Distribuované transakce

Lukáš Petrlík*
luki@kiv.zcu.cz

1 Úvod

Pojem transakce pochází původně z obchodního světa. Předpokládejme, že firma *A* hledá dodavatele pro jistou zakázku. V úvahu přichází firma *B*, ovšem k podmínkám dodávky má jisté výhrady. Po nějaké době dohadování dojdou obě strany ke konsensu a podepíší kontrakt. Do podepsání kontraktu mají obě strany svobodu ukončit vyjednávání a chovat se, jako by se nic nestalo. Po podepsání jsou ovšem obě strany vázány splnit svou část úmluvy.

Databázové transakce se chovají podobně. Proces *A* oznámí, že chce provést transakci s jedním nebo více dalšími procesy. Procesy pak mohou provádět operace – vytvářet, rušit a modifikovat objekty. Nakonec iniciátor oznámí přání, aby se všechny strany zavázaly (to commit – zavázat se) k provedeným změnám. Pokud některý z procesů odmítne (nebo havaruje před potvrzením transakce), situace se vrátí přesně do stavu před započetím transakce, tj. všechny změny na objektech jsou zrušeny.

1.1 Primitiva pro programování transakcí

Jazyky pro manipulaci dat musejí obsahovat základní primitiva pro programování transakcí. Konkrétní seznam primitiv samozřejmě závisí na jazyce. Jako příklad pro další diskusi zvolíme těchto 5 primitiv:

START – začátek transakce, následující příkazy budou tvořit transakci. Příkaz pro start transakce může být zadán explicitně nebo je implicitně předpokládán.
COMMIT – ukončení transakce a pokus o zajištění skutečného provedení změn v databázi.
ABORT – zrušení transakce, návrat do stavu před započetím transakce
READ – přečtení dat
WRITE – zápis dat

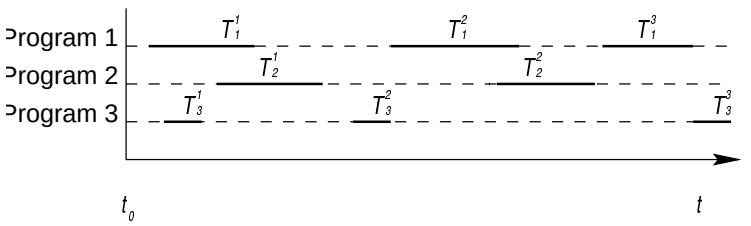
Uvnitř transakce je možné použít libovolný příkaz pro modifikaci databáze, v našem jednoduchém modelu však budeme předpokládat, že se složitější operace skládají z primitivních operací čtení a zápisu.

Provádění aplikačního programu v databázovém prostředí můžeme chápat jako posloupnost atomických transakcí, mezi kterými se provádí nedatabázové operace.

Příklad:

Mějme bankovní aplikaci, ve které chceme přesunout jistou částku mezi dvěma účty. Operace bude provedena v těchto krocích:

* Napsáno v prosinci 1996, poslední úprava: leden 2003. Sazba TEXem.



Obrázek 1: Transakce (tlustou čarou) jsou prokládány nedatabázovými operacemi.

```
START T1
  READ KONTO1
  KONTO1 := KONTO1 - ČÁSTKA
  WRITE KONTO1
  READ KONTO2
  KONTO2 := KONTO2 + ČÁSTKA
  WRITE KONTO2
COMMIT T1
```

Je zřejmé, že transakce musí být provedena atomicky, jinak by mohlo dojít ke „ztrácení“ peněz.

1.2 Vlastnosti transakcí

Transakce musí mít tyto vlastnosti:[1]

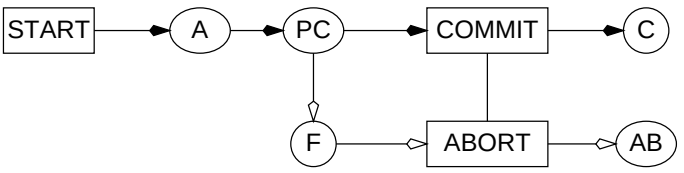
1. **atomičnost** (atomicity) – pro „vnější svět“ je transakce atomická, tj. tvoří nedělitelnou jednotku
2. **izolovaná vratnost** – vrácení (zrušení) transakce nemá vliv na jiné transakce
3. **trvalost** (durability) – úspěšná transakce způsobí uložení změn v bázi dat.
4. **upořádatelnost** (serializability) – výsledek souběžně prováděných transakcí je stejný jako některé (systémově závislé) sériové provedení těchto transakcí.

1.3 Stavy transakce

Transakce se po dobu svého života může dostat do jednoho z pěti stavů:

- aktivní (A) – stav od počátku provádění transakce
- částečně potvrzený (PC) – stav po provedení poslední operace transakce
- chybný (F) – v průběhu transakce nelze pokračovat
- zrušený (AB) – stav po skončení operace ABORT
- potvrzený (C) – stav po úspěšném skončení, tj. po dokončení operace COMMIT

[1] Nutno poznamenat, že různí autoři vycházejí při diskuzi transakcí z poněkud odlišných množin základních vlastností – pravděpodobně neexistuje všeobecně uznávaná minimální množina.

