Adatbázisok elmélete 25. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu
http://www.cs.bme.hu/˜kiskat

2004

Védekezés hibák ellen, helyreállítás (emlékeztető)

Alapprobléma: nem fut le valamelyik tranzakció (sérül az atomiság) és emiatt inkonzisztens lesz az adatbázis.

Cél az, hogy újra konzisztens állapotba hozzuk az adatbázist (visszacsinálás vagy befejezés) úgy, hogy a tartósság megmaradjon: ha egy tranzakció már befejezte a munkáját, akkor annak hatása ne vesszen el.

Alapfogalmak (emlékeztető)

COMMIT pont: az a pont, amikor a tranzakció minden érdemi munkával megvan, programhiba vagy ütemező miatt ABORT már biztos nem lehet.
Nem biztos, hogy ekkor minden hatása látszik is már a tranzakciónak, lehet, hogy nincs minden írása véglegesítve, de minden készen áll már erre.

Piszkos adat: Olyan adat, amit még nem COMMIT-ált tranzakció (azaz olyan, aki még meghalhat) írt az adatbázisba.
Ha ilyet olvas egy másik tranzakció, akkor baj lehet, ha az első ABORT-ál, de a második nem.

Lavina: egymás után kell ABORT-okat elrendelni a tranzakcióknál piszkos adatból eredő hiba miatt.

Megoldások piszkos adat és lavina ellen (emlékeztető)

Különböző megoldások a tranzakcióhibákból (programhiba vagy rendszer általi ABORT) származó problémákra:

- Olyan tranzakciótól, aki nem COMMIT-ált, nem olvasunk. (Nem olvasunk olyan értéket, amit olyan tranzakció írt, akinek még nem volt COMMIT).
- Hagyjuk, hogy minden tranzakció azt csináljon, amit akar, ha lavina lesz, akkor majd megoldjuk (UNDO protokoll)
- Zárolási protokollt kényszerítünk a tranzakciókra, ami biztosítja, hogy nem lesz piszkos adatból probléma, lavina:

  szigorú 2PL:
  - ⋆ 2PL
  - ⋆ DB-be írás csak COMMIT után
  - ⋆ zárak elengedése csak írás után

Tétel. *Ha mindegyik tranzakció a szigorú 2PL protokollt követi, akkor az ütemezés sorosítható lesz és lavinamentes.*

Rendszerhibák utáni helyreállítás

Az eddigiekben azzal foglalkoztunk, hogy a tranzakciók hibái esetén mit lehet tenni. Erre a szigorú 2PL jó megoldást nyújt, de mi van a komolyabb hibák, a rendszerhibák esetén?

Azt mindig feltesszük, hogy a háttértár nem sérül, csak a belső memória, a puffer egy része száll el.

A belső tár sérülése elleni védekezés két részből áll:

1. Felkészülés a hibára: **naplózás**
2. Hiba után helyreállítás: **a napló segítségével egy konzisztens állapot helyreállítása**

Természetesen a naplózás és a hiba utáni helyreállítás összhangban vannak, de van több különböző naplózási protokoll (és ennek megfelelő helyreállítás).

Napló

A tranzakciók legfontosabb történéseit írjuk ide, például:

- T_i kezd: (T_i, BEGIN)
- T_i írja A-t: (T_i, A, régi érték, új érték)
  (néha elég csak a régi vagy csak az új érték, a naplózási protokolltól függően)
- T_i COMMIT-ál: (T_i, COMMIT)
- T_i ABORT-ál: (T_i, ABORT)

A napló időrendben tartalmazza a történéseket és tipikusan a háttértáron tartjuk, amiről feltesszük, hogy nem sérül.

Fontos, hogy a naplóbejegyzéseket mikor írjuk át a pufferből a lemezre.

UNDO protokoll-naplózás

Fő szabályok:

- Ha a T tranzakció módosítja az X adatbáziselemet, akkor a (T, X, régi érték) naplóbejegyzést azelőtt kell a lemezre írni, mielőtt az X új értékét a lemezre írná a rendszer.
- Ha a tranzakció hibamentesen befejeződött, akkor a COMMIT naplóbejegyzést csak azután szabad a lemezre írni, ha a tranzakció által módosított összes adatbáziselem már a lemezre íródott, de ezután rögtön.

