Téma 12 – Architektury DBMS

Obsah

- Architektury databázových systémů
- Systémy klient-server
- Transakční servery
- Paralelní systémy
- Distribuované systémy
- Distribuované databázové systémy
- Homogenní a heterogenní DBMS
- Distribuované ukládání dat a replikace
- Distribuované transakce a jejich atomicita
- Potvrzovací protokoly
- Řízení souběhu v distribuovaných systémech
- Další otázky a problémy v distribuovaných systémech

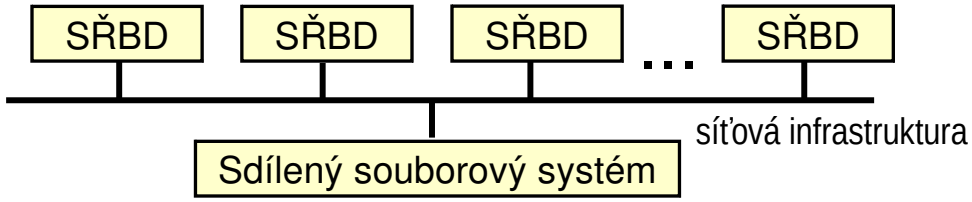
Architektury databázových systémů

Centralizované databázové systémy

- Běží na jediném izolovaném počítači
  - nespolupracuje a neinteraguje se s jinými počítači
- Centralizované databáze běží na standardním počítači a za podpory obecného operačního systému
  - dokonce i monoprogramního
- Všechny databázové akce jsou prováděny lokálně
  - Paralelní zpracování zde není buď vůbec, nebo jen velmi omezené
  - Databázové technologie popisované v tomto kurzu jsou používány jen pro zrychlení přístupu k datům
- Příklady použití
  - domácí správa CD disků
  - jednouživatelské účetní systémy
  - evidence skladových zásob v obchodě s jedinou pokladnou
- Z pohledu DBMS jsou takové systémy nezajímavé

Systémy file-Server

- Server poskytuje sdílený souborový systém pro lokální SŘBD



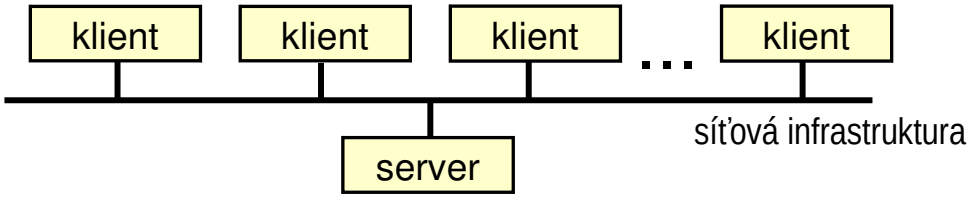
- Každý klient obsahuje svůj SŘBD, který používá sdílený souborový systém
  - Uživatel zadá na svém počítači dotaz SŘBD
  - SŘBD získá data ze sdíleného souborového systému
  - SŘBD lokálně vyhodnotí transakci

- Výhody:
  - Jednoduchá implementace

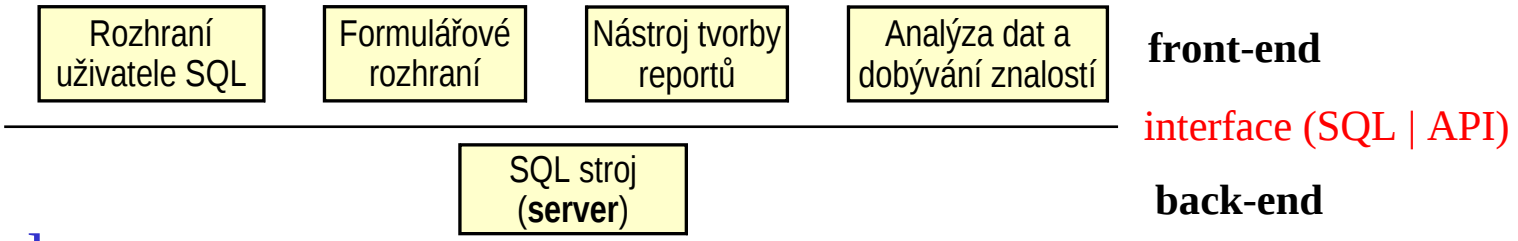
- Nevýhody:
  - Přenosy velkých objemů dat – přenáší se celá relace, zpracování probíhá lokálně

Systémy Klient-Server

- Server plní požadavky zasílané *m* klientskými systémy



- Funkcionalitu databáze lze rozdělit na
  - **Back-end**: spravuje přístup k datovým strukturám, optimalizuje a vyhodnocuje dotazy, řídí souběh a zotavování
  - **Front-end**: obsahuje nástroje typu formulářů a GUI
    - Rozhraním mezi těmito komponentami je buď SQL nebo databázové API



- Výhody:
  - dobrý poměr cena/výkon a relativně snadná údržba dat
  - pružnost při změnách v datech a různorodost aplikaci "nad daty"
  - oddělenost uživatelských rozhraní (GUI)
- Typy serverů
  - transakční servery – pro relační databáze
  - datové servery – obvykle jen pro databáze objektové

