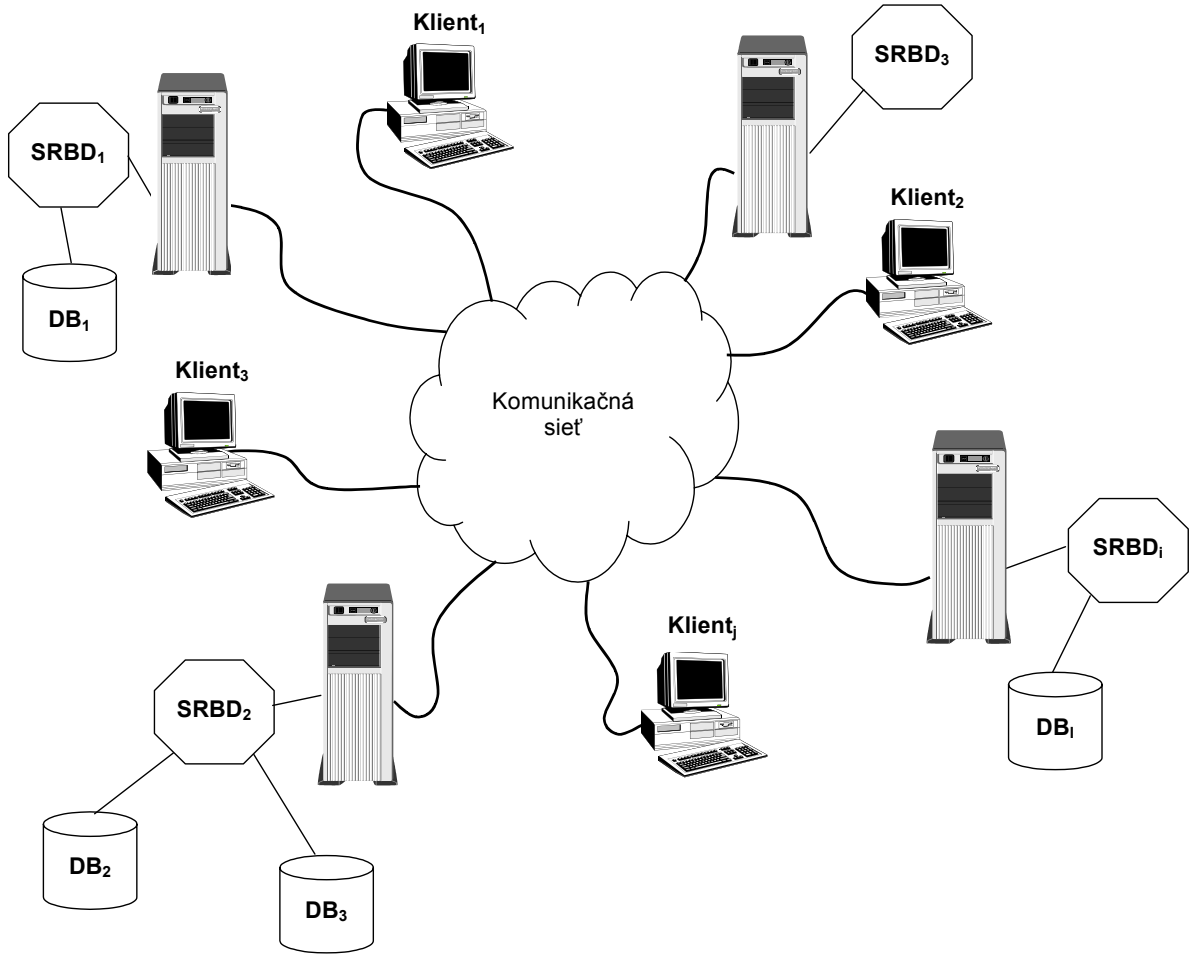
12. Distribuované databázové systémy

Definícia – Distribuovaný databázový systém
Distribuovaný databázový systém je množina uzlov počítačovej siete, navzájom prepojených v komunikačnej sieti, pričom:

- každý z uzlov je samostatný databázový systém, ale tieto uzly navzájom spolupracujú tak, že z každého uzla je možné sprístupniť údaje uložené na inom uzle presne tak, akoby boli umiestnené na vlastnom uzle
- samotná databáza je množina navzájom prepojených databáz, ktoré sú umiestnené na rôznych uzloch tak, že užívateľ s nimi manipuluje akoby boli umiestnené v centralizovanej databáze.

♦



Obr. 12.1 Architektúra distribuovaného databázového systému

Jeho hlavné funkcie sú:

1. Riadenie globálnych katalógov dát o distribuovaných dátach
2. Definovanie distribuovaných dát
3. Distribuovaná sémantická kontrola
4. Distribuované spracovanie dotazov, vrátane optimalizácie dotazov a vzdialeného prístupu k dátam
5. Distribuované riadenie transakcií, vrátane zabezpečenia paralelného spracovania.

12.1 Dvanásť vlastností DDBS

1. Lokálna autonómnosť
2. Nezávislosť od centrálnehoo uzla
3. Súvislá prevádzka
4. Nezávislosť od umiestnenia dát
5. Nezávislosť od fragmentácie dát
6. Nezávislosť od replikácie dát
7. Spracovanie distribuovaných dotazov
8. Distribuované riadenie transakcií
9. Nezávislosť od technického vybavenia
10. Nezávislosť od operačného systému
11. Nezávislosť od siete
12. Nezávislosť od lokálnych databázových systémov

12.2 Klasifikácia DDBS

Definícia – Homogénny DDBS
DDBS, v ktorom všetky lokálne databázy sú riadené rovnakým SRBD, je homogénny DDBS.

♦

Definícia – Heterogénny DDBS
DDBS, v ktorom všetky lokálne databázy nie sú riadené rovnakým SRBD, je heterogénny DDBS.

♦

Poznámka
Definície homogenity a heterogenity DDBS sa v niektorej literatúre líšia podľa toho, či sa vyžaduje aj rovnorodosť operačného systému, prípadne aj hardwaru, na ktorom je DDBS prevádzkovaný.

12.3 Výhody a nevýhody DDBS

Výhody:

- lokálna autonómnosť
- zvýšenie výkonu
- zvýšenie spoľahlivosti
- zvýšenie dostupnosti dát
- vyššia zdieľatelnosť dát
- rozširovatelnosť
- ekonomická efektívnosť

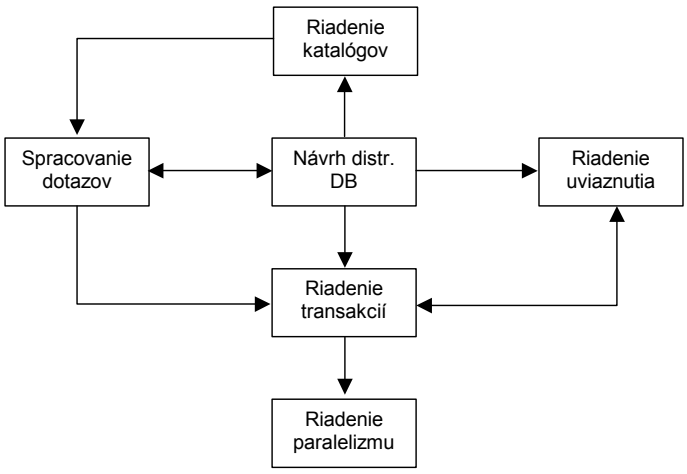
Samozrejme, že distribuované spracovanie dát prináša aj celý rad **nevýhod**, ktoré možno zhrnúť nasledovne:

- nedostatok skúseností
- zložitosť nového systému
- zvýšenie ceny
- riadenie distribuovaného spracovania
- vyššie nároky na bezpečnosť

12.4 Klasifikácia problémov DDBS

Tieto problémy by sme mohli klasifikovať nasledovne:

1. Návrh distribuovaných databáz
2. Distribuované spracovanie dotazov
3. Distribuované riadenie katalógov
4. Distribuované riadenie transakcií
5. Distribuované riadenie paralelného spracovania transakcií



Obr. 12.2 Problémy DDBS

12.4.1 Návrh distribuovaných databáz

Z dôvodu, že aplikácie využívajúce DDBS môžu využívať dáta umiestené na viacerých uzloch, je možné dáta rozdeliť na:

- Replikované
- Nereplikované

V prípade, že uvažujeme s **replikáciou** databáz môžeme hovoriť o:

- Plnej replikácii – ak replikujeme celú databázu na viacej uzlov
- Čiastočnej replikácii – ak replikujeme len časť databázy

Naviac niektoré objekty môžeme deliť na menšie časti, tzv. **fragmenty** a podľa toho, ako sú fragmenty umiestnené, delíme dáta na:

- Fragmentované
- Nefragmentované

12.4.2 Distribuované spracovanie dotazov

Distribuované spracovanie dotazov sa týka návrhu samotných algoritmov, ktoré analyzujú požiadavky a transformujú ich na množinu úloh, zabezpečujúcich vykonanie príslušnej požiadavky. Ide o problém návrhu stratégie vykonania každého dotazu v DDBS s ohľadom na minimalizáciu nákladov (času vykonania).

12.4.3 Distribuované riadenie katalógov

Katalógy sú metadáta v DDBS. Obsahujú schémy lokálnych databáz, ktoré sú súčasťou DDBS a vlastne reprezentujú samostatnú distribuovanú databázy, pre ktorú je nutné navrhnúť umiestnenie katalógov dát. Tieto katalógy sú často replikované na všetky uzly DDBS a takisto existujú riešenia, keď katalógy sú centralizované na jednom z uzlov a poskytované ďalším uzlom podľa požiadaviek.

12.4.4 Distribuované riadenie transakcií

Distribuované riadenie transakcií prináša nové problémy súvisiace so zabezpečením obnovy databázy v prípade chýb, alebo havárie spracovania aplikácií. Pretože v DDBS transakcia môže spúšťať podtransakcie na viacerých uzloch, je nutné koordinovať prácu podtransakcií a zabezpečiť konzistenciu DB aj v prípade havárie jednej z jej podtransakcií.

12.4.5 Distribuované riadenie paralelného spracovania transakcií

Paralelné spracovanie transakcií v DDBS prináša nutnosť synchronizovať všetky prístupy k distribuovanej databáze tak, aby bola zabezpečená konzistencia DB. Zabezpečenie paralelizmu v DDBS je oveľa zložitejšie ako v centralizovaných DBS.