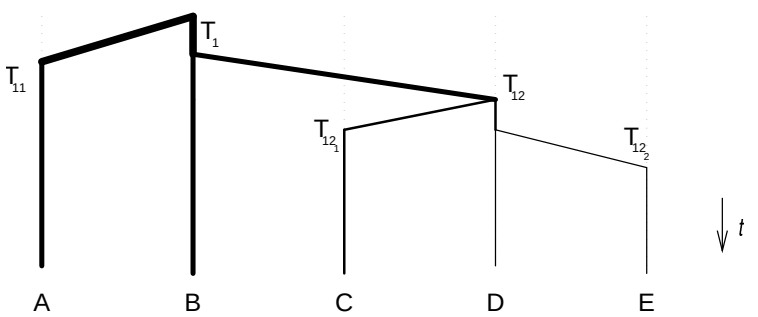


Obrázek 2: Stavy transakce.

1.4 Lokální a distribuované transakce

V distribuovaném databázovém prostředí může transakce přistupovat k datům uloženým ve více uzlech. Takové transakci říkáme **distribuovaná** nebo **globální**. Naproti tomu **lokální** transakce přistupuje pouze k datům jediného uzlu a nevyžaduje spolupráci s transakcemi v jiných uzlech.

Distribuované transakce jsou spuštěny z jednoho uzlu a podle potřeby spouštějí další, **dílčí** transakce (sub-transactions). Původní (primární) transakce většinou přebírá koordinaci distribuované transakce. Tato koordinace zahrnuje synchronizaci se souběžnými lokálními transakcemi a s ostatními globálními transakcemi, které jsou v systému aktivní.



Obrázek 3: Spouštění podtransakcí v distribuovaném systému.

Z toho je zřejmé, že distribuovanost podstatně zvyšuje složitost koordinace transakcí.

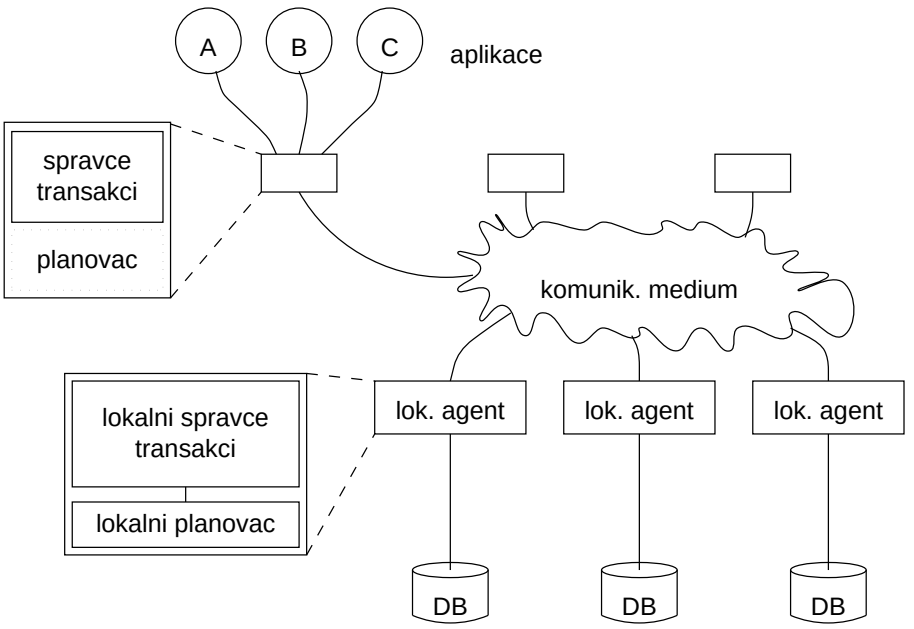
2 Implementace

Modul, který je v systému řízení báze dat (SŘBD) zodpovědný za synchronizaci se nazývá **plánovač** (scheduler), protože rozvrhuje transakce tak, aby nedošlo k jejich vzájemnému ovlivnění. Aplikační programy komunikují s plánovačem prostřednictvím **správce transakcí** (transaction manager), který pro aplikace provádí koordinaci databázových operací.

Nyní se budeme zabývat tím, jak se výše uvedené vlastnosti implementují. Nejprve budou stručně popsány mechanismy implementace lokálních transakcí (v distribuovaném systému je provádí lokální agent), poté budou popsány mechanismy, používané v distribuovaném prostředí.

2.1 Lokální transakce

V následujícím oddíle popíšeme některé mechanismy pro implementaci lokálních transakcí – mechanismus soukromé pracovní oblasti a tvorbu žurnálu. Pro zajištění konzistence databáze i v případě výpadku se tyto metody často kombinují s mechanismem stabilní paměti.



Obrázek 4: Zjednodušená koncepce distribuovaného SŘBD.

