12. Distribuované databázové systémy (DDBS)

12.1. Úvod

Distribuovaná databáze je kolekce dat, které logicky patří do jednoho systému, ale fyzicky jsou rozprostřena mezi jednotlivými místy (uzly) počítačové sítě. K rozvoji DDBS vedla řada faktorů – potenciální výhody DDBS zahrnují následující :

- *distribuovaná povaha některých databázových aplikací* – mnoho databázových aplikací je přirozeně rozděleno do různých míst. Například firma lokalizovaná v různých městech či banka s několika pobočkami. Je pochopitelně přirozené, že databáze používaná v těchto případech je sama rozdělena (rozmístěna) podle těchto umístění. Mnoho **lokálních uživatelů** tak může přistupovat pouze k datům, které jim v rámci svého umístění příslušejí, zatímco ovšem **globální uživatelé** (vedení firmy …) mohou využívat data rozmístěná v nejrůznějších místech (uzlech). Všimněme si, že data v každém lokálním sídle představují jakýsi „minisvět“ v tomto sídle, přičemž zdroje dat a většina uživatelů a aplikací lokální databáze se zde fyzicky nacházejí.
- *zvýšená spolehlivost a dostupnost*. Jedná se o dvě z nejčastěji citovaných potenciálních výhod DDBS. **Spolehlivost** je obecně definována jako pravděpodobnost, že systém je v každém dílčím okamžiku korektní a aktuální, zatímco **dostupnost** je pravděpodobnost, že systém průběžně (nepřetržitě) dostupný během určitého časového intervalu. Pokud jsou data a software SŘBD distribuována do různých míst, může nastat situace, že jedno místo selže, zatímco ostatní pokračují v práci. Tj. pouze data a software na zmíněném problematickém místě nejsou dostupná, což zvyšuje jak spolehlivost, tak i dostupnost. (Selže-li centralizovaná databáze, je celý systém nedostupný pro všechny uživatele). Vyšší spolehlivosti je možné dosáhnout, pokud se data i software replikují do více než jednoho místa – určitá data tak existují vícekrát.
- *řízené sdílení dat* – v některých DDBS je možné kontrolovat data a software lokálně v každém uzlu a jistá data mohou být zpřístupněna i uživatelům ze vzdálených uzlů prostřednictvím DSŘBD. To umožňuje řízené sdílení dat v rámci distribuovaného systému
- *zvýšený výkon* – je-li velká databáze rozmístěna (rozdělena) do více míst, v každém z těchto míst existuje menší databáze. Provádění lokálních dotazů a transakcí v každém jednotlivém uzlu je výkonnější, neboť se odehrává nad menší databází. Mimoto v každém uzlu se obecně provádí méně transakcí, než kdyby se veškeré transakce potvrzovaly v jedné centrální databázi

Distribuce vede ke zvýšené složitosti systémového návrhu a implementace. K zajištění výše uvedených předností DDBS je nezbytné, aby DSŘBD software byl schopen realizovat následující přídavné funkce (ke stávajícím, které normálně dává k dispozici centralizovaný systém)

- zpřístupnit vzdálené uzly, zajištění přenosu dotazů a dat mezi nejrůznějšími místy prostřednictvím komunikační sítě
- sledovat distribuce a replikace dat (evidence v katalogu DŘBD)
- stanovit výkonnou strategii pro provádění dotazů a transakcí týkajících se dat z různých uzlů

- rozhodnout, která kopie (replikovaná datová položka) bude použita pro určitou operaci
- zajistit konzistenci kopií replikovaných datových položek
- zajistit zotavení ze selhání individuálního uzlu a zvládnutí nových typů chyb jako je např. selhání komunikačního vedení

Tyto funkce zvyšují složitost DSŘBD oproti centralizovanému SŘBD. Dříve než však můžeme realizovat plné výhody distribučního systému, musíme najít uspokojivé řešení problémů s tím souvisejících. Zajistit veškerou požadovanou dodatečnou funkcionalitu systému není jednoduchá záležitost, a najít optimální řešení, je dalším krokem kupředu. Další složitosti se objevují, pokud zvažujeme návrh distribuované databáze. Specielně se musíme rozhodnout, jak rozdělit data do jednotlivých míst a zda budeme replikovat, a pokud ano, tak která data.

Na fyzické (hardwarové) úrovni existují následující odlišnosti:

- více počítačů označených jako **místa** či **uzly**
- uzly musejí být propojeny nějakým rozumným typem komunikační sítě

Podle potřeby se může jednat o **lokální síť** (LAN) či **rozsáhlou síť** (WAN) s nejrůznější topologií. Typ a topologie použité sítě mají pochopitelně významný vliv na výkon a odtud na strategii provádění distribuovaných dotazů a transakcí a návrh samotné distribuované databáze. V nejvyšší úrovni architektury DDBS není rozhodující, jaký typ sítě je použit, důležité je hlavně to, že každý uzel je schopen komunikovat (přímo nebo nepřímo) s libovolným jiným uzlem sítě.

12.2. Architektura klient – server

Architektura klient – server se používá pro práci v počítačových systémech, v nichž je řada osobních počítačů, pracovních stanic, file serverů, tiskáren a dalších zařízení propojena pomocí sítě. Základní ideou je definovat **specializované servery** zajišťující specifické funkce. Typickým příkladem jsou třeba tiskové servery zajišťující tisk pro celou řadu klientů (stanic, PC …) nebo file servery poskytující soubory oprávněným klientům - uživatelům bezdiskových stanic či servery nabízející specifický software (SŘBD , CAD …), který na běžné stanici není k dispozici.

Architektura klient – server je zahrnuta do řady komerčních SŘBD (Oracle, MS SQL Server, Sybase, …). Základní myšlenka je rozdělit SŘBD software do dvou úrovní – klient a server a snížit tak složitost systému. Na některých místech pak může být klient, na jiných (nebo pouze na jediné) bude server. Některá místa mohou fungovat jako klient i jako server, to není nikterak vyloučeno.