Podobne, ako v centralizovaných DBS, aj tu sa využívajú techniky zamykania a časových pečiatok, avšak s nutnosťou zabezpečenia synchronizácie operácií.

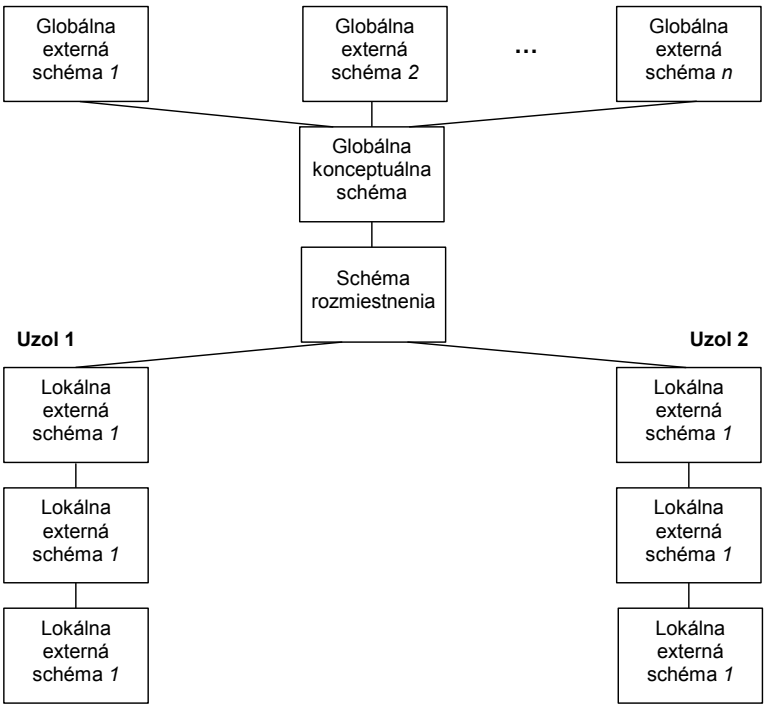
Špeciálnym stavom systému je situácia, keď je potrebné zabrániť uviaznutiu systému.

12.5 Architektúra databáz pre distribuované spracovanie

Základné rozdelenie uvažuje definovanie:

- globálnej architektúry
- množiny lokálnych architektúr pre jednotlivé uzly DDBS

Globálna architektúra reprezentuje globálny pohľad užívateľa s tým, že globálna konceptuálna schéma je rozšírená o schému rozmiestnenia jednotlivých lokálnych databáz. Úrovne tejto schémy popisujú distribuovanú nezávislosť databázy od jednotlivých uzlov. Globálna konceptuálna schéma definuje všetky vzťahy, ktoré sa vyskytujú v DDBS. Schéma rozmiestnenia nám popisuje, akým spôsobom sú rozmiestňované dáta na jednotlivých uzloch, hovorí nám aj o spôsobe fragmentácie a replikácie databáz. Každý z uzlov DDBS má architektúru lokálneho DBS v zmysle základnej definície architektúry DBS.



Obr. 12.3 Základná architektúra DDBS

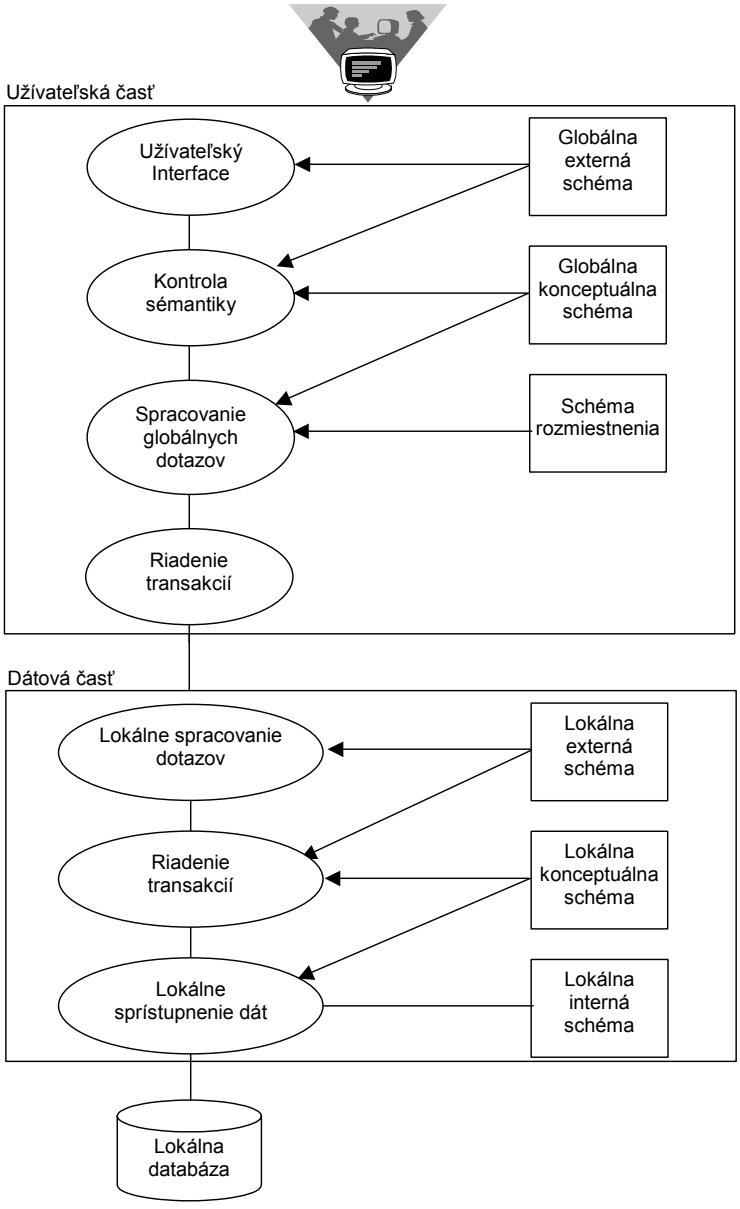
Typickú architektúru distribuovaného SRBD predstavuje obrázok 12.4, ktorý rozdeľuje funkčnosť DDBS na dve časti:

- Užívateľskú
- Dátovú

Užívateľská časť zabezpečí analýzu požiadavky a návrat výsledkov užívateľovi. Sémantickú kontrolu dát vyplývajúcich z požiadavky, autorizáciu a kontrolu integritných obmedzení zabezpečí sémantická dátová kontrola, na základe ktorej sa zabezpečí dekompozícia dotazu na množinu úloh, ktoré budú spustené na jednom alebo viacerých uzloch. Spustenie týchto úloh zabezpečí transakčný manažér, ktorý spustí potrebné množstvo podtransakcií na uzloch, podľa rozmiestnenia dát.

Dátová časť zabezpečí spracovanie lokálneho dotazu, ktorý bol definovaný pre príslušnú podtransakciu, dekomponuje ho a optimalizuje lokálnu požiadavku. Transakčný manažér zabezpečí spracovanie tejto požiadavky, vrátane komunikácie s hlavnou

transakciou. Transakčný manažér taktiež zabezpečí komunikáciu s operačným systémom, ktorý sprístupní požadované dáta príslušnej lokálnej databázy s pomocou lokálne definovanej organizácie dát a prístupových metód, vrátane monitorovania transakcií do logických žurnálov.



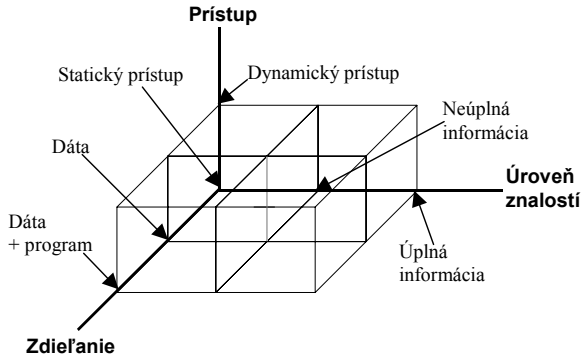
Obr. 12.4 Základná architektúra distribuovaného SRBD

12.6 Modelovanie distribuovaných databáz

12.6.1 Problematika modelovania databáz v distribuovanom informačnom systéme alokovania databáz (Database Allocation Problem (DAP))

Levin a Morgan v roku 1975 [LEVIN75] navrhli tvorbu distribuovaných systémov podľa troch hľadísk (viď Obr. 12.5):

1. **Zdieľanie** - zdieľanie dát, alebo programov
2. **Spôsob prístupu** – statický alebo dynamický pohľad na aplikáciu
3. **Úroveň poznania** - znalosť návrhára databázy, akým spôsobom bude užívateľ pristupovať k databáze



Obr. 12.5 Úrovne zložitosti DDBS

V súčasnej dobe väčšina prác zaoberajúcich sa danou problematikou je sústredená na problém návrhu dátového modelu v nasledovných krokoch a to:

- **Návrh konceptuálnej schémy databázy**
- **Návrh fyzickej databázy**
- **Fragmentácia dátového modelu a objektov databázy**
- **Alokácia fragmentov databázy**

12.6.2 Fragmentácia dátového modelu

- **Horizontálnou fragmentáciou**
- **Vertikálnou fragmentáciou**

Ich kombináciou je možné vytvoriť tzv**. hybridnú fragmentáciu**.

12.6.2.1 Podmienky fragmentácie

1. Kompletnosť
2. Rekonštruovateľnosť
3. Nespojiteľnosť

Kompletnosť

Definícia - Kompletnosť
Ak výskyt relácie ***R*** je dekomponovaný na fragmenty $R_1, R_2, \ldots R_n$, potom každá položka sa vyskytuje v ***R*** a taktiež sa vyskytuje aspoň v jednom z fragmentov R_i.

♦

Poznámka
V prípade horizontálnej fragmentácie je položkou celá n-tica a v prípade vertikálnej fragmentácie atribút.

Rekonštruovateľnosť

Definícia - Rekonštruovateľnosť
Ak relácia ***R*** je dekomponovaná na fragmenty $R_1, R_2, \ldots R_n$, malo by byť možné definovať relačný operátor ∇ taký, že platí

$R = \nabla R_i \quad \forall i = 1,2,\ldots n$

♦

Operátor ∇ sa líši podľa spôsobu fragmentácie. Je iný pre horizontálnu a iný pre vertikálnu fragmentáciu.

