ARIES

(*Algorithm for Recovery and Isolation Exploiting Semantics*)

algorytm odtwarzania i semantyka wykorzystania izolacji

prosta i efektywna metoda odtwarzania transakcji wspierająca blokady o wysokiej granularności i częściowe rollbacki z użyciem rejestrowania zapisów z wyprzedzeniem (WAL – *Write-Ahead Logging*)

Beata Golichowska & Maja Perycz

Przed czym chcemy się bronić?

Podstawowe rodzaje awarii to:

- Błąd transakcji
- Awaria systemu
- Uszkodzenie nośnika

ACID – Atomowość, Spójność, Izolacja, Trwałość

Kryteria oceniania rozwiązań:

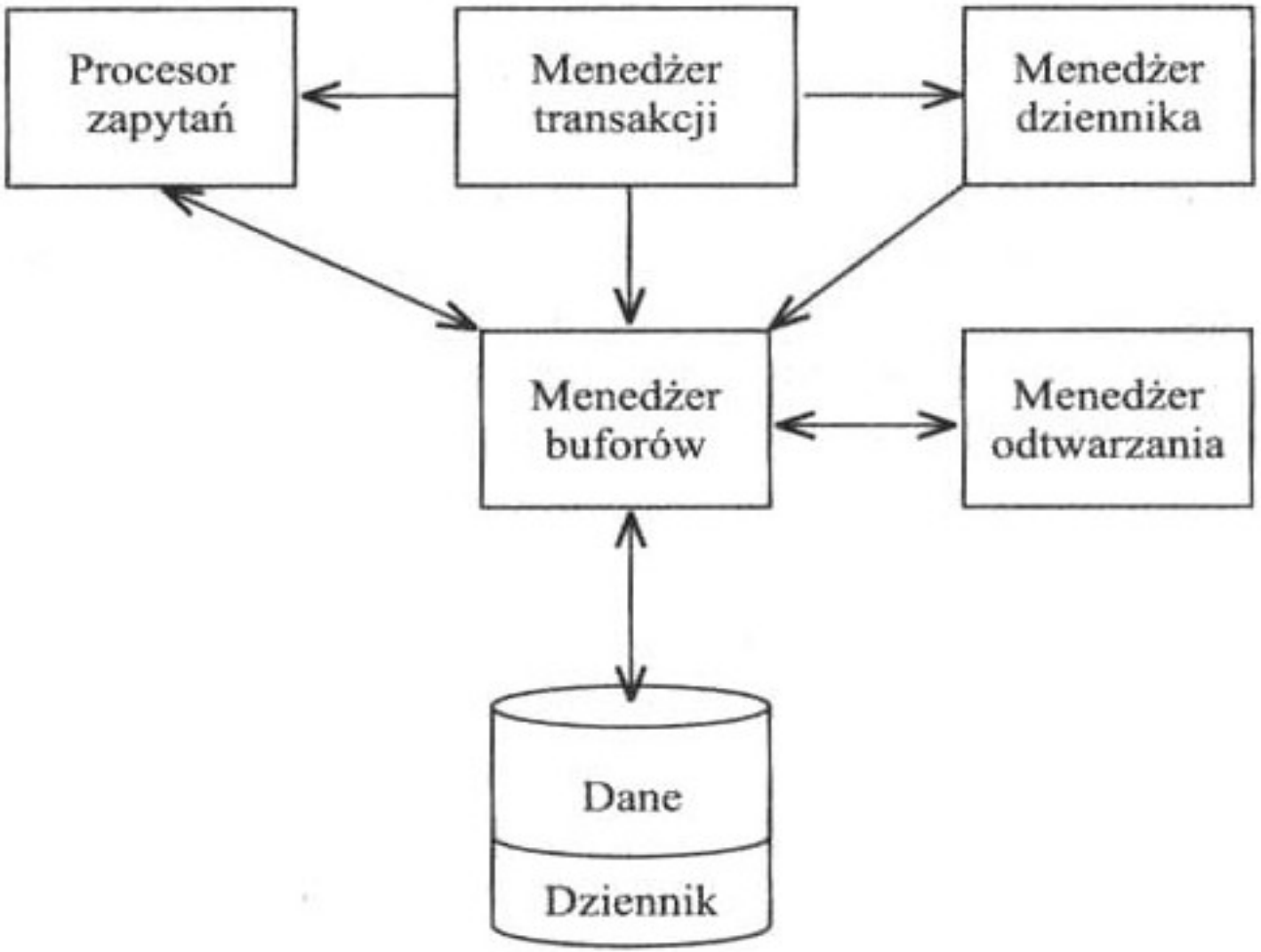
- ✓ stopień współbieżności
- ✓ złożoność logiki
- ✓ narzut na pamięć ulotną i nieulotną przy zapisie danych i logów
- ✓ narzut na wejście/ wyjście (synchroniczne i asynchroniczne)
- ✓ wspierane funkcjonalności
- ✓ ilość przetwarzanych danych przy restarcie po awarii (i stopień współbieżności)
- ✓ zasięg wywoływanych przez system rollbacków spowodowanych zakleszczeniami lub ograniczeniami nałożonymi na dane
- ✓ wspieranie nowatorskich trybów zakładania blokad (współbieżność korzystająca z przemienności i innych właściwościach operacji takich jak inc/dec realizowanych na tych samych danych przez różne transakcje)
- ✓ wykorzystywanie szczególnych właściwości niektórych operacji w celu zwiększenia współbieżności
- ✓ itd…

WAL – Write-Ahead Logging

- rekordy dziennika opisujące zmiany związane z pewnymi danymi muszą trafić do pamięci stabilnej zanim te dane będą mogły zostać nadpisane na dysku
- system nie może zapisać zmodyfikowanej strony na dysk zanim przynajmniej informacje undo nie zostaną umieszczone w pamięci trwałej
- by umożliwić wymuszenie tego protokołu, na każdej stronie zapisywany jest LSN rekordu dziennika opisującego ostatnią modyfikację tej strony

Systemy wykorzystujące ARIES

- DB2 Universal Database for z/OS and OS/390; DB2 Universal Database for Linux, Windows, AIX, Solaris, HP-UX, NUMA-Q, OS/2; DB2 Server for VM and VSE
- Cloudscape
- Lotus Domino
- MQSeries persistent messaging and queuing system
- Tivoli Storage Manager (TSM)
- Encina recoverable file system
- VM Shared File System
- Starburst extensible DBMS
- QuickSilver distributed operating system
- Microsoft SQL Server and NT file system
- Sybase ASE 11.9.2
- Clustra DBMS
- University of Wisconsin's Gamma Database Machine; Shore persistent object system; EXODUS extensible DBMS; Paradise GIS system; Minirel DBMS; Minibase DBMS
- Cornell University's PREDATOR object-relational DBMS; Mars ARIES simulator; Seoul National University's; Soprano client-server object storage system; SNU RDBMS Platform (SRP)
- University of Pisa's Java Relational System (JRS)
- University of Melbourne's Aditi deductive database system (see also)
    - Australian National University's Platypus Object Store
        - Johns Hopkins University's ARIES Project


Procesor zapytań
Menedżer transakcji
Menedżer dziennika
Menedżer buforów
Menedżer odtwarzania
Dane
Dziennik

(rys. Ullmann)

Pojęcia wstępne

- koncepcja dziennika
- blokady a zatrzaski
- blokady o wysokiej granularności
- zarządzanie buforami
- logiczne a fizyczne undo /redo

Koncepcja dziennika

- zapewnienie bezpieczeństwa poprzez zapis postępu transakcji i dokonanych przez nią modyfikacji
- dziennik stanowi źródło pewności, czy zatwierdzone działania transakcji są odzwierciedlone w bazie mimo awarii, a te niezatwierdzone zostały wycofane
- jego rekordy stanowią też się źródło rekonstrukcji uszkodzonych bądź straconych danych
- każdy rekord oznaczony jest unikalnym numerem LSN
- koncepcyjnie stanowi nieskończenie rozszerzający się plik sekwencyjny
- nieulotna wersja dziennika jest przechowywana na dysku
  - archiwizowane rekordy mogą zostać usunięte gdy tylko przestaną być potrzebne, tzn. odpowiednie kopie zapasowe bazy danych zostaną utworzone

Koncepcja dziennika – cd.

- każdy rekord trafia najpierw do pamięci wirtualnej buforów dziennika
- zrzucanie dziennika do danego numeru LSN – wymuszenie sekwencyjnego zapisu jego rekordów na dysk w związku z czynnościami transakcji i zarządcy buforów, lub okresowo, w tle, w celach porządkowych
- 3 rodzaje rekordów: undo-redo, undo-only i redo-only
- informacja undo/redo w rekordzie mówi, jak wycofać/powtórzyć zmiany dokonane przez transakcję
  - w zależności od przeprowadzanej operacji może być zapisywana fizycznie lub operacyjnie
- w dzienniku przechowywany jest także status transakcji i żadna z nich nie może być traktowana jako całość zanim nie zostanie zatwierdzona a dotyczące jej wpisy w dzienniku nie trafią na dysk
  - oznacza to, że transakcji nie wolno zakończyć zatwierdzania się dopóki informacje redo w jej rekordach dziennika nie zostaną zapisane do pamięci stabilnej

Blokady a zatrzaski

- są zazwyczaj do kontroli dostępu do pamięci dzielonej

cecha	zatrzaski	blokady
Gwarantują zazwyczaj:	fizyczną spójność danych	logiczną spójność danych
Zakładane są na czas:	krótszy	dłuższy
Detektor zakleszczeń a zawieszone oczekiwania:	detektor nie jest informowany	detektor jest informowany
Koszt zakładania i zwalniania:	niższy	wyższy
Alokacja pamięci na informacje kontrolne:	statyczna	dynamiczna
Ilość obiektów, na które można je nakładać:	mniejsza	większa

- całość informacji związanych z blokadami przechowywana jest w centralnej tablicy haszującej

- zwykle przy alokacji bloku kontrolnego blokady należy użyć zatrzasków (wiele transakcji może jednocześnie korzystać z tablicy blokad)

Blokady a zatrzaski – cd.