Transakční servery

- Alternativní název SQL server
  - Klienti posílají serveru požadavky (dotazy)
  - Server vyhodnocuje dotazy jako samostatné transakce
  - Výsledky zasílá zpět klientům
- Požadavky jsou formulovány v SQL a zasílány serveru
  - nejčastějším mechanismem je *remote procedure call* (RPC) nebo jiný podobný protokol předávání zpráv
- *Open Database Connectivity* (ODBC)
  - je API standard pro jazyk C vyvinutý Microsoftem pro připojení klienta k serveru, zasílání požadavků v SQL a příjem výsledků
- JDBC standard
  - je analogie s ODBC pro klienty v Javě vyvinutý Sun Microsystems

Struktura procesů transakčního serveru

- Typický transakční server obsahuje řadu procesů přistupujících ke sdílené paměti (*shared memory*)
  - Servisní procesy
    - Přijímají požadavky klientů, vyhodnocují je jako transakce a zasílají výsledky zpět
    - Procesy bývají vícevláknové, což jednomu procesu vyhodnocovat několik požadavků souběžně
  - Správce zámků ➔
  - Proces writer (*data writer*)
    - Jeho úkolem je kontinuální provádění operací **`output`** na modifikovaných vyrovnávacích pamětech bloků
  - Proces log writer, též logger
    - Servisní procesy prostě přidávají protokolové záznamy do vyrovnávací paměti protokolu
    - Úkolem procesu *log writer* je včasné a koordinované vypisování záznamů na stabilní paměť
  - Proces pro správu kontrolních bodů (*checkpointing process*)
    - Periodicky generuje kontrolní body způsobem popsaným dříve
  - Monitorovací proces (*process monitor*)
    - Hlídá běh ostatních procesů, testuje uváznutí a jiné chyby a zajišťuje zotavení
      - Např. ruší transakce servisních procesů a restartuje je

Procesy transakčního serveru

Procesy transakčního serveru (pokr.)

- Sdílená paměť obsahuje
  - *Buffer pool* – prostor, z nějž se alokují vyrovnávací paměti
  - *Log buffer* – vyrovnávací paměť protokolu
  - Tabulku zámků – viz ←
  - *Cached query plans* – jednou vytvořené plány vyhodnocení dotazů mohou být použity znovu
- Všechny procesy mohou přistupovat ke sdílené paměti
  - K zábraně problémů se souběhem na sdílených datových strukturách se používají mutexy na bázi
    - semaforů OS nebo atomických instrukcí typu TSL ←
  - Aby se redukovala režie spojená s meziprocesní komunikací se správcem zámků, přistupují všechny procesy k tabulce zámků přímo
    - místo aby posílaly žádosti správci zámků
    - Správce zámků pracuje obvykle ve funkci detektoru uváznutí a spolupracuje s monitorem procesů

Paralelní systémy

- Paralelní databázové systémy jsou složeny ze sady procesorů s příslušnými diskovými poli a vše je propojeno rychlou sítí
  - „Hrubozrnné“ paralelní stroje jsou tvořeny malým počtem vysoce výkonných procesorů
  - Masivně paralelní (nebo též jemnozrnné) stroje používají velké množství (stovky) malých procesorů
- Architektury paralelních databází
  - Se sdílenou pamětí – procesory sdílejí společnou paměť
    - těsně vázané systémy (*tightly coupled*) ←
  - Se sdílenými disky – procesory sdílejí společnou sadu disků
  - Bez sdílení – procesory nesdílí ani paměť ani disky
    - jde o soustavu samostatných počítačů
  - Hierarchická architektura – hybrid shora uvedených architektur

Architektury paralelních systémů

sdílená paměť

sdílené disky

bez sdílení

hierarchická
organizace

Systémy se sdílením

- Systémy se sdílenou pamětí
  - Procesory a disky mají společnou paměť přístupnou přes rychlou sběrnici
  - Extrémně rychlá komunikace mezi procesory
    - nic se nemusí kopírovat
  - Nevýhoda
    - Architektura není použitelná při větším počtu procesorů, neboť sběrnice se stává úzkým místem systému
    - Užíváno pro systémy s méně než cca 16 procesory
- Systémy se sdílenými disky
  - Procesory mohou přímo přistupovat k diskům prostřednictvím komunikačního propojení, avšak mají svoje privátní paměti
    - Není problém s „ucpáváním“ sběrnice
    - Architektura poskytuje jistý stupeň odolnosti vůči chybám (*fault-tolerance*) – zhavaruje-li jeden procesor, ostatní mohou převzít jeho práci, neboť data jsou na discích
  - Nevýhoda
    - Úzkým místem je zde propojení k diskovému subsystému
    - Systémy se sdílenými disky lze škálovat na více procesorů, avšak komunikace mezi procesory je pomalejší