Rekonštruovateľnosť globálnej relácie z jej fragmentov zaisťuje zachovanie integritných obmedzení definovaných pre dáta relácie.

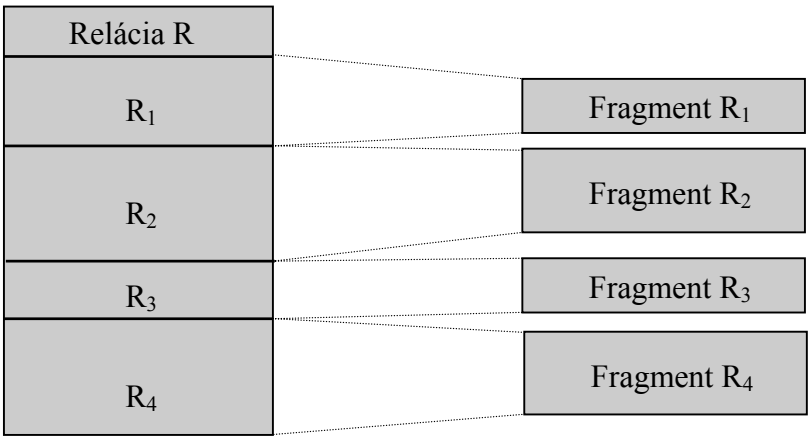
Nespojiteľnosť

Definícia - Nespojiteľnosť
Ak relácia ***R*** je horizontálne dekomponovaná na fragmenty $R_1, R_2, \ldots, R_n$ a dátová položka d_i patrí do fragmentu R_j, potom sa nevyskytuje v žiadnom z fragmentov R_k $(k<>j)$.

♦

Poznámka
Upozorňujeme, že táto definícia má vzťah len k horizontálnej fragmentácii.

12.6.2.2 Horizontálna fragmentácia



Obr. 12.6 Horizontálna fragmentácia relácie R na fragmenty R_1, R_2, R_3, R_4

Definícia – Horizontálna fragmentácia

Relácia ***R*** je horizontálne dekomponovaná na fragmenty $R_1, R_2, \ldots R_n$, pričom

$$R_i = \sigma_{Ci}(R) \quad \forall i = 1,2,\ldots n$$

a pre rekonštrukciu relácie platí

$$R = Union\ (R_i) \quad \forall i = 1,2,\ldots n$$

kde

σ - je operácia relačnej algebry výber s podmienkou C_i v konjuktívno normálnej forme pre daný fragment R_i.

Union - je operácia relačnej algebry zjednotenie.

♦

Príklad 12.1

Máme rozdeliť reláciu STUDENT na tri fragmenty ST_Z, ST_PD, ST_R, podľa pracoviska, ktoré študent navštevuje (Žilina, Prievidza, Ružomberok). Informáciu, ktoré pracovisko študent navštevuje, získame z druhého znaku študijnej skupiny. Pri fragmentácii teda dostávame tri definície:

```
DEFINE FRAGMENT ST_Z AS
            SELECT *
              FROM Student
              WHERE st_skupina[2,2] = "Z";

DEFINE FRAGMENT ST_PD AS
            SELECT *
              FROM Student
              WHERE st_skupina[2,2] = "P";

DEFINE FRAGMENT ST_R AS
            SELECT *
              FROM Student
              WHERE st_skupina[2,2] = "R";
```

a pre vytvorenie pôvodnej relácie musí platiť

```
CREATE VIEW Student AS
SELECT * FROM ST_Z
      UNION
SELECT * FROM ST_PD
      UNION
SELECT * FROM ST_R;
```

Pre fragmentovanie relácií, ktoré neobsahujú atribúty, podľa ktorých by bolo možné fragmentovať, sa vykonáva, tzv. **odvodená fragmentácia**.

Odvodená horizontálna fragmentácia

Definícia – Odvodená horizontálna fragmentácia

Relácia ***R`*** je odvodene horizontálne dekomponovaná na fragmenty $R_1, R_2, \ldots R_n$, ktoré vzniknú z relácie ***R***, pre ktorú platí, že ju získame projekciou a polospojením (SEMIJOIN) relácie ***R`***, ktorú fragmentujeme a relácie ***S***, o ktorej vieme, že obsahuje atribúty, podľa ktorých je možné fragmentáciu vykonať.

$$R = \pi_{R`}(R` \bowtie_K S) \quad \forall i = 1,2,\ldots n$$

$$R_i = \sigma_{Ci}(R) \quad \forall i = 1,2,\ldots n$$

alebo

$$R_i = \sigma_{Ci}(\pi_{R`}(R` \bowtie_K S)) \quad \forall i = 1,2,\ldots n$$

a pre rekonštrukciu relácie platí

$$R` = Union\ (R_i) \quad \forall i = 1,2,\ldots n$$

kde

π- je operácia relačnej algebry projekcia

$\bowtie_K$ - operácia relačnej algebry spojenie cez množinu atribútov K

σ_{Ci} - je operácia relačnej algebry výber s podmienkou C_i v konjuktívno normálnej forme pre daný fragment R_i

Union - je operácia relačnej algebry zjednotenie

♦

Príklad 12.2

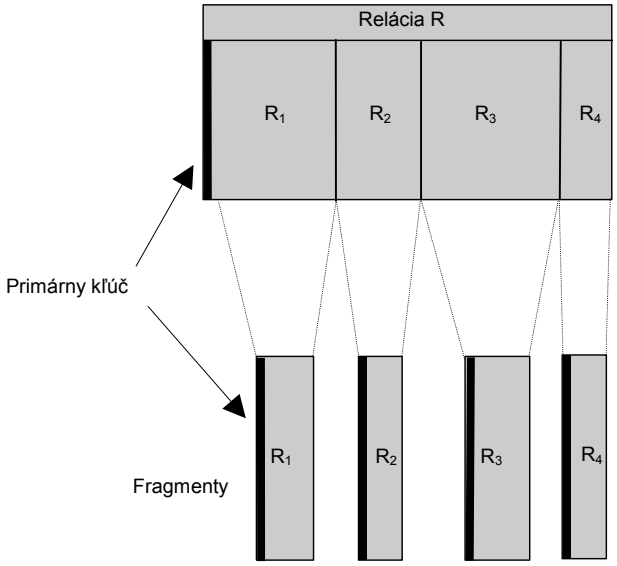
Podobne ako v predchádzajúcom príklade kritériom rozdelenia relácií bude pracovisko, ktoré študent navštevuje. Tento krát však nebudeme fragmentovať reláciu Student, v ktorej sa nachádza informácia o pracovisku študenta, ale reláciu ZAP_PREDMETY.

```
DEFINE FRAGMENT ZP_Z AS
            SELECT zp.*
              FROM Student st, zap_predmety zp
              WHERE st_skupina[2,2] = "Z";
DEFINE FRAGMENT ZP_PD AS
            SELECT zp.*
              FROM Student st, zap_predmety zp
              WHERE st_skupina[2,2] = "P";

DEFINE FRAGMENT ZP_R AS
            SELECT zp.*
              FROM Student st, zap_predmety zp
              WHERE st_skupina[2,2] = "R";
```

a pre vytvorenie pôvodnej relácie musí platiť

```
CREATE VIEW zap_predmety AS
SELECT * FROM ZP_Z
      UNION
SELECT * FROM ZP_PD
      UNION
SELECT * FROM ZP_R;
```

12.6.2.3 Vertikálna fragmentácia



Obr. 12.7 Vertikálna fragmentácia relácie R na fragmenty R_1, R_2, R_3, R_4

Definícia – Vertikálna fragmentácia
Relácia ***R*** je vertikálne dekomponovaná na fragmenty $R_1, R_2, \ldots R_n$, pričom

$$R_i = \pi(R) \qquad \forall i = 1,2,\ldots n$$

a pre rekonštrukciu relácie platí

$$R = JOIN_K(R_i) \qquad \forall i = 1,2,\ldots n$$

kde

π - je operácia relačnej algebry projekcia.
$JOIN_K$ - operácia relačnej algebry spojenie cez množinu atribútov ***K***.

♦

Príklad 12.3
Pri rozdelení relácie Os_udaje (#rod_cislo, meno, priezvisko, ulica, psc, mesto) na dva fragmenty OU_1,OU_2, máme dve definície, v ktorých sú použité dve navzájom disjunktné podmnožiny atribútov s podmienkou, že primárny kľúč musí byť súčasťou každého nového fragmentu.

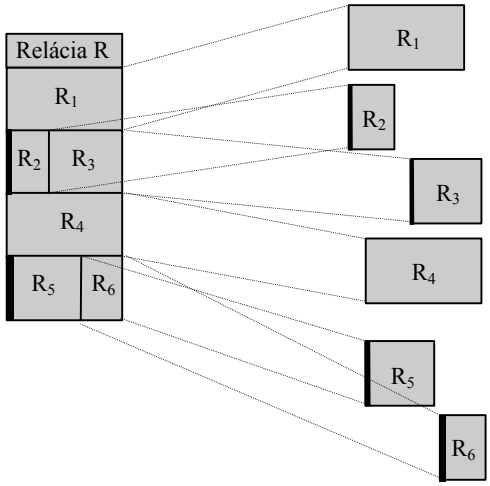
```
DEFINE FRAGMENT OU1 AS
        SELECT rod_cislo, meno, priezvisko
        FROM Os_udaje;

DEFINE FRAGMENT OU2 AS
        SELECT rod_cislo, ulica, psc, mesto
         FROM Os_udaje;
```

a pre vytvorenie pôvodnej relácie musí platiť

```
CREATE VIEW Os_udaje AS
        SELECT OU1.rod_cislo, meno, priezvisko, ulica, psc, mesto
        FROM OU1, OU2
        WHERE OU1.rod_cislo = OU2.rod_cislo;
```

12.6.2.4 Hybridná fragmentácia

- Horizontálne vertikálnu fragmentáciu
- Vertikálne horizontálnu fragmentáciu



Obr. 12.8 Hybridná fragmentácia relácie R na fragmenty R_1, R_2, R_3, R_4, R_5, R_6