2.1.1 Stabilní paměť

Prvořadou funkcí kvalitního SŘBD je zabezpečit zotavení z případných chyb. Některé systémy proto pro uchování kritických informací o průběhu transakce používají tzv. stabilní paměť.

Stabilní paměť je možné implementovat pomocí dvojice disků (viz obr. 5), kde druhý disk je přesnou kopií prvního. Každý blok se se nejprve zapíše na první disk a přečtením se ověří bezchybnost zápisu, pak se provede totéž pro druhý disk.

Předpokládejme, že systém havaruje po zápisu bloku na první disk. Při zotavování se z chyby systém porovná obsah obou disků. Pokud nalezne rozdíl, bude předpokládat že blok na na prvním disku je správný, protože byl zapsán dříve. Při chybě kontrolního součtu může být vadný blok regenerován z odpovídajícího bloku na druhém disku.

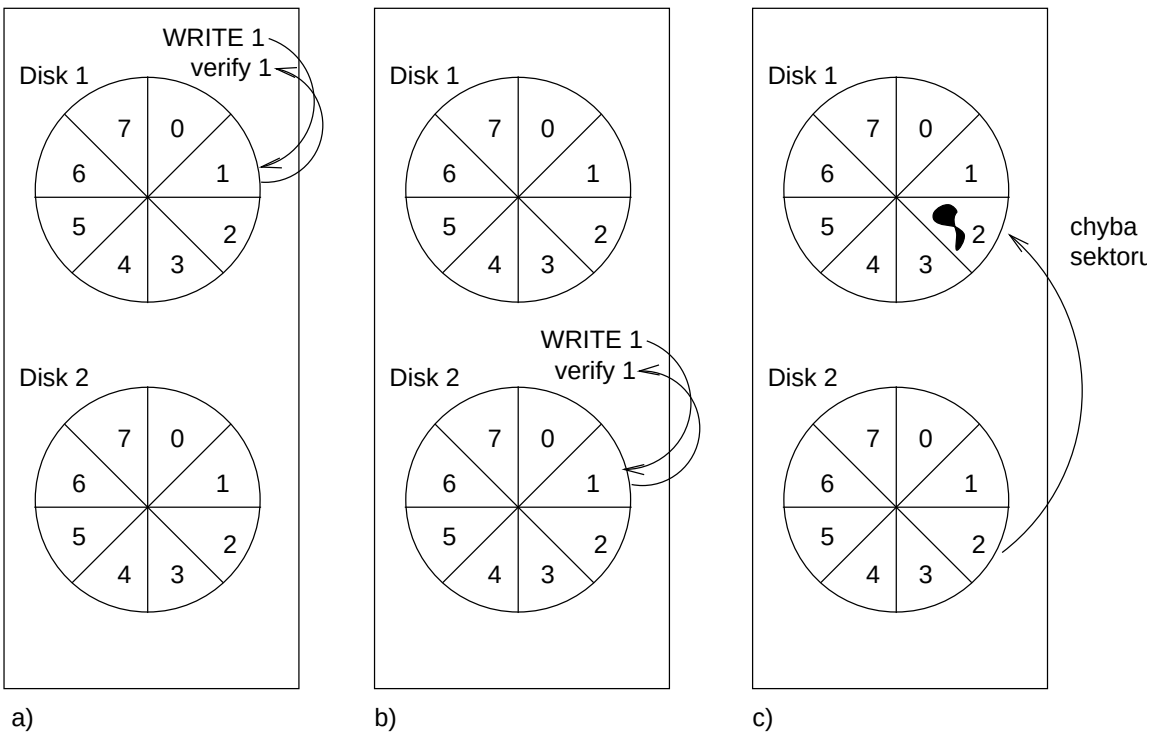
2.1.2 Soukromá pracovní oblast

Při započetí transakce je procesu koncepčně dán k dispozici soukromý pracovní prostor, obsahující všechny objekty, ke kterým má proces přístup. Veškeré operace pak provádí v tomto pracovním prostoru až do dokončení transakce.

Toto pozorování vede k první implementační metodě, totiž vytvoření skutečného soukromého pracovního prostoru pro transakci. Ve skutečnosti by ovšem cena takové operace byla příliš vysoká, proto se používá několik optimalizací.

Při spuštění transakce je vytvořena prázdná pracovní oblast (viz obr. 6), která obsahuje pouze odkaz na skutečnou databázi (nebo – pro podtransakci – na rodičovskou pracovní oblast). Pokud je záznam požíván pouze pro čtení, není nutno vytvářet jeho kopii. Systém sleduje odkazy, dokud nedojde k záznamu v pracovní oblasti některého předka.

V případě zápisu je záznam nalezen stejným způsobem jako pro čtení a jeho kopie je vložena do soukromého pracovního prostoru spolu s indexem příslušné databáze. Díky soukromému indexu je možné číst data obvyklým způsobem, protože obsahuje odkazy na původní záznamy. Při pokusu o zápis je vytvořena nová kopie záznamu a odkaz na ní je vložen do indexu.