Problémem ovšem je, jak rozdělit funkčnost SŘBD mezi server a klienta, jak stanovit optimální „dělicí čáru“ (otázka tenkého a tlustého klienta). Jedna z variant je přenést veškerou funkčnost standardního centralizovaného systému na server – tuto možnost využívá řada relačních databázových systémů poskytujících SQL server. Klient musí zajistit určité uživatelské rozhraní dovolující uživateli v rozumné podobě komunikovat s databází – výsledkem této komunikace je pak dotaz formulovaný v SQL, který je odeslán k provedení na server. Klient se také může obracet na databázový katalog, který obsahuje informace o distribuci dat mezi jednotlivými SQL servery a moduly pro dekompozici globálních dotazů na lokální dotazy, které se pak mohou provádět na různých místech. Komunikace mezi klientem a serverem může vypadat následovně:

1. Klient rozparsuje uživatelský dotaz a rozloží ho na řadu nezávislých dílčích dotazů podle míst. Každý dílčí dotaz je poslán odpovídajícímu serveru.
2. Každý server zpracuje lokální dotaz a výsledek pošle zpět klientovi.
3. Klient zkombinuje výsledky poddotazů a vytvoří tak odpověď na původní dotaz.

V tomto postupu bývá server označován jako **databázový procesor** nebo **back-end stroj**, zatímco klient bývá nazván jako **aplikační procesor** nebo také **front-end stroj.** Interakce mezi klientem a serverem může být specifikována na klientské úrovni nebo prostřednictvím specielního klientského modulu. Například uživatel může vědět, která data jsou uložena na kterém serveru, může pak rozložit dotaz na lokální poddotazy ručně, předložit je příslušným serverům a pomocí dalšího uživatelem formulovaného dotazu zkombinovat jednotlivé výsledkové tabulky na úrovni klienta. Druhá varianta je mít k dispozici klientský modul, který toto provádí automaticky.

Jiný postup, používaný některými objektově orientovanými DSŘBD, rozděluje softwarové moduly DSŘBD mezi server a klienta daleko komplexněji. Na úrovni serveru může být zahrnuta část softwaru zodpovídající za ukládání dat na disk, řízení lokálního konkurenčního přístupu a obnovování, bufferování a cachování diskových stránek atd. Klientská úroveň může zajišťovat uživatelské prostředí, funkce datového slovníku, interakci s překladači programovacích jazyků, SŘBD optimalizaci, řízení víceuživatelského přístupu, obnovovací systém v rámci globálních dotazů atd. V tomto případě je interakce server-klient daleko těsněji spojena a je realizována vnitřně pomocí SŘBD modulů (nikoliv manuálně samotným uživatelem).

V typickém DSŘBD je obvyklé rozdělit softwarové moduly do tří úrovní:

- **serverový software** je zodpovědný za správu lokálních dat v uzlu podobně jako centralizovaný SŘBD software
- **klientský software** zodpovídá za většinu distribuovaných funkcí, zpřístupňuje informace o distribuci dat z DSŘBD katalogu a zpracovává veškeré požadavky, které vyžadují přístup k více než jednomu místu (uzlu)
- **komunikační software** zajišťuje základní komunikační operace a služby potřebné pro přenos příkazů a dat mezi jednotlivými místy. (Nemusí být nutně částí DSŘBD).

Klient je zodpovědný za vytvoření distribuovaného prováděcího plánu pro provádění dotazů a transakcí týkajících se více míst a za dohled nad samotným distribuovaným prováděním prostřednictvím příkazů posílaných na jednotlivé servery. Tyto příkazy zahrnují lokální dotazy a transakce stejně tak jako přenos dat k jiným klientům či serverům. Tj. klientský software musí být všude tam, kde se pracuje s dotazy obracejícími se na více míst.

Další funkce řízená klientem je zajištění konzistence replikovaných kopií datových položek s použitím distribuovaných či globálních technik pro řízení víceuživatelského přístupu. Pokud určité místo vypadne (selže), pak musí klient zajistit atomicitu globálních transakcí použitím globálního obnovovacího systému.

Jednou z možných funkcí klienta je skrýt před uživatelem detaily o distribuci dat – tj. umožnit uživateli psát globální dotazy a transakce, aniž musí specifikovat umístění jednotlivých dat (jako by databáze byla centralizovaná) – **distribuční transparence**. Některé DSŘBD neposkytují distribuční transparenci, takže se uživatel musí starat o to, jak jsou data distribuována a musí tedy v příslušných dotazech a transakcích uvádět umístění dat. Většinou se to realizuje tak, že se přidává jméno místa k odkazu na relaci, soubor či záznam. Pokud

naopak databáze poskytuje plnou transparenci, uživatel má k dispozici schéma neobsahující údaje o distribuci dat – příslušné informace jsou v DSŘBD katalogu. Tyto informace jsou využívány klientským softwarem již výše zmíněným způsobem ke zpracování globálních dotazů. Uživatel má jednodušší práce , ale samozřejmě příslušný software je složitější.

12.3. Fragmentace dat, replikace, alokační techniky v návrhu DD

12.3.1. Fragmentace dat

V DDBS se mimo jiné jedná o to, jak optimálně rozdělit data na jednotlivá místa. Pro jednoduchost zatím předpokládejme, že data nejsou replikovaná tj. že každý soubor či část souboru je uložena na právě jediném místě. Začněme s relačním databázovým schématem daným na příkladu 12.1 a zkusme určit, jak rozdělit data na místa.

Příklad 12.1 – Firma

ZAMĚSTNANEC	CisloZ	PrijmeniZ	JmenoZ	AdresaZ
	DatNarZ	PohlaviZ	OddZ	PlatZ

ODDĚLENÍ	CisloO	NazevO	MistoO	VedouciO

PROJEKT	CisloP	OddP	NazevP

PRACUJE_NA	Zam	Proj	Hodiny

DÍTĚ	CisloZ	JmenoD	DatNarD

Příklad 12.2 - Konkrétní instance databáze firmy

ODDĚLENÍ	**CisloO**	**NazevO**	**MistoO**	**VedouciO**
	10	Účtárna	Č. Budějovice	11
	20	Projekce	Č. Krumlov	08
	30	Analýza	Velešín	07