12.6.2.5 Horizontálne vertikálna fragmentácia

Definícia – Horizontálne veritkálna fragmentácia
Relácia ***R*** je horizontálne dekomponovaná na fragmenty $R_1, R_2, \ldots R_n$, pričom
$R_i = \sigma_{Ci}(R) \quad \forall i = 1,2,\ldots n$
a každý fragment R_i je vertikálne dekomponovaný na fragmenty R_{ij}
$R_{ij} = \pi (R_i) \quad \forall j = 1,2,\ldots m$
a pre rekonštrukciu relácie platí
$R_i = JOIN_K (R_{ij}) \quad \forall j = 1,2,\ldots m$
$R = Union (R_i) \quad \forall i = 1,2,\ldots n$

♦

12.6.2.6 Vertikálne horizontálna fragmentácia

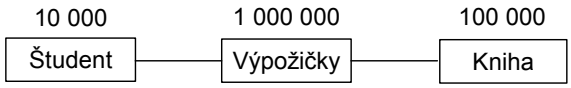
Definícia – Veritkálne horizontálna fragmentácia
Relácia ***R*** je vertikálne dekomponovaná na fragmenty $R_1, R_2, \ldots R_n$, pričom
$R_i = \pi (R) \quad \forall i = 1,2,\ldots n$
a každý fragment R_i je horizontálne dekomponovaný na fragmenty R_{ij}
$R_{ij} = \sigma_{Cj}(R_i) \quad \forall j = 1,2,\ldots m$
a pre rekonštrukciu relácie ***R*** platí
$R_j = Union (R_{ij}) \quad \forall j = 1,2,\ldots m$
$R = JOIN_K (R_i) \quad \forall i = 1,2,\ldots n$

♦

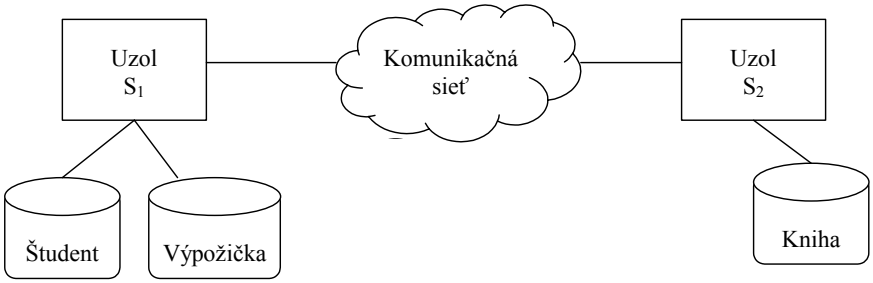
12.7 Spracovanie dotazov

Príklad 12.4

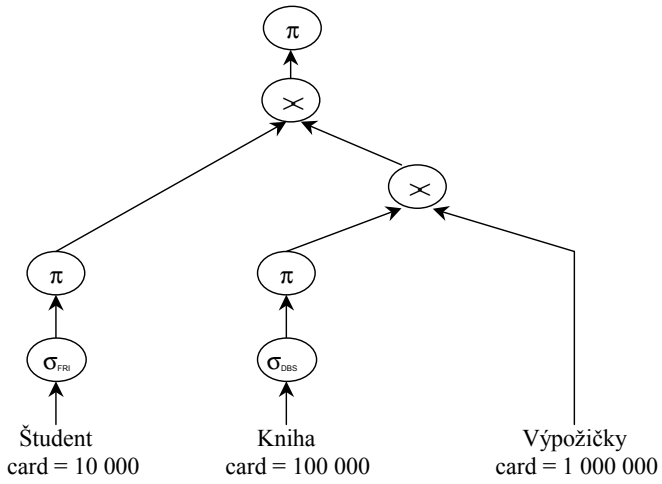
Majme databázu o univerzite s tabuľkami:
Študent(os_číslo, fakulta, meno, priezvisko, ...) s kardinalitou 10 000 riadkov na uzle S_1 ,
Kniha(ev_číslo, názov, ...) s kardinalitou 100 000 riadkov na uzle S_2 ,
Výpožičky (os_číslo, ev_číslo, dátum_pôžičky, dátum_vrátenia, ...) s kardinalitou 1 000 000 riadkov na uzle S_1 .



Formulujem dotaz, ktorým chcem sprístupniť zoznam osobných čísel študentov, ktorí mali požičanú knihu obsahujúcu v názve reťazec „Databázové systémy“ a sú z fakulty FRI (fakulta riadenia a informatiky).



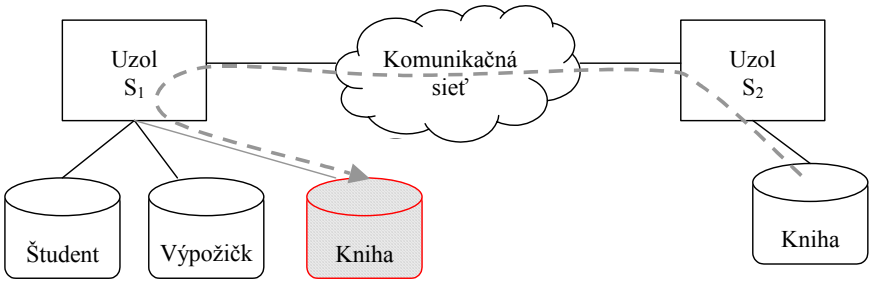

```
SELECT os_číslo FROM študent, výpožičky, Kniha
      WHERE Študent.os_číslo=Výpožičky.os_číslo AND
             Výpožičky.ev_číslo=Kniha.ev_číslo AND
             Kniha.názov  LIKE „%databázové systémy%” AND
             Študent.fakulta = “FRI”
```

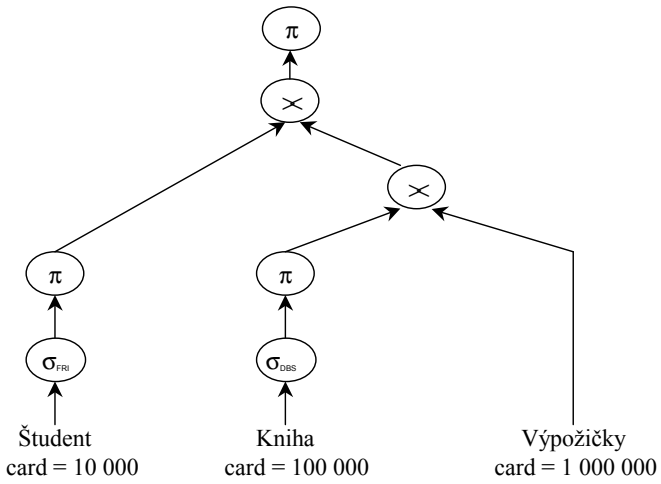


Odhad počtu kníh relevantných s názvom je 10.
Počet výpožičiek študentov fakulty riadenia a informatiky je 100 000.
Prenosová rýchlosť je 50 000 bitov/sek.
Oneskorenie pri odosielaní správ medzi uzlami siete je 0,1 sek.

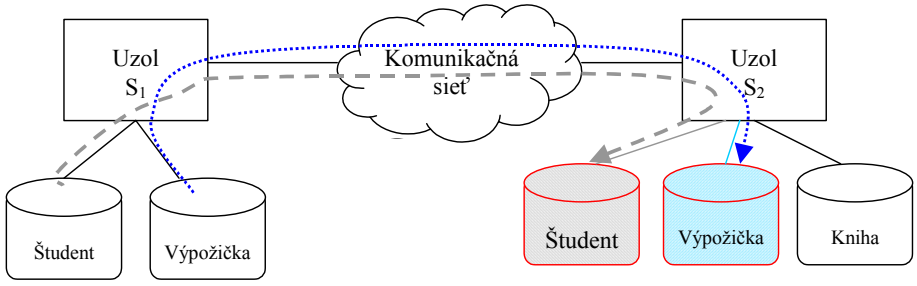
T = celková doba na odoslanie správ + doba prenosu dát

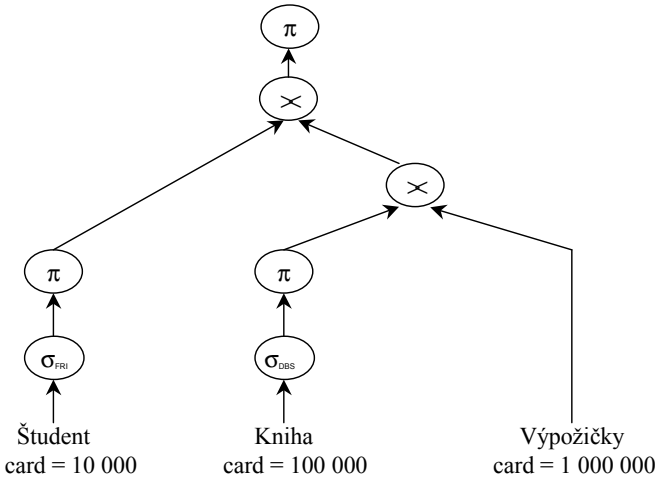
$$T_1 = t_{AD} * počet\ správ + OD \div Vp = t_{AD} + t_{prenos} = 0.1 + (100000 * 200) \div 50000 = 400\,\text{sek} = 6.67\,\text{min}$$



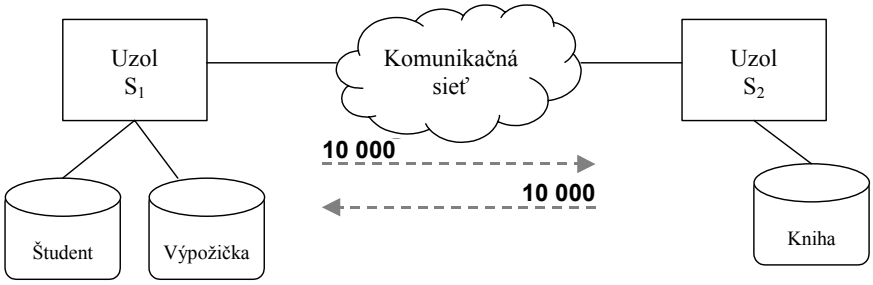


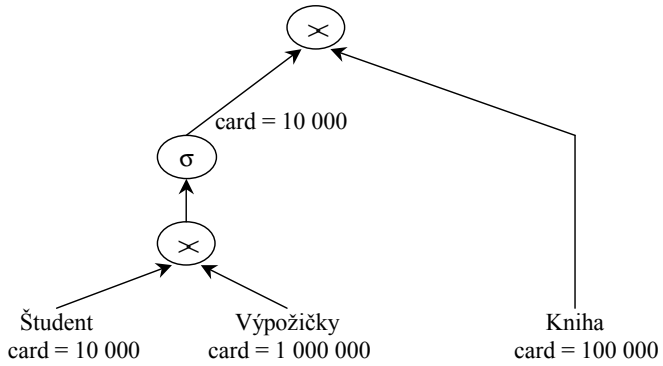
$$T_2 = t_{AD} * počet\ správ + OD \div Vp = 2 * t_{AD} + OD \div Up =$$
$$2 * 0.1 + ((10\,000 + 100\,000) * 200) \div 50\,000 = 4040\ \text{sek} = 1.26\ \text{hod}$$