Podstawowe tryby zakładania blokad i zastrzasków: S (shared), X (exclusive), IN (intension exclusive), IS (intention shared), SIX (shared intension exclusive)

blokady i zatrzaski na danym obiekcie mogą być utrzymywane przez wiele transakcji jednocześnie tylko, jeśli ich tryby są kompatybilne

Różne poziomy granularności, np. rekord, tabela (relacja) czy plik (tablespace).

- zamykanie hierarchiczne – tryby 'z zamiarem' stosowane są dla wyższych poziomów hierarchii, a prostsze dla niższych

  zwolnienie blokady trybu prostego bezwarunkowo zwalnia blokady na podobiektach, a w przypadku trybów 'z zamiarem' daje im jedynie prawo żądania zwolnienia blokad

- możliwość pobierania warunkowego i bezwarunkowego oraz utrzymywania przez różny okres czasu:

  - blokady chwilowe – ich bezwarunkowe żądanie daje zawsze pozytywną odpowiedź, ewentualnie opóźnioną do czasu zwolnienia się blokady
    - blokady ręczne – zwalniane po jakimś czasie od pobrania, zwykle długo przed zakończeniem transakcji
      - blokady typu 'commit' – zwalniane dopiero po zakończeniu transakcji

Blokady o wysokiej granularności

- wzrost znaczenia wspierania wysokiego poziomu współbieżności i popularyzacja systemów zorientowanych obiektowo

- konieczność wymyślenia sposobów kontroli współbieżności i metod odtwarzania, które poradzą sobie z semantyką operacji na danych i będą wydajnie wspierać zamki o wysokiej granularności

- w zorientowanym obiektowo logicznym obrazie bazy danych strona jest mało naturalną jednostką dla blokad

  jeśli jednostką zamykania jest strona, a czas utrzymywania blokad długi, czas oczekiwania na zwolnienie blokady i prawdopodobieństwo zakleszczenia rosną.

- popularyzacja systemów relacyjnych – konieczność ułatwienia zarządzania pamięcią i definiowania danych

  - przy indeksowaniu potrzebna jest elastyczna metoda odtwarzania wspierająca wysoki poziom współbieżności, a blokady na poziomie stron są nie do przyjęcia

Zarządzanie buforami

- zarządca buforów zarządza pulą buforów i związanymi z tym operacjami we/wy
- operacja *fix* – uzyskanie dostępu do strony (być może sprowadzenie jej z dysku, zwiększenie licznika odwołań)
- po wykonaniu operacji *fix* na stronie, odpowiadający jej bufor nie może być wykorzystany przez inną stronę dopóki nie zostanie wykonana operacja *unfix.*
- dla każdej strony zarządca przechowuje licznik odwołań (fix count)
- operacja *fix* jest też wykorzystywania do komunikowania intencji modyfikacji strony
- brudna strona nie mogą być kopiowana na dysk jak długo fix count z intencją modyfikacji jest większy od zera (strona czytana może być modyfikowana w tle)
- zarządca buforów systematycznie kopiuje brudne strony na dysk z zachowaniem protokołu WAL
- zarządca buforów dostarcza także wsparcie dla zatrzasków stron (zatrzaski tak na prawdę na buforze zawierającym daną stronę)

Zarządzanie buforami – cd

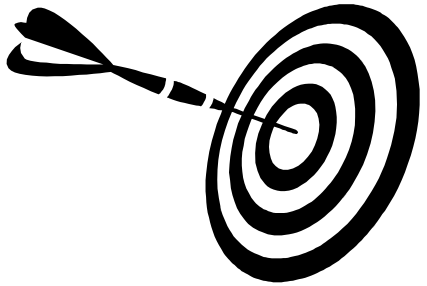
Popularne strategie zarządzania buforami:

- steal – dopuszcza się, żeby strona zmodyfikowana przez transakcję została zapisana na trwały nośnik zanim transakcja zostanie zacommitowana
- no-steal (przeciwieństwo steal)
- force – przy commitowaniu wymuszasza się zapisanie wszystkich zmodyfikowanych stron na trwały nośnik (nie trzeba przeprowadzać powtórzeń dla zatwierdzonych transakcji)
- no-force (przeciwieństwo force)
- deferred updating – modyfikacje danych są wykonywane (nawet w pam.operacyjnej) dopiero gdy wiadomo, że transakcja zostanie zatwierdzona; do tego czasu przechowywana jest lista operacji oczekujących na wykonanie (skutek: transakcja nie widzi własnych modyfikacji)

Logiczne a fizyczne undo/redo

- redo zorientowane stronowo opisuje modyfikację poszczególnych stron
- w systemach z redo logicznym powtarzanie wymaga odwołania się do kilku deskryptorów i stron bazy danych
  - zmiany w indeksach nie są logowane osobno tylko powtarzane przy użyciu rekordów dziennika dla stron z danymi
- możliwość przeprowadzania stronowo zorientowanych redo pozwala systemowi zapewnić niezależność odtwarzania pomiędzy obiektami
- możliwość przeprowadzania logicznych undo zwiększa poziom współbieżności
  - undo logiczne (z odpowiednimi protokołami kontroli współbieżności) pozwala na przenoszenie niezatwierdzonych zmian jednej transakcji na inną stronę przez inną transakcję
- stronowo zorientowane redo i undo pozwalają na szybsze odtwarzanie, jeśli nie ma odwołań do innych stron bazy danych niż te wspomniane w rekordach dziennika.
- ARIES wspiera redo zorientowane stronowo w celu zwiększenia wydajności, a undo logiczne w celu zwiększenia współbieżności.

Cele



- ✓Prostota
- ✓Logowanie operacji
- ✓Elastyczne zarządzanie pamięcią
- ✓Częściowe rollbacki
- ✓Elastyczne zarządzanie buforami
- ✓Niezależność odtwarzania
- ✓Logiczne undo
- ✓Równoległość i szybkie odtwarzanie
- ✓Minimalne koszta

Prostota

- współbieżność i odtwarzanie to kwestie złożone na tle innych aspektów zarządzania danymi
- skomplikowane algorytmy często powodują błędy
- dążymy do uzyskania algorytmu prostego, acz potężnego i elastycznego

Logowanie operacji

- bogate semantycznie tryby zakładania blokad
  - jedna transakcja może modyfikować dane zmienione wcześniej przez inną, jeszcze nie zatwierdzoną, gdy ich akcje są kompatybilne semantycznie (np. inc/dec)
- albo logowanie wartości i stanów albo logowanie operacji
  - odnosi się to też do systemów stosujących fizyczne, zorientowane bajtowo, logowanie wszystkich zmian na stronie
- konieczność precyzyjnego śledzenia stanu strony w stosunku do dziennika
  - pozwala to uniknąć prób zbędnego powtarzania i wycofywania operacji
- konieczność logowania zmian dokonanych podczas rollbacków
  - przy wycofywaniu transakcji, która wcześniej modyfikowała stronę, potrzebujemy informacji o naniesionych zmianach i postępie trwającego rollbacku
- oszczędność przestrzeni dziennika – możliwość logowania logicznego
  - tzn. takiego, że nie każda zmiana strony musi być logowana explicite; przykładem są zmiany informacji kontrolnych, jak ilość wolnego miejsca na stronie
    - możliwość logicznego przeprowadzania undo i redo

Elastyczne zarządzanie pamięcią

- wydajne wsparcie dla pamięci i operowania danymi zmiennej długości
- możliwość uniknięcia potrzeby wstrzymywania systemu w celu reorganizacji, odśmiecania pamięci i odzyskiwania każdego wolnego fragmentu
- logiczna natura logowania i zakładania blokad
  - przesuwanie danych w obrębie strony w ramach odśmiecania pamięci nie powoduje zakładania blokad na dane ani logowania tych przesunięć
- logiczna natura wycofywania – ważne dla indeksów
  - dana transakcja musi być w stanie podzielić stronę-liścia, nawet jeśli zawiera ona niezatwierdzone dane wstawione przez inną transakcję, co może prowadzić do problemów przy przeprowadzaniu stronowo zorientowanych wycofań z użyciem dziennika
    - transakcja, która zwolniła trochę pamięci, powinna móc użyć jej ponownie przy ewentualnej późniejszej operacji insert

Częściowe rollbacki

- koncepcja savepointów i wycofywania się do nich
- przyjazny użytkownikom brak wymogu wycofania całej transakcji
- radzenie sobie z naruszaniem integralności danych i problemami wynikającymi z przestarzałych sposobów przechowywania informacji

Elastyczne zarządzanie buforami

- jak najmniej założeń ograniczających w praktyce politykę zarządzania buforami, a zarazem umiejętność dostosowania się do każdej z nich
- elastyczność powodująca zwiększenie współbieżności, zmniejszenie liczby wyjść/wejść i wydajne użycie pamięci buforowej
- zależność skomplikowania czynności podejmowanych podczas odtwarzania przy restarcie po awarii lub przy odzyskiwaniu danych z dysku od przyjętej strategii zarządzania buforami
- polityka inna niż steal powoduje dodatkowy koszt związany ze śledzeniem, czy strona zawiera niezatwierdzone modyfikacje
- logowanie wycofywania i modyfikacje w miejscu.
- logiczna natura logowania i zakładania blokad
  - przesuwanie danych w obrębie strony w ramach odśmiecania pamięci nie powoduje zakładania blokad na dane ani logowania tych przesunięć

Niezależność odtwarzania

- możliwość tworzenia kopii zapasowych i odtwarzania danych z dysku lub pamięci przy restarcie na niższym poziomie granularności, niż cała baza

- odzyskiwanie jednego obiektu nie wymusza odzyskiwania innego

- podczas równoległego odtwarzania powiązanych ze sobą obiektów przy restarcie istnieje niebezpieczeństwo niepoprawnej synchronizacji deskryptorów

- możliwość odtwarzania wybiórczego i opóźniania odtwarzania niektórych obiektów do pewnego momentu

- odtwarzanie zorientowane stronowo – możliwość odtworzenia pojedynczej, uszkodzonej strony

  łącznie z zapisem CLRów upraszcza to znacznie odzyskiwanie danych z dysku i pozwala na tworzenie niezależnych kopii zapasowych różnych obiektów z różną częstotliwością

