチャ

3.1　時制永続オブジェクトの管理

全ての時制永続オブジェクトは、2 次記憶上に格納される。上に述べたように、2 次記憶上では OID として 2 次記憶上の物理アドレスを使用する。また、オブジェクトのヘッダに現在オブジェクトの OID と現在インターバルの情報を追加する。現在オブジェクトと履歴オブジェクトの対応を管理しなければならないが、OID だけでは時間に依存する情報は得られない。そこで、現在オブジェクトの OID と時刻をキーとして、その時刻を現在インターバルに含む当該履歴オブジェクトの OID を得るための履歴索引を用意する。

3.2　時制永続オブジェクトの格納

現在オブジェクトと履歴オブジェクトは以下のように性質が違う。

- 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
  - 更新が行なわれ得る
- 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、高速性も要求されない
  - 読出しのみが可能である

  - データベース操作に伴い量は膨大な量になりうる

こうした、性質の違いを踏まえて、現在オブジェクトと履歴オブジェクトをそれぞれ現在ファイル、履歴ファイルと呼ぶ別々の2次記憶に格納することにする。したがって、オブジェクトをページ単位でメモリにマップする際、現在オブジェクトと履歴オブジェクトが同一のページ中に混在することはない。また、2次記憶上の1ページは、チャンクと呼ぶ単位に等分割し、各オブジェクトは必要最小サイズのチャンクに格納する。これは、領域管理を容易にすると共に、トランザクションのコミットの時に更新されたオブジェクトを効率良く検出するためである。

3.3 コミット時の処理

更新されたオブジェクトは更新トランザクションのコミット時に検出され、履歴オブジェクトが履歴ファイルに格納されると共に履歴インデックスにその情報が挿入される。POST/C++では、UNIXシステムコール`mprotect`を用いて更新されたオブジェクトが格納されているページを検出する。次に、更新前のページイメージは2次記憶上に、更新後のページイメージは主記憶上にあるので、両者の差をチャンク単位で調べて更新されたオブジェクトを検出する。

3.4 実装

POST/C++の実装には、Texas [2]をベースシステムとして用いた。POST/C++の実現のために行なった主な変更と拡張は以下の通りである。

- オブジェクトヘッダの拡張
- 履歴オブジェクト操作関数の提供
- 履歴索引の構築
- 履歴ファイルの処理の構築
- コミット処理の変更と拡張
- リカバリ処理の変更と拡張

開発環境は、SunOS 4.1.2上でGNU g++コンパイラを用いて実装している。

4 基本性能測定

本章では、ODBMSの性能測定のためのベンチマークであるOO1を拡張したベンチマークによる、基本性能測定について述べる。

4.1 OO1ベンチマークの拡張

OO1ベンチマーク [1]はODBMSの性能測定用ベンチマークとしては最も基本的なものである。ベンチマークで用いるデータベースはデータモデルとは独立したものであり、Part型という単一のオブジェクト型で定義される。操作としては、オブジェクトの検索、トラバース、挿入操作である。Part型の定義は、以下のようになる。本研究では、オブジェクトのサイズが管理用の情報を格納するヘッダ部分を含めて128bytesまたは256 bytesの場合について測定を行なった。

```
class Part {
    int id;
    char type[10];
    int x,y;
    int build;
    Part* to[3];
    Part* from[...];
}
```

このデータベースの中には、全部で2万個のPartオブジェクトがある。それぞれのPartは他のランダムに選ばれた3つのPartを参照している。この参照は、参照の90%が全体の1%にあたるidが近いオブジェクトを参照する。本研究では、オブジェクトの更新履歴を取り扱うため、挿入操作の代わりに更新操作を行なう。また、検索、トラバース操作は、OO1では現在オブジェクトに対しての検索であるが、これを過去のある時点でのデータベースの状態を表す履歴オブジェクトを対象とした場合も含めるように拡張する。まとめると、以下の処理の実行時間を測定する。LookupとTraverssalについては、現在オブジェクトを検索する場合と履歴オブジェクトを検索する場合を測定する。

Lookup 1,000個のランダムに選ばれたPartを検索する

Traversal ランダムにPartを選び、そのPartから接続されているPartを検索する。7回接続をたどり、全部で3,820個のPartを検索する。

Update 100個のPartをランダムに選び、x、yの属性をランダムに更新する。ただし、更新対象は現在オブジェクトのみ

4.2 測定結果

測定は、ワークステーション Sun-4 SLC (主記憶8MBytes、SunOS 4.1.2)を用いて行なった。比較のために、POST/C++で履歴管理をした場合と履歴管理をしなかった場合について測定した。また、LookupとTraversalにいては、Cold、Warm、Hotについて測定した。Coldは、システムキャッシュがクリアされた状態、Warmは徐々ににキャッシュが効いてきている状態、Hotは同じオブジェクトアクセスを繰り返した場合である。現在オブジェクトが格納されるデータベースサイズは、Partオブジェクトのサイズが128bytesの場合は約2.5Mbytes、256bytesの場合は、約5Mbytesとなる。ただし、オブジェクトのサイズを変えた場合に結果の傾向は変らなかったので以下では、256 bytesの場合のみ示す。結果は、各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。