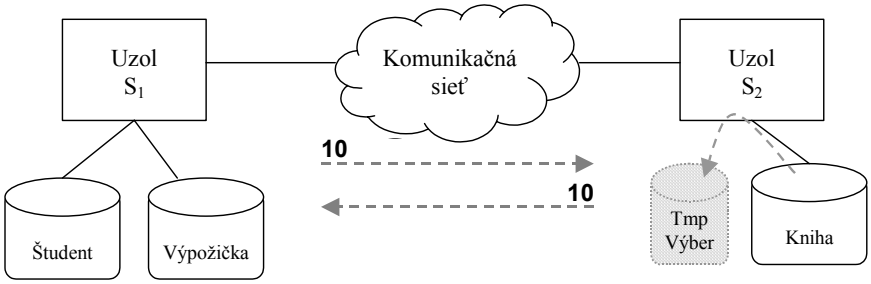


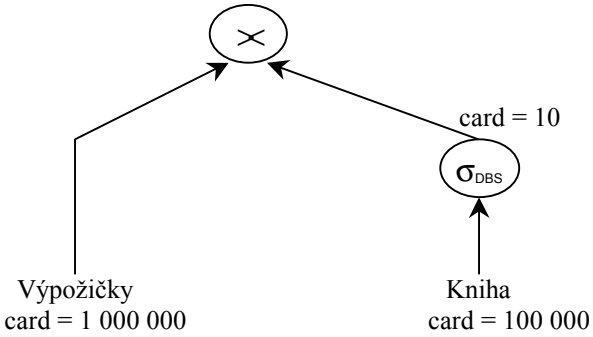
$T_3 = t_{AD} * počet\ správ + OD \div Vp = 2 * počet\ správ = 2 * 10\,000 = 20\,000$ sek $= 5.56$ hod



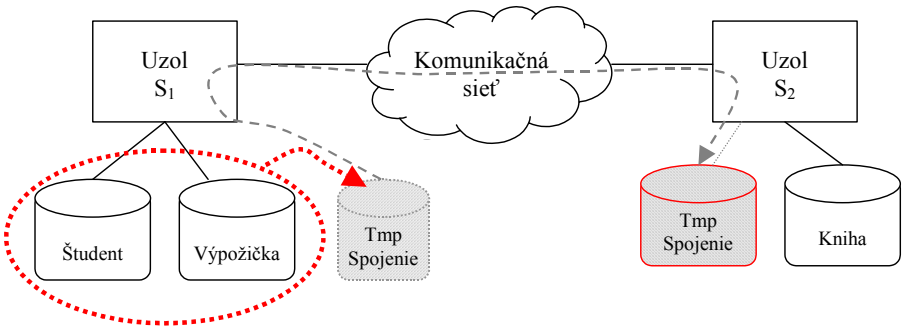


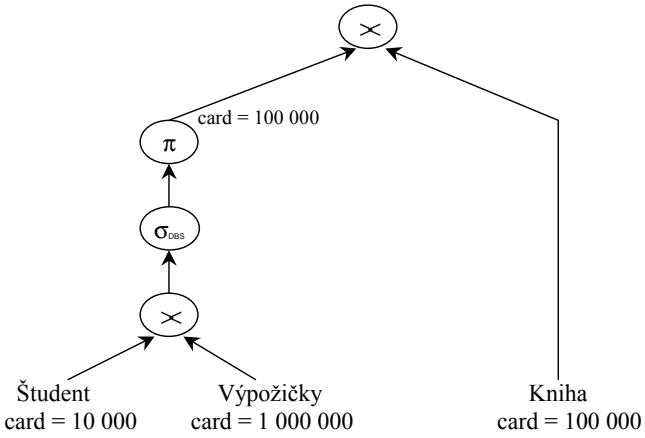
$T_4 = t_{AD} * počet\ správ + OD \div Vp = 2 * t_{DA} * počet\ správ = 2 * 0.1 * 10 = 2$ sek



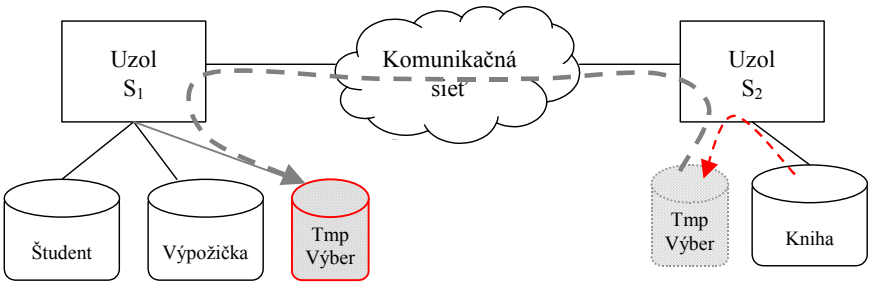


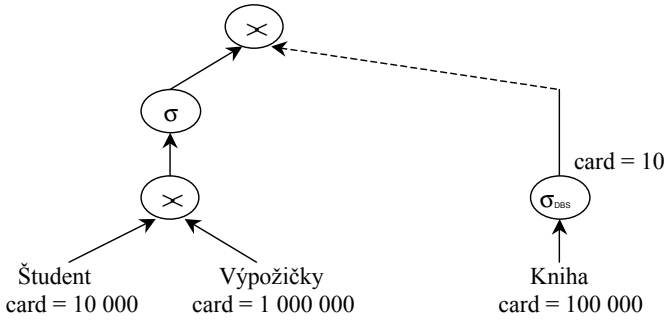
$T_5 = t_{DA} * počet\ správ + OD \div Vp = 0.1 * 1 + (100\,000 * 200) \div 50\,000 = 400$ sek $= 6.67$ min





$T_6 = t_{AD} * počet\ správ + OD \div Vp = 0.1 * 1 + (10 * 200) \div 50\,000 = 0.1$ sek





Z predchádzajúceho vyplýva, že máme nasledujúce možnosti, pričom najvýhodnejšie je použiť šiestu stratégiu.

$T_1 = t_{AD} + t_{prenos} = 0.1 + (100\,000 * 200) \div 50\,000 = 400$ sek $= 6.67$ min

$T_2 = 2 * t_{AD} + OD \div Up = 2 * 0.1 + ((10\,000 + 100\,000) * 200) \div 50\,000 = 4040$ sek $= 1.26$ hod

$T_3 = 2 * počet\ správ = 2 * 10\,000 = 20\,000$ sek $= 5.56$ hod

$T_4 = 2 * t_{DA} * počet\ správ = 2 * 0.1 * 10 = 2$ sek

$T_5 = t_{DA} * počet\ správ + OD \div Up = 0.1 * 1 + (100\,000 * 200) \div 50\,000 = 400$ sek $= 6.67$ min

$T_6 = t_{DA} * počet\ správ + OD \div Up = 0.1 * 1 + (10 * 200) \div 50\,000 = 0.1$ sek

Z predchádzajúceho príkladu je zrejmé, že pre každú požiadavku je možné vypracovať niekoľko stratégií spracovania, z ktorých bude potrebné vybrať tú, ktorá je najvýhodnejšia. Rozhodujúcim kritériom pre výber bude doba celkového spracovania, ktorá je ovplyvnená prenosovými rýchlosťami medzi uzlami v DDBS. Samotné výpočty ovplyvnené aj vstupno/výstupnými operáciami, je možné v niektorých prípadoch aj zanedbať.

Okrem toho je z prechádzajúceho textu zrejmé, že niektoré zo stratégií umožnia paralelné spracovanie častí dotazu, spracovávaných na viacerých uzloch DDBS, čo môže taktiež prispieť k skráteniu doby spracovávania dotazu.

12.8 Riadenie katalógov

- **centralizované** – existuje jeden centrálny katalóg umiestnený na jednom z uzlov DDBS
- **plne replikované** – úplný katalóg je umiestnený na všetkých uzloch DDBS
- **čiastočné** – každý uzol obsahuje vlastný katalóg len pre objekty umiestnené na tomto uzle, úplný dostaneme zjednotením všetkých čiastočných katalógov
- **kombinované** – každý uzol obsahuje vlastný katalóg a pre celý systém existuje centrálny katalóg obsahujúci kópie všetkých lokálnych katalógov.

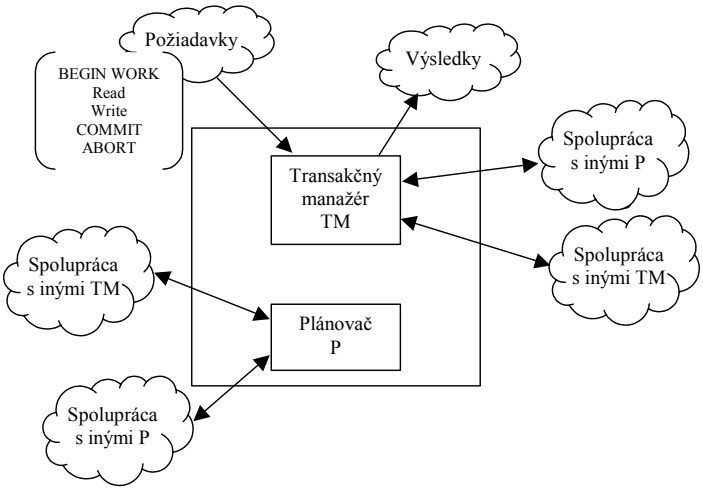
12.9 Riadenie transakcií v distribuovanom systéme

Je nutné zabezpečiť vlastnosti transakcie:

- **Atomicita** – celá postupnosť operácií je chápaná ako logická jednotka, aj v prípade, že transakcia spúšťa podtransakcie. V tom prípade transakcia môže byť potvrdená, len vtedy, ak všetky podtransakcie boli potvrdené.
- **Konzistencia** – DB vo všetkých uzloch musí byť vždy v konzistentnom stave (t. j. ak robíme deštruktívne operácie, tak sa musia vykonávať vo všetkých replikách.)
- **Izolovaná vratnosť** – zrušením transakcie nesmie byť postihnutá iná transakcia a zároveň musí byť zabezpečená konzistencia celej distribuovanej databázy
- **Trvanlivosť** (durability) – zmeny sú skutočne zapísané do všetkých častí distribuovanej databázy

Pretože v distribuovanom systéme rozlišujeme dva typy transakcií, môžeme ich klasifikovať ako:

- Globálna transakcia
- Lokálna transakcia



Obr. 12.9 Práca SRBD v DDBS

Definícia – Lokálna transakcia
Lokálna transakcia k svojmu vykonaniu potrebuje prístup k dátam jedného uzla a nespolupracuje s transakciami na iných uzloch

♦

Definícia – Globálna transakcia
Globálna (distribuovaná) transakcia k svojmu vykonaniu potrebuje spoluprácu s viacerými uzlami.

♦

Globálna transakcia začína pracovať na jednom uzle a podľa potreby spúšťa podtransakcie na iných uzloch.

Pre globálnu transakciu (GT) platí:

- Transakcia číta len jednu kópiu log. objektu X
- Transakcia zapisuje zmenu na všetky kópie log. objektu X

12.10 Potvrdzovacie protokoly

5 základných operácií:

- BEGIN_TRANSACTION (BOT) – táto operácia oznamuje transakčnému manažérovi, že nová transakcia štartuje. Tento manažér jej pridelí jednoznačné číslo, zabezpečí synchronizáciu atď.
- READ – zabezpečí načítanie dátového objektu v rámci transakcie. V prípade, že objekt umiestnený na danom uzle načíta ho priamo, v opačnom prípade musí požiadať iný transakčný manažér o sprístupnenie daného dátového objektu
- WRITE – zabezpečí zápis dátových objektov do všetkých uzlov, kde majú byť umiestnené
- COMMIT – zabezpečí koordináciu zápisu všetkých zmien databáz rozmiestnených na uzloch distribuovaného systému
- ABORT – zabezpečí, že transakcia, ani žiadna z jej podtransakcií, nebude potvrdená

12.10.1 Dvojfázový potvrdzovací protokol

Pre tento protokol platia dve pravidlá:

1. Globálna transakcia môže potvrdiť svoje zmeny, len ak všetky podtransakcie dosiahli bod RC (READY TO COMMIT) a vyslali hlavnej transakcii operáciu COMMIT
2. Ak ktorákoľvek z podtransakcií vyšle operáciu ABORT, globálna transakcia musí vysielať aj ostatným podtransakciám ABORT, na základe ktorého budú hlavná transakcia aj podtransakcie zrušené

Existuje niekoľko variantov ako zabezpečiť dvojfázový potrvdzovací protokol. Základný pracuje na základe nasledovného algoritmu, ktorý je zabezpečovaný v dvoch fázach:

1. **fáza**

   Globálna transakcia spúšťa všetky podtransakcie a zbiera informácie o ukončení podtransakcií. Počas 1. fázy sa nesmú uvoľňovať žiadne objekty, zamknuté ktoroukoľvek podtransakciou.

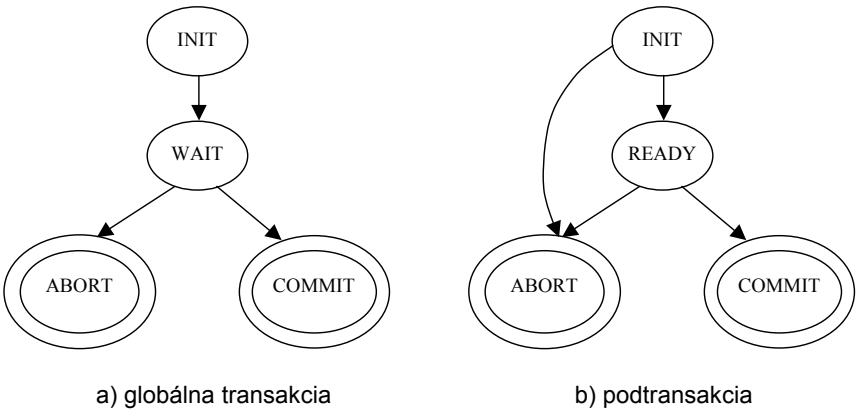
2. **fáza**

   Globálna transakcia vysiela informáciu všetkým podtransakciám ako majú skončiť (COMMIT, alebo ABORT)

Podľa spôsobu zabezpečenia vykonania týchto fáz rozdeľujeme protokoly na:

- **Centralizované** – pri ktorých sú výsledky (COMMIT, ABORT) podtransakcií sústreďované v uzle, ktorý spustil globálnu transakciu a globálna transakcia zabezpečí vysielanie správy o spôsobe ukončenia podtransakciám. Najpoužívanejšie centralizované protokoly sú protokoly 6-n a 4-n
- **Lineárne** – pri ktorých jedna podtransakcia môže komunikovať len s jednou susednou podtransakciou, kde obe patria k tej istej globálnej transakcii. Tieto podtransakcie sú usporiadané v systéme v poradí ich spracovania
- **Distribuované** – pri tomto protokole sa vyžaduje komunikácia medzi všetkými podtransakciami spustenými globálnou transakciou

Spôsob spracovania protokolu 2PC si môžeme znázorniť aj stavovými diagramami, pričom globálna transakcia aj podtransakcia majú svoj stavový diagram.

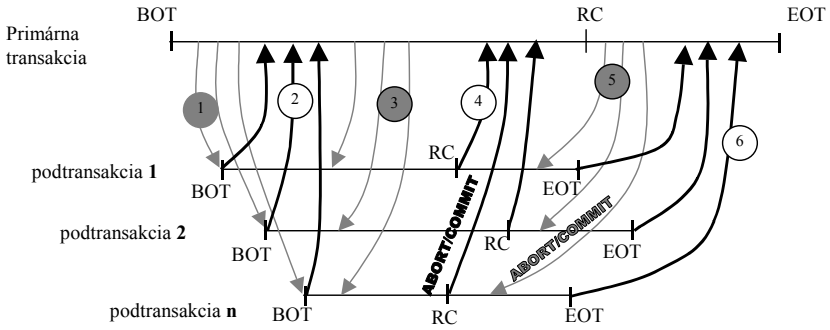


a) globálna transakcia b) podtransakcia

Obr. 12.10 Stavové diagramy transakcií

12.10.1.1 Protokol 6-n správ

Nasledujúci obrázok popisuje spôsob zabezpečenia tohto protokolu.



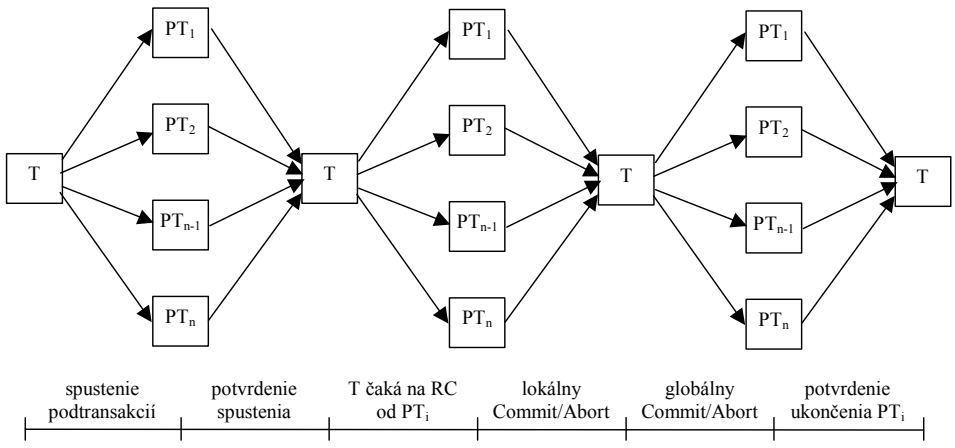
Obr. 12.11 Správy protokolu 6-n správ

Popis spracovania:

1. fáza
- 1 globálna transakcia spúšťa podtransakciu
- 2 potvrdenie spustenia podtransakcie
- 3 oznam, že globálna transakcia čaká na dosiahnutie bodu RC od podtransakcií
- 4 odpoveď o ukončení podtransakcie – lokálny COMMIT / lokálny ABORT

2. fáza
- 5 príkaz k ukončeniu podtransakciám – globálny COMMIT / globálny ABORT
  (stačí ak jedna podtransakcia poslala lokálny ABORT – všetky musia ukončiť svoju prácu ABORT- globálny)
- 6 potvrdenie ukončenia podtransakcií

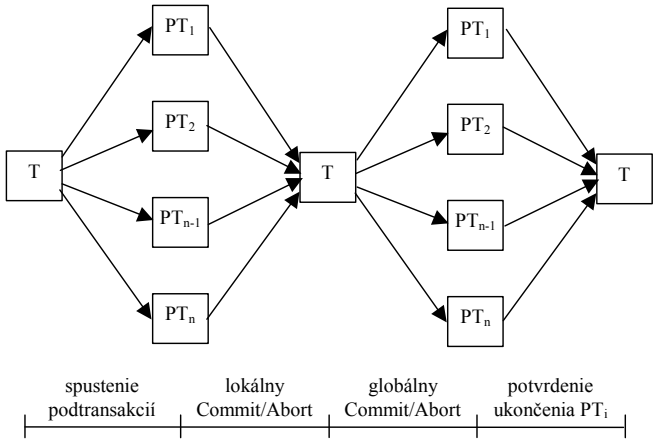
Iná možnosť znázornenia tohto protokolu je na nasledujúcom obrázku, kde T je globálna transakcia, ktorá spúšťa podtransakcie PT_1 …PT_n.



Obr. 12.12 Protokol 6-n správ

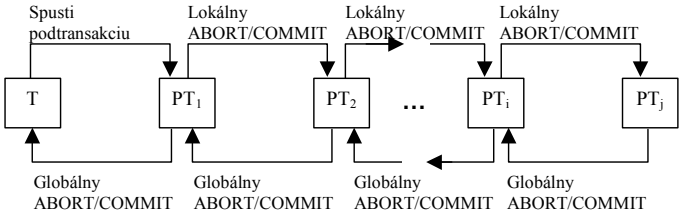
12.10.1.2 Protokol 4-n správ

Protokol 4-n získame zjednodušením protokolu 6-n tak, že vynecháme správu 2 (potvrdenie spustenia podtransakcie) a správu 3 (oznam, že globálna transakcia čaká na dosiahnutie bodu RC). Teda tento protokol vyzerá nasledovne:



Obr. 12.13 Protokol 4-n správ

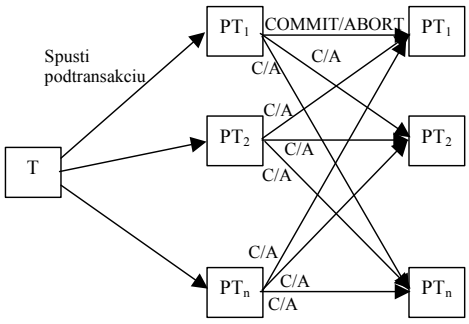
12.10.1.3 Lineárny dvojfázový potvrdzovací protokol (L2PC)



Obr. 12.14 Lineárny dvojfázový potvrdzovací protokol

Tento protokol je nevýhodný, pretože neumožňuje paralelizmus spracovania podtransakcií a využíva sa hlavne v takom prostredí, kde je počítačová sieť málo výkonná.

12.10.1.4 Distribuovaný dvojfázový potvrdzovací protokol (D2PC)



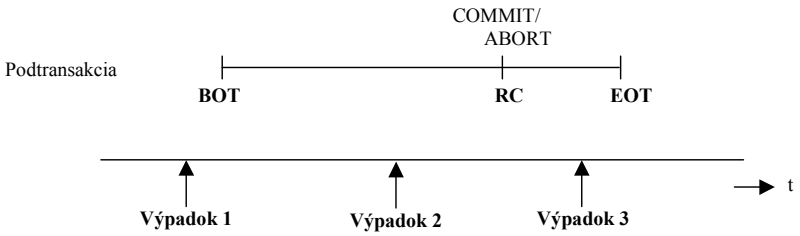
Obr. 12.15 Distribuovaný dvojfázový potvrdzovací protokol

12.11 Výpadky uzlov

12.11.1Výpadky podtransakcií

Výpadok podtransakcie môže nastať z pohľadu globálnej transakcie v troch prípadoch (viď obrázok 12.16):

1. Výpadok typu jeden znamená, že podtransakciu nebolo možné spustiť a globálna transakcia nedostane správu, že podtransakcia bola spustená
2. Výpadok typu dva znamená, že globálna transakcia nedostane informáciu o dosiahnutí bodu RC podtransakcie. Nevieme, či došlo k výpadku uzla, alebo či uzol s podtransakciou nie je schopný poslať informáciu o dosiahnutí bodu RC
3. Výpadok typu tri znamená, že globálna transakcia získala od podtransakcie správu lokálny COMMIT alebo lokálny ABORT, ale nemôže vyslať signál globálny COMMIT alebo globálny ABORT, alebo globálna transakcia nedostala správu o ukončení podtransakcie.



Obr. 12.16 Výpadky podtransakcií

V prípade výpadku typu 1 musí globálna transakcia vyslať všetkým ostatným podtransakciám signál globálny ABORT. V niektorých systémoch (hlavne pri operácii READ) je možné sa pokúsiť spustiť podtransakciu na inom uzle, na ktorom existuje replika dát, avšak transakčný manažér globálnej transakcie si musí pamätať, že bola spustená náhradná podtransakcia, aby v prípade obnovenia práce SRBD, na ktorom nebola spustená podtransakcia, nedošlo ku skresleniu výsledkov.

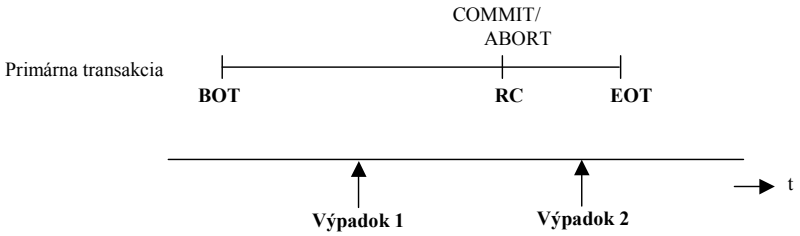
V prípade výpadku typu 2, globálna transakcia bude čakať po dobu určeného časového limitu a ak nedostane správu od podtransakcie, bude predpokladať, že podtransakcia skončila signálom ABORT, a podľa toho ukončí aj globálnu transakciu.

V prípade výpadku typu 3 nevieme, či lokálny uzol prijal správu globálny COMMIT, alebo globálny ABORT, a preto musí globálna transakcia ostať aktívna aj naďalej, pokiaľ sa neukončí komunikácia medzi globálnou transakciou a podtransakciou.

12.11.2 Výpadky globálnej transakcie

Na uzle, na ktorom beží globálna transakcia, rozpoznávame dva typy výpadkov (obr. 12.17):

1. Výpadok typu jeden znamená, že globálna transakcia nebola schopná vyslať globálny COMMIT alebo globálny ABORT
2. Výpadok typu dva znamená, že globálna transakcia už začala vysielať príkazy globálny COMMIT alebo ABORT.



Obr. 12.17 Výpadky globálnej transakcie

V prípade výpadku typu 1 rozpoznávame tri typy podtransakcií:

- Podtransakcie, ktoré nahlásili dosiahnutie bodu RC globálnej transakcii – v tomto prípade podtransakcie musia čakať na obnovenie činnosti uzla s globálnou transakciou
- Podtransakcie, ktoré poslali správu lokálny ABORT – v tomto prípade nemôže nastať globálny COMMIT, a teda tieto podtransakcie môžu vykonať lokálny ABORT
- Podtransakcie, ktoré ešte nevyslali žiadny signál globálnej transakcii – v tomto prípade sa podtransakcie môžu sami rozhodnúť, či budú čakať, alebo vyšlú správu ABORT. V takom prípade sa pri obnovení globálnej transakcie musí zabezpečiť vysielanie globálneho ABORT-u

V prípade výpadku typu 2 máme nasledovné situácie:

- Globálna transakcia začala vysielať príkaz globálny COMMIT alebo ABORT

  Ak všetky podtransakcie dosiahli bod RC, tak musia čakať na obnovenie činnosti globálnej transakcie. Svoju činnosť môžu ukončiť len tie podtransakcie, ktorým stihla globálna transakcia vyslať globálny COMMIT.

  V prípade, že globálna transakcia vyslala príkaz ABORT, tak svoju činnosť môžu ukončiť len tie transakcie, ktoré dostali túto správu alebo sami vyslali lokálny ABORT.
- Globálna transakcia ešte nevyslala príkaz globálny COMMIT alebo ABORT.

V prípade, že podtransakcia vyslala lokálny ABORT, môže ukončiť svoju činnosť, ale ostatné podtransakcie musia čakať na obnovenie činnosti globálnej transakcie.

12.12 Paralelizmus a synchronizácia v DDBS

- Zamykanie
- Časové pečiatky

Rozdiely:

a) Zamykanie zaručí, že spracovanie je porovnateľné s náhodne usporiadaným spracovaním.

   Časové pečiatky zaručia, že poradie transakcií bude zodpovedať poradiu vzniku požiadaviek na spracovanie t. j. definovaným poradím.

b) Zamykanie môže spôsobiť uviaznutie, časové pečiatky nie.

12.12.1 Zamykanie

- Centrálne
  - tabuľka o zamknutých objektoch je uložená na centrálnom uzle
  - jednoducho rozpozná globálne uviaznutie
  - nevýhodou sú vysoké komunikačné náklady
  - zraniteľnosť pri výpadku centrálneho uzla
  - jednoduchosť
- Decentralizované
  - každý uzol má vlastnú tabuľku zámkov
  - problémy s rozpoznaním globálneho uviaznutia

Základným princípom zamykania v DDBS je, že každá transakcia

a) zamkne daný objekt, ktorý sprístupní
b) zamknuté objekty uvoľní až po ukončení transakcie

Pre dosiahnutie paralelného spracovania je nutné, aby vždy vykonali tieto činnosti:

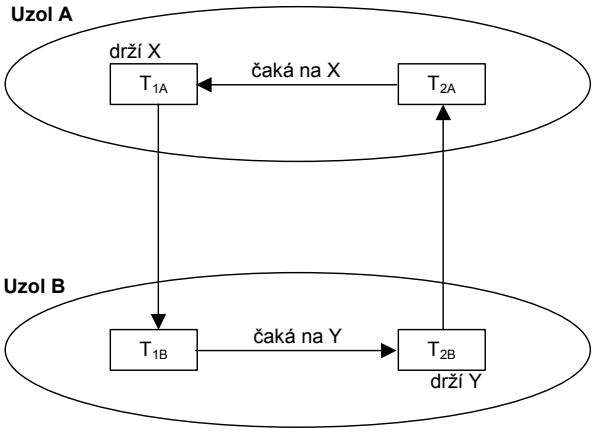
- pred čítaním dátového objektu je nutné uzamknúť zdieľateľným zámkom (S-lock - shared) aspoň jednej kópie dát
- pred operáciou, ktorá mení databázu je nutné zabezpečiť vykonanie výlučného uzamknutia (X-lock - exclusive) na každý taký objekt
- všetky objekty sú uvoľnené až po dosiahnutí COMMIT primárnej transakcie

Je potrebné si uvedomiť, že pri zmenách v databáze môžu byť kópie daných objektov umiestnené na iných uzloch nedostupné.