Obrázek 5: Zápis a zotavení stabilní paměti.

Při zrušení transakce stačí zrušit soukromou pracovní oblast. Dokončení transakce znamená atomický zápis soukromého indexu do rodičovského pracovního prostoru.

Stejným způsobem se někdy implementují transakční souborové systémy. Jako index zde slouží i-uzel souboru (nebo obdobná datová struktura) a jako záznamy se chápou logické bloky souboru.

2.1.3 Tvorba žurnálu

Další známé strategie využívají tvorby žurnálu, a to s bezprostřední nebo odloženou realizací změn. Žurnál (log) je obvykle zapisován do stabilní paměti.

Inkrementální tvorba žurnálu s bezprostřední realizací změn.

V této metodě je databáze modifikována na místě, ale před každou fyzickou změnou databáze je do žurnálu zapsán záznam obsahující čtveřici:

<*transakce, záznam, stará-hodnota, nová-hodnota*>

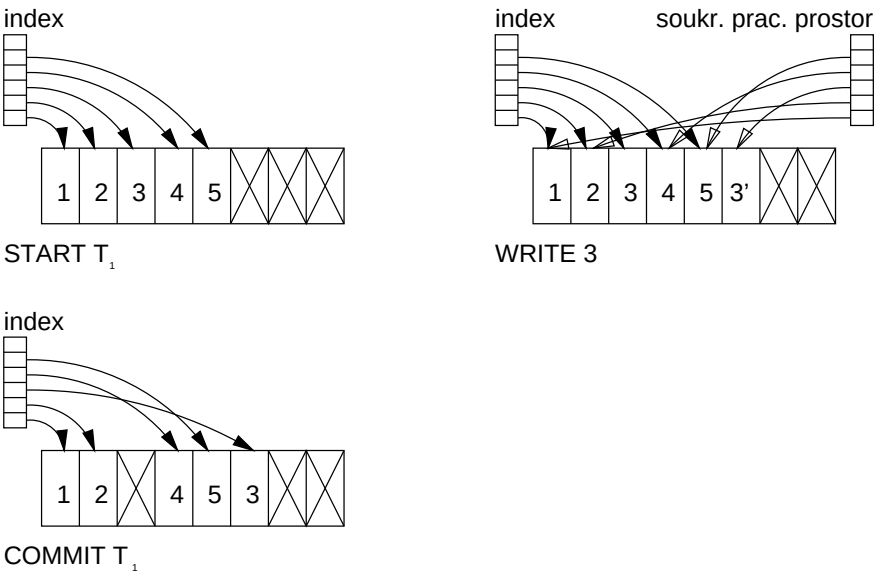
Při startu a dokončení transakce jsou do žurnálu zapsány záznamy

<*transakce,* START>

resp.

<*transakce,* COMMIT>.

Při dokončení transakce není třeba databázi modifikovat. V případě zrušení transakce je možné pomocí žurnálu dojít k původnímu stavu. Systém prochází žurnál od konce a ruší změny, které transakce provedla v databázi. Tato akce se nazývá **zpětný chod** (rollback).



Obrázek 6: Implementace soukromé pracovní oblasti.

Žurnál je také možno použít pro zotavení z chyb. Ve fázi zotavení systém projde žurnál, aby zjistil, zda v době havárie nebyla některá transakce aktivní a nedokončenou transakci odčiní. Dále je nutné provést znovu ty transakce, které byly před havárií úplné (existuje pro ně záznam COMMIT), ale nebyly dokončeny fyzicky (nebyl proveden zápis vyrovnávacích pamětí na disk).

Inkrementální tvorba žurnálu s odloženými realizacemi změn.

V této metodě se všechny změny zapisují do žurnálu, skutečné změny databáze jsou odloženy až do okamžiku dokončení transakce. Pro každou operaci změny je do žurnálu zapsána trojice

<*transakce*, *záznam*, *nová-hodnota*>.

Dokončení transakce zapíše záznam COMMIT a spustí zápis změn do databáze.

Předpokládejme, že zhroucení systému nastalo po zápisu záznamu COMMIT do žurnálu, ale před provedením fyzické změny v databázi. V takovém případě stačí znovu spustit zápis operací, zaznamenaných v žurnálu.

Některé systémy místo stabilní paměti využívají mechanismus **kontrolních bodů** (checkpoints), které se vytvářejí automaticky v jistých časových intervalech v průběhu provádění transakce. Kontrolní body zahrnují zápis vyrovnávacích pamětí na disk (flush) a zápis kontrolního záznamu (checkpoint record) do žurnálu. V takovém systému lze provést zotavení z chyb jen do posledního kontrolního bodu.