Logiczne undo

- podczas wycofywania umożliwia zmianę strony, która jest inna od tej zmodyfikowanej podczas działania w przód, co jest potrzebne w kontekście podzielenia przez jedną transakcję indeksowanej strony, która zawiera niezatwierdzone dane innej transakcji

- pozwala na wspieranie współbieżności na wyższym poziomie, zwłaszcza w strukturach wyszukiwania

- jeśli podczas działania wstecz nie odbywałoby się logowanie, undo logiczne byłoby bardzo trudne do zastosowania przy potrzebie niezależności odtwarzania i odtwarzania zorientowanego stronowo

Równoległość i szybkie odtwarzanie

- wykorzystywanie równoległości w różnych etapach odtwarzania i odzyskiwaniu danych z dysku w związku z upowszechnianiem się wieloprocesorów
- szybkość samego odtwarzania - podejście hot-standby (istnienie zapasowej bazy, na którą przełącza się system transakcji w razie awarii tej głównej)
- zorientowanie stronowe powtarzania i wycofywania zmian
- pozwolenie odtwarzanemu systemowi na przetwarzanie nowych transakcji, nawet przed zakończeniem wycofywania przerwanych

Minimalne koszta

- wysoka wydajność podczas normalnego działania i odtwarzania
- minimalny narzut na pamięci ulotne i nieulotne
- minimalizacja liczby stron modyfikowanych podczas restartu
  - zmniejszenie ilości stron, które muszą zostać wymiecione na dysk, oraz narzutu na CPU
- brak wycofywania i powtarzania zatwierdzonych zmian i niepotrzebnego wycofywania nieobecnych
- brak zakleszczeń wycofujących się transakcji
  - zapis CLRów w sytuacji wystąpienia zagnieżdżonych rollbacków lub powtarzających się awarii przy wycofywaniu nie powien powodować pojawienia się nieskończonej ilości rekordów w dzienniku
  - możliwość tworzenia checkpointów i kopii zapasowych bez wyciszania istotnych czynności systemu

Struktury danych

- Rekordy dziennika
- Struktura strony
- Tablica transakcji
- Tablica dirty_pages

Rekordy dziennika

- **LSN** (Log Sequential Number) – adres pierwszego bajtu rekordu dziennika w jego przestrzeni adresowej
- **Type** – określa typ rekordu, np.: ‘compensation’, ‘update’, ‘prepare’, ’begin’, ’end’, ’commit’, ’rollback’, rekordy nietransakcyjne (np. ‘OSfile_return’)
- **TransID** – identyfikator transakcji dokonującej wpisu.
- **PrevLSN** – LSN poprzedzającego rekordu dziennika wpisanego przez tą samą
- **PageID** – (tylko w rekordach typu ‘update’ i ‘compensation’) – wskazuje na stronę, która została zmodyfikowana w wyniku logowanej operacji
  - złożony z 2 części: objectID (numer obiektu) i numer strony wewnątrz tego obiektu
- **UndoNxtLSN** (tylko w CLRach) – LSN pierwszego w kolejności do wykonania podczas rollbacku rekordu wpisanego przez daną transakcję
  - UndoNxtLSN to wartość PrevLSNa rekordu dziennika kompensowanego przez aktualny rekord.
  - **Data** (opis zmiany) – instrukcje, jak wykonać redo i/lub undo danej modyfikacji

zmiany mogą być zapisywane logicznie, a niektórych w ogóle nie trzeba logować

Struktura strony

- **page_LSN** – występujący na każdej stronie numer zawierający LSN rekordu dziennika opisującego ostatnią zmianę dokonaną na tej stronie (‘update’ lub CLR)

- brak ograniczeń na zasady wymiany stron w buforze (poza zastosowaniem protokołu WAL) – można użyć np. strategii *steal*

- nadpisywanie odbywa się w pamięci nieulotnej
  - zmiany są nanoszone natychmiast i bezpośrednio do buforowej wersji strony zawierającej dany obiekt

- brak modyfikacji opóźnionych
  - w razie potrzeby, opóźnione modyfikowanie, a więc i opóźnione logowanie, można zaimplementować

Tablica transakcji

- używana podczas odtwarzania przy restarcie do śledzenia stanu aktywnych transakcji
- inicjalizowana w etapie analizy z rekordu ostatniego checkpointa i modyfikowana podczas analizy rekordów zapisanych po jego rozpoczęciu
- tej samej tablicy używa podczas normalnego działania zarządca transakcji

Oto jej najważniejsze pola:

- **TransID** – identyfikator transakcji
- **State** – stan zatwierdzenia transakcji; P – 'prepared' oraz U – 'unprepared'.
- **LastLSN** –LSN ostatniego rekordu dziennika wpisanego przez daną transakcję
  - **UndoNxtLSN** –LSN pierwszego w kolejności rekordu do przetworzenia podczas rollbacku

    dla rekordów nie będących CLRami będzie to wartość LastLSN

Tablica dirty_pages

- przedstawia informacje o zabrudzonych buforach stron podczas normalnego działania, włączana do rekordów checkpointowych
- każdy wpis zawiera 2 pola: PageID (identyfikator strony) i RecLSN (LSN odtwarzania)
- RecLSN wskazuje, gdzie w dzienniku mogą zaczynać się zmiany, których być może nie ma jeszcze w dyskowej wersji strony
  - jego wartość to LSN końca dziennika z momentu, w którym następuje pierwszy zapis do czystej strony
- minimalna wartość RecLSN w tablicy wskazuje na punkt startowy dla etapu powtarzania zmian przy odtwarzaniu po restarcie
- używana podczas odtwarzania przy restarcie tablica *restart dirty_pages* jest inicjalizowana z rekordu ostatniego checkpointa i modyfikowana podczas fazy analizy innych rekordów

Działanie normalne

- modyfikacja danych
- pełne i częściowe rollbacki
- zakończenie transakcji
- checkpointy

Modyfikacja danych

- jeżeli blokady zakładane są na poziomie rekordów, to:
  - zakładana jest blokada na ten rekord
  - uzyskiwany jest dostęp do odpowiedniej strony w buforze (jeśli czysta, to w polu *RecLSN* umieszczany jest LSN końca dziennika)
  - uzyskiwany jest zatrzask tej strony (w trybie X)
  - rekord jest modyfikowany
  - dodawany jest nowy rekord do dziennika, a jego LSN umieszczany w polu *PageLSN* strony i tablicy transakcji
  - zatrzask strony jest zwalniany i zmniejszany licznik odwołań do strony

- dla zachowania w dzienniku chronologii zmian w stronie (istotne przy fizycznych redo) ważne jest, by dodać wpis do dziennika przed zwolnieniem zatrzasku strony (zatrzask nie jest potrzebny podczas ewentualnej modyfikacji indeksów)

- dzięki obowiązkowi uzyskiwania zatrzasków całych stron przy operacjach czytania i pisania do pojedynczych rekordów, możliwe jest przemieszczanie rekordów w stronie podczas pisania (unikanie fragmentacji)

Modyfikacja danych – cd

- czasem nie wystarczy jeden rekord w dzienniku do opisania jakiejś operacji na danych (np. zapisany zostanie jeden rekord z informacją undo i osobny z redo); w takiej sytuacji:
  - rekord undo-only musi poprzedzać redo-only (na wypadek awarii po zapisaniu pierwszego)
  - w polu page_LSN umieścić LSN rekordu redo-only (żeby wskazywał ostatni rekord dziennika opisujący ostatnią modyfikację strony)

- bywa, że do dziennika zapisywane są tylko rekordy typu redo-only (dla operacji takich jak prepare, inwentaryzacja wolnej pamięci itp.)

Modyfikacja danych – cd

- czasem nie wiadomo z góry, który rekord (dane) będą modyfikowane (np. podczas operacji instert trzeba najpierw uzyskać dostęp do strony i znaleźć w niej wolne miejsce) – wtedy blokadę na rekord można próbować założyć (w trybie nieblokującym !) dopiero po uzyskaniu zatrzasku strony

- jeżeli blokady zakładane są na poziomie stron (lub czegoś większego), to wtedy wystarczy blokada strony (bez zatrzasku) – chyba że system dopuszcza dirty reads

Rollbacki – ogólnie

- rekordy dziennika dodane przez tą samą transajcję połączone są w listę za pomocą wskaźników PrevLSN (w kolejności odwrotnie chronologicznej)

- wskaźniki *UndoNxtLSN* w CLRach tworzą „skróty” w tej liście, pozwalając „przeskoczyć” sekwencję CLRów i kompensowanych przez nie rekordów – czyli nie wycofywać wycofanych już zmnian

- aby wycofać transakję ARIES (w uproszczeniu):
  - z tablicy transakcji odczytuje LSN ostatniego rekordu zapisanego do dziennika przez tę transakcję
  - korzystając ze wskaźników *PrevLSN* (lub *UndoNxtLSN* w przypadku CLRów) przechodzi po liście rekordów dziennika stopniowo wycofując opisywane przez nie zmiany (i zapisując CLRy dla wycofywanych akcji)