ZAMĚSTNANEC	CisloZ	PrijmeniZ	JmenoZ	DatNarZ	PohlaviZ	AdresaZ	OddZ	PlatZ
	01	Bláha	Jan	05.04.72	M	Č. Budějovice	20	
	02	Adamec	Josef	15.07.80	M	Tábor	30	
	03	Berková	Jana	25.11.71	Z	Č. Budějovice	10	
	04	Kysilka	Petr	14.09.78	M	Velešín	30	
	05	Marková	Radka	22.02.72	Z	Č. Budějovice	10	
	06	Součková	Eva	13.12.80	Z	Velešín	10	
	07	Novotný	Pavel	19.01.70	M	Kaplice	20	
	08	Peterka	Jakub	15.06.79	M	Kaplice	20	
	09	Suchánek	Petr	04.10.71	M	Tábor	20	
	10	Zelenka	Jiří	14.07.77	M	Č. Budějovice	30	
	11	Hanousek	Pavel	07.07.74	M	Č. Budějovice	20	
	12	Pavelka	Jiří	18.09.70	M	Soběslav	30	
	13	Straka	David	12.03.68	M	Č. Budějovice	30	
	14	Vávra	Pavel	18.04.80	M	Trhové Sviny	20	
	15	Červenka	Jan	23.10.69	M	Trhové Sviny	20	

DÍTĚ	CisloZ	JmenoD	DatNarD
	05	Kateřina	17.04.00
	03	Jakub	19.11.98
	07	Daniel	11.05.99
	09	Petra	14.04.96
	09	Pavel	14.04.96
	13	Alena	24.02.91

PROJEKT	CisloP	OddP	NazevP
	1001	10	Mzdové náklady
	2001	20	Analýza prodeje
	2002	20	Výrobní náklady
	2003	20	Úspora spotřeby energie
	2004	20	Skladové hospodářství
	3001	30	Analýza cen
	3002	30	Fakturace
	3003	30	Bezpečnost systému

PRACUJE_NA	Zam	Proj	Hodiny
	05	2004	20
	05	3002	20
	06	1001	15
	08	2004	28
	10	3003	24
	11	2001	10
	12	2002	12
	13	2002	18
	13	3002	16
	13	3003	14
	14	2003	25
	15	3001	10

Dříve než se rozhodneme, jak distribuovat data, je třeba určit **logickou jednotku** (část databáze), která se bude distribuovat. Nejjednodušší logickou jednotkou je relace tj. každá relace je umístěna v nějakém lokálním uzlu. V našem příkladu je třeba rozhodnout, kam umístíme relace ZAMĚSTNANEC, ODDĚLENÍ, PROJEKT, PRACUJE_NA a DÍTĚ. (V řadě případů je relace v rámci distribuce rozdělena na menší logické jednotky). Předpokládejme, že firma má pouze tři oddělení s čísly 10, 20, 30 a má tedy tři uzly – jeden pro každé oddělení. Můžeme chtít uložit informace týkající se každého oddělení do příslušného uzlu, v tom případě potřebujeme rozdělit každou relaci pomocí techniky označované jako **horizontální fragmentace.**

12.3.2. Horizontální a vertikální fragmentace

Horizontální fragment relace je podmnožina n-tic (záznamů) v relaci. N-tice patřící do horizontálního fragmentu je dána podmínkou na jeden nebo více atributů v relaci (selekce). V našem příkladu můžeme definovat tři podmínky pro fragmentaci relace ZAMĚSTNANEC: (OddZ = 10), (OddZ = 20), (OddZ = 30), každý fragment obsahuje záznamy pracovníků zařazených na příslušné oddělení. Podobně samozřejmě můžeme definovat fragmenty relace PROJEKT podmínkami : (CisloO = 10), (CisloO = 20), (CisloO = 30). Horizontální fragmentace rozděluje relaci horizontálně – vznikají tak skupiny řádků (podmnožiny relace), které k sobě logicky patří. Tyto fragmenty pak mohou být umístěny každý do jiného uzlu.

Vertikální fragment relace obsahuje pouze vybrané atributy relace (projekce). Mohli bychom například rozdělit relaci ZAMĚSTNANEC na dva vertikální fragmenty, z nichž první by obsahoval personální informace (JmenoZ, PrijmeniZ, AdresaZ, DatNarZ, PohlaviZ) a druhý pracovní informace (CisloZ, OddZ, PlatZ). Toto rozdělení ovšem není správné, protože pokud jsou tyto fragmenty uloženy samostatně, již nikdy nezískáme původní informace, protože v nich není obsažen rozumný společný (spojující) atribut. Do každého vertikálního fragmentu je tedy nezbytné zahrnout atributy primárního klíče relace, jen tak je možné z fragmentů rekonstruovat původní relaci.

Uvědomme si, že každý horizontální fragment relace R je dán selekcí $\sigma_{Ci}(R)$. Množina horizontálních fragmentů, jejíž podmínky C1, C2 … Cn zahrnují všechny n-tice relace R tj.

každá n-tice vyhovuje podmínce (C1 OR C2 OR …Cn) , je nazvána jako **kompletní horizontální fragmentace** relace R. V řadě případů je kompletní horizontální fragmentace disjunktní tj. žádná n-tice nesplňuje podmínku (Ci AND Cj) pro všechna i ≠ j. Oba výše uvedené příklady horizontální fragmentace relací ZAMĚSTNANEC a PROJEKT byly kompletní i disjunktní. Pokud potřebujeme rekonstruovat původní relace z kompletní horizontální fragmentace, stačí použít operaci sjednocení (UNION).

Vertikální fragment je dán operací $\pi_{Li}(R)$. Množina vertikálních fragmentů, v níž projekční seznamy atributů L1, L2, …, Ln zahrnují všechny atributy relace R a mají společné pouze atributy primárního klíče, se nazývá **kompletní vertikální fragmentace** relace R. Tj. musí platit:

- $L1 \cup L2 \cup \ldots \cup Ln = ATTRS(R)$
- $Li \cap Lj = PK(R)$ pro všechny $i \neq j$, kde ATTRS(R) je množina všech atributů R a PK(R) je primární klíč R

K rekonstrukci relace z kompletní vertikální fragmentace je třeba aplikovat operaci vnější sjednocení OUTER UNION (nebo plné vnější spojení FULL OUTER JOIN). Dva vertikální fragmenty relace ZAMĚSTNANEC s projekčními seznamy L1 = {CisloZ, JmenoZ, PrijmeniZ, AdresaZ, DatNarZ, PohlaviZ} a L2 = {CisloZ, OddZ, PlatZ} představují kompletní vertikální fragmentaci.