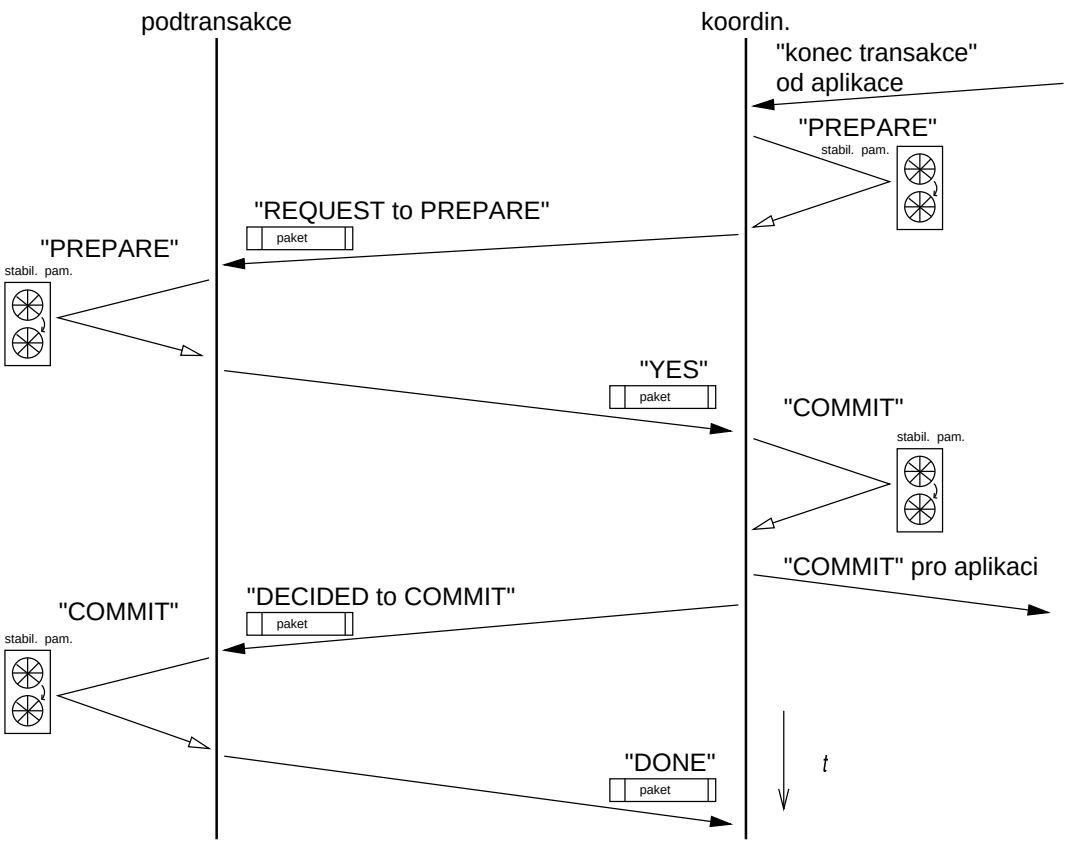
2.2 Dvoufázový potvrzovací protokol

Již bylo řečeno, že z vnějšího pohledu musí být transakce atomická. V distribuovaném databázovém systému však může potvrzení transakce vyžadovat spolupráci více podtransakcí běžících v různých uzlech, kde každý proces vlastní některé objekty modifikované transakcí. Pro atomické potvrzení distribuované transakce se nejčastěji používá **dvoufázový potvrzovací protokol** (viz obr. 7).

Jeden z procesů přebírá úlohu **koordinátora** globální transakce. Obvykle je to proces řídící primární transakci. Potvrzovací protokol začíná koordinátor zápisem záznamu o začátku potvrzovacího protokolu do žurnálu. Poté koordinátor pošle všem podřízeným transakcím zprávu, žádající podřízené transakce o potvrzení jejich připravenosti zveřejnit provedené změny.

Každý podřízený proces po přijetí zprávy rozhodne, zda může potvrdit dokončení transakce a své rozhodnutí zapíše do žurnálu a pošle o něm zprávu koordinátorovi. Ten po přijetí všech zpráv určí, zda má globální transakci dokončit nebo zrušit. Transakci je možné dokončit jen tehdy, jsou-li rozhodnutí všech podřízených kladná. Pokud některý proces rozhodnul záporně nebo neodpoví-li ve stanovené lhůtě, koordinátor transakci zruší.

Koordinátor zapíše do žurnálu záznam o svém rozhodnutí a pošle zprávu všem podřízeným transakcím. Podřízené procesy po přijetí zprávy zveřejní své změny.



Obrázek 7: Dvoufázové potvrzení.

Při havárii v kterémkoli kroku potvrzování je možné určit z žurnálu, ve kterém stavu se transakce v době havárie nacházela a případě provést příslušný krok znovu.

Transakce je dokončena již zápisem záznamu do žurnálu, a to bez ohledu na další události. Pokud koordinátor po zápisu havaruje, ve fázi zotavení pouze znovu pošle své rozhodnutí všem podřízeným. Pokud havaruje některý podřízený po odeslání odpovědi na první zprávu, koordinátor se bude bude opakovaně pokoušet zaslat mu zprávu o svém rozhodnutí.

V reálných systémech se dvoufázové potvrzování používá zřídka v této čisté podobě. Obvyklou optimalizací je, že dvoufázového protokolu se neúčastní transakce, skládající se pouze z operací READ.