UNDO protokoll

A (nem teljesen szigorú) 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron:
   (T_i, BEGIN), (T_i, A régi érték) vagy (T_i, ABORT)
2. Tényleges írás az adatbázisba a háttértáron, nem a pufferben: OUTPUT(A)
3. COMMIT után a napló háttértárra írása.
4. Zárak elengedése

Megjegyzések

- nincs lavina, mert zárelengedés csak COMMIT után (azaz piszkos adatot nem olvashatunk)
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt az UNDO helyreállítás

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	$(T, A, 8)$
READ(B, t)	8	16		8	8	
$t := t \cdot 2$	16	16		8	8	
WRITE(B, t)	16	16	16	8	8	$(T, B, 8)$
FLUSH LOG						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
						(T, COMMIT)
FLUSH LOG						
UNLOCK(A)						
UNLOCK(B)						

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - ★ Ha van (T, BEGIN) és van (T, COMMIT) $\Longrightarrow$ minden változás a lemezen van $\surd$
  - ★ Ha van (T, BEGIN), de nincs (T, COMMIT) $\Longrightarrow$ lehet olyan változás, ami nem került még a lemezre, de lehet olyan is ami kikerült $\Longrightarrow$ ezeket vissza kell állítani
- Második feladat: visszaállítás
  A napló végétől visszafelé (pontosabban a hibától) haladva: megjegyezzük, hogy mely T_i-re találtunk (T_i, COMMIT) és (T_i, ABORT) bejegyzéseket.
  Ha van egy (T_i, X, v) bejegyzés:
  - ★ Ha láttunk már (T_i, COMMIT) bejegyzést (visszafelé haladva), akkor T_i már befejeződött, értékét kiírtuk a tárra $\Longrightarrow$ nem csinálunk semmit
  - ★ Minden más esetben (vagy volt (T_i, ABORT) vagy semmi) $\Longrightarrow$ X-be visszaírjuk v-t
- Harmadik feladat: Ha végeztünk, minden nem teljes T_i-re írjunk (T_i, ABORT) a napló végére.

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	$(T, A, 8)$
READ(B, t)	8	16		8	8	
$t := t \cdot 2$	16	16		8	8	
WRITE(B, t)	16	16	16	8	8	$(T, B, 8)$
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
						(T, COMMIT)
(FLUSH LOG)						
UNLOCK(A)						
UNLOCK(B)						

Megjegyzések

- Mi van ha a helyreállítás közben hiba történik? Már bizonyos értékeket visszaállítottunk, de utána elakadunk.
  $\Longrightarrow$ Kezdjük előről a visszaállítást! Ha már valami vissza volt állítva, legfeljebb még egyszer „visszaállítjuk” $\Longrightarrow$ nem történik semmi.
- **Ez így nagyon sokáig tarthat, mert el kell mennünk a napló elejéig.**

CHECKPOINT képzése

Meddig menjünk vissza a naplóban? Honnan tudjuk, hogy mikor vagyunk egy biztosan konzisztens állapotnál?
Erre való a **CHECKPOINT**. Ennek képzése:

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz
4. A naplóba beírjuk, hogy (CHECKPOINT)
5. A naplót is háttértárra írjuk: (FLUSH LOG)

Ezután nyilván elég az első CHECKPOINT-ig visszamenni, hiszen előtte minden T_i már valahogy befejeződött.

$\Longrightarrow$ Teljesen le kell állítani a rendszert.

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT $(T_1, \dots, T_k)$), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Megvárjuk, hogy minden fenti T_i végetérjen. (Közben más tranzakciók elindulhatnak.)
4. Ha mind befejeződött: (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van $\Longrightarrow$ elég visszamenni a következő START CHECKPOINT-ig.
- Ha előbb (START CHECKPOINT $(T_1, \dots, T_k)$)-ot találunk $\Longrightarrow$ ezek nem mindegyike fejeződött be (meg esetleg mások sem, amik még később kezdődtek) $\Longrightarrow$ elég visszamenni a legkorábban kezdődött T_i elejére

Példa

$(T_1$, BEGIN)
$(T_1, A, 5)$
$(T_2$, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2)) $\leftarrow$
$(T_2, C, 15)$
$(T_3$, BEGIN)
$(T_1, D, 20)$
$(T_1$, COMMIT)
$(T_3, E, 25)$ $\leftarrow$
$(T_2$, COMMIT)
(END CHECKPOINT)
$(T_3, F, 30)$ $\leftarrow$