Systémy bez sdílení

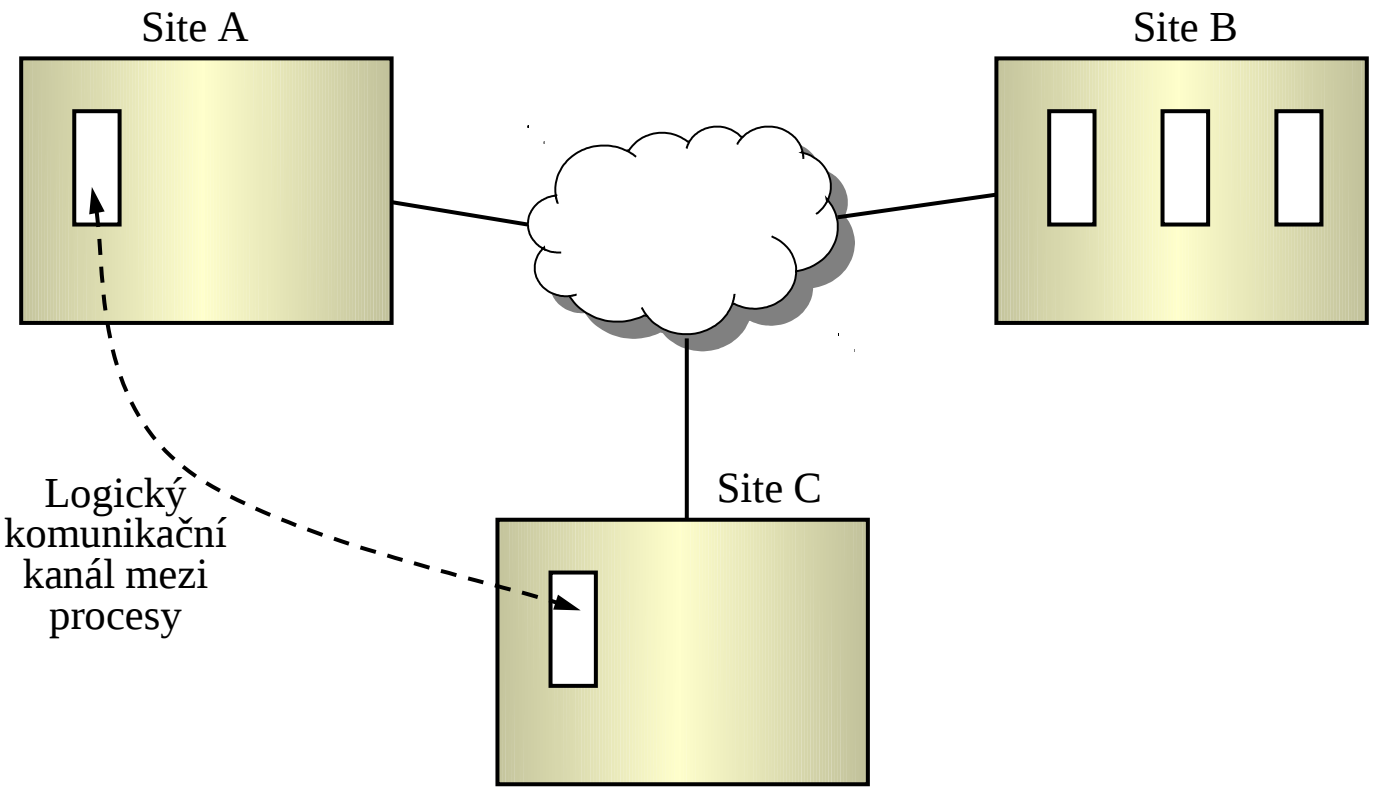
- Každý uzel této architektury je tvořen procesorem, pamětí a jedním či několika disky
  - Procesor komunikuje s procesorem v jiném uzlu prostřednictvím propojovací sítě
  - Každý uzel tak pracuje jako server pro data, která se nacházejí na disku či discích, příslušejících daném uzlu
  - Data vybavovaná z lokálního disku (a z lokální paměti) se nikam nepřenášejí
    - Minimalizuje se tak „cena“ za přenosy a interference sdílených zdrojů (netřeba „tolik zamykat“, a tím nehrozí tak častá uváznutí)
  - Systémy bez sdílení lze použít i pro multiprocesory se stovkami či tisíci procesních jednotek bez rizika přetížení komunikačních cest či sdílených prostředků
- Hlavní nevýhoda
  - cena za přístup k datům uloženým v jiném uzlu
    - zasílání dat zahrnuje softwarové řízení na obou koncích, tj. v obou dotčených uzlech

Hierarchická organizace

- Kombinuje vlastnosti systémů se sdílením a bez sdílení
- Nejvyšší úroveň je obvykle architektura bez sdílení
  - Uzly propojené komunikační sítí nesdílejí ani disky ani paměť
  - Každý uzel může být systém s několika procesory a sdílenou pamětí, nebo
  - Každý uzel může být systémem se sdílenými disky, přičemž dílčí systémy sdílející disky mohou být multiprocesory sdílející paměť
- Základní nevýhodou takových systémů je složitost jejich programování
  - K redukci složitosti programování byly vyvinuty zvláštní techniky označované jako architektury s distribuovanou virtuální pamětí
    - známé též pod zkratkou NUMA (*non-uniform memory architecture*)

Distribuované systémy

- Data jsou rozptýlena po celé řadě počítačů obecně s různými architekturami propojenými sítí
- Data jsou sdílena uživateli jednotlivých dílčích systémů



Distribuované databáze

- Distribuované databáze jsou obecně dvou typů
  - Homogenní distribuované databáze
    - Ve všech uzlech je shodné databázové schéma a struktura software
      - data mohou být rozložena mezi uzly
    - Cíl: vytvořit dojem jediné databáze, kde detaily distribuce dat jsou skryty
  - Heterogenní distribuované databáze
    - V různých uzlech jsou různá schémata dat a různé softwarové prostředky
    - Cíl: integrace existujících databází za účelem užitečné funkcionality
- Je třeba odlišit *lokální* a *globální* transakce
  - Lokální transakce přistupuje k datům v tom uzlu sítě, kde byla transakce iniciována
    - Dotaz na lokální data
  - Globální transakce buď přistupuje k datům uloženým v jiném (vztaženo k místu, kde transakce vznikla) uzlu nebo dokonce v několika uzlech

