メモリマップに基づく永続オブジェクト管理のためのトランザクション機構

6G－2

原 政博　山本 耕平　上原 敬太郎　宮澤 元　猪原 茂和　益田 隆司
東京大学 大学院 理学系研究科 情報科学専攻

1 序論

従来、トランザクション機構は銀行口座の入出金の管理や、データベースの一貫性を保持するために用いられてきた。近年、このようなビジネスアプリケーション分野に加え、グループウェアや協調作業といった新たなアプリケーション分野（以後、協調アプリケーションと呼ぶ）でも、トランザクション技術が用いられ始めている。協調アプリケーションでは、従来のトランザクションに比べて複雑な構造や参照関係をもつ二次記憶上のデータ（永続オブジェクト）を分散環境上で共有することが多い。そこで、アプリケーションがより自由にかつ効率よく分散永続オブジェクトにアクセスするために、ハードウェアのメモリ管理ユニット（MMU）が利用できる、ページサイズを１つのブロックとして管理するページサーバーアーキテクチャ[3]が考え出されている。同時に、永続オブジェクトを仮想記憶にマップするメモリマップに基づく永続オブジェクト管理システムが使われることが多い（O_2, EXODUS, ObjectStore, Eosなど）。

本稿では、ページサーバーアーキテクチャとメモリマップに基づく永続オブジェクト管理システム上で効率よく実装できる新たなトランザクションの並行性制御機構を提案する。また、提供するトランザクション機構のプロトタイプを用いて提案方式の実行効率についても述べる。

2 従来の並行性制御機構の問題点

現代の代表的なトランザクション機構にはTwo Phase Locking (2PL), Time-stamp Ordering (TO), Optimistic Method (OM) [2]などの並行性制御機構が使われている。しかし、これら従来の方法は、ページサーバーアーキテクチャ及びメモリマップに基づく永続オブジェクト管理システム上では十分効率よく動作しないという問題がある。

2PLではページ単位でロックをかけると、他のトランザクションのロックと衝突（Conflict）する可能性が極めて高くなってしまうため、並行性の低下が起こる。TOでは、ページ単位でトランザクションのタイムスタンプの管理を行うと、同時に多くのトランザクションが同一のページを読み書きしたときに、１つのトランザクションのアボートが他のトランザクションのアボートを引き起こしてしまう（Cascading abort）危険性が大きい。

また、OMではロックをかけずにデータを読み書きするため、他のトランザクションがコミットしたときに、不当なデータを読む可能性がある。つまり、本来なら見えてはいけないページを読み書きしてしまうことによって、不当な領域へのアクセスが起こり、アボートする前にトランザクションを実行中のプログラム全体が異常終了する可能性がある。

そこで我々は、OMにバージョン化された永続オブジェクト管理機構を組み合わせることによりページサーバアーキテクチャ及びメモリマップ環境上でのOMの持つ問題点を解消した。本システムでは新しく書き込まれたページは別の領域に新しいバージョンが与えられて書き込まれるために、書き込みの内容が読み出し専用のトランザクションには影響しない。そのため、頻度の高い読み出し専用のトランザクションがアボートすることがないとう利点が生じる。

3 バージョン付き Optimistic Method

本研究では、トランザクションシステムをトランザクションマネージャ（TM）、ストレージサーバ（SS）の組み合わせで実現している（図1）。

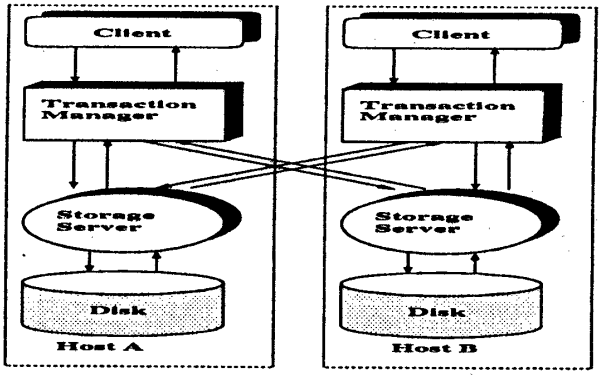

図 1: トランザクションシステムの概念

TMはクライアントとのインターフェイスとトランザクション機構を提供し、SSは二次記憶上の永続オブジェクトのバージョン管理をする。

OMでは、トランザクションの処理をRead-phase, Validation-phase, Write-phaseの３つのphaseに分けて管理している。

クライアントがトランザクションを開始すると、TMと通信を行って、仮想記憶上に永続オブジェクトのための領域を確保する。そして、永続オブジェクトにアクセスすると、TMは、オペレーティングシステムの仮想記憶機構を用いて永続オブジェクトの最初のアクセ

本研究の一部はIPAの独創的情報技術育成事業の支援を受けて行われています。

Transaction System for Memory-mapped Persistent Objects

M. Hara, K. Yamamoto, K. Uehara, H. Miyazawa, S. Inohara, and T. Masuda
The Department of Information Science, Graduate School of Science, the University of Tokyo