3 Interference souběžných transakcí

V dosavadní diskusi jsme dosud neuvažovali souběžné provádění transakcí. Je zřejmé, že operace souběžných transakcí nelze prokládat libovolně, protože by mohlo dojít k porušení konzistence databáze. Dva příklady takové interference je ztracený zápis a nekonzistentní čtení.

Ztráta zápisu by nastala v případě, že by úspěšný zápis byl zvrácen zápisem jiné transakce. Nekonzistentní čtení by mohlo nastat v případě, že by transakci byly přístupné částečné výsledky některé nedokončené transakce.

Tyto problémy by nemohly nastat, kdyby SŘBD dovolil pouze sériový běh transakcí. Cílem SŘBD je však také zajistit co nejvyšší průchodnost, je třeba nalézt takové uspořádání operací, které dovolí souběžný běh transakcí bez vzájemné interference.

3.1 Uspořadatelnost rozvrhů

Jedním ze základních požadavků na provádění posloupnosti transakcí je tedy **uspořadatelnost** (serializability), tj. výsledek souběžně prováděných transakcí musí být stejný jako některé sériové provedení těchto transakcí. Stanovené pořadí provádění operací více transakcí v čase nazýváme rozvrhem.

Rozvrh S zapisujeme obecně

$$S = [O^1, O^2, O^3, \ldots O^m]$$

kde O^i znamená operaci READ nebo WRITE a O^i předchází v čase operaci O^j, jestliže $i < j$.

Sériový rozvrh je takový rozvrh, který zachovává operace každé transakce pohromadě. Abychom potvrdili správnost rozvrhu, stačí ukázat, že výsledek jím určené posloupnosti operací je ekvivalentní výsledku operací některého sériového rozvrhu.

Máme-li rozvrhy S_1 a S_2 pro transakce $T_1, \ldots, T_n$, pak rozvrhy S_1 a S_2 jsou **ekvivalentní**, jsou-li splněny tyto podmínky:

1. Jestliže v jenom rozvrhu je operace READ A v transakci T_i a její hodnota vznikla zápisem WRITE A v T_j, pak totéž musí být zachováno ve druhém rozvrhu.
2. Jestliže v jednom rozvrhu je poslední instrukce WRITE A v transakci T_i, pak totéž musí být zachováno ve druhém rozvrhu.

Rozvrh je **uspořadatelný**, jestliže pro něj existuje ekvivalentní sériový rozvrh. Každý uspořadatelný rozvrh je správný, mohou však existovat i správné rozvrhy, které nejsou ekvivalentní se žádným sériovým rozvrhem.

Testování uspořadatelnosti lze provádět algoritmicky, ale pro obecný rozvrh je to NP-úplný problém. Při zavedení dodatečné podmínky, že každému WRITE A předchází READ A, lze uspořadatelnost otestovat v čase $O(n^2)$ konstrukcí grafu závislosti transakcí.

Graf závislosti transakcí konstruujeme takto:

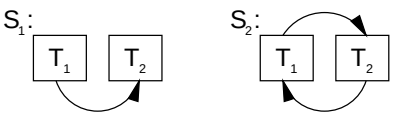
1. Do grafu vložíme uzel pro každou transakci.
2. Jestliže v rozvrhu je operace WRITE A transakce T_i dříve než operace READ A transakce T_j, vložíme hranu (T_j, T_i).
3. Jestliže v rozvrhu je operace READ A transakce T_i dříve než operace WRITE A transakce T_j, vložíme hranu (T_i, T_j).

Hrana (T_i, T_j) v grafu indikuje, že v kterémkoli ekvivalentním sériovém rozvrhu musí T_i předcházet T_j. Rozvrh je uspořadatelný, jestliže v jeho grafu závislosti transakcí není žádný cyklus.

Příklad:

$S_1 = [T_1, T_2]$

$S_2 = [\mathrm{READ}_1\ A, \mathrm{READ}_2\ A, \mathrm{WRITE}_2\ A, \mathrm{WRITE}_1\ A]$



3.2 Synchronizace přístupu k databázi

Protože distribuované transakce jsou prováděny souběžně různými procesy v různých uzlech, vzniká potřeba mechanismu synchronizace přístupu k databázi. V dalším textu popíšeme tři běžně používané metody synchronizace – dvoufázové zamykání, optimistické řízení a metodu časových razítek.

3.2.1 Zamykací protokoly

Zamykání je akce, kterou si proces vyhrazuje přístup k objektu. Zamykání a odemykání objektů je v databázových systémech prováděno obvykle automaticky prostřednictvím správce transakcí, takže nevyžaduje žádné úsilí ze strany aplikačního programátora.

Každý objekt musí být uzamčen dříve, než k němu bude transakce přistupovat. Uzamykací modul prohledá tabulku zámků a pokud objekt již není zamčen, jeho zamčení provede. V případě, že objekt nelze uzamknout, může proces čekat na uvolnění zámku (pak ale hrozí uvíznutí) nebo transakci zrušit.