- T_3 nem fejeződött be $\Longrightarrow F \to 30$
- T_3 nem fejeződött be $\Longrightarrow E \to 25$
- (START CHECKPOINT) $\Longrightarrow \surd$

Példa

$(T_1$, BEGIN)
$(T_1, A, 5)$
$(T_2$, BEGIN)
$(T_2, B, 10)$ $\leftarrow$
(START CHECKPOINT (T_1, T_2)) $\leftarrow$
$(T_2, C, 15)$ $\leftarrow$
$(T_3$, BEGIN)
$(T_1, D, 20)$
$(T_1$, COMMIT) $\leftarrow$
$(T_3, E, 25)$ $\leftarrow$

- T_3 nem fejeződött be $\Longrightarrow E \to 25$
- T_1 befejeződött $\Longrightarrow$ T_1-et nem bántjuk
- T_2 nem fejeződött be $\Longrightarrow C \to 15$
- (START CHECKPOINT) $\Longrightarrow$ elég visszamenni T_2 elejéig
- T_2 nem fejeződött be $\Longrightarrow B \to 10$

REDO protokoll-naplózás

Fő szabály:

- Mielőtt az a lemezen módosítunk egy X adatelemet, a (T, X, v) és a $(T$, COMMIT) bejegyzést is ki kell írnunk a naplóba.

REDO protokoll

Ez a szigorú 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron: $(T_i$, BEGIN), $(T_i$, A, új érték), $(T_i$, ABORT)
2. COMMIT után a napló háttértárra írása
3. Tényleges írás az adatbázisba a háttértáron, nem a pufferben
4. Zárak elengedése

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt a REDO helyreállítás
- Különbség a az UNDO protokollhoz képest:
  - ★ Az adat változás utáni értékét jegyezzük fel a naplóba
  - ★ Máshová rakjuk a COMMIT-ot, a kiírás elé $\Longrightarrow$ megtelhet a puffer
  - ★ Az UNDO protokoll esetleg túl gyakran akar írni $\Longrightarrow$ itt el lehet halasztani az írást

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	$(T, A, 16)$
READ(B, t)	8	16		8	8	
$t := t \cdot 2$	16	16		8	8	
WRITE(B, t)	16	16	16	8	8	$(T, B, 16)$
						(T, COMMIT)
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
UNLOCK(A)						
UNLOCK(B)						

REDO helyreállítás

Ha rendszerhiba történt és megsérült a belső tár, akkor az alábbiakat tesszük:

1. Minden zárat feloldunk
2. A napló mentett részét nézzük visszafele, megkeressük azokat a tranzakciókat, amikre volt már COMMIT (a többi nem érdekes, mert ha még nem volt a COMMIT-juk kimentve, akkor nem is írtak a DB-be)
3. Addig megyünk vissza a naplóban, amíg biztosan konzisztens állapotot nem találunk (eleje vagy CHECKPOINT)
4. A COMMIT-tált tranzakciók írásait elölről kezdve (a legelső COMMIT-ált elejétől) megismételjük (ha már egyszer be volt írva, az se baj, akkor csak felülírjuk ugyanazzal). Ezt meg tudjuk tenni, mert ismerjük az új értékeket.
5. Minden nem befejezett T_i tranzakcióra (T_i,ABORT)-ot írunk a napló végére, (FLUSH LOG)

Megjegyzések a REDO helyreállításhoz

- Ha a napló háttértáron van, akkor mindent újra tudunk csinálni, ami meg még nem került ki, azzal kapcsolatban változtatás se történt, nem kell visszacsinálni semmit.
- Ha a helyreállítás során lenne újra hiba, akkor a napló marad, mert az már kint van, ez alapján újra kezdhetjük a helyreállítást.
- Eredmény: a háttértárra kikerült COMMIT-oknak megfelelő tranazakciók eredménye látszik, a többiekéből pedig semmi.

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	$(T$, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	$(T, A, 16)$
READ(B, t)	8	16		8	8	
$t := t \cdot 2$	16	16		8	8	
WRITE(B, t)	16	16	16	8	8	$(T, B, 16)$
						$(T$, COMMIT)
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
UNLOCK(A)						
UNLOCK(B)						