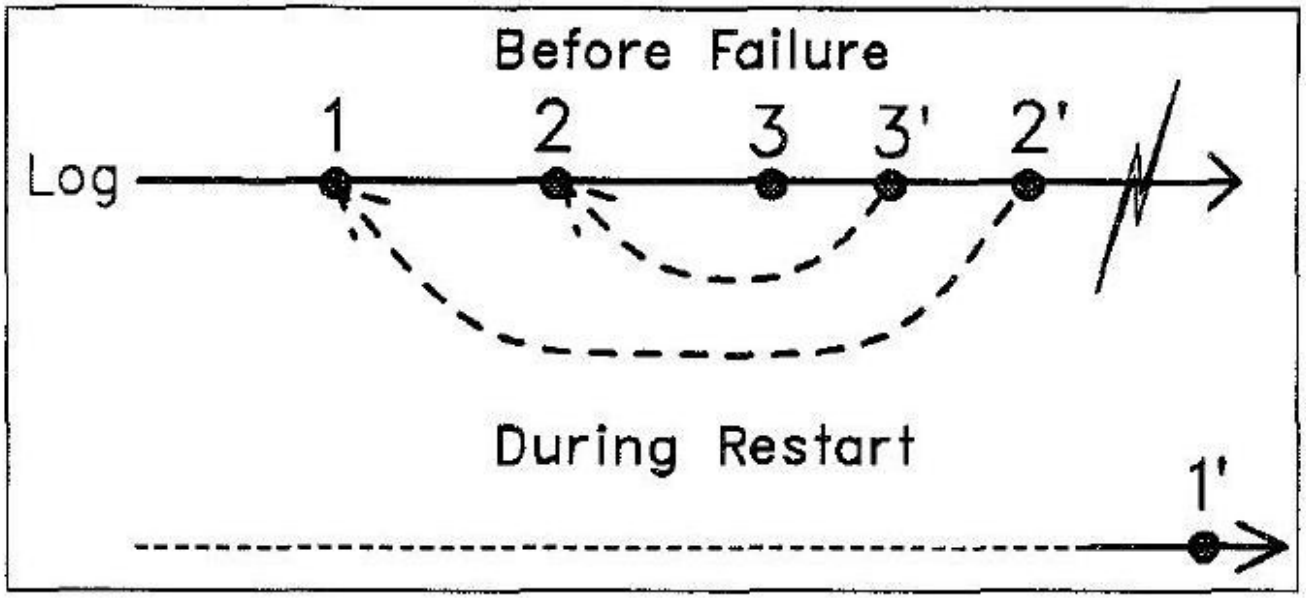
Rollbacki – ogólnie

- jeśli ARIES przechodząc po tej liście otworzonej za pomocą tych wskaźników natrafi na rekord nie CLR typu undo-redo lub undo-only, to wykonuje operację undo i przechodzi dalej po wskażniku *PrevLSN*

- jeśli natrafi na rekord nie CLR typu redo-only, to ignoruje go i przechodzi dalej

- jeśli natomiast natrafi na CLR, to przechodzi po wskaźniku *UndoNxtLSN*, omijając w tej sposób wszystkie zmiany wycofane we wcześniejszym rollbacku – właśnie to pozwala na ograniczenie liczby CLRów zapisywanych do dziennika, nawet w przypadku powtarzających się awarii podczas restartu lub zagnieżdżonych rollbacków.

Rollbacki – ogólnie



I′ is the Compensation Log Record for I
I′ points to the predecessor, if any, of I

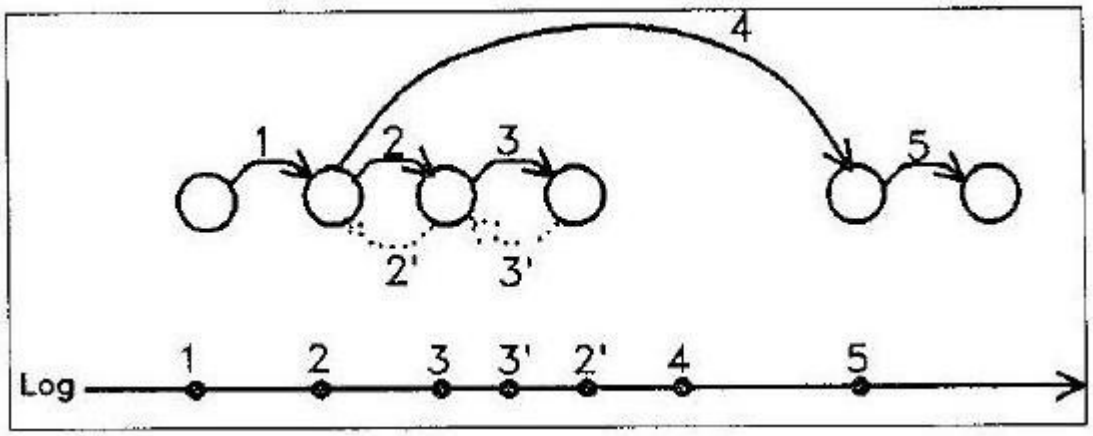
ARIES' technique for avoiding compensating compensations and duplicate compensations.

Rys. C. Mohan et al.

Pełne i częściowe rollbacki

- w każdym momencie wykonywania transakcji można dodać savepoint
- transakcję można wycofać do wybranego savepointu, po czym może ona kontynuować działanie
- jeśli transakcja została wycofana do jakiegoś savepointu lub poza niego, to przestaje on istnieć
- podczas tworzenia savepointu w pamięci zapisywany jest LSN ostatniego rekordu (lub zero jeśli początek transakcji) dziennika należącego do tej transakcji – dalej SaveLSN
- mechanizm savepointów może być widoczny lub niewidoczny dla użytkownika

Rys. C. Mohan et al.



Pełne i częściowe rollbacki – cd

```
ROLLBACK(SaveLSN,TransID);
UndoNxt := Trans_Table[TransID].UndoNxtLSN;                    /* addr of 1st record to undo   */
WHILE SaveLSN < UndoNxt DO;                                /* loop thru all relevant records   */
  LogRec := Log_Read(UndoNxt);                             /* read record to be processed      */
  SELECT (LogRec.Type)
    WHEN('update') DO;
      IF LogRec is undoable THEN DO;
        Page := fix&latch(LogRec.PageID,'X');
        Undo_Update(Page,LogRec);
        Log_Write('compensation',LogRec.TransID,Trans_Table[TransID].LastLSN,
                LogRec.PageID,LogRec.PrevLSN, ...,LgLSN,Data);                  /* write CLR */
        Page.LSN := LgLSN;
        Trans_Table[TransID].LastLSN := LgLSN;
        unfix&unlatch(Page);
      END;
      UndoNxt := LogRec.PrevLSN;
    END; /* WHEN('update') */
    WHEN('compensation')  UndoNxt := LogRec.UndoNxtLSN;          /* a CLR - nothing to undo */
    OTHERWISE  UndoNxt := LogRec.PrevLSN                 /* skip record and go to previous one */
  END; /* SELECT */
  Trans_Table[TransID].UndoNxtLSN := UndoNxt;
END;   /* WHILE  */
RETURN;
```

Rys. C. Mohan et al.

Pełne i częściowe rollbacki – cd

- podczas wycofywania transakcji nie są zakładane żadne blokady (tylko zatrzaski modyfikowanych stron) – wycofywana transakcja nie może ulec zakleszczeniu (same zatrzaski nie powodują zakleszczeń)

- dla każdej wycofywanej operacji do dziennika dodawany jest rekord CLR (dla uproszczenia zakładamy, że wystarczy jeden rekord)

- jeśli wykonywane jest logiczne undo mogą być też dodawanie inne rekordy (nie CLR)

- ograniczenie na liczbę rekordów dodawanych do dziennika podczas rollbacków

- rozwiązywanie zakleszczeń za pomocą częściowych rollbacków: kiedy transakcja zostanie częściowo wycofana do pewnego savepointu, można zwolnić blokady założone przez transakcję po ustanowieniu tego savepointu (bo nie wycofuje się CLRów)

Zakończenie transakcji

- załóżmy, że używana jest jakaś forma dwufazowego protokołu commit i że rekord *prepare* w dzienniku zawiera listę blokad (do pisania) trzymanych przez transakcję (na wypadek awarii po przejściu transakcji w stan prepare – do ochrony niezacommitowanych zmian)

- alternatywą jest określanie potrzebnych blokad na podstawie dziennika

- blokady pobrane w trybie do odczytu można zazwyczaj zwolnić

- operacje, które mogą się wiązać z usuwaniem plików i wymagałyby obszernych wpisów (undo) w dzienniku, można opóźnić aż do chwili, w której uzyskamy pewność, że trakcja zostaje zatwierdzona

- informacje o odłożonych na później operacjach umieszcza się w rekordzie *prepare*

Zakończenie transakcji – cd

- zatwierdzenie transakcji obejmuje:
  - dodanie do dziennika rekordu *end*
  - i zwolnienie blokad

- po zapisaniu rekordu *end* należy wykonać wszystkie odłożone na później operacje; dla tych operacji dodajemy do dziennika rekordy typu redo-only

- wycofanie transakcji znajdującej sie w stanie *prepare* obejmuje:
  - dodanie do dziennika rekordu rollback
  - wycofanie całej transakcji
  - porzucenie listy operacji odłożonych na później
  - zwolnienie blokad
  - i na koniec zapisanie w dzienniku rekordu *end*

Checkpointy

- okresowe tworzenie się punktów kontrolnych powoduje zmniejszenie nakładu pracy (konieczność przeglądania mniejszej lub większej części dziennika, ilość stron, które trzeba wczytać z trwałego nośnika itd.) przy restarcie

- checkpointy mogą być tworzone asynchronicznie (tzn. współbieżnie z wykonywaniem transakcji) – niespoczynkowe punkty kontrolne (fuzzy checkpoints)

- ARIES nie wymaga kopiowania żadnych brudnych stron na dysk przy okazji checkpointu; regularne przenoszenie zawartości brudnych stron na nośnik trwały należy do zarządcy buforów

Checkpointy – cd

- punkt kontrolny rozpoczyna się od dopisania do dziennika rekordu *begin_chkpt*

- następnie konstruowany jest rekord *end_rekord* (zakładamy dla uproszczenia, że wystarczy jeden rekord), zawierający:
  - tablicę transakcji
  - tablicę *dirty_pages* managera buforów
  - informacje związane z mapowaniem plików na obiekty, jeśli posiadają one jakieś brudne strony w buforach