表 1　**Lookup** の測定結果

	履歴なし	履歴あり	
		(現在)	(履歴)
cold	41.22	44.47	69.74
warm	14.27 – 0.01	14.24 – 0.10	43.65 – 37.75
hot	0.01	0.01	38.78

表 1は Lookup の結果であるが、まず現在オブジェクトの検索について比較する。Cold の時は、ディスクからの読込みが大量に起こり実行時間がかかる。しかし、Warm で繰返して実行するとキャッシュ上に載っているオブジェクトへの検索があるのでページフォールトの回数が減少していき実行時間が短くなる。Hot では、オブジェクトが既にキャッシュ上に載っているためにディスクへのアクセスがないために実行時間が短い。履歴を取っていても現在オブジェクトのアクセスは、ほとんど影響を受けない。それに対して、履歴オブジェクトを検索した場合は、キャッシュの状態により実行時間が短くなっていくのは、現在オブジェクトの検索と同じ傾向を示すが、各検索において一定回数のディスクからの読み込みがあるために、実行時間はあまり短くはならない。これは、履歴オブジェクトの参照にいくと、現在オブジェクトも 1 度参照に行ってしまうことと、コミット毎に履歴オブジェクトを格納したページを解放しているので、再度の読み込みが必要になるからである。履歴オブジェクトのページを解放するのは、同じ履歴オブジェクトへのアクセスの頻度は現在オブジェクトに比べて低いことが予想されるのでコミット毎に解放するほうがメモリの効率的な利用になると考えたからである。

表 2　**Traversal** の測定結果

	履歴なし	履歴あり	
		(現在)	(履歴)
cold	30.35	47.72	96.64
warm	9.92 – 0.17	9.67 – 0.12	74.40 – 50.50
hot	0.04	0.12	50.67

表 2は Traversal の結果である。まず、現在オブジェクトの検索の場合に注目すると、Lookup に対して 3 倍ものオブジェクトをアクセスに行くのに実行時間が短くなっている。これは、今回作成したデータベースは、オブジェクトの id が近いと現在ファイル中でも近い位置に配置されるからである。つまり、現在オブジェクトに対しては Part 間の参照の局所性が、そのままファイル上での位置の局所性となっているためであると考えられる。これに対して、履歴ファイル上では、履歴オブジェクトの参照の局所性がページ内での参照の局所性に反映されないため、履歴オブジェクトの検索では、読出す履歴オブジェクトの数の増加が読み込むページ数の増加につながり実行時間を長くしている。

表 3　**Update** の測定結果

	履歴なし	履歴あり
open	0.22	0.26
updates	10.56	10.63
commit	10.21	13.12

表 3では更新トランザクションを、3 つのフェーズに分けて測定している。Update では履歴を管理するためのオーバヘッドはコミット時に集中している。今回、1 回のトランザクションで 100Part オブジェクトの更新をおこなっているが、更新時のオーバーヘッドは、コミット部で 30% 程度、全体で 15% 程度である。

5　結　論

時制永続オブジェクト管理システム POST/C++ の概要、基本設計、ならびに実装について述べた。また、システムの基本性能測定結果を示した。

今後の課題としては以下のものがある。

- 履歴オブジェクトへのアクセスが、ある程度時間がかかるが、より高速化するため、履歴オブジェクトをマップしたページを解放しないオプション、参照の局所性を反映させる履歴ファイルの構成法などを検討しなければならない。
- 更新を繰返すことにより、履歴ファイルと履歴索引の大きさは単調に増加していく。このファイル長の増大がどの程度性能に影響するか調べる必要がある。
- 現在の実装では同時実行制御を行なっていない。履歴管理を行なった場合の同時実行制御の方法について検討が必要である。

参考文献

[1] Gray, J. (ed.): *The Benchmark Handbook,* Morgan Kaufmann Publishers,Inc.,1993.

[2] Singhal, V., Kakkad, S.V. and Wilson, P.R.: Texas: An Efficient, Portable Persisten Object Store, *Proc. of 5th Int'l. Workshop of Persistent Object Systems,* 1992.

[3] 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.

[4] 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.