Příkladem dvou horizontálních fragmentů, které nejsou ani kompletní, ani disjunktní, jsou fragmenty dané podmínkou C1 = (PlatZ > 20000) a C2 = (OddZ = 20)

Kombinovaná (mixed) fragmentace vznikne jako kombinace obojího výše uvedeného. Původní relaci pak získáme pomocí operací sjednocení a vnější sjednocení (vnější spojení) v odpovídajícím pořadí. Obecně takovýto fragment relace se dá získat pomocí kombinace selekce a projekce $\pi_L(\sigma_C (R))$. Pokud je C = TRUE a L ≠ ATTRS(R), pak získáme vertikální fragment, je-li naopak C ≠ TRUE a L = ATTRS(R), pak získáme horizontální fragment. Konečně je-li C ≠ TRUE a L ≠ ATTRS(R), pak získáme kombinovaný fragment. (Celá relace je jediný fragment s C = TRUE a L = ATTRS(R).)

Fragmentační schéma databáze je definice množiny fragmentů, které zahrnují všechny atributy a n-tice (záznamy) databáze a splňují podmínku, že celá databáze může být rekonstruována z fragmentů aplikováním posloupnosti operací OUTER UNION (nebo OUTER JOIN) a UNION. Je užitečné, ale nikoliv nezbytné, aby všechny fragmenty byly disjunktní (s výjimkou atributů primárního klíče).

Alokační schéma popisuje uložení fragmentů v jednotlivých místech (uzlech) databáze tj. pro každý fragment je specifikován uzel, kde je uložen. Je-li fragment uložen na více než jednom místě, hovoříme o **replikaci.**

12.4. Replikace a alokace dat

Replikace dat je užitečná z důvodu zvýšení dostupnosti dat. Nejextrémnější případ představuje situaci, kdy celá databáze je uložena v každém uzlu distribuovaného systému – **plně replikovaná** distribuovaná databáze. To samozřejmě značně zvyšuje dostupnost dat, neboť systém může pokračovat v provádění operací tak dlouho, dokud aspoň jedno místo funguje. Dále to zvyšuje výkon při provádění globálních dotazů, neboť v každém uzlu jsou k dispozici všechna data, takže není nutné obracet se do jiných uzlů a zdržovat se přenosem po síti. Nevýhodou plné replikace je drastické snížení výkonu při provádění aktualizačních operací – každá lokální aktualizace v jediném uzlu musí následně vyvolat aktualizaci ve všech

ostatních místech distribuovaného systému tak, aby data byla stále konzistentní. Plná replikace tak komplikuje a prodražuje řízení víceuživatelského přístupu a obnovovací systém.

Druhým extrémem je **žádná replikace** – každý fragment je uložen právě jednou (v jistém uzlu). To vyžaduje disjunktní fragmenty (s výjimkou atributů primárního klíče) – tzv. **neredundantní alokace.**

Mezi těmito výše uvedenými extrémy se nalézá široké spektrum **částečných replikací** dat – některé fragmenty databáze jsou replikovány, zatímco jiné nikoliv, počet replikací se může pohybovat někde mezi jedničkou a počtem uzlů v distribuovaném systému – **replikační schéma**.

Každý fragment (každá kopie fragmentu) musí být přiřazen na určité místo v distribuovaném systému. Tento proces je nazýván **distribuce dat** (či **alokace dat**). Výběr míst a stupeň replikace závisí na výkonu a dostupnosti míst v systému a na typech a frekvenci transakcí prováděných v jednotlivých místech. Například je-li požadována vysoká dostupnost a transakce mohou být jsou prováděny v každém uzlu a většina transakcí pouze získává data (a neaktualizuje), pak je dobrou volbou plná replikace. Pokud určité transakce pracují jen s některými daty a provádějí se jen v určitém místě, pak je rozumné provést odpovídající částečnou replikaci v daném místě. Data, která jsou potřebná v určitých místech, jsou pak replikována v těchto místech. Pokud se provádí hodně aktualizací, je třeba omezit počet replikací. Nalezení optimálního či dobrého řešení distribuované alokace dat představuje složitý optimalizační problém.

Příklad 12.3

Uvažujme databázi firmy dle příkladu 12.2 a předpokládejme, že máme pouze 3 oddělení s čísly 10, 20, 30.

Předpokládejme tím pádem, že firma má tři uzly s čísly 1, 2, 3 (pro každé oddělení jeden). V každém oddělení očekáváme častý přístup k informacím o příslušných zaměstnancích a projektech, které spadají pod příslušné oddělení. Uzel 1 je navíc k dispozici vedení firmy, které požaduje přístup k informacím o všech zaměstnancích a projektech i o dětech zaměstnanců.

V souladu s těmito požadavky uložíme celou databázi do uzlu 1 a určíme, které fragmenty budou replikovány v uzlech 2 a 3. Nejprve horizontálně rozdělíme relace ZAMĚSTNANEC, PROJEKT a ODDĚLENÍ podle čísla oddělení (položky OddZ, OddP a CisloO). Pak vertikálně rozdělíme fragmenty relace ZAMĚSTNANEC na fragmenty obsahující atributy {CisloZ, JmenoZ, PrijmeniZ, OddZ, PlatZ} a uložíme je do příslušných uzlů podle oddělení. Získáváme tak fragment ZAM20 uložený v uzlu 2 a ZAM30 v uzlu 3, oba fragmenty jsou replikovány, neboť jsou vlastně uloženy i v uzlu 1.

Řešení 12. 3

Kombinovaná fragmentace relace ZAMĚSTNANEC

ZAM20	CislоZ	PrijmeniZ	JmenoZ	OddZ	PlatZ
	01	Bláha	Jan	20	
	07	Novotný	Pavel	20	
	08	Peterka	Jakub	20	
	09	Suchánek	Petr	20	
	11	Hanousek	Pavel	20	
	14	Vávra	Pavel	20	
	15	Červenka	Jan	20	

ZAM30	CislоZ	PrijmeniZ	JmenoZ	OddZ	PlatZ
	02	Adamec	Josef	30	
	04	Kysilka	Petr	30	
	10	Zelenka	Jiří	30	
	12	Pavelka	Jiří	30	
	13	Straka	David	30	

Dále je třeba rozdělit relaci PRACUJE_NA, což se může jevit jako problém, neboť žádný z atributů příslušné relace nemá žádnou souvislost s oddělením tj. nemůžeme přímo určit, kam každou n-tici relace PRACUJE_NA umístit. Každá n-tice se vztahuje k zaměstnanci **z**, který pracuje na projektu **p**, můžeme tedy fragmentovat relaci PRACUJE_NA podle toho, ve kterém oddělení **o** pracuje výše uvedený zaměstnanec **z** nebo podle oddělení **d'**, které řídí projekt **p**. Pokud by bylo dáno omezení, že zaměstnanec může pracovat pouze na projektech svého vlastního oddělení tj. d = d', pak by byla fragmentace jednoduchá. Toto omezení ovšem v našem případě neplatí viz např. záznam <13, 2002, 18>, kde zaměstnanec s číslem 13 patří na oddělení 30 a pracuje na projektu řízeném oddělením 20.