V distribuovaných systémech může být správa zámků buď centrální nebo v každém uzlu pro lokání data. V případě centrálního uzamykání je možné uvíznutí detekovat pomocí konstrukce grafu vzájemných závislostí,[2] nevýhodou jsou vysoké komunikační náklady a zranitelnost systému výpadkem uzlu spravujícího zámky. Obvykle však lze graf vzájemných závislostí konstruovat bez přílišných přídavných nákladů alespoň pro lokální podtransakce. Pro globální transakce se často určuje časový limit, jehož vypršení se považuje za uvíznutí.

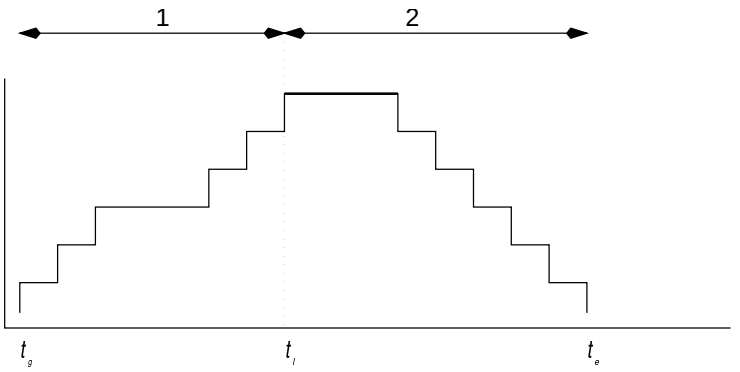
Mnohem lepší metodou je **dvoufázové zamykání**, ve kterém zámky alokujeme v nějakém kanonickém pořadí.[3] V první fázi transakce uzamyká všechny objekty, ke kterým bude přistupovat, ve druhé fázi zámky pouze uvolňuje.

V tomto základním schematu můžeme zvýšit stupeň paralelismu transakcí, pokud budeme rozlišovat zámky **sdílené** (pro operaci čtení) a **výlučné** (pro zápis). Pokud je záznam zamčen sdíleným zámkem a jiná transakce k němu potřebuje přístup pro čtení, může jej také zamknout sdíleným zámkem. Sdílené zámky zajistí, že záznam nebude modifikován (vyloučí všechny snahy o zápis), ale dovolí ostatním transakcím záznam číst. Pokud je záznam zamčen výlučným zámkem, žádné další zamčení již není možné.

Samotné dvoufázové zamykání je poněkud silnější, než je nutné. Pokud transakce záznam pouze čte, může ho uvolnit, jakmile ho přestane potřebovat. Pokud byl záznam naopak modifikován, jeho uvolnění je možné až ve druhé fázi.

[2] Prevence a detekce uvíznutí je klasické téma operačních systémů, viz např. [6], kapitola 3.

[3] Kanonické pořadí alokace zámků je postačující podmínka pro zabránění uvíznutí.



Obrázek 8: Dvoufázové zamykání. Počet zámků alokovaných procesem je největší v bodě t_l, tzv. bodě uzamčení.

Předpokládejme, že by transakce zámek uvolnila po provedení zápisu. Některá jiná transakce mohla záznam zamknout, provést nějakou operaci a transakci potvrdit. Kdyby pak byla původní transakce zrušena, musela by být zrušena i druhá (již potvrzená) transakce, protože prováděla operaci nad hodnotami, které neměly existovat. Této situaci se říká **dominový efekt** (cascaded abort) a jejímu vzniku se snažíme zamezit např. již popsaným dvoufázovým zamykáním.

Dvoufázové zamykání se používá velmi často, mimo jiné i proto, že pokud všechny transakce používají dvoufázové zamykání, je každý jejich legální rozvrh uspořadatelný.

3.2.2 Optimistické řízení

Myšlenka metody optimistického řízení je jednoduchá – transakce provádí čtení a zápis podle potřeby, operace zápisu se provádějí do soukromé pracovní oblasti. Případné konflikty se řeší až ve fázi potvrzení.

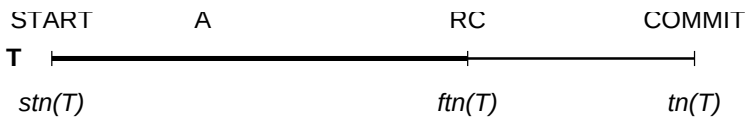
Každá transakce musí uchovávat informaci o množině čtených a zapisovaných záznamů. Transakci je možné potvrdit, jestliže pro všechny starší transakce platí alespoň jedna z následujících podmínek:

- Starší transakce skončila před spuštěním testované transakce.
- Starší transakce skončila dříve, než aktuální transakce zveřejní své změny a množina záznamů čtených aktuální transakcí je disjunktní s množinou záznamů, které zapsala starší transakce. To znamená, že aktuální transakce nemohla přepsat starší transakci (starší transakce skončila dříve), a data zapsaná starší transakcí neovlivnila aktuální transakci.
- Starší transakce přejde do fáze potvrzení dříve, než aktuální transakce, a množina záznamů čtených nebo zapisovaných aktuální transakcí je disjunktní s množinou záznamů, zapsaných starší transakcí.

Pro testování výše uvedených podmínek se zavádí časová razítka. Při spuštění transakce a při přechodu do fáze potvrzení jsou transakci přiřazena časová razítka $stn(T)$ a $fnt(T)$, která se používají pouze v potvrzovací fázi. Po potvrzení je transakci přiřazeno definitivní časové razítko $tn(T)$.

Výše uvedené podmínky pak můžeme testovat podle následujícího algoritmu:

Nechť $R(T)$ a $W(T)$ je množina dat čtených, resp. zapisovaných transakcí T. Pak aktuální transakce T_a je platná, platí-li pro každou již potvrzenou transakci T_i alespoň jedna z následujících podmínek:



Obrázek 9: Časová razítka přiřazená transakci v metodě optimistického řízení.

- $tn(T_i) \prec stn(T_a)$
- $tn(T_i) \prec ftn(T_a) \wedge W(T_i) \cap R(T_a) = \emptyset$
- $ftn(T_i) \prec ftn(T_a) \wedge W(T_i) \cap (R(T_a) \cup W(T_a)) = \emptyset$

V distribuovaných systémech se nejprve ověřují lokální podtransakce potvrzované transakce. Poté je provedeno globální ověření, zda je uspořádání podtransakcí ve všech lokálních uzlech stejné.

Pokud nenastal konflikt, transakce zviditelní svá data, v opačném případě je pracovní prostor zrušen a transakce spuštěna znovu.

Metoda optimistického řízení dovoluje maximální paralelismus, protože transakce na sebe nemusí nikdy čekat. Metoda je vhodná, pokud jsou transakce vzdáleny v čase a dominují operace čtení. Se zvyšujícím se zatížením systému však roste pravděpodobnost konfliktu, což prudce zhoršuje výkonnost systému. Proto ji žádný z běžných SŘBD nepoužívá.

3.2.3 Metoda časových razítek

Další optimistickou metodou je metoda časových razítek. Každé transakci T přiřadíme v okamžiku spuštění jedinečné časové razítko $ts(T)$, např. pomocí Lamportova algoritmu pro uspořádání událostí v distribuovaném systému.

Transakce zapisují do soukromé pracovní oblasti. S každým záznamem x je sdruženo časové razítko $ts_r(x)$ pro čtení a $ts_w(x)$ pro zápis, které říká, která transakce záznam četla nebo zapisovala naposledy.

Pokud chce transakce záznam číst a záznam byl modifikován některou mladší transakcí (tj. $ts(T_a) \prec ts_w(x)$), je transakce zrušena. Pokud některá starší nepotvrzená transakce záznam modifikovala ve svém soukromém pracovním prostoru, musí operace READ čekat do potvrzení nebo zrušení této starší transakce. Provedení operace READ pak nastaví novou hodnotu časového razítka $ts_r(x)$.

Při pokusu o zápis je transakce zrušena, pokud byl záznam zapisován některou mladší transakcí (tj. pokud $ts(T_a) \prec ts_r(x) \vee ts(T_a) \prec ts_w(x)$). Provedení zápisu nastaví novou hodnotu razítka $ts_w(x)$.

Aby byl dodržen požadavek uspořadatelnosti, musí být při operaci COMMIT zviditelnění dat z pracovní oblasti provedeno v pořadí časových razítek transakcí. Pokud má potvrzovaná transakce ve své pracovní oblasti záznam, který ve své pracovní oblasti modifikovala některá starší, dosud nepotvrzená transakce, musí transakce čekat na potvrzení nebo zrušení této starší transakce.

Metoda časových razítek produkuje uspořadatelné rozvrhy, ekvivalentní rozvrhu, definovanému posloupností potvrzených transakcí v pořadí jejich časových razítek.

4 Závěr

Zpracování transakcí je centrální součástí současných databázových systémů. V tomto článku jsme provedli diskusi některých metod, kterými se transakční zpracování implementuje.

Literatura

1. *Bell, David – Grimson, Jane:* **Distributed Database Systems.** Addison Wesley, Reading 1992.
2. *Lain, William A. – Johnson, James E. – Landau, Robert V.:* **Transaction Management Support in the VMS Operating System Kernel.** Digital Technical Journal, vol. 3 no. 1 (Winter 1991): 33–44.
3. **Oracle7 Server Distributed Systems, vol. I: Ditributed Data.** On-line manuál. April 1995.
4. *Pokorný, Jaroslav:* **Počítačové databáze.** Kancelářské stroje, Praha 1991.
5. *Sokolowsky, Peter – Pokorný, Jaroslav – Peterka Jiří:* **Distribuované databázové systémy.** Skripta MFF UK, Praha 1992.
6. *Tanenbaum, Andrew S.:* **Modern Operating Systems (2nd ed.).** Prentice-Hall, Upper Saddle River 2001.