Výhody a nevýhody distribuovaných databází

- Výhody
  - Sdílení dat
    - uživatelé v jednom uzlu mohou přistupovat k datům uloženým jinde
  - Autonomie
    - Každý uzel je schopen podržet si jistý stupeň řízení a kontroly lokálně uložených dat
  - Lepší a trvalejší dostupnost díky redundanci
    - Data mohou být replikována na vzdálených počítačích a systém muže fungovat, i když některý uzel zhavaruje
- Zásadní nevýhoda
  - Složitost koordinace dat ukládaných v různých uzlech
    - Náklady na vývoj software
    - Potenciálně mnohem více chyb
    - Nárůst režie zpracování

Implementační otázky distribuovaných databází

- Atomicitu je nutno zajistit i pro globální transakce
  - Transakce může měnit data uložená v několika uzlech
- Dvoufázový protokol potvrzování
  - *two-phase commit protocol* = 2PC
  - Základní myšlenka: Každý uzel provádí transakce až do okamžiku těsně před potvrzením (**`commit`**), konečné rozhodnutí o potvrzení je však přenecháno tzv. koordinátoru
  - Každý uzel se musí řídit rozhodnutími koordinátora
    - dokonce i při havárii, která vznikne v době čekání na rozhodnutí koordinátora
  - 2PC není vždy vhodný; vyvinuty jiné transakční mechanismy
    - persistentní zasílání zpráv
    - modely toku akcí (*workflow*)
- Jsou nutné i distribuované algoritmy řízení souběhu a detekce uváznutí
- Data mohou být replikována kvůli zlepšení dostupnosti
  - Avšak složitost koordinace obsahu kopií roste

Distribuované DBMS

- Distribuovaný databázový systém je tvořen sadou volně provázaných strojů – uzlů, které nesdílejí žádnou fyzickou složku
  - Systémy běžící v jednotlivých uzlech pracují nezávisle
  - Transakce přistupují k datům umístěným na jednom nebo více uzlech
- Homogenní distribuované databáze
  - Na všech strojích běží identický software
  - Jednotlivé systémy jsou připraveny vzájemně kooperovat při vyřizování uživatelských požadavků
  - Každý stroj se vzdává části své "autonomie" ve smyslu změn schémat dat či software
  - Uživateli se množina propojených uzlů jeví jako jeden celek
- Cílem je, aby uživatel nepoznal, že je databáze distribuovaná

Distribuované ukládání dat

- Předpokládejme relační model dat
- Replikace
  - Systém udržuje několik kopií dat uložených v různých uzlech distribuovaného systému kvůli rychlejším přístupům, odolnosti proti chybám a minimalizaci přenosů dat
- Fragmentace
  - Relace je rozdělena do několika fragmentů uložených na různých strojích
- Replikaci a fragmentaci lze kombinovat
  - Relace je rozdělena do několika fragmentů a systém tyto fragmenty replikuje

Replikace dat

- Úplná replikace nastává, když relace je redundantně replikována na všech strojích tvořících distribuovaný DBMS
  - Plně redundantní databáze jsou pak ty, kde každý stroj má úplnou kopii celé databáze
- Výhody replikace
  - Dostupnost: chyba uzlu držícího relaci *r* nezpůsobí nedostupnost dat, existují-li repliky *r*
  - Paralelismus: dotazy na *r* lze zpracovávat paralelně v uzlech
- Nevýhody replikace
  - Nákladné aktualizace: Všechny repliky relace *r* nutno změnit
  - Nárůst složitosti řízení souběhu: Souběžné aktualizace jednotlivých kopií si vynucují použití speciálních technik řízení souběhu, aby nevznikly nekonzistence dat
    - Možné řešení: jedna kopie se zvolí jako primární kopie a řízení souběhu se děje jen nad primární kopií; po aktualizaci primární kopie relace *r* se pak primární kopie replikuje do ostatních uzlů

Fragmentace dat

- Rozklad relace *r* na fragmenty r_1, r_2, …, r_n obsahující dostatek informací pro rekonstrukci celé relace *r*
- Horizontální fragmentace: každá *n*-tice *r* je uložena v jedno či více fragmentech
- Vertikální fragmentace: schéma relace *r* je rozděleno do několika menších schémat
  - Všechna schémata musí obsahovat společný klíč k zajištění bezeztrátové rekonstrukce celé relace
    - Lze přidat i dodatečný atribut, který poslouží jako takový klíč
- Příklad: relace "*account*" se schématem

  account = (*branch_name*, *account_number*, *customer_name*, *balance*)

Horizontální fragmentace relace *account*

branch_name	*account_number*	*customer_name*	*balance*
Praha	A-305	Novák	500
Praha	A-226	Hájek	336
Praha	A-155	Zavadil	80

$$account_1 = \sigma_{branch\_name=\text{“Praha”}}(account)$$

branch_name	*account_number*	*customer_name*	*balance*
Brno	A-177	Krejčí	205
Brno	A-402	Kovář	1000
Brno	A-408	Kolář	1234
Brno	A-639	Švec	567

$$account_2 = \sigma_{branch\_name=\text{“Brno”}}(account)$$

Vertikální fragmentace relace *account*

branch_name	*customer_name*	*tuple_id*
Praha	Novák	1
Brno	Krejčí	2
Praha	Hájek	3
Brno	Kovář	4
Praha	Zavadil	5
Brno	Kolář	6
Brno	Švec	7

$accnt_1 = \Pi_{branch\_name,\ customer\_name,\ tuple\_id}(account)$

account_number	*balance*	*tuple_id*
A-305	500	1
A-177	205	2
A-226	336	3
A-402	1000	4
A-155	80	5
A-408	1234	6
A-639	567	7

$accnt_2 = \Pi_{account\_number,\ balance,\ tuple\_id}(account)$

Výhody fragmentace

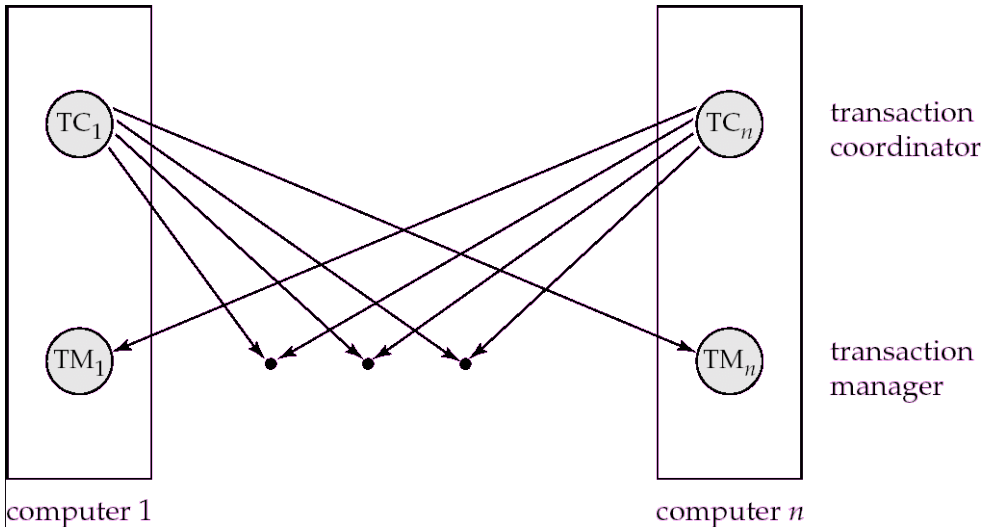
- Horizontální
  - umožňuje paralelní zpracování na fragmentech relace, které jsou umístěny tam, kde se k nim nejčastěji přistupuje
    - např. údaje o lokálních účtech se používají na pobočce nejčastěji
- Vertikální
  - umožňuje umístit atributy tam, kde se nejčastěji potřebují
    - na přepážce se nejčastěji potřebuje zůstatek účtu
  - přidaný atribut *tuple_id* dovoluje snadné spojování vertikálních fragmentů a tím i paralelní zpracování
- Vertikální a horizontální fragmentaci lze kombinovat
  - Fragmenty mohou být nadále fragmentovány do libovolné hloubky

Jména datových položek

- Položky musí mít v celém systému jedinečná jména
  - Musí být možné efektivně lokalizovat datovou položku a transparentně změnit její umístění
  - Každému uzlu musí být dovoleno autonomně vytvářet novou datovou položku
- Centralizovaný přístup – Name Server
  - Princip: "Name server" přiřazuje všechna jména
    - Každý uzel zná své lokální datové položky a žádá "Name server" o informaci, kde jsou jiná data
  - Výhody
    - Jednoznačnost jmen, rychlost přístupu, transparence přesunu položek
  - Nevýhody
    - Nedovoluje autonomní vytváření položek
    - Name server je úzkým místem celého systému (z pohledu zatížení) a jeho havárie (nebo nedostupnost) zastaví celý distribuovaný systém
- Distribuovaná alternativa
  - Každý uzel prefixuje data svým jménem (např. *Brno.account*)
    - Jména jsou jedinečná, není problém s centralizací, transparentní však není přesun dat
  - Řešení: "přezdívky" (aliasy)
    - Položky mají své přezdívky a mapování na skutečná jména jsou v uzlech

Distribuované transakce

- Transakce může přistupovat k datům v různých uzlech
- Každý uzel má svého správce transakcí odpovědného za
  - vedení protokolu pro účely zotavení
  - účast na koordinaci souběžných transakcí vyhodnocovaných v lokálním uzlu
- Každý uzel má koordinátor transakcí, který odpovídá za
  - Spouštění běhu transakcí, které vznikly v tomto uzlu
  - Distribuci subtransakcí do jiných uzlů
  - Koordinaci ukončení transakcí vzniklých v tomto uzlu, což může mít za následek potvrzení (commit) či zrušení (abort) subtransakcí v mnoha jiných uzlech



Chyby distribuovaných systémů

- Chyby specifické pro distribuované systémy
  - Havárie počítače v uzlu
  - Ztráta přenášených zpráv
    - Zpravidla řešeno zabezpečeným síťovým přenosovým protokolem, např. TCP-IP
  - Selhání komunikační linky
    - Řešeno síťovými směrovacími protokoly, kdy se najde spojení po alternativních komunikačních cestách
  - Rozštěpení sítě
    - Síť se rozpadne na dvě nebo více částí vlivem ztráty konektivity. Jednotlivé subsítě a v nich obsažené uzly pracují samostatně dále
      - Poznámka: "Subsítí" může být i jediný uzel
  - Obecně platí, že rozštěpení sítě a havárii jediného uzlu nelze vzájemně odlišit

Potvrzovací protokoly

- Potvrzovací (commit) protokoly slouží k zajištění atomicity transakcí, které v distribuovaných systémech probíhají ve více uzlech
  - Transakce, která se vyhodnocuje ve více uzlech, musí být buď všude potvrzena nebo všude zrušena
  - Není přípustné, aby v jednom uzlu byla potvrzena (commited) a v jiném zrušena (aborted)
- Nejužívanější je dvoufázový potvrzovací protokol [*two-phase commit* (2PC) *protocol*]
  - Existuje i třífázový (3PC) protokol, který odstraňuje drobné nedostatky 2PC, avšak pro svoji složitost a komunikační náročnost se prakticky nepoužívá

Dvoufázový potvrzovací protokol (2PC)

- Předpokládá se tzv. *fail-stop* chování
  - Chybné (*failed*) uzly přestanou (*stop*) pracovat a nepáchají tak další škody
    - např. neposílají chybné zprávy ostatním uzlům
- Realizace 2PC protokolu je zahájena koordinátorem poté, co transakce dosáhla svého posledního kroku
  - Protokol zahrnuje všechny uzly, kterých se transakce týkala
  - Nechť T je transakce iniciovaná v uzlu S_i a nechť koordinátorem transakce v S_i je C_i

2PC protokol

- Fáze 1: Příprava
  - Koordinátor C_i požádá všechny participanty, aby připravili (*prepare*) "commit" transakce *T*
    - C_i zaprotokoluje záznam <***prepare*** *T*> a zapíše protokol na stabilní paměť a rozešle zprávu "***prepare*** *T*" všem uzlům, kde *T* probíhala
  - Správce transakcí v uzlu při přijetí "***prepare*** *T*" zkoumá, zda lze transakci *T* potvrdit
    - pokud ne, zaprotokoluje záznam <***no*** *T*> a pošle zprávu ***abort*** *T* do C_i
    - pokud ano, pak:
    - zaprotokoluje <***ready*** *T*>, *všechny záznamy* protokolu o transakci *T* uloží na stabilní paměť a pošle zprávu ***ready*** *T* do C_i
- Fáze 2: Rozhodnutí
  - Koordinátor C_i může *T* potvrdit, pokud dostane zprávy ***ready*** *T* od všech zúčastněných uzlů; jinak musí *T* zrušit
    - Přidá záznam <***commit*** *T*> nebo <***abort*** *T*> do lokálního protokolu a uloží ho na stabilní paměť
    - Pošle zprávy ***commit*** *T* nebo ***abort*** *T* všem zúčastněným uzlům a informuje je tak o výsledku
  - Zúčastnění pak provedou příslušné lokální akce

Řízení souběhu

- Pro distribuované prostředí je nutno modifikovat principy řízení souběhu
  - Předpokládáme, že uzly participují na potvrzovacím protokolu, aby byla zajištěna globální atomicita transakcí
  - Rovněž předpokládáme, že repliky dat budou aktualizovány
- Přístup s centrálním správou zámků
  - Systém má jediného správce zámků, který běží v uzlu S_i
  - Potřebuje-li transakce zámek, pošle žádost S_i a správce rozhodne, zda lze zámek přidělit okamžitě
    - Pokud ano, správce zámků odpoví kladně; jinak je odpověď pozdržena, dokud se zámek neuvolní
  - Transakce může číst data z *kterékoliv* repliky. Zápisy musí zaručovat aktualizaci všech replik datových položek
  - Výhoda:
    - Snadná implementace a správa zámků
  - Nevýhody:
    - Správce zámků se stává úzkým místem systému
    - Havárie či nedostupnost uzlu se správcem zámků zastaví celý systém

Distribuovaná správa zámků

- Funkcionalita zamykání je implementována lokálními správci zámků v každém uzlu
  - Správci zámků řídí přístup k lokálním datům
    - Repliky však užívají speciální protokoly
- Výhoda: robustní distribuované řízení
- Nevýhoda: komplikovaná detekce uváznutí
  - Lokální správci zámků musí spolu kooperovat při odhalování uváznutí
- Existuje několik variant distribuované správy zámků
  - Primární kopie
  - Majoritní protokol
  - Upřednostňující protokol
  - Rozhodné kvorum

Primární kopie

- Jedna replika se zvolí jako primární kopie dat
  - Příslušný uzel je pak primárním uzlem pro tato data
  - Různé datové položky mají tak různé primární uzly
  - Když transakce potřebuje zamknout datovou položku *Q*, požádá primární uzel této položky
    - Implicitně jsou tak zamčeny všechny repliky
- Přínos
  - Řízení souběhu je stejné, jakoby data nebyla replikována – jednoduchá implementace
- Nedostatek
  - Když zhavaruje primární uzel *Q*, pak *Q* se stane nedostupným, i když existují repliky (které jsou jinde, a tudíž dostupné)

Majoritní protokol

- Lokální správci zámků v jednotlivých uzlech administrují žádosti o zamykání a odmykání lokálních dat
  - Když transakce chce zamknout nereplikovaná data Q uložená v uzlu S_i, zašle žádost správci zámků v S_i
    - Je-li Q zamčeno nekompatibilním způsobem, žádost bude vyřízena, až když zamčení bude možné; jinak se vyřídí okamžitě
  - Pro replikovaná data je situace složitější
    - Je-li Q replikováno v n uzlech, pak se žádost posílá aspoň $n/2$ uzlům, na nichž jsou data replikována
    - Transakce nepokračuje, dokud nezíská zámek na většině uzlů, kde jsou data Q replikována
    - Při zápisu se aktualizují *všechny* repliky dat
  - Výhoda
    - Lze pracovat, i když jsou některé uzly nedostupné
      - problém s nedostupnými replikami pro zápisu (obnova po připojení)
  - Nevýhoda
    - Potřeba $2*(n/2 + 1)$ zpráv při zamykání a $(n/2 + 1)$ zpráv při odmykání
    - Nebezpečí uváznutí: např. 3 transakce a každá z nich drží 1/3 zámků na replikách

Upřednostňující protokol

- Lokální správci zámků v jednotlivých uzlech pracují stejně jako u majoritního protokolu, avšak rozlišují se sdílené (*S*-lock) zámky od výlučných (*X*-lock)
  - Sdílené zámky: Když transakce potřebuje sdíleně uzamknout položku *Q*, pak si prostě vyžádá zámek od správce zámků v uzlu, kde je replika *Q* umístěna
  - Výlučné zámky: Když transakce potřebuje uzamknout položku *Q* výlučně, pak musí získat *X*-zámky ze všech uzlů, kde je *Q* replikováno
- Vlastnosti
  - nižší režie při upřednostňovaných operací čtení
  - vysoká režie u operací zápisových
    - Srovnejte s úlohou "Čtenáři a písaři" při přednosti čtenářů

Rozhodné kvorum

- Zobecnění majoritního a upřednostňujícího protokolu
  - Každý uzel má přiřazenu svoji váhu
    - Nechť S je součet vah všech uzlů
  - Zvolí se dvě hodnoty: "čtecí kvorum" Q_r a "zápisové kvorum" Q_w, přičemž $Q_r + Q_w > S$ a $2 * Q_w > S$
    - Váhy uzlů a kvora mohou být vybírána pro každou položku samostatně
  - Každé čtení musí zamknout více než Q_r replik
  - Každý zápis musí zamknout více než Q_w replik
    - tedy nadpoloviční většinu uzlů s uvažováním jejich vah

Správa uváznutí

- Uvažme dvě transakce pracující s položkou P uloženou v uzlu U_1 a s položkou Q uloženou v uzlu U_2. Transakce T_1 běží v U_1 a transakce T_2 v uzlu U_2:

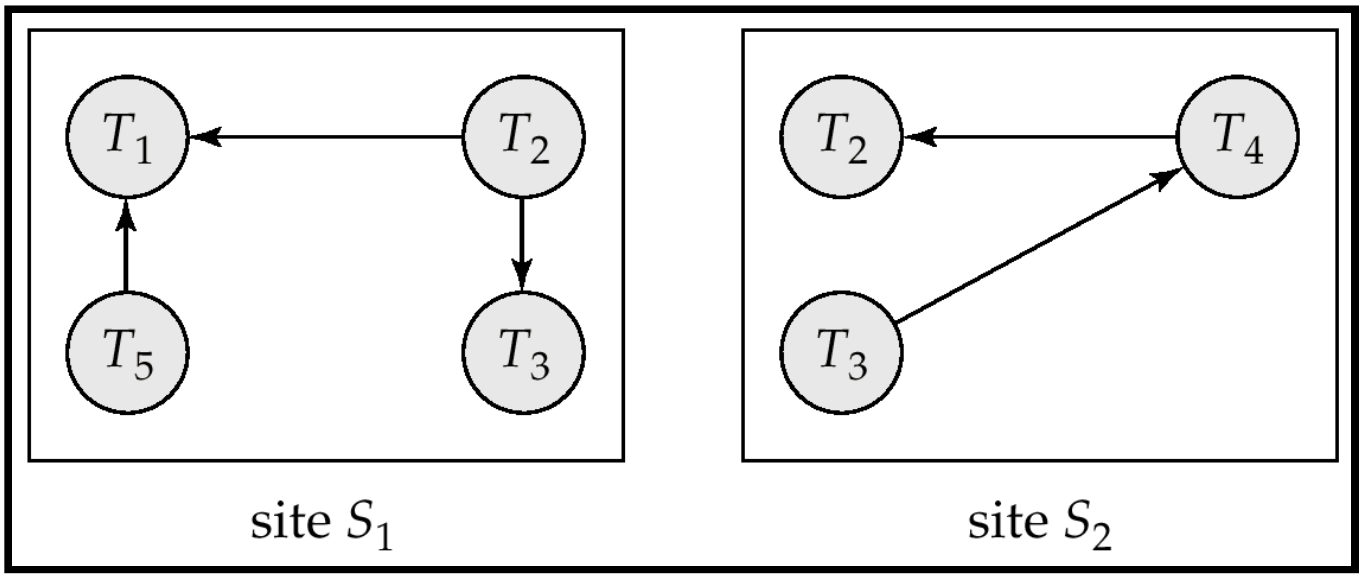
T_1: write (P) write (Q)	T_2: write (Q) write (P)
X-lock on P write (P)	*X*-lock on Q write (Q) wait for *X*-lock on P
wait for *X*-lock on Q	

Výsledek: Uváznutí, které nelze detekovat v žádném z uzlů

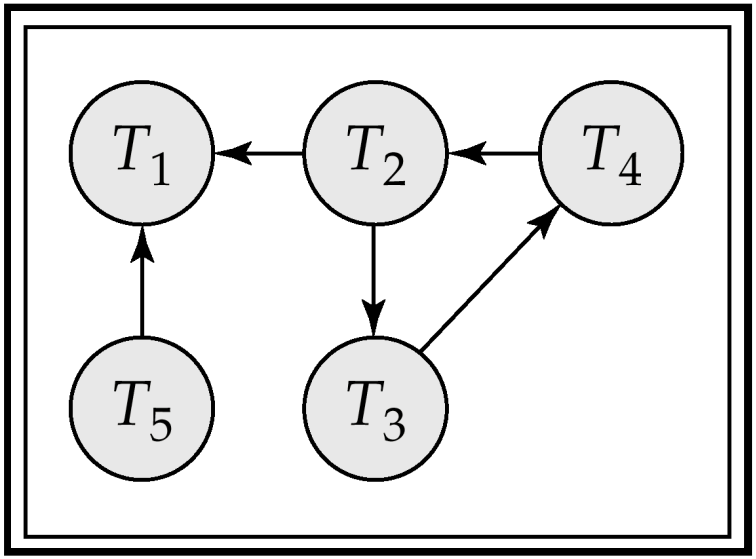
Centralizovaný přístup

- Jeden vybraný uzel – koordinátor detekce uváznutí – konstruuje globální čekací graf
  - *Reálný graf* nelze zkonstruovat, neboť stav systému se dynamicky mění a změny nastávají rychleji, než se o tom koordinátor může dovědět
  - *Konstruovaný graf*: Aproximace generovaná koordinátorem na základě informací z uzlů řídících transakce během jejich práce
  - Globální čekací graf lze konstruovat z informací, které koordinátor detekce získá z jednotlivých uzlů
    - Je přidána nová hrana nebo zmizí nějaká hrana v některém z lokálně budovaných čekacích grafů (tedy při každé změně kteréhokoliv grafu)
    - Lokální čekací graf změnil svůj tvar natolik, že uzel nahlásí tuto změnu koordinátoru
    - Koordinátor potřebuje spustit algoritmus detekce cyklů, a proto si vyžádá stav lokálních čekacích grafů ze všech uzlů
- Když koordinátor najde cyklus – detekuje uváznutí
  - Vybere oběť a oznámí ji všem zúčastněným uzlům. Tyto uzly pak zruší obětovanou transakci a obnoví data

Lokální a globální čekací grafy

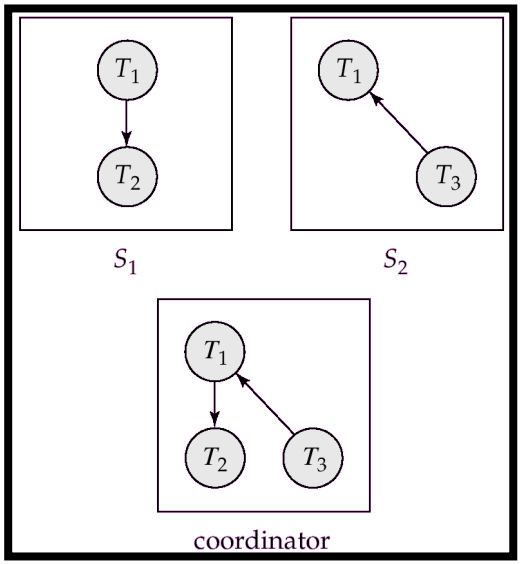


Local



Global

Falešné cykly



- Mějme počáteční stav dle obrázku, Pak
  1. T_2 uvolní položku v S_1
     - manažer transakcí v S_1 pošle zprávu "remove $T_1 \rightarrow T_2$" koordinátoru
  2. a T_2 požádá o položku zamčenou T_3 v S_2
     - tj. koordinátorovi odejde z S_2 zpráva "insert $T_2 \rightarrow T_3$"
- Předpokládejme, že zpráva "insert" dorazí dříve než zpráva "remove"
  - To může být důsledkem zpoždění v síti
- Koordinátor v tomto okamžiku zdetekuje falešný cyklus

  $$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1,$$

  který nikdy v reálu neexistoval
- Falešné cykly nevznikají, používá-li se ve všech uzlech striktní dvoufázové zamykání

- Praktické zkušenosti ukazují, že falešné cykly se detekují jen zřídka
  - Pozor ale na bezpečnostně kritické aplikace

Další otázky distribuovaných databází

- Problematika distribuovaných databází a distribuovaných systémů vůbec je velmi široká
- Heslovitě některé pojmy:
  - Dostupnost uzlů
    - detekce, rekonfigurace
  - Zajištění plnohodnotných replikací
    - šíření kopií primárních dat, číslování verzí replik
    - reintegrace systému po obnově dostupnosti uzlů
  - Havárie či nedostupnost koordinátora
    - záložní koordinátory
    - volby na téma: Kdo bude novým koordinátorem
  - Distribuované vyhodnocování dotazů
    - dotaz lze dekomponovat tak, aby se co nejvíce dat získávalo v uzlech lokálně a iniciátoru transakce se posílají jen výsledky lokálních částí
    - založeno na dobré fragmentaci dat a strategiích využívajících dobře formulovaná relační spojení (join)
  - Heterogenní distribuované systémy (databáze)
    - Hierarchicky nadřazená spojovací softwarová vrstva, které umí komunikovat s jednotlivými subsystémy a integruje je
    - Donedávna předmět výzkumu a stále problematická efektivita
    - Jedna z cest: LDAP: Lightweight Directory Access Protocol

Dotazy