Příklad 12.4

Kompletní a disjunktní fragmentace PRACUJE_NA

a) fragmenty pro zaměstnance oddělení 10

```
SELECT * FROM PRACUJE_ NA
WHERE   (Zam IN (SELECT CislоZ FROM ZAMĚSTNANEC WHERE OddZ = 10))
AND     (PROJ IN (SELECT CislоP FROM PROJEKT WHERE OddP=10))
```

G1	Zam	Proj	Hodiny
	06	1001	15

```
SELECT * FROM PRACUJE_ NA
WHERE   (Zam IN (SELECT CisloZ FROM ZAMĚSTNANEC WHERE OddZ = 10))
AND     (PROJ IN (SELECT CisloP FROM PROJEKT WHERE OddP=20))
```

G2	Zam	Proj	Hodiny
	05	2004	20

```
SELECT * FROM PRACUJE_ NA
WHERE   (Zam IN (SELECT CisloZ FROM ZAMĚSTNANEC WHERE OddZ = 10))
AND     (PROJ IN (SELECT CisloP FROM PROJEKT WHERE OddP=30))
```

G3	Zam	Proj	Hodiny
	05	3002	20

b) fragmenty pro zaměstnance oddělení 20

```
SELECT * FROM PRACUJE_ NA
WHERE   (Zam IN (SELECT CisloZ FROM ZAMĚSTNANEC WHERE OddZ = 20))
AND     (PROJ IN (SELECT CisloP FROM PROJEKT WHERE OddP=10))
```

G4	Zam	Proj	Hodiny

```
SELECT * FROM PRACUJE_ NA
WHERE   (Zam IN (SELECT CisloZ FROM ZAMĚSTNANEC WHERE OddZ = 20))
AND     (PROJ IN (SELECT CisloP FROM PROJEKT WHERE OddP=20))
```

G5	Zam	Proj	Hodiny
	08	2004	28
	11	2001	10
	14	2003	25

```
SELECT * FROM PRACUJE_ NA
WHERE   (Zam IN (SELECT CisloZ FROM ZAMĚSTNANEC WHERE OddZ = 20))
AND     (PROJ IN (SELECT CisloP FROM PROJEKT WHERE OddP=30))
```

G6	Zam	Proj	Hodiny
	15	3001	10

c) fragmenty pro zaměstnance oddělení 30

```
SELECT * FROM PRACUJE_ NA
WHERE   (Zam IN (SELECT CisloZ FROM ZAMĚSTNANEC WHERE OddZ = 30))
AND     (PROJ IN (SELECT CisloP FROM PROJEKT WHERE OddP=10))
```

G7	Zam	Proj	Hodiny

```
SELECT * FROM PRACUJE_ NA
WHERE   (Zam IN (SELECT CisloZ FROM ZAMĚSTNANEC WHERE OddZ = 30))
AND     (PROJ IN (SELECT CisloP FROM PROJEKT WHERE OddP=20))
```

G8	Zam	Proj	Hodiny
	12	2002	12
	13	2002	18

```
SELECT * FROM PRACUJE_ NA
WHERE   (Zam IN (SELECT CisloZ FROM ZAMĚSTNANEC WHERE OddZ = 30))
AND     (PROJ IN (SELECT CisloP FROM PROJEKT WHERE OddP=30))
```

G9	Zam	Proj	Hodiny
	10	3003	24
	13	3002	16
	13	3003	14

V příkladu 12.4 dává sjednocení fragmentů G1, G2 a G3 všechny n-tice (záznamy) relace PRACUJE_NA týkající se zaměstnanců zařazených na oddělení 10, podobně sjednocení fragmentů G4, G5 a G6 dává všechny záznamy relace PRACUJE_NA týkající se zaměstnanců zařazených na oddělení 20 a samozřejmě sjednocení fragmentů G7, G8 a G9 dává všechny záznamy relace PRACUJE_NA týkající se zaměstnanců oddělení 30. Na druhé straně sjednocení fragmentů G1, G4, G7 dává všechny záznamy relace PRACUJE_NA týkající se projektů řízených oddělením 10. Relace reprezentující vztah M : N jako je v našem případě PRACUJE_NA se dá fragmentovat různými způsoby.

V příkladu 12.5 s distribučním schématem databáze firmy zahrneme do distribuce všechny fragmenty, které se dají připojit buďto k záznamům relace ZAMĚSTNANEC nebo relace PROJEKT umístěných v uzlech 2 a 3. Umístíme tedy sjednocení fragmentů G2, G4, G5, G6 a G8 do uzlu 2 a sjednocení fragmentů G3, G6, G7, G8 a G9 do uzlu 3. (Všimněme si, že fragmenty G6 a G8 jsou replikovány v obou uzlech.) Tato alokační strategie dovoluje spojení mezi lokálními fragmenty ZAMĚSTNANEC a PROJEKT v uzlu 2 nebo uzlu 3 s tím, že lokální fragment PRACUJE_NA může být zpracováván kompletně lokálně.