スを検知し、SS に必要なページを要求する。SS は最も適したバージョンのページデータを返し、TM がそれをクライアントの仮想記憶に読み込む。以降、永続オブジェクトへは普通の仮想記憶上のデータと同じようにアクセスできる。トランザクションを終了すると、バリデーションのチェックを行い、コミットならばディスク上に書き込む。

本研究では、ページにタイムスタンプを付加して、バージョン管理をしている。アルゴリズムは、以下の通りである。

Read-phase: (1) クライアントが初めて永続オブジェクトにアクセスすると、現在のタイムスタンプ (view-TS) がトランザクションに与えられる。(2) 永続オブジェクトのあるページに初めてアクセスすると、TM は SS に view-TS とともにページ要求を送る。(3) SS は view-TS より古いバージョンのうちで最も大きなタイムスタンプを持つページを返す。それを、TM はクライアントの仮想記憶に読み込む。(4) 永続オブジェクトのページに初めて書き込みが起こると、ページはコピーされ、全ての書き込みはそのコピーに対して行われる。

Validation-phase: (1) トランザクションに現在のタイムスタンプ (commit-TS) が与えられる。(2) 永続オブジェクトに書き込みがなければ、バリデーション成功とみなす。(3) 永続オブジェクトに書き込みがあれば、TM は view-TS と commit-TS とページの読み書き情報とともに“バリデーション開始”要求を各 SS に送る。このとき、書き込みが起こったページの内容も伴う。(4) SS は view-TS がページの最新バージョンのタイムスタンプよりも大きいかどうか調べる。もし、そのタイムスタンプが view-TS よりも大きければ、それは、別のトランザクションが既に書き込みを行ったことを示しているので、TM に“バリデーション失敗”メッセージを返す。そうでなければ、“バリデーション成功”メッセージを返す。(5) TM がトランザクションに関係ある全ての SS から“バリデーション成功”メッセージを得るとバリデーションは成功し、そうでなければ、バリデーションは失敗に終わる。

Write-phase: (1) バリデーションが成功ならば、TM は commit-TS とともに“コミット”メッセージを各 SS に送り、失敗なら“アボート”メッセージを送る。(2) SS が“コミット”メッセージを受け取ると、書き込みが起こったページは、commit-TS をタイムスタンプとする新しいバージョンとなり、全てのトランザクションに見えるようになる。(3) SS が“アボート”メッセージを受け取ると、書き込みが起こったページは捨てられる。

上記アルゴリズムの実装では、並行性制御に要する通信を減らし、ボトルネックを生じないようにする必要がある。そこで、本研究では各ホストのローカルクロックをタイムスタンプとして用い、必要に応じて分散したクロックの同期を行う方法も考えている [1]。

4 プロトタイプ版の性能測定

本研究では、3節で述べた機構を、DEC-station 5000 上の Mach 3.0 micro-kernel で実装した。TM はトランザクションの処理において、オブジェクトのマップ、ページフォルト、オブジェクトのアンマップ、バリデーションの 4つの処理を行う。ここでは、TM がページのマッピング、アンマッピングに要する時間、ページフォルトに要する時間を計測した (表 1)。

処理	時間 (μsec)
n ページ分の領域をメモリにマップ	$0.41n + 108$
カーネル内処理	1270
TM の処理	+420
1 ページの内容を仮想記憶に読み込む	1690

表 1: 処理に要する時間

n ページ分の領域をメモリにマップするのに要する時間は $(0.41n + 108)\mu$sec である。クライアントのアクセスでページフォルトが起こったときにページデータを仮想記憶に読み込むのにかかる時間は 1.69msec であり、そのうちカーネルが処理する時間は 1.27msec である。つまり、実質 TM が処理する時間は、わずか 420μsec に過ぎないのである。このトランザクション機構によるコストは、ごくわずかであるということがわかる。

5 結論

Optimistic Method にバージョンを付加した、このメモリマップに基づくトランザクション機構を用いると、他のトランザクションの影響による不当なアクセスや、最も多い読み出し専用のトランザクションのアボートがなくなるため、効率のよいトランザクションが行える。

また、現段階での問題点として、読み書きのある長いトランザクションは、読み書きのある短いトランザクションに先にコミットされて、アボートしやすいという傾向がある。そのため、飢餓状態 (starvation) が起こる可能性が高い。そこで、今後の課題としてトランザクションに優先順位をつけて、何回もアボートするようなトランザクションは優先順位を高くし、コミットしやすくするする手法を考えている。そして、トランザクションに時間の概念を導入して、より優先順位の高いトランザクションは、時間内にコミットできるような機構を考えたい。

参考文献

[1] Inohara, S., Y. Shigehata, K. Uehara, H. Miyazawa, K. Yamamoto, and T. Masuda, “Page-Based Optimistic Concurrency Control for Memory-Mapped Persistent Object Systems,” in *Proc. of the 28th Hawaii Int'l Conf. on System Sciences* vol. II, pp. 645–654, Jan. 1995.

[2] Kung, H. T., and J. T. Robinson, “On Optimistic Methods for Concurrency Control,” *ACM Trans. on Database Systems,* vol. 6, pp. 213–226, June 1981.

[3] Özsu, M. T., U.Dayel, and P. Valduriez, *Distributed Object Management.* San Metro: Morgan Kaufmann Publishers, 1994.