- kiedy rekord *end_chkpt* jest gotowy, zapisuje się go na dysk

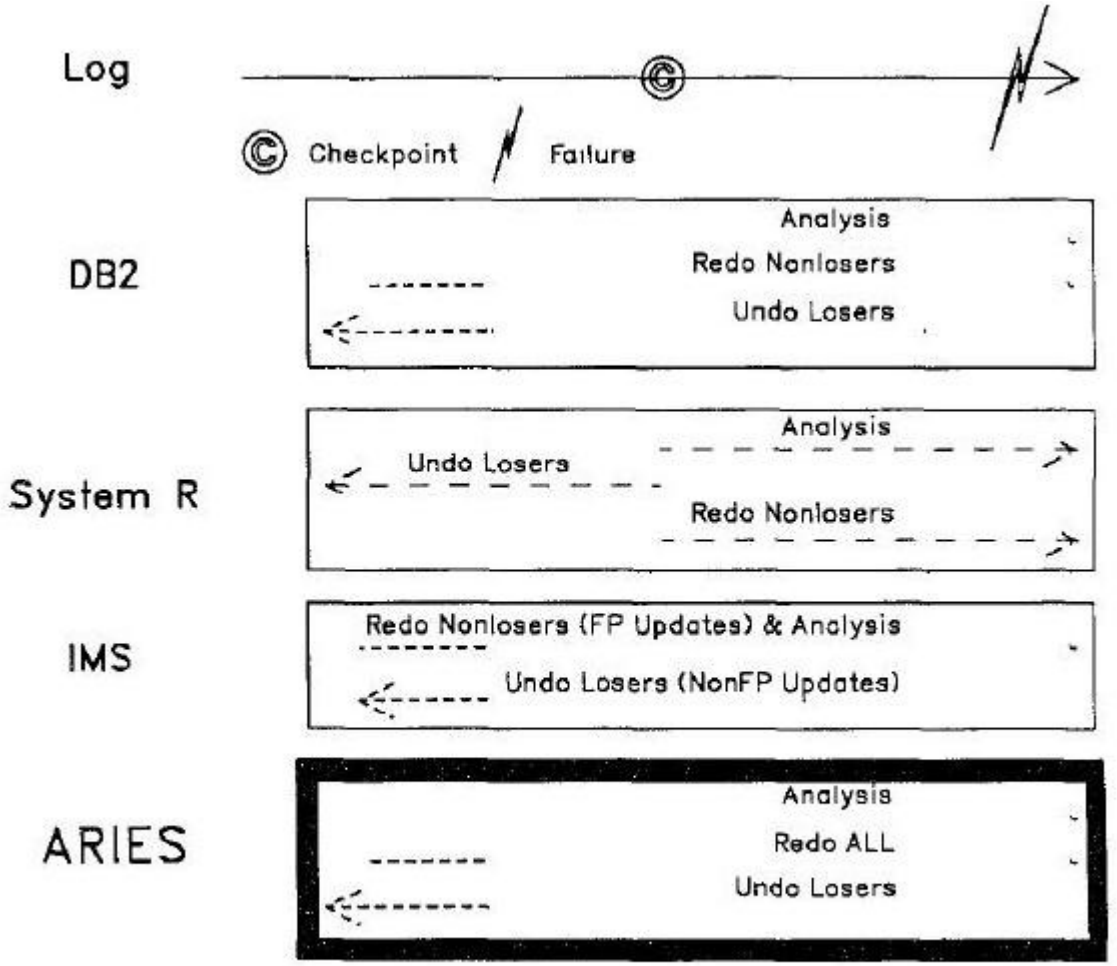
- LSN rekordu *begin_chkpt* umieszcza się w stałym i dobrze znanym miejscu na dysku (master record) – do momentu zapisania checkpoint traktowany jest jako niepełny

- rekordy *begin_chkpt* i *end_chckpt* mogą być rozdzielone wpisami dodanymi w międzyczasie przez transakcje – dlatego podczas ustalania miejsca, od którego ma się rozpocząć etap redo odtwarzania systemu, oprócz minimum z *RecLSN* brudnych stron, brane są pod uwagę rekordy dziennika, które zostały zapisane po rekordzie *begin_chkpt*

Restart

- Etap analizy
- Etap powtarzania zmian
- Etap wycofywania zmian
- Wybiórczy lub opóźniony restart

Restart – ogólnie



Restart processing in different methods.

Rys. C. Mohan et al.

Restart – ogólnie

- podczas restartu systemu po awarii trzeba doprowadzić dane do spójnego i aktualnego stanu, aby zachować własności atomowości i trwałości transakcji

- odtwarzanie systemu składa się z 3 faz:
  1. analizy
  2. fazy redo
  3. fazy undo

  } można w zasadzie połączyć

przy czym dwie pierwsze można w zasadzie połączyć. Na koniec tworzony jest checkpoint.

- aby skrócić ten proces, niektóre czynności można wykonywać równolegle (wymaga to jednak pobierania zatrzasków podczas restartu)

- można też dopuścić przetwarzanie nowych transakcji jeszcze przed zakończeniem restartu, albo wykonać częściowe odtworzenie danych, z opóźnionym odzyskaniem reszty

Restart – pseudokod

```
RESTART(Master_Addr);
Restart_Analysis(Master_Addr,Trans_Table, Dirty_Pages, RedoLSN);
Restart_Redo(RedoLSN, Trans_Table, Dirty_Pages);
buffer pool Dirty_Pages table := Dirty_Pages;
remove entries for non-buffer-resident pages from the buffer pool Dirty_Pages table;
Restart_Undo(Trans_Table);
reacquire locks for prepared transactions;
checkpoint();
RETURN;
```

Rys. C. Mohan et al.

Etap analizy

- w fazie analizy przeglądany jest dziennik w celu zebrania następujących informacji:
  - tablicy transakcji opisującej wszystkie niezakończone transakcji z momentu awarii (na podstawie tablicy transakcji z ostatniego checkpointu oraz późniejszych wpisów do dziennika)
  - tablicy *dirty_pages*, zawierającej listę wszystkich potencjalnie brudnych stron z buforów z momentu awarii (jest to lista stron, których zawartość trzeba być może odtworzyć w fazie redo)
  - wyznaczenie *RedoLSN*, od którego rozpocznie się faza redo

- jedyne rekordy jakie mogą być dodane do dziennika w tej fazie to rekordy end transakcji, które zakończyły działanie, ale nie zdążyły zapisać rekordu end przed awarią

Etap analizy – cd

- jeśli przeglądając dziennik znaleziony rekord dziennika dotyczący strony, której nie ma w *dirty_pages*, dodaje się ją do tablicy, jako *RecLSN* wpisując LSN tego rekordu

- przeglądając dziennik uaktualnia się także tablicę transakcji, śledząc stan transakcji i uaktualniając LSN ostatnich operacji wykonanych przez transakcje

- usuwa się z *dirty_pages* strony należące po pliku, jeśli znaleziony zostanie rekord *OSfile_return* dla tego pliku

- *RedoLSN* to minimum z *RecLSN* w tablicy *dirty_pages* z końca fazy analizy. Jeśli *dirty_pages* jest pusta, to można pomiąć fazę redo

- fazę analizy można połączyć z fazą redo, o ile w checkpoincie zapisane są blokady trzymane przez transakcje w stanie prepare (inaczej nie wiadomo skąd zacząć przeglądanie dziennika, bo trzeba uwzględnić rekordy transakcji w stanie prepare jeszcze sprzed checkpointu, a nie wiadomo, które to będą transakcje)

Rys. C. Mohan et al.

```
RESTART_ANALYSIS(Master_Addr, Trans_Table, Dirty_Pages, RedoLSN);
initialize the tables Trans_Table and Dirty_Pages to empty;
Master_Rec := Read_Disk(Master_Addr);
Open_Log_Scan(Master_Rec.ChkptLSN);                    /* open log scan at Begin_Chkpt record */
LogRec := Next_Log();                                  /* read in the Begin_Chkpt record */
LogRec := Next_Log();                                  /* read log record following Begin_Chkpt */
WHILE NOT(End_of_Log) DO;
  IF trans related record & LogRec.TransID NOT in Trans_Table THEN /* not chkpt/OSfile_return*/
    insert (LogRec.TransID,'U',LogRec.LSN,LogRec.PrevLSN) into Trans_Table;    /* log record */
  SELECT(LogRec.Type)
    WHEN('update' | 'compensation') DO;
      Trans_Table[LogRec.TransID].LastLSN := LogRec.LSN;
      IF LogRec.Type = 'update' THEN
        IF LogRec is undoable THEN Trans_Table[LogRec.TransID].UndoNxtLSN := LogRec.LSN;
      ELSE Trans_Table[LogRec.TransID].UndoNxtLSN := LogRec.UndoNxtLSN;
                                  /* next record to undo is the one pointed to by this CLR */
      IF LogRec is redoable & LogRec.PageID NOT IN Dirty_Pages THEN
        insert (LogRec.PageID, LogRec.LSN) into Dirty_Pages;
    END; /* WHEN('update' | 'compensation') */
    WHEN('Begin_Chkpt') ;  /* found an incomplete checkpoint's Begin_Chkpt record. ignore it */
    WHEN('End_Chkpt') DO;
      FOR each entry in LogRec.Tran_Table DO;
        IF TransID NOT IN Trans_Table THEN DO;
          insert entry(TransID,State,LastLSN,UndoNxtLSN) in Trans_Table;
        END;
      END; /* FOR */
      FOR each entry in LogRec.Dirty_PagLst DO;
        IF PageID NOT IN Dirty_Pages THEN insert entry(PageID,RecLSN) in Dirty_Pages;
        ELSE set RecLSN of Dirty_Pages entry to RecLSN in Dirty_PagLst;
      END; /* FOR */
    END; /* WHEN('End_Chkpt') */
    WHEN('prepare' | 'rollback') DO;
      IF LogRec.Type = 'prepare' THEN Trans_Table[LogRec.TransID].State := 'P';
      ELSE Trans_Table[LogRec.TransID].State := 'U';
      Trans_Table[LogRec.TransID].LastLSN := LogRec.LSN;
    END; /* WHEN('prepare' | 'rollback') */
    WHEN('end') delete Trans_Table entry for which TransID = LogRec.TransID;
    WHEN('OSfile_return') delete from Dirty_Pages all pages of returned file;
  END; /* SELECT */
  LogRec := Next_Log();
END; /* WHILE */
FOR EACH Trans_Table entry with (State = 'U') & (UndoNxtLSN = 0) DO; /* rolled back trans    */
  write end record and remove entry from Trans_Table;            /* with missing end record */
END; /* FOR */
RedoLSN := minimum(Dirty_Pages.RecLSN);                  /* return start position for redo */
RETURN;
```