Příklad 12.5

a) **fragmenty v uzlu 1**

celá databáze

b) **fragmenty v uzlu 2**

ZAM20	**CisloZ**	**PrijmeniZ**	**JmenoZ**	**OddZ**	**PlatZ**
	01	Bláha	Jan	20	
	07	Novotný	Pavel	20	
	08	Peterka	Jakub	20	
	09	Suchánek	Petr	20	
	11	Hanousek	Pavel	20	
	14	Vávra	Pavel	20	
	15	Červenka	Jan	20	

ODD20	**CisloO**	**NazevO**	**MistoO**	**VedouciO**
	20	Projekce	Č. Krumlov	08

PROJ20	**CisloP**	**OddP**	**NazevP**
	2001	20	Analýza prodeje
	2002	20	Výrobní náklady
	2003	20	Úspora spotřeby energie
	2004	20	Skladové hospodářství

PRAC20	**Zam**	**Proj**	**Hodiny**
	05	2004	20
	08	2004	28
	11	2001	10
	12	2002	12
	13	2002	18
	14	2003	25
	15	3001	10

c) fragmenty v uzlu 3

ZAM30	CisloZ	PrijmeniZ	JmenoZ	OddZ	PlatZ
	02	Adamec	Josef	30	
	04	Kysilka	Petr	30	
	10	Zelenka	Jiří	30	
	12	Pavelka	Jiří	30	
	13	Straka	David	30	

PROJ30	CisloP	OddP	NazevP
	3001	30	Analýza cen
	3002	30	Fakturace
	3003	30	Bezpečnost systému

ODD30	CisloO	NazevO	MistoO	VedouciO
	30	Analýza	Velešín	07

PRAC30	Zam	Proj	Hodiny
	05	3002	20
	10	3003	24
	12	2002	12
	13	2002	18
	13	3002	16
	13	3003	14
	15	3001	10

12.5. Typy distribuovaných databázových systémů

Pojem **distribuované systémy řízení báze dat** (DSŘBD s anglickou zkratkou DDBMS) může zahrnovat celou řadu systémů, které se v mnoha aspektech odlišují. Společným myšlenkou všech těchto systémů je fakt, že data a software je distribuován (rozdělen a rozmístěn) do jednotlivých míst (uzlů), které jsou vzájemně propojeny nějakým druhem komunikační sítě. V následujícím výkladu uvedeme kritéria a faktory, podle kterých se jednotlivé distribuované systémy odlišují.

Prvním faktorem je **stupeň homogenity** DSŘBD softwaru. Pokud všechny servery používají stejný software a všichni klienti používají stejný software, pak označujeme DSŘBD jako **homogenní**, není-li tomu tak, pak se jedná o **heterogenní** systém.

Dalším faktorem je **stupeň lokální autonomie**. Pokud se veškeré přístupy k DSŘBD odehrávají prostřednictvím klientů, pak systém nemá **žádnou lokální autonomii**. Na druhé

straně je-li povolen jistý *přímý přístup* lokálních transakcí k serveru, pak se dá hovořit o určité lokální autonomii.

Extrémním příkladem lokální autonomie je takový systém, který se vůči uživateli tváří jako centralizovaný systém, kdy existuje jednoduché konceptuální schéma a veškeré přístupy jsou realizovány přes klienta tj. neexistuje žádná lokální autonomie. Opačným extrémem je typ DSŘBD označovaný jako **federativní DSŘBD** (nebo **multidatabázový systém**). V tomto systému představuje každý server nezávislý a autonomní centralizovaný SŘBD, který má své vlastní lokální uživatele, lokální transakce a DBA a tím pádem dosahuje vysokého stupně lokální autonomie. Každý server může autorizovat přístup k jisté podmnožině dat ze své databáze tím, že stanoví **exportní schéma**, které specifikuje tu část databáze, co má být zpřístupněna jisté skupině nelokálních uživatelů. Klient v takovémto systému musí zahrnovat nezbytné rozhraní k výše zmíněným serverům, které dovolují „multidatabázovému" (globálnímu) uživateli přistupovat k datům uloženým v některých z těchto autonomních databází. Všimněme si, že federativní systém je něco jako kříženec mezi distribuovaným a centralizovaným systémem – pro lokální uživatele se tváří jako centralizovaný systém a pro globální uživatele jako distribuovaný systém.

V **heterogenním databázovém systému** může být jeden server relační SŘBD, jiný server síťový SŘBD a další třeba hierarchický SŘBD. V takovém případě je ovšem nezbytné mít jistý kanonický jazykový systém a zahrnout pak do klientského softwaru překladače, které překládají dotazy z kanonického jazyka do jazyka každého serveru.

Třetím faktorem, který se používá pro kategorizaci distribuovaných databázových systémů, je **stupeň distribuční transparence,** nebo **stupeň integrace schématu**. Pokud uživatel „vidí" jednoduché nedílné schéma bez jakékoli informace týkající se fragmentace, replikace či distribuce, pak se jedná o systém s *vysokým stupněm distribuční transparence*. Pokud uživatel „vidí" veškerou fragmentaci, alokaci a distribuci, pak systém nemá *žádnou distribuční transparenci.* V tomto případě, pokud se uživatel odkazuje na určitou kopii fragmentu v určitém uzlu, musí uvádět při formulaci dotazu též jméno uzlu před jménem relace či fragmentu. Jedná se o tzv. problém **pojmenovávání** v distribučním systému. V případě DSŘBD neposkytujícího distribuční transparenci je na uživateli jednoznačně specifikovat jméno jednotlivé relace či kopie fragmentu. Tento úkol je mnohem problematičtější v multidatabázovém systému, protože lokální SŘBD software každého serveru byl pravděpodobně vyvíjen samostatně, takže může dojít ke konfliktu jmen. U distribuovaných systémů poskytujících integrované schéma je pojmenovávání vnitřní systémovou záležitostí, protože uživateli je předkládáno jednoduché jednoznačné schéma. DSŘBD musí ukládat veškeré souvislosti mezi objekty integrovaného schématu a objekty distribuovanými mezi různými DPs v **distribučním katalogu**.

12.6. Provádění dotazů v distribučním databázovém systému

12.6.1. Cena datového přenosu při provádění distribuovaných dotazů

Provádění a optimalizace dotazů v centralizovaných databázích bylo rozebíráno již dříve v souvislosti s formulací dotazů v SQL. V distribuovaném systému je řada dalších faktorů, které provádění dotazů komplikují. Prvním z nich je cena přenosu dat po síti. Data zahrnují jak dočasné soubory, které se přenášejí k určitým uzlům k dalšímu zpracování, tak i soubory s konečnými výsledky dotazů, které se musejí přenést do místa, kde je výsledek dotazu požadován. Cena datového přenosu nehraje roli, pokud jsou jednotlivá místa propojena výkonnou lokální sítí, ale v ostatních případech se jedná o významný faktor. Algoritmus pro

optimalizaci DSŘBD dotazů bere tím pádem snížení *množství přenášených dat* jako důležité optimalizační kritérium při výběru strategie provádění dotazů.

Předpokládejme zjednodušenou situaci podle obrázku 12.6 a zkusme formulovat dva jednoduché dotazy.