Jedným z problémov zamykania v DDBS je množstvo správ odovzdávaných medzi uzlami, obsahujúcich dátové objekty. Ak transakcia mení dátový objekt, ktorý má kópie na iných uzloch, je možné uvažovať až 5n správ:

1. Požiadavka na zamknutie
2. Potvrdenie zamknutia
3. Správa o zmene
4. Potvrdenie zmeny
5. Uvoľnenie

12.12.2Globálne uviaznutie



Obr. 12.18 Príklad globálneho uviaznutia

12.12.3Rozpoznanie uviaznutia

Definícia – Čakací graf
Čakací graf (Wait-for graph) – je orientovaný graf, v ktorom transakcia reprezentovaná ako vrchol a vzťah vyjadrujúci čakanie transakcie T_i na udalosť, ktorou je v DDBS aj ukončenie transakcie T_j, reprezentovaný orientovanou hranou $v_{ij} = [T_i, T_j]$.

♦

Ak čaká transakcia T_1 na uvoľnenie objektu držaného transakciu T_2 môžeme písať

$$\mathbf{T_1} \longrightarrow \mathbf{T_2}$$

všeobecne

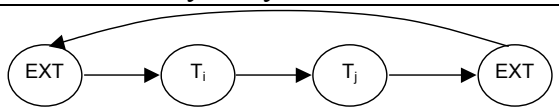
$$\boldsymbol{T_i} \longrightarrow \boldsymbol{T_j}$$

Rozpoznanie uviaznutia sa rieši hľadaním cyklov v čakacom grafe. V distribuovaných systémoch musíme uvažovať s globálnym čakacím grafom, ktorý vznikne zjednotením všetkých lokálnych čakacích grafov.

Rozpoznanie sa môže diať:

- Centralizovane
- Decentralizovane
- Zmiešane

$$\boldsymbol{EXT} \longrightarrow \boldsymbol{T_i} \longrightarrow \boldsymbol{T_j} \longrightarrow \boldsymbol{EXT}$$



Obr. 12.19 Upravený čakací graf

Centrálne rozpoznávanie

Lokálne čakacie grafy sú zasielané centrálnemu uzlu, ktorý vytvorí globálny čakací graf a v ňom hľadá cykly.

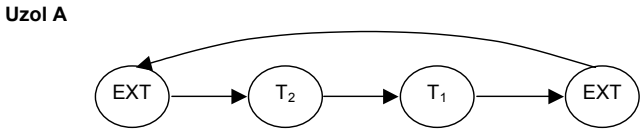
Nevýhodou sú vysoké komunikačné náklady.

Decentralizované rozpoznávanie

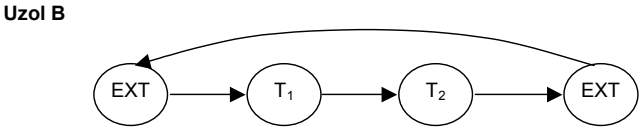
Pri decentralizovanom rozpoznávaní uviaznutia sa na rozpoznávaní môžu podieľať všetky uzly. Preto každý SRBD, ktorý používa zamykanie, musí zabezpečiť rozpoznanie uviaznutia. To sa zabezpečí periodicky aktivovaným algoritmom, ktorý analyzuje množinu transakcií bežiacich v systéme, pokúsi sa rozpoznať uviaznutie a odstrániť ho. Algoritmus rozpoznania je nasledovný:

1. Vytvor lokálny čakací graf (doplnený v prípade potreby s EXT vrcholmi.)
2. V prípade, že niektorá z transakcií je spustená z iného uzla, pridaj k lokálnemu čakaciemu grafu čakací graf vyslaný z daného uzla
3. Zisti, či v grafe neexistuje cyklus, alebo potencionálny cyklus
4. Ak existuje cyklus v grafe, odstráň jednu z transakcií z cyklu
5. Ak existuje potencionálny cyklus v grafe, odošli lokálny čakací graf na uzol, na ktorom bola spustená podtransakcia
6. Opakuj kroky 2-5 pokiaľ sa neodstráni uviaznutie, alebo nie je zostrojený globálny čakací graf bez cyklu

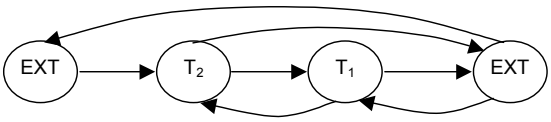
Príklad 12.5 Vytvorenie globálneho čakacieho grafu
Majme lokálny čakací graf na uzle A:



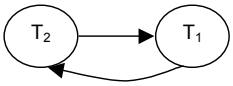
a lokálny čakací graf na uzle B:



Potom pri tvorbe globálneho čakacieho grafu najprv spojíme lokálne čakacie grafy:



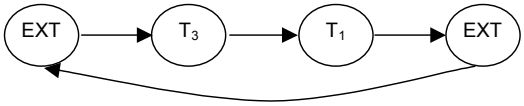
a následne odstránime fiktívne vrcholy EXT:



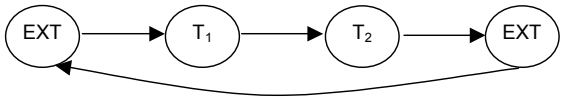
Takto sme dostali globálny čakací graf, z ktorého je zrejmé, že došlo ku globálnemu uviaznutiu a jedna z transakcií T_1, alebo T_2 musí byť zrušená.

Príklad 12.6 – Rozpoznanie uviaznutia
Majme nasledovné lokálne čakacie grafy na jednotlivých uzloch:

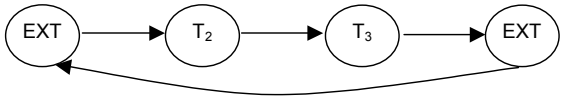
Uzol A



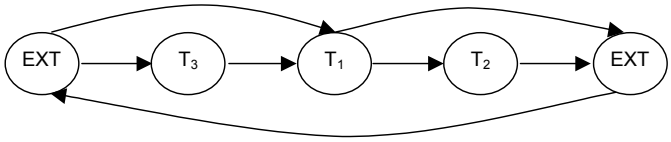
Uzol B



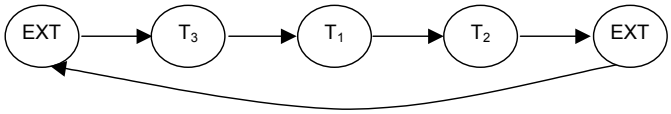
Uzol C



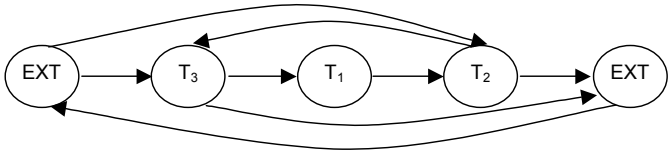
Keďže na uzle A neexistuje cyklus, ale existuje potenciálny cyklus, vyšleme lokálny čakací graf z uzla A na uzol B a tam ich spojíme.



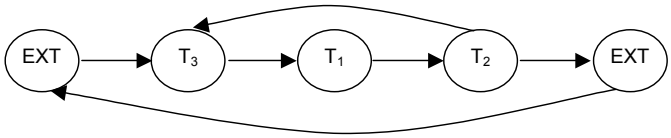
Tento čakací graf sa dá nasledovne zjednodušiť:



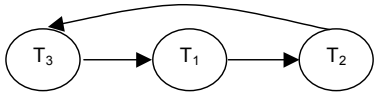
Vidíme, že ani tu sa nám ešte nepodarilo odhaliť cyklus, takže je nutné vyslať čakací graf na ďalší uzol C.



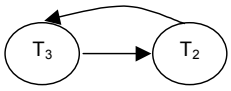
Tento čakací graf môžeme opäť zjednodušiť:



Pretože ide už o globálny čakací graf, môžeme vrcholy EXT vybrať a dostaneme nasledovný globálny čakací graf.



V čakacom grafe existuje cyklus, ktorý pozostáva zo všetkých transakcií T_3, T_1, T_2, ale keďže sa momentálne nachádzame na uzle C a na tomto uzle transakcia T_1 nebeží, vyberieme ju z grafu.



Teraz je zrejmé, že SRBD na uzle C musí zrušiť jednu z transakcií T_2 alebo T_3, aby sme odstránili uviaznutie.

12.12.4 Časové pečiatky

Základná myšlienka pre používanie časových pečiatok vychádza z úvahy, že pre dvojicu udalostí v systéme platí:

- Ak na tom istom uzle udalosť **i** predchádza udalosť **j**, tak pre časové pečiatky týchto udalostí platí

  TS(i) < TS(j)

- Ak udalosť spôsobí vyslanie správy z uzla S_1 na uzol S_2, tak čas prijatia správy na uzle S_2 musí byť väčší ako čas odoslania správy z uzla S_1, čím zabezpečíme, aby v celom systéme bolo poradie transakcií zachované. Pre samotné označovanie časových pečiatok je teda potrebné zabezpečiť synchronizáciu hodín v systéme.

12.12.4.1 Synchronizácia hodín

Zabezpečiť synchronizáciu na všetkých uzloch v systéme je možné pomocou viacerých metód alebo mechanizmov (GPS, …).

Metóda 1

Existuje jeden z uzlov S v sieti, ktorý periodicky synchronizuje čas na všetkých ostatných uzloch.

Metóda 2

Lokálne hodiny na každom uzle sa inkrementujú medzi dvojicou udalostí tak, že časová pečiatka udalosti $\mathbf{t_1}$ odoslaná z S_1 nastaví lokálne hodiny na uzle S_2, ktoré majú hodnotu $\mathbf{time_2}$:

$$\mathbf{time_2 := MAX(t_1, time_2)}$$

12.12.4.2 Predchádzanie uviaznutiu pomocou časových pečiatok

Druhou metódou ako sa vysporiadať s uviaznutím je možnosť prechádzaniu uviaznutiu. To sa dá zabezpečiť len v tom prípade, že používame časové pečiatky pre označovanie objektov a transakcií a systém je synchronizovaný.

Potom na predídenie uviaznutia môžeme použiť jednu z metód:

- WAIT – DIE

- WOUND – WAIT.

Prvá z týchto metód odstráni zo systému mladšiu transakciu, v prípade, že sa pokúša pracovať s objektom, s ktorým pracuje staršia transakcia. Pri druhej metóde staršia transakcia spôsobí zrušenie mladšej transakcie, v prípade, že staršia transakcia požaduje objekt, s ktorým pracuje mladšia transakcia.

Existuje aj celá rada iných metód (konzervatívne pečiatkovanie, používanie verzií objektov, …), ktoré niektoré systémy používajú na to, aby nedošlo k uviaznutiu transakcií.