Etap redo

- przeglądanie dziennika rozpoczyna się od *RedoLSN* z fazy analizy
- po napotkaniu rekordu typu redo (redo-only lub undo-redo) sprawdza się, czy odpowiednia strona znajduje się w *dirty_pages i czy* LSN rekordu jest >= *RecLSN* odpowiedniego wpisu w *dirty_pages*
- jeśli tak, to zagląda się do strony i porównuje *pageLSN* LSNem rekordu. Jeśli *page_LSN* jest mniejszy, to trzeba odtworzyć zmiany
- w fazie redo odtwarzany jest stan systemu z chwili awarii – nawet modyfikacje wykonane przez przerwane transakcje (wydaje się to zbędne, ale bardzo upraszcza algorytm – pozwala uniknąć wielu komplikacji i przypadków szczególnych)
- w fazie redo trzeba także odtworzyć operacje odłożone przez transakcje na później, które być może nie zostały jeszcze wykonane

Etap redo – cd

```
RESTART_REDO(RedoLSN, Dirty_Pages);
Open_Log_Scan(RedoLSN);                          /* open log scan and position at restart pt */
LogRec := Next_Log();                             /* read log record at restart redo point   */
WHILE NOT(End_of_Log) DO;                            /* look at all records till end of log */
  IF LogRec.Type = ('update'|'compensation') & LogRec is redoable &
     LogRec.PageID IN Dirty_Pages & LogRec.LSN >= Dirty_Pages[LogRec.PageID].RecLSN
    THEN DO;                  /* a redoable page update. updated page might not have made it to */
                              /* disk before sys failure. need to access page and check its LSN */
      Page := fix&latch(LogRec.PageID,'X');
      IF Page.LSN < LogRec.LSN THEN DO                  /* update not on page. need to redo it */
        Redo_Update(Page,LogRec);                                          /* redo update */
        Page.LSN := LogRec.LSN;
      END;                                                                 /* redid update */
      ELSE Dirty_Pages[LogRec.PageID].RecLSN := Page.LSN+1;       /* update already on page */
                    /* update dirty page list with correct info. this will happen if this */
                    /* page was written to disk after the checkpt but before sys failure  */
      unfix&unlatch(Page);
    END;                                                  /* LSN on page has to be checked */
  LogRec := Next_Log();                                            /* read next log record */
END;                                                             /* reading till end of log */
RETURN;
```

Rys. C. Mohan et al.

Etap redo – cd

- w fazie redo do dziennika nie są dodawane żadne wpisy

- ograniczenie dostępów do stron:
  - w ARIES redo dotyczą zawsze konkretnej strony, więc tylko strony z tablicy *dirty_pages* mogą wymagać odtworzenia – *dirty_pages* wraz z *RecLSN znacząco* ograniczja liczbę stron, do których niepotrzebnie uzyskuje się dostęp w fazie redo
  - możnaby jeszcze skomplikować ten mechanizm, ale czy warto? (koszt)

- współbieżność:
  - informacje z tablicy *dirty_pages* można wykorzystać do wcześniejszego zainicjowania operacji we/wy na tych stronach.
  - możliwe jest też współbieżne przetwarzanie różnych rekordów dziennika przez różne procesy pod warunkiem, że każda strona będzie modyfikowana tylko przez jeden proces i że zachowana będzie chronologia zmian w ramach każdej strony

Etap undo

- wykorzystywana jest tablica transakcji, natomiast niepotrzebna jest już tablica *dirty_pages*, ani porównywanie LSNów

- wycofywane są przerwane transakcje:
  - w kolejnych krokach wybierane jest maksimum z *UndoNxtLSN* (następna operacja do wycofania) wycofywanych transakcji
  - wycofywanie transakcji przebiega dokładnie tak jak normalny rollback, łącznie z zapisywaniem w dzienniku CLRów, z zachowaniem protokoły WAL

- współbieżność:
  - fazę undo również można wykonywać równolegle, pod warunkiem, że każdą transakcję będzie wycofywał tylko jeden proces
  - można też np. najpierw zapisać same CLRy bez rzeczywistego nanoszenia zmian na dane, a następnie współbieżnie wykonać te modyfikacje na podstawie dziennika

- ARIES pozwala na kontynuację przerwanych transakcji po awarii – wystarczy rollback do ostatniego savepointu (ale do tego potrzebna jest informacja o trzymanych przez nie blokadach oraz możliwość odtworzenia sesji z aplikacją kliencką)

```
RESTART_UNDO(Trans_Table);
WHILE EXISTS(Trans with State = 'U' in Trans_Table) DO;
  UndoLSN := maximum(UndoNxtLSN) from Trans_Table entries with State = 'U';
                          /* pick up UndoNxtLSN of unprepared trans with maximum UndoNxtLSN */
  LogRec := Log_Read(UndoLSN);                          /* read log record to be undone or a CLR */
  SELECT(LogRec.Type)
    WHEN('update') DO;
      IF LogRec is undoable THEN DO;          /* record needs undoing (not redo-only record) */
        Page := fix&latch(LogRec.PageID,'X');
        Undo_Update(Page,LogRec);
        Log_Write('compensation',LogRec.TransID,Trans_Table[LogRec.TransID].LastLSN,
              LogRec.PageID,LogRec.PrevLSN, ...,LgLSN,Data);                    /* write CLR */
        Page.LSN := LgLSN;                                        /* store LSN of CLR in page */
        Trans_Table[LogRec.TransID].LastLSN := LgLSN;           /* store LSN of CLR in table */
        unfix&unlatch(Page);
      END;                                                          /* undoable record case */
      ELSE;                                                /* record cannot be undone - ignore it */
      Trans_Table[LogRec.TransID].UndoNxtLSN := LogRec.PrevLSN; /* next record to process is */
                                    /* the one preceding this record in its backward chain */
      IF LogRec.PrevLSN = 0 THEN DO;                    /* have undone completely - write end  */
        Log_Write('end',LogRec.TransID,Trans_Table[LogRec.TransID].LastLSN,...);
        delete Trans_Table entry where TransID = LogRec.TransID;  /* delete trans from table */
      END;                                                            /* trans fully undone */
    END; /* WHEN('update') */
    WHEN('compensation') Trans_Table[LogRec.TransID].UndoNxtLSN := LogRec.UndoNxtLSN;
                                                        /* pick up addr of next record to examine */
    WHEN('rollback'|'prepare') Trans_Table[LogRec.TransID].UndoNxtLSN := LogRec.PrevLSN;
                                                        /* pick up addr of next record to examine */
  END; /* SELECT */
END;   /* WHILE */
RETURN;
```

Rys. C. Mohan et al.

Wybiórczy lub opóźniony restart

- można odzyskać najpierw tylko krytyczne dane i pozwolić systemowi na przetwarzanie nowych transakcji operujących na tych danych

- możliwe (w pewnym zakresie; w DB2 nie ma z tym problemu, bo undo jest tam dokładnym przeciwieństwem redo) jest też opóźnione odtwarzanie danych, które są chwilowo offline

- można rozpocząć przetwarzanie nowych transakcji równolegle z wycofywaniem transakcji przerwanych (trzeba odtworzyć ich blokady)

Algorytm opóźnionego odtwarzania niedostępnych danych:

1. powtórz historię dla dostępnych obiektów jak zwykle, opóźnij ją dla niedostępnych, zapamiętując zakresy nie odtworzonych rekordów w dzienniku

2. wykonuj undo jak zwykle, ale po napotkaniu pierwszej modyfikacji, dla której nie da się wygenerować CLRa, przerwij wycofywanie danej transakcji (transakcja zatrzymana – stopped). Kontynuuj wycofywanie pozostałych transakcji

3. dla zatrzymanych transakcji trzeba pobrać blokady, by chronić zmiany w dostępnych obiektach (tym w niedostępnych i tak nic nie grozi). Można to osiągnąć np. przechodząc dalej po wskaźnikach jak przy wycofywaniu i odnajdując w ten sposób wszystkie potrzebne blokady

4. kiedy restart systemu zostanie zakończony, a po pewnym czasie wcześniej niedostępne dane staną się dostępne, najpierw powtórz historię dla tych danych, a następnie kontynuuj wycofywanie zatrzymanych transakcji, zwalniając po drodze blokady

5. kiedy jakiś obiekt stanie się już dostępny i zostanie dla niego powtórzona historia, można pozwolić nowym transakcjom na dostęp do tego obiektu równolegle z wycofywaniem zatrzymanych transakcji

Wybiórczy lub opóźniony restart – cd

- algorytm ten wymaga możliwości określenia potrzebnych blokad na podstawie dziennika

- analogicznie można też traktować część dostępnych obiektów, opóźniając w ten sposób ich odtworzenie i przyspieszając restart systemu

Checkpointy podczas restartu

- W fazie analizy:

  Można dodać punkt kontrolny na koniec fazy analizy. Powinien on zawierać tablicę transakcji i tablicę *dirty_pages* utworzone w tej fazie.

- W fazie redo:

  Można dodawać punkty kontrolne w dowolnym momencie fazy redo. Nie będą one jednak miały żadnego sensu (tablice *dirty_pages* i transakcji nie są normalnie modyfikowane w tej fazie), chyba że manager buforów będzie na bieżąco uaktualniał pola *RecLSN* wpisów w tablicy restart *dirty_pages*.

- W fazie undo:

  Na początku tej fazy restart *dirty_pages* używana w poprzednich etapach staje się tablicą *dirty_pages* managera buforów. Dalej wycofywanie transakcji przebiega tak samo jak przy normalnym działaniu systemu, a zatem i checkpointy tworzymy normalnie.

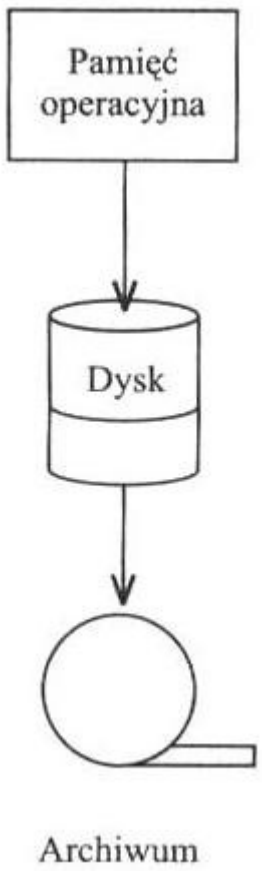
Odtwarzanie danych z nośnika

- by umożliwić późniejsze odtwarzanie danych, tworzy się ich kopie zapasowe – niespoczynkowe archiwa, tworzone współbieżnie z operacjami na tych samych danych (wydajność)

- archiwum może zawierać pewne niezacommitowane zmiany (możliwe jest też tworzenie archiwów nie zawierających takich zmian)

- dane do archiwum można kopiować bezpośrednio z dysku, bez udziału managera buforów (szybsze i prostsze) lub też z jego udziałem (wymaga pewnej synchronizacji)

- wraz z kopią danych w archiwum zapisuje się położenie rekordu *begin_checkpoint* najświeższego checkpointu



Rys. Ullman

Odtwarzanie danych z nośnika – cd

- odtwarzanie uszkodzonych danych polega na:
  - pobraniu z archiwum starszej wersji odtwarzanego obiektu/encji
  - przywróceniu go do stanu aktualnego (checkpointu + dziennik). Dzieje się to podobnie jak podczas omówionego wcześniej restartu, z tym że:
    - ograniczamy się oczywiście do odtwarzanego obiektu
    - pomijamy fazę analizy (nie warto w tej sytuacji) – przez to w fazie redo nie możemy założyć, że jeśli strona nie występuję w *dirty_pages* z checkpointu, to jest aktualna
    - wycofujemy wszystkie niezakończone (i nie będące w stanie prepare) transakcje, które modyfikowały odtwarzany obiekt. Informację o tym, które to transakcje można gdzieś gromadzić lub uzyskać wykonując fazę analizy (ale tylko od ostatniego checkpointu).

- informacje redo są w dzienniku związane z konkretnymi stronami – w prosty sposób można odzyskać nawet pojedynczą stronę, bez konieczności odtwarzania całego obiektu

- można też odtworzyć uszkodzoną/niespójną stronę w pamięci głównej na podstawie jej spójnej wersji z dysku (przegląda się dziennik poczynając od *RecLSN* pamiętanego dla tej strony w managerze buforów)

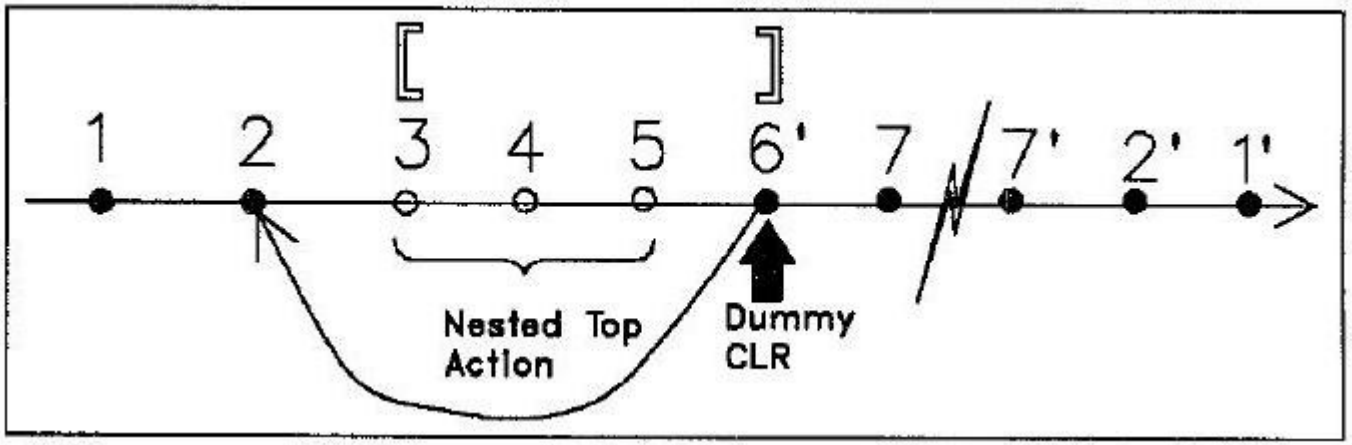
Zagnieżdżone top actions

- ciąg operacji commitowanych niezależnie od wyniku całej transakcji
- własność atomowości
- jeśli zakończą się powodzeniem, to nie zostaną już wycofane
- przykładem jest sytuacja, kiedy transakcja powiększa jakiś plik i chcemy pozwolić innym transakcjom wykorzystać nową przestrzeń w pliku jeszcze przed zacomittowaniem pierwszej transakcji
- możliwe rozwiązanie: wykonanie top action jako osobnej transakcji (drogie, możliwe zakleszczenie z udziałem obu transakcji itd.)

Zagnieżdżone top actions – cd

- W ARIES wykonanie takiej zagnieżdżonej top action wymaga:
  - zapamiętania LSNu ostatniego rekordu dziennika tej transakcji
  - zapisywania w dzienniku informacji redo i undo dla kolejnych czynności w ramach top action
  - po zakończeniu, zapisania do dziennika atrapy CLRa, jako UndoNxtLSN wpisując LSN zapamiętany w kroku 1.

- Jeśli zagnieżdżona top action składa się z tylko jednej operacji modyfikującej, to można ją opisać pojedynczym rekordem dziennika typu redo-only i uniknąć dodawania atrapy CLR.



Paradygmaty odtwarzania

- Wybiórcze powtarzanie
- Stan rollbacka
- Zarządzanie przestrzenią
- Wielokrotne LSNy

Wybiórcze powtarzanie

Wybiórcze powtarzanie (w fazie redo powtarzane są tylko operacje transakcji, które nie wymagają wycofania) wprowadzone zostało w wielu systemach, nie jest ono jednak łatwe do pogodzenia z protokołem WAL przy dużej granularności blokad, dlatego metoda ta nie została przyjęta w ARIES.

Wybiórcze powtarzanie - problemy

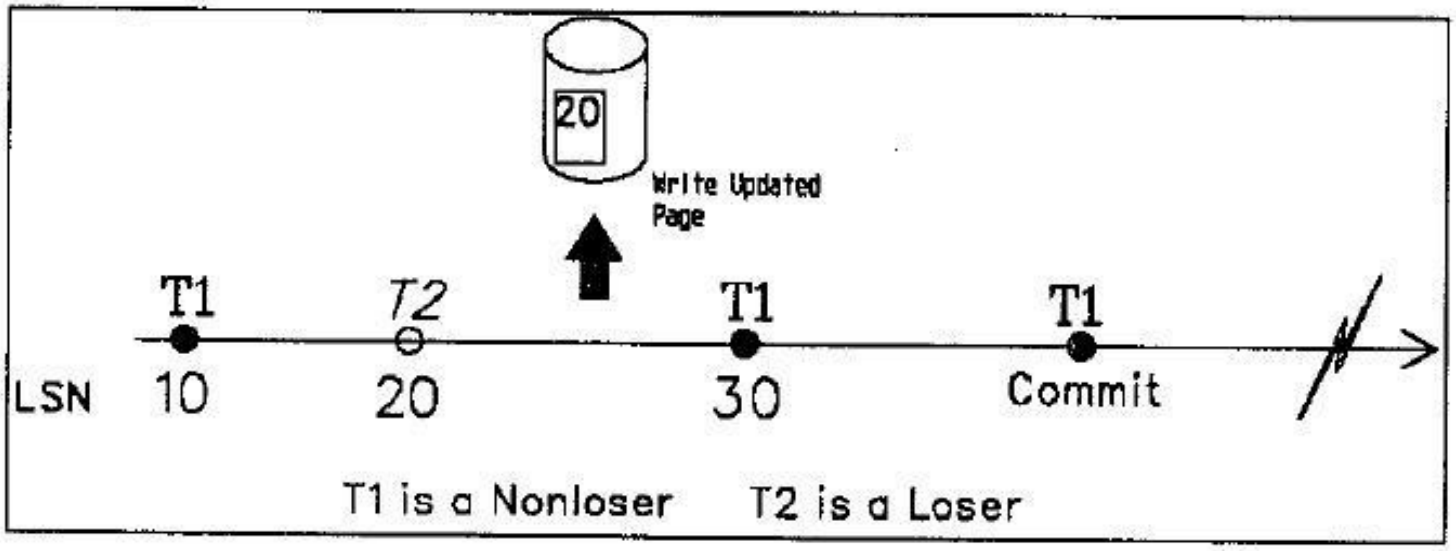
- gubimy informację o tym, które zmiany zostały zapisane w stronie, a które nie. Normalnie służy do tego pole pageLSN, co oczywiście nie wystarczy jeśli ma być stosowane wybiórcze redo, chyba że blokady zakładane są przez transakcje co najmniej na całe strony (bo wtedy tylko jedna transakcja modyfikuje na raz daną stronę).

- jeżeli utracona zostanie informacja o tym, które zmiany zostały już naniesione na stronę, a które nie, wtedy w fazie undo nie wiadomo, które zmiany należy wycofać. Dwukrotne wycofywanie tej samej modyfikacji danych może być niebezpieczne jeśli stosowane jest logowanie operacji (np. zwiększenie/zmniejszenie licznika).

- przy odpowiednim prowadzeniu dziennika wielokrotne wycofywanie modyfikacji może być niegroźne, nakłada to jednak duże ograniczenia i nie pozwala np. na logiczne undo.

- wykonanie faz redo i undo (wybiórczych) w odwrotnej kolejności także nie rozwiąże problemu. Tu też tracona jest informacja o tym, które zmiany są obecne w stronie i na podstawie pageLSN nie można już określić, które operacje należy powtórzyć w fazie redo (w fazie undo dodajemy do dziennika rekordy CLR zwiększając pageLSN).

Wybiórcze powtarzanie – problemy

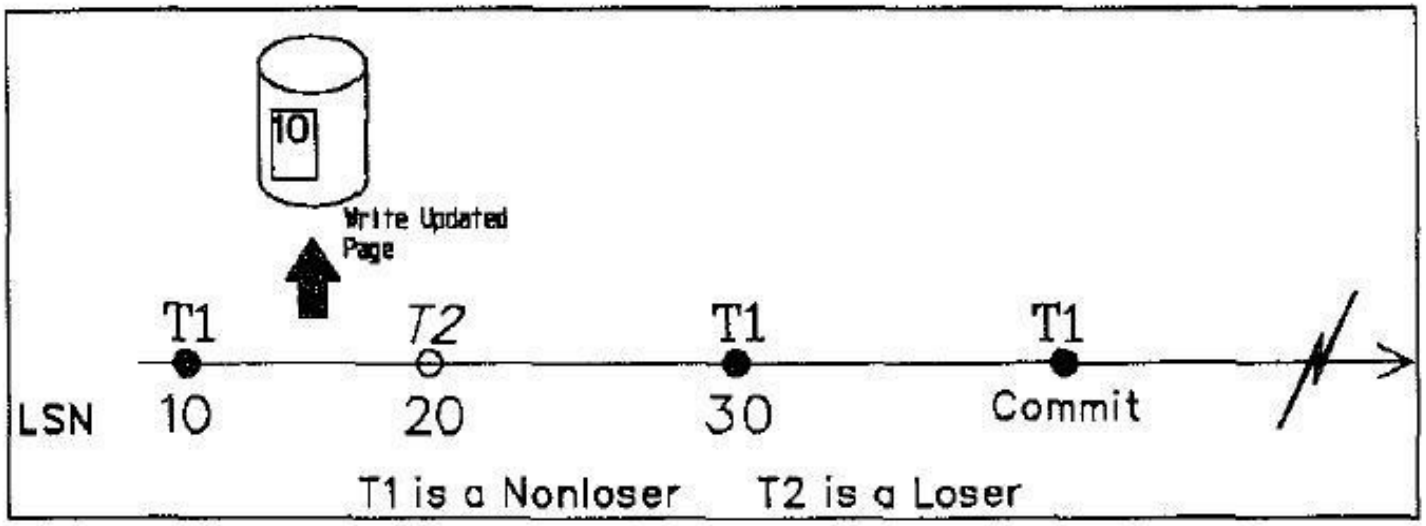


REDO Redoes Update 30
UNDO Undoes Update 20

Selective redo with WAL—problem-free scenario.

Rys. C. Mohan et al.

Wybiórcze powtarzanie – problemy



REDO Redoes Update 30

UNDO Will Try to Undo 20 Even Though Update Is NOT on Page

ERROR?!

Selective redo with WAL—problem scenario.

Rys. C. Mohan et al.

Wybiórcze powtarzanie

ARIES nie wykonuje wybiórczego redo, ale powtarza historię aż do momentu awarii. Ma to jeszcze tę zaletę, że pozwala w prosty sposób zatwierdzać część operacji wykonanych przez transakcję niezależnie od losów całej transakcji (zagnieżdzone top actions).

Oczywiście wadą tego rozwiązania jest zbędne powtarzanie niektórych operacji tylko po to, by móc je później wycofać.

Stan rollbacka

Nie zapisywanie CLRów sprawia, ze tracimy informację o tym, w jaki sposób wycofywanie zmian transakcji przeplatało się w czasie z operacjami wykonywanymi przez inne transakcje (niekoniecznie wycofywane) na tych samych danych, co uniemożliwiłoby fizyczne (w przeciwieństwie do logicznego) logowanie informacji redo.

W ARIES natomiast postawiono na fizyczne logowanie operacji redo, które pozwala na znacznie prostsze i szybsze odtwarzanie tych operacji.

Zarządzanie przestrzenią

Problemy związane z zarządzaniem przestrzenią jeśli blokady zakładane są na obiekty mniejsze niż strona i przy rekordach zmiennej długości:

- problem zapewnienia, że przestrzeń zwolniona przez jedną transakcję nie zostanie wykorzystana przez inna dopóki pierwsza transakcja nie zostanie zatwierdzona

- dopuszczenie wykorzystania przestrzeni zwolnionej przez niezacommitowaną transakcję w przypadku indeksów w celu zwiększenia poziomu współbieżności

- „logiczne adresy” rekordów w ramach strony – zapewnienie możliwości swobodnego przemieszczania rekordów w stronie w ramach odśmiecania

Zarządzanie przestrzenią – cd

Zazwyczaj dla każdego pliku zawierającego rekordy jednej lub wielu relacji utrzymuje się pewną liczbę stron z informacją o zajętości stron – tzw. free space inventory pages (FSIPs).

Informacje te są szacunkowe, np. zajęta w co najmniej 0%, co najmniej 25% itd.

Aby uniknąć specjalnego traktowania stron FSIP podczas redo i undo oraz by zachować niezależność odtwarzania (odrębnych obiektów), zmiany stron FSIP też trzeba rejestrować w dzienniku. W tej sytuacji muszą to być jednak rekordy typu redo-only, a do tego logiczne undo operacji na opisywanych przez tą FSIP stronach powinny w razie potrzeby modyfikować stronę FSIP, dopisując do dziennika odpowiednie rekordy.

Przykład: strona jest zajęta w 24%. T1 dopisuje kolejne 10 % (modyfikując odpowiednią FSIP), a T2 5%. Potem T1 zostaje wycofana. Nie powinna ona jednak wycofać zmiany we FSIP.

Wielokrotne LSNy

Pomysł jest taki, żeby rozwiązać problemy i ograniczenia związane z tylko jednym LSNem na stronę przy dużej granularności blokad poprzez związanie osobnego pola LSN z każdym obiektem.

Problemy:

- koszt pamięciowy
- problem z pamiętaniem LSNów dla usuniętych obiektów

Atrybuty ARIES – podsumowanie

- ✓ Wsparcie dla kontroli współbieżności na poziomie niższym niż strona i różnych stopni granularności blokad.
- ✓ Elastyczne zarządzanie buforami podczas restartu i normalnego działania.
- ✓ Minimalny koszt pamięciowy – po jednym LSNie na stronę.
- ✓ Gwarancja idempotentności powtarzania i cofania logowanych akcji bez nakładania ograniczeń na dane.
- ✓ Akcje podejmowane podczas wycofywania modyfikacji nie muszą być dokładną odwrotnością akcji podejmowanych przy modyfikacji oryginalnej.
- ✓ Wsparcie dla logowania operacji i nowatorskich trybów zakładania blokad.
- ✓ Użycie rekordów typu redo-only undo-only.
- ✓ Wsparcie dla częściowych i pełnych rollbacków transakcji.
- ✓ Wsparcie dla obiektów wielostronicowych.
- ✓ Pozwala na swobodną wymianę plików z systemem operacyjnym.
- ✓ Część operacji z transakcji może być zatwierdzona nawet jeśli transakcja jako całość została wycofana.

Atrybuty ARIES – podsumowanie

- ✓ Wydajne checkpointy (w tym podczas odtwarzania przy restarcie).
- ✓ Jednoczesne działanie wielu transakcji (zarówno wprzód jak i rollbacków) odwołujących się do tej samej strony.
- ✓ Brak nowych blokad i zakleszczania podczas rollbacków.
- ✓ Ograniczenie logowania podczas restartu mimo powtarzających się awarii lub zagnieżdżonych rollbacków.
- ✓ Dopuszcza wykorzystanie współbieżności i działania wybiórczego lub opóźnionego, co przyspiesza restart systemu.
- ✓ Tworzenie kopii zapasowych danych z dysku.
- ✓ Kontynuacja przerwanych transakcji po restarcie.
- ✓ Tylko jedno przejście wstecz po dzienniku podczas restartu lub odtwarzania danych.
- ✓ W CLRach potrzebna jest tylko informacja typu redo.
- ✓ Wsparcie dla transakcji rozproszonych.
- ✓ Wcześniejsze zwalnianie blokad podczas wycofywania transakcji i rozwiązywanie zakleszczeń przy użyciu częściowych rollbacków.

Podsumowanie – c.d.

Dokładne powtarzanie historii, z którego wynika użycie LSNów i zapis CLRów, umożliwia:

- ✓ wspieranie blokad na poziomie rekordów i przesuwanie rekordów w obrębie strony
- ✓ używanie jednej zmiennej pamiętającej stan i jednego LSN na stronę
- ✓ ponowne użycie zwolnionej pamięci dla późniejszych akcji tej samej lub innych transakcji
- ✓ odróżnianie się odwrotności danej operacji przy działaniu transakcji w przód od akcji podejmowanych przy jej wycofywaniu
- ✓ wycofywanie się transakcji ze strony, po której współbieżnie inne działają w przód
- ✓ odtwarzanie każdej strony niezależne od innych stron lub rekordów dziennika powiązanych ze stanem transakcji
- ✓ kontynuację transakcji, które zostały przerwane w wyniku awarii
- ✓ wybiórczy lub opóźniony restart i wycofywanie strat współbieżnie z działaniem nowych transakcji
- ✓ częściowe rollbacki

Podsumowanie – c.d.

Łączenie, dzięki polu UndoNxtLSN, CLRów z rekordami dziennika zapisanymi przy działaniu w przód umożliwia (dostarczane przez protokół powtarzania historii):

- ✓ uniknięcie wycofywania akcji CLRów
- ✓ uniknięcie wycofywania tego samego rekordu dziennika więcej niż raz
- ✓ zwalnianie blokady z obiektu po wycofaniu wszystkich jego modyfikacji podczas rollbacku
- ✓ radzenie sobie z częściowym wycofywaniem
- ✓ czynienie stałymi niektórych zmian dokonanych przez transakcję

Podsumowanie – c.d.

Przeprowadzanie fazy analizy przed powtarzaniem historii umożliwia:

- ✓ tworzenie checkpointów w każdym momencie fazy redo i undo
- ✓ dynamiczne zwracanie plików do systemu operacyjnego
- ✓ odtwarzanie informacji o plikach współbieżnie z danymi użytkownika
- ✓ rozpoznawanie stron z dużym prawdopodobieństwem wymagających powtarzania
- ✓ wykorzystanie możliwości unikania powtarzania zmian na niektórych stronach
- ✓ wykorzystanie możliwości unikania czytania niektórych stron podczas fazy powtarzania
- ✓ rozpoznawanie transakcji w stanach in-doubt i in-progress

Bibliografia

- C. Mohan, Donald J. Haderle, Bruce G. Lindsay, Hamid Pirahesh, Peter M. Schwarz: ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. (1992)
- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom: Systemy baz danych. Pełny wykład.
- http://wikipedia.org
- http://www.almaden.ibm.com/u/mohan/ARIES_Impact.html