Příklad 12.6

Uzel 1

ZAMĚSTNANEC	CisloZ	PrijmeniZ	JmenoZ	DatNarZ	PohlaviZ	AdresaZ	OddZ	PlatZ

10 000 záznamů
1 záznam 100 B
CisloZ 5 B OddZ 3 B
PrijmeniZ 15 B JmenoZ 15 B

Uzel 2

ODDĚLENÍ	CisloO	NazevO	MistoO	VedouciO

100 záznamů
1 záznam 58 B
CisloO 3 B NazevO 25 B
MistoO 25 B VedouciO 5 B

Uzel 3 a další uzly (k následujícímu výpočtu nepotřebujeme znát fragmenty v těchto uzlech)

Předpokládejme, že žádná relace není fragmentována tj. celkové množství dat je

uzel 1 : 10 000 * 100 B = 1 000 000 B (relace ZAMĚSTNANEC)

uzel 2 : 100 * 58 B = 5 800 B (relace ODDĚLENÍ)

Příklad 12.7:

Uvažujme dotaz Q1 : Pro každého zaměstnance zjistěte jeho jméno a příjmení a jméno oddělení, pro které pracuje

Q1: $\P_{JmenoZ, PrijmeniZ, NazevO}$ (ZAMĚSTNANEC $|x|_{OddZ=CisloO}$ ODDĚLENÍ)

Výsledkem našeho dotazu bude 10 000 záznamů, neboť každý zaměstnanec je povinně zařazen na právě jediné oddělení

Předpokládejme dále, že výše zmíněný dotaz vyšel z uzlu 3 a odpověď tedy chceme dodat také do uzlu 3 (**výsledkový uzel**).

Můžeme použít tři následující strategie provádění dotazu.

1. Přenést obě relace (ZAMĚSTNANEC i ODDĚLENÍ) do výsledkového uzlu a provést operaci spojení v tomto uzlu.
   Přenášíme tedy celkem 1 000 000 B + 5 800 B = **1 005 800 B**.

2. Přenést relaci ZAMĚSTNANEC do uzlu 2, provést operaci spojení (JOIN) v uzlu 2 a přenést výsledek k uzlu 3. Přenášíme tedy nejprve 1 000 000 B a po provedeném spojení a projekci na atributy JmenoZ, PrijmeniZ a NazevO je nutné přenést do uzlu 3 10 000 * 55 B tj. celkem 1 000 000 B + 550 000 B = **1 550 000 B**.
3. Přenést relaci ODDĚLENÍ do uzlu 1, tam provést spojení a výsledek dodat do uzlu 3. Tj. celkový přenos činí 5 800 B + 550 000 B = **555 800 B**.

Vezmeme-li jako hlavní kritérium minimum přenášených dat, pak **strategie 3 je nejlepší.**

Příklad 12.8:

Uvažujme dotaz Q2: Pro každé oddělení zjistěte jeho název a jméno a příjmení jeho vedoucího

Q2: ¶ $_{\text{JmenoZ, PrijmeniZ, NazevO}}$ (ODDĚLENÍ |x| $_{\text{VedouciO=CisloZ}}$ ZAMĚSTNANEC)

a) výsledek požadujeme v uzlu 3

Strategie:

1. Přenést obě relace (ZAMĚSTNANEC i ODDĚLENÍ) do výsledkového uzlu a provést operaci spojení v tomto uzlu.
   Přenášíme tedy celkem 1 000 000 B + 5 800 B = **1 005 800 B**.
2. Přenést relaci ZAMĚSTNANEC do uzlu 2, provést operaci spojení (JOIN) v uzlu 2 a přenést výsledek k uzlu 3. Přenášíme tedy nejprve 1 000 000 B a po provedeném spojení a projekci na atributy JmenoZ, PrijmeniZ a NazevO je nutné přenést do uzlu 3 100 * 55 B tj. celkem 1 000 000 B + 5 500 B = **1 005 500 B**.
3. Přenést relaci ODDĚLENÍ do uzlu 1, tam provést spojení a výsledek dodat do uzlu 3. Tj. celkový přenos činí 5 800 B + 5 500 B = **11 300 B**.

Opět se ukazuje jako **nejlepší strategie 3**.

b) výsledek požadujeme v uzlu 2

Strategie:

4. Přenést relaci ZAMĚSTNANEC do uzlu 2 a provést operaci spojení v tomto uzlu. Přenášíme tedy celkem **1 000 000 B**.
5. Přenést relaci ODDĚLENÍ do uzlu 1, provést operaci spojení (JOIN) v uzlu 1 a přenést výsledek k uzlu 2. Přenášíme tedy nejprve 5 800 B a po provedeném spojení a projekci na atributy JmenoZ, PrijmeniZ a NazevO ještě 100 * 55 B
   tj. celkem 5 800 B + 5 500 B = **11 300 B**.

Složitější strategie, která může dávat lepší výsledky než výše uvedené, používá operaci polospojení.

12.6.2. Provádění distribuovaných dotazů s použitím polospojení

Základní ideou je pomocí polospojení redukovat počet záznamů v relaci ještě před vlastním přenosem k jinému uzlu. Tj. budeme-li spojovat relace R a S, pak pošleme spojující sloupec z relace R do uzlu, kde je umístěna relace S a spojíme ho s touto relací. Následně je provedena projekce spojujících atributů spolu s atributy požadovanými ve výsledku a vše je posláno zpět do uzlu s relací R a je provedeno spojení. Čili jedním směrem (od R k S) se

posílá pouze spojující sloupec a zpět se pošle už jen podmnožina S bez zbytečných záznamů. Pokud se spojení účastní pouze malá podmnožina relace S, pak je tato metoda efektivním způsobem, jak snížit počet přenášených dat.

