Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Dr. G. Alonso *C. Hagen, B. Kemme*

AR 1: Recovery

Theoretische Übung 3 — Beispiellösung

Aufgabe 1: Undo/Redo

Wir betrachten einen Recovery-Algorithmus, der UNDO und REDO durchführt, aber die Datenobjekte nicht mit der *log sequence number* (LSN) des Log-Eintrags der letzten Änderung markiert. Um Log-Speicherplatz zu sparen, werden nur *after-images* aufgezeichnet. Es wird *commit consistent* Checkpointing verwendet.

Definieren Sie eine Datenstruktur für das Log (Update-Records, Checkpoint-Records, etc.). Beschreiben Sie die Algorithmen für die Standard-Recovery-Prozeduren *Write, Read, Commit, Abort, und Restart* sowie für das Checkpointing. Beschreiben Sie insbesondere, wie diese Prozeduren die Log-Struktur verwenden.

Welche Probleme treten auf, wenn *cache consistent* Checkpointing verwendet wird?

Lösung: Die Besonderheit dieses Algorithmus ist, daß er zum einen keine Before Images speichert (dies erschwert den UNDO-Algorithmus), und zum anderen keine LSN-Einträge in Datenobjekten ablegt.

Die folgenden Typen von Log-Records werden benötigt:

1. Update: *Für jede Schreiboperation werden die folgenden Informationen ins Log geschrieben (Update Record):*
   - *Transaktionsname*
   - *Name oder Adresse des geschriebenen Objekts*
   - *Neuer Wert des Objekts (after image)*
   - *Zeiger auf den zuletzt geschriebenen Update Record derselben Transaktion*
2. Commit: *Der Commit-Record enthält lediglich den Namen der Transaktion.*
3. Abort: *Der Abort-Record enthält lediglich den Namen der abgebrochenen Transaktion.*
4. Checkpoint: *Da wir* commit consistent *Checkpointing verwenden, müssen wir nur die Tatsache festhalten, daß der Checkpoint stattgefunden hat. Weitere Informationen werden nicht gespeichert.*

Wir betrachten nun die Operationen, die der Resource Manager durchzuführen hat. Die Operationen RM-Read, RM-Write *und* RM-Commit *unterscheiden sich nicht vom Standard-Algorithmus. Deswegen geben wir hier nur die anderen zwei Operationen an:*

1. RM-Abort(T):
   (a) *Finde den neuesten Update Record von T im Log.*

   (b) *Durchwandere das Log rückwärts, indem Du den Zeigern in den Update Records folgst. Für jeden Record [T,x,v] durchsuche das Log rückwärts, um den letzten Update Record für x, der von einer anderen Transaktion stammt, zu finden. Falls Striktes 2-Phase-Locking verwendet wird, ist das After Image dieses Records "Before Image" für die abzubrechende Operation. Schreibe dieses Image in den Cache.*
   (c) *Füge t in die Abort List ein.*
   (d) *Melde das Abort zurück und entferne T aus der Active List.*
2. Restart: *Der restart-Algorithmus hat zwei wesentliche Unterschiede zu dem in [BHG87] (Seite 188) vorgestellten:*
   - *Es werden keine LSNs in Datenobjekten gespeichert. Um den letzten gültigen Wert im Falle eines UNDO zu finden, müssen wir deshalb das Log durchsuchen.*
   - *Da wir* Commit Consistent *Checkpointing verwenden, wissen wir, dass Redo keine Informationen benötigt, die älter sind als der letzte Checkpoint. Für UNDO ist es jedoch notwendig, das Log auch vor dem letzten Checkpoint zu durchsuchen, da wir keine Before Images speichern.*

Der Algorithmus durchläuft das Log rückwärts und verwaltet dabei drei Listen: TA_UNDO enthält die abzubrechenden Transaktionen, TA_REDO enthält die committeten Transaktionen, und REPLACE enthält die Datenobjekte, deren Werte wiederhergestellt werden müssen.

Wir durchlaufen das Log rückwärts bis zum neuesten Checkpoint Record. Für jeden Logeintrag führen wir dabei die folgenden Aktionen durch, abhängig vom Typ des Eintrags:

- *Ein Commit Record für Transaction T: Hinzufügen von T zur TA_REDO-Liste.*
- *Ein Abort Record für Transaction T: Hinzufügen von T zur TA_UNDO-Liste.*
- *Ein Update-Record für Transaction T und Datenobjekt x: Falls T in der TA_UNDO-Liste ist, füge x in die REPLACE-Liste ein (falls es nicht schon drin ist). Falls T nicht in der UNDO-Liste ist und x ist in der REPLACE-Liste, dann schreibe das After Image, das im Log Record enthalten ist, in den Cache und entferne x aus der REPLACE-Liste.*

Nachdem der neueste Checkpoint-Eintrag erreicht ist, durchlaufen wir das Log weiter rückwärts, bis die REPLACE-Liste leer ist. Für jeden Update-Eintrag, der zu einem Objekt in REPLACE gehört und von einer comitteten Transaktion geschrieben wurde, schreiben wir das After-Image in den Cache und entfernen x aus der REPLACE-Liste.

Sobald die REPLACE-Liste leer ist, starten wir mit der REDO-Phase. Beginnend beim neuesten Checkpoint-Record, durchlaufen wir das Log vorwärts. Für jeden Update-Record, der zu einer Transaktion in TA_REDO gehört, schreiben wir das After Image in den Cache.

Bei der Verwendung von Cache Consistent *Checkpointing verändern sich die Algorithmen insofern, als es Transaktionen geben kann, die vor dem letzten Checkpoint begonnen haben, aber nachher abgebrochen wurden. UNDO muss also auch Operationen berücksichtigen, die vor dem letzten Checkpoint stattfanden. Dies gilt jedoch nur für diejenigen Transaktionen, die erst nach diesem Checkpoint beendet wurden.*

Aufgabe 2: Versionen

Im No-Undo/Redo-Algorithmus ist es möglich, *shadowing* zu verwenden, um es einer Transaktion zu ermöglichen, die Daten zu lesen, die sie selbst geändert hat: Wenn eine Transaktion T_i das Objekt x zum ersten mal schreibt, wird eine neue Version von x angelegt und in den Cache geschrieben. Die vorherige (Schatten-) Version von x wird nicht überschrieben.

Arbeiten Sie die Details dieses Ansatzes aus, indem sie Datenstrukturen und Algorithmen für die Recovery-Prozeduren (Write, Read, Commit, Abort, Restart) angeben.

Lösung: Wir skizzieren eine sehr einfache Lösung des Problems. Optimierungen sind in vielerlei Hinsicht möglich. Unser Algorithmus verwendet ein Schattenverzeichnis (shadow directory, SD(T)) *für jede Transaktion, das für jede geschriebene Seite die Adresse der entsprechenden Schattenseite enthält. Dieses Verzeichnis kann im Hauptspeicher gehalten werden. Wir gehen davon aus, daß die Speicherverwaltung Operationen zum Anfordern und Freigeben von Schattenseiten anbietet.*

Die Resource-Manager-Operationen sehen wie folgt aus:

1. RM-Write(T,x,v): *Falls es einen Eintrag (x,y) in SD(T) gibt, schreibe v in y. Andernfalls fordere eine neue Schattenseite z an, schreibe v in z, und füge (x,z) in SD(T) ein. Writes auf Schattenseiten müssen nicht geloggt werden.*
2. RM-Read(T,x): *Falls es einen Eintrag (x,y) in SD(T) gibt, lese y. Andernfalls lese x.*
3. RM-Abort(T): *Freigabe aller Schattenseiten von T und Löschen von SD(T).*
4. RM-Commit(T): *Für jeden Eintrag (x,y) in SD(T): Schreibe den Wert von y in x. Nach dieser Kopierphase wird der Commit-Record geschrieben. Anschliessend werden alle Schattenseiten freigegeben und SD(T) gelöscht. Die Writes auf dem original-Seiten werden geloggt.*
5. Restart: *Es ist lediglich REDO notwendig. Dies wird in der üblichen Weise durchgeführt (Durchlaufen des logs und Installation von After Images).*

Zusätzlch benötigt dieser Algorithmus einen Garbage-Collection-Mechanismus für Schattenseiten: Falls das System abstürzt, geht die Information über Schattenseiten aller gerade aktiven Transaktionen verloren. Möglich ist dies z.B. durch das Speichern der TransaktionsID in den Schattenseiten. Ein Garbage-Collection-Mechanismus kann dann alle Schattenseiten von nicht mehr aktiven Transaktionen periodisch freigeben.