

データベースサーバ「わかし」上への 4 A a - 1 ログ, リカバリ, 2相コミットの実装

田村 慶一, 金子 邦彦*, 牧之内 顕文*

九州大学情報工学科, 九州大学大学院システム情報科学研究科*

1. はじめに

データベース分野において, トランザクション管理は重要なテーマである。本稿では, トランザクション障害及びシステム障害からのリカバリに焦点を当て, 新しいリカバリ方式を提案する。トランザクション障害では, トランザクションがコミットせずに終了(トランザクションが意図的に処理を中止する場合も含む)するため, トランザクションが実行されなかったのと同じ状態に戻す必要がある。システム障害では, システムがダウンし, 実メモリ上のデータベースキャッシュは破壊されるため, ディスク上のログから正しい状態に修復する必要がある。我々のリカバリ方式では OS のメモリマップドファイルの機能を利用し, データベースファイルを PRIVATE モードでマップする(シャドウキャッシュ)ことで, トランザクション障害に効果的に対処する。システム障害のために, 更新した部分を差分として取り出し, 更新ログを生成する。本リカバリ方式は, 従来の Redo/No-Undo リカバリよりも高速な redo-only リカバリを実現している。

2. メモリアーキテクチャ

我々のリカバリ方式では, 単一メモリによりデータベースが管理されていることを前提としている。すなわち, データベースファイルは実メモリ上にマップされ, データベースアクセスは, 普通のメモリアccessと同様に行われるものとする。言い換えると, 実メモリはデータベースキャッシュとして機能する。また, 遠隔データベースへのアクセスでは, 各サイトにデータベースキャッシュが作成されお互いにマップされているものとする。

i) シャドウキャッシュ

トランザクションによる更新は, 実メモリ上のデータベースキャッシュに対して行われ, データベースキャッシュのフラッシュはユーザから透過的に行われる。もし, フラッシュがトランザクションコミット前に行われるとしたら, トランザクション障害に備えてトランザクション開始時のデータを Before イメージとして保存せねばならない。従来のリカバ

リ方式の多くでは, 更新ごとに, どういった更新を行ったかの更新ログを作成し, データベースキャッシュのフラッシュの前に, 更新ログのフラッシュを行う(write ahead log)。この方式ではディスク上の更新ログとデータベースから Before イメージを再現できる。但し, 従来のログは更新ごとに更新ログ生成を行わなければならない。我々は, OS の mmap() を利用して before イメージを保持する。すなわちデータベースオープン時に, データベースファイルを mmap() の SHARED モードでマップすると同時に, PRIVATE モードでもマップする(シャドウキャッシュ)。トランザクションはシャドウキャッシュに対して更新を行う。シャドウキャッシュは(1)通常のメモリ空間と全く同じように扱え, (2)トランザクション終了前にはデータベースファイルへフラッシュが行われないことを保証する。また, (3)シャドウキャッシュの更新時にはリカバリマネージャによって自動的に Before イメージが作成される。正確には, あるページが最初にアクセスされたときのみそのページのバックアップが作成され, Before イメージとされる。また, ミラーサイトのデータベースキャッシュもシャドウキャッシュとして機能する。

ii) ログ

CT-Log と HT-Log の2種類のログを生成する。CT-Log はデータベースファイルが存在するサイトとクライアントが実行されるサイトの両方に作られる。CT-Log には, トランザクションの開始・コミット・アボートに関する情報が格納される。HT-Log は更新ログに相当し, データベースファイルのあるサイトのみに作られる。HT-Log は, Check-Point フラグ, HT-LogID, 当該データベースファイルを更新したトランザクションのID, 更新されたページ番号, 更新情報が格納される。

iii) コンパクトコミット

HT-Log は Before イメージとコミット時のシャドウキャッシュの差分から作成される。HT-Log の1レコードは1ページ分の更新結果に対応する。以上のように, 個々の更新ごとに更新ログを作成するのではなくコミット時にのみ更新ログを作成する。これは更新ログ処理コストを軽減する。例えば, トランザクションが100ページを平均100回書き換えたとして, 更新ごとにログを取れば10000レコードの更新

Implementation of logging, recovery and two-phase-commitment on a Database Server Wakashi

Keiichi Tamura, Kunihiko Kaneko*, Akifumi Makinouchi*
Information Engineering, Kyushu University Graduate School
of Information Science and Electrical Engineering, Kyushu University*

ログが必要であるが、本方式では100レコード以下の更新ログで済むことになる。我々は本方式をコンバクトコミットと呼ぶ。

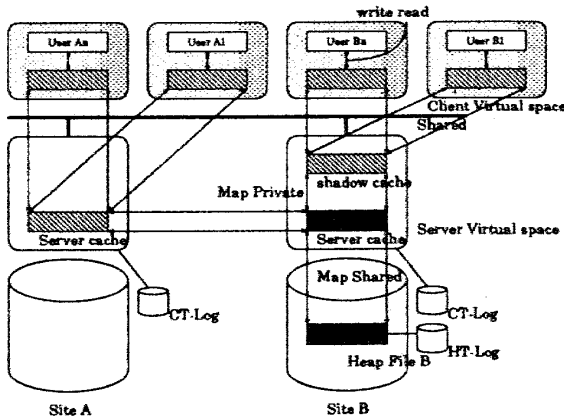


図 1: シャドウキャッシュ

3. コミット時の処理

- i) データベースファイルの存在するサイト
トランザクションのコミットでは、まずトランザクションが変更したページごとにシャドウキャッシュとBeforeイメージの比較を行い、差分を取り更新ログを生成する。この更新ログはHT-Logとしてディスクにフラッシュされる。その後、CT-LogにCommitレコードを書き込む。この時点でトランザクションはコミットされたことになる。最後に、データベースキャッシュの更新を行う。これはSharedモードでマップされたメモリ空間を更新することで行う。このことは更新ログが書き込まれた後にはじめてデータベースが更新されることを意味する。
- ii) 遠隔アクセス
遠隔アクセスの場合のコミットは、2相コミットで行う。HT-Logはトランザクション実行サイトで作られ、主サイトに転送される。このことで、CPUコストは増えるがネットワークコストを削減できる。2相コミットはVote-Commit, Do-Commitの2つの相からなる。Vote-CommitでHT-Logが主サイトに送られ、Do-Commit時にHT-Logをもとに主サイトでデータベースキャッシュの更新を行う。

4. サイトクラッシュとリカバリ

サイトクラッシュが発生したとき、実メモリ上のデータは失われるがCT-Log, HT-Logはディスクに保存されているので、これらのログをもとにリカバリが可能である。CT-LogとHT-Logを使用したリカバリの流れを以下に示す。

- i) CT-Logを走査し最新のCheck-Pointレコードを探す。
(チェックポイントについては5章で説明する)

- ii) チェックポイントからCT-Logの最終レコードまで走査し、トランザクションのCommitレコードとAbortレコードを探す。

(1) もし、Commitレコードなら、トランザクションIDに対応するHT-Logを走査し、HT-Logごとに用意したリカバリリストにページ番号とHT-Log識別子を入れていく。但し、Check-Pointフラグが立っているエントリは飛ばす。

(2) もし、Abortレコードなら、トランザクションIDに対応するHT-Logを走査し当該エントリを消していく。但し、Check-Pointフラグが立っているエントリは飛ばす。

- iii) リカバリリストをページ番号、HT-Log識別子の順に昇順にソートし、各ページ番号のHT-Log識別子がもっとも新しいものだけを再実行する。

5. チェックポイント

我々は、データベースファイルへのフラッシュをトランザクションコミット時には行わず、チェックポイントでフラッシュを行う。トランザクションがすべてコミットかアポートするのを待ってチェックポイントを行うわけではない(キャッシュコンシステントチェックポイント)。2章で示したように更新はシャドウキャッシュのみに行われる。そのために、チェックポイント処理中は、新たなトランザクションのアポート、コミットを禁止する。次にチェックポイント時の処理の流れを示す。

- i) 新たなトランザクションのアポート、コミットを禁止する。
- ii) CT-Logを最後のCheck Pointレコードがあるところまで走査し、Abortレコードをみつけたら、そのトランザクションIDの当該HT-Logを消去する。Commitレコードをみつけたら、そのトランザクションIDの当該HT-LogレコードのCheck-Pointフラグを立てる。
- iii) データベースキャッシュをフラッシュする。
- iv) CT-LogにCheck-Pointレコードを書き込む。

6. 終りに

シャドウキャッシュを用いた新しいリカバリ方式は、従来のRedo/No-Undoリカバリ方式に比べると、より高速なリカバリが可能となっている。この新しいリカバリ方式を我々の研究室で研究開発中のデータベースサーバ「わかし」に実装していきたい。

参考文献

- [1] Yu, Ge., A. Makinouchi, "Transaction Management for a Distributed Object Storage System WAKASHI - Design, Implementation and Performance", Proc. of 12th Int'l Conf. on Data Engineering, New Orleans, USA, February 1996
- [2] Franklin, M.J., et al, "Crash recovery in client-server EXODUS." Proc. of SIGMOD, San Diego, June 1992. pp.165-174.