Příklad 12.8

a) výše zmiňovaný dotaz Q1

strategie:

1. Provést projekci spojujících atributů relace ZAMĚSTNANEC v uzlu 2 a přenést ji do uzlu 1. Přenášíme tedy C1 = ¶ $_{CislоO}$ (ODDĚLENÍ), což je 100 * 3 B = **300 B**
2. Spojíme přenesená data s relací ZAMĚSTNANEC v uzlu 1 a přeneseme požadované atributy do uzlu 2. Přenášíme vlastně množinu Z1 = ¶ $_{JmenoZ, PrijmeniZ, OddZ}$ (C1 |x| $_{CislоO=OddZ}$ ZAMĚSTNANEC) o velikosti 10 000 * 33 B = **330 000 B.**
3. Nyní spojíme Z1 spolu s relací ODDĚLENÍ a máme výsledek – přeneslo se celkem **330 300 B dat**

b) výše zmiňovaný dotaz Q2

strategie:

1. Provést projekci spojujících atributů relace ZAMĚSTNANEC v uzlu 2 a přenést ji do uzlu 1. Přenášíme tedy C2 = ¶ $_{VedouciO}$ (ODDĚLENÍ), což je 100 * 5 B = **500 B**
2. Spojíme přenesená data s relací ZAMĚSTNANEC v uzlu 1 a přeneseme požadované atributy do uzlu 2. Přenášíme vlastně množinu Z2 = ¶ $_{VedouciO, JmenoZ, PrijmeniZ}$ (C2 |x| $_{VedouciO=CislоZ}$ ZAMĚSTNANEC) o velikosti 100 * 35 B = **3 500 B.**
3. Nyní spojíme Z2 spolu s relací ODDĚLENÍ a máme výsledek – přeneslo se celkem **4 000 B dat**.

S použitím této strategie jsme zúžili množství přenášených záznamů relace ZAMĚSTNANEC v kroku 2 pouze na ty, které se dají spojit se záznamy relace ODDĚLENÍ v kroku 3. U dotazu Q1 jsme dosáhli mírné zlepšení, v případě Q2 se jedná o výrazné zlepšení.

Připomeňme definici operace **polospojení** (semijoin):

Mějme relace R a S. Operace polospojení **R |x $_{A = B}$ S**, kde A a B jsou doménově kompatibilní atributy relací R a S, je dána jako výsledek následujícího výrazu relační algebry

$_{<Attr(R)>}$ (R |x|$_{A=B}$ S)

Uvědomme si, že pro operaci polospojení neplatí komutativita tj. **R |x S ≠ S |x R**

12.6.3. Dekompozice dotazů a aktualizací

V DSŘBD bez distribuční transparence uživatel musí formulovat dotazy rovnou s odkazem na jednotlivé fragmenty. Chceme-li například zjistit pro projekty řízené oddělením 2 jména zaměstnanců, kteří na nich pracují a počty hodiny, pak musíme určit, zda se odkazujeme na

fragmenty uložené v uzlu 2 nebo v uzlu 1, kde jsou replikována a umístěna příslušná data. Mimoto by v těchto systémech měl uživatel zajistit i konzistenci dat.

Na druhé straně máme-li DSŘBD poskytující plnou distribuční, fragmentační a replikační transparenci, pak se vše vůči uživateli tváří jako centralizovaný systém, navíc tento systém zajišťuje i konzistenci jednotlivých replik a řízení víceuživatelského přístupu. Při provádění dotazů hraje důležitou roli **dekompoziční modul**, který zajišťuje rozložení dotazu na poddotazy, které se pak provádějí v jednotlivých uzlech, a vhodně kombinuje dílčí výsledky a získává tak odpověď na původní dotaz. Kdykoliv DSŘBD zjistí, že položka uvedená v dotazu je replikovaná, musí vybrat specifickou (konkrétní) repliku, se kterou se bude pracovat během provádění dotazu.

K určení repliky, na kterou se budeme při zpracování dotazu odkazovat, se používají fragmentační, replikační a distribuční informace uložené v DSŘBD katalogu – v případě vertikální fragmentace je pro každý fragment k dispozici seznam atributů, kdežto pro horizontální fragmentaci se ukládá pro každý fragment podmínka určující, který záznam smí být uložen v příslušném fragmentu. (Tato podmínka se někdy označuje jako **strážce** – guard). V případě kombinované fragmentace se samozřejmě do katalogu ukládá jak seznam atributů, tak i selekční podmínka.

Příklad 12.9.

Informace o fragmentaci databáze z příkladu 12.6

a) uzel 1

celá databáze
- podmínka: TRUE
- seznam atributů: * (všechny atributy)

b) uzel 2

ZAM20
- podmínka: OddZ = 20
- seznam atributů: CisloZ, JmenoZ, PrijmeniZ, PlatZ, OddZ

ODD20
- podmínka: CisloO = 20
- seznam atributů: *

PROJ20
- podmínka: OddP = 20
- seznam atributů: *

PRAC20
- podmínka: (Zam IN (¶ $_{\text{CisloZ}}$ (ZAM20))) OR (Proj IN (¶ $_{\text{CisloP}}$ (PROJ20)))
- seznam atributů: *

c) uzel 3

ZAM30
- podmínka: OddZ = 30
- seznam atributů: CisloZ, JmenoZ, PrijmeniZ, PlatZ, OddZ

ODD30
- podmínka: CisloO = 30
- seznam atributů: *

PROJ30
podmínka: OddP = 30
seznam atributů: *

PRAC30
podmínka: (Zam IN (¶ $_{CislozZ}$ (ZAM30))) OR (Proj IN (¶ $_{CislozP}$ (PROJ30)))
seznam atributů: *

Pokud DSŘBD provádí dekompozici požadavku na aktualizaci dat, může podle podmínek daných v katalogu určit, které fragmenty se budou aktualizovat. Představme si, že chceme zaevidovat nového zaměstnance tj. snažíme se vložit do relace ZAMĚSTNANEC následující záznam <20, "Kudláček", "Jiří", 12.04.1968, "M", "Tábor",30, 25000>. Tato aktualizace vyžaduje dvě operace – jednak celý záznam musí být vložen v uzlu 1 do fragmentu ZAMĚSTNANEC, jednak podle oddělení 30, kam je nový zaměstnanec zařazen, je třeba vložit do fragmentu ZAM30 v uzlu 3 následující projekci původně vkládaného nového záznamu <20, "Kudláček", "Jiří", 30, 25000>.

Pro dekompozici dotazu musí DSŘBD určit, které fragmenty obsahují požadované záznamy tj. porovnat podmínky uvedené v dotazu s „strážnými" podmínkami.

Příklad 12.10

Uvažme následující dotaz: Zjistěte jména, čísla projektů a počty odpracovaných hodin pro každého zaměstnance, který pracuje na projektu řízeném oddělení 30. Formulujme tento dotaz pomocí SQL

Q : SELECT JmