6D－4

時制永続オブジェクト管理システム POST/C++ の開発 — 基本機能の実装と性能評価 —

鈴木 孝幸 † 北川 博之 †† 林 恵将 ‡ 柳川 史彦 ‡

† 筑波大学 工学研究科 †† 筑波大学電子・情報工学系 ‡ 筑波大学 理工学研究科

E-mail : {suzu, kitagawa, rinn, yana}@dblab.is.tsukuba.ac.jp

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として、時間情報管理やオブジェクトの更新履歴管理の要求もある。これまで、主に関係データベースを基礎にした多くの研究が時制データベース [5] の研究分野で行なわれてきている。しかし、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である。これまでいくかのモデルの提案はあるが、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。

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については、[3][4] で述べた。2 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++ の実装について述べ、3 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 POST/C++ の実装

[3] 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++ の実装ついて述べる。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。より詳細については、[3][4] を参照されたい。

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。

- 履歴オブジェクトの情報を格納するための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
- 現在オブジェクトと履歴オブジェクトの関連をとる履歴索引の構築
- 履歴オブジェクトを格納する履歴ファイルの処理機構の構築
- コミット処理に履歴管理機能を拡張
- 履歴管理を行なうことによる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。

3 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。

3.1 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ベンチマークである。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。このデータベースの中には、全部で 2 万個の Part オブジェクトがあり、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。本研究では、OO1 を拡張したベンチマークで基本性能測定を行なった。操作としては、オブジェクトの検索、トラバース、挿入、履歴トラバースである。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[...];
}
```

本研究では、以下の処理の実行時間を測定する。Lookup と Traversal 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。また、接続を逆にたどることも行なう。

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。

Hisory Traversal 1 つの Part をランダムに選び、iterator を作成し、現在オブジェクトから履歴を順にたどり、生成時の履歴オブジェクトまでたどる。

Development of a Temporal Persistent Object Management System POST/C++ - Implemantation and Basic Performance Analysis -
Takayuki SUZUKI†, Hiroyuki KITAGAWA††, Yoshiyuki HAYASHI‡ and Fumihiko YANAGAWA‡
†Doctral Degree Program in Eng., Univ. of Tsukuba
††Institute of Info. Sci. and Elec., Univ. of Tsukuba
‡Master's Degree Program in Sci. and Eng., Univ. of Tsukuba

3.2 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。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 5M bytes となる。256bytes の場合の Lookup、Traversal、Update の測定結果については、[4] で報告した。以下では、128 bytes の場合についてのみ、Update、Travesal、History Traversal の結果について示す。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: Update の測定結果

更新回数	open	updates	commit
5000	0.28	8.28	11.19
10000	0.30	8.26	11.33
15000	0.29	8.22	11.28
20000	0.32	8.30	11.64
平均	0.29	8.27	11.59

表 1 は、Update の結果である。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。今回、更新トランザクションを 2 万回繰り返した。すなわち、各現在オブジェクトが平均 100 回更新されたことになる。更新トランザクションの回数が増加しても 1 回の更新トランザクションの実行時間はほとんど変化しない。

表 2: Traversal の測定結果

	cold	warm	hot
履歴なし	14.35	4.58 − 0.04	0.04
履歴あり (現在)	19.32	1.03 − 0.12	0.12
履歴あり (履歴)	317.95	305.49 − 275.14	260.59
履歴あり (逆・現在)	19.82	0.46 − 0.39	0.08

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を考える。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メモリ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実行時間が短くなる。Hot では、オブジェクトが既にメモリ上に載っているためにディスクアクセスがないために実行時間が短い。また、今回のデータベースは、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。したがって、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格納位置の局所性となっている。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。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。しかし、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ので、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り、実行時間はあまり短くはならない。また、接続を逆にたどる場合もほぼ同じ傾向を示す。

表 3: History Traversal の測定結果

iterator の作成	0.10
全履歴をたどった結果	6.10
1 履歴オブジェクトアクセスの平均	0.06
snapshot で 1 履歴オブジェクトをアクセス	0.23

表 3 は、History Traversal の結果である。比較のため、snapshot オペレータを利用してオブジェクト作成時の履歴オブジェクトにアクセスする時間を測定している。iterator の作成には、履歴索引上でクラスタリングの効果のため時間がかかっていない。しかし、オブジェクトの最初の履歴オブジェクトまでたどると今回の場合、平均 100 履歴オブジェクトたどることになるので、実行時間が長くなっている。それに対して、snapshot オペレータを利用した履歴オブジェクトへの直接的アクセスの時間は、iterator を用いたほぼ 4 世代分の履歴オブジェクトアクセスに必要な時間に相当する。

4 結論

時制永続オブジェクト管理システム POST/C++ の実装ならびに、基本性能測定について述べた。

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は、ある程度時間がかかってしまうが、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。
- 今回の実現では、複数トランザクションの同時実行が考慮されていない。同時実行を行なった環境での検討が必要である。

参考文献

[1] Cattell, R.G.G. and Skeen, J.: Object Operations Benchmark, *ACM Trans. Database Systems*, Vol.17, No.1(1992), pp. 1–31.

[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), pp. 57 − 654.

[4] 林, 鈴木, 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, 日本ソフトウエア科学会第 12 回大会予稿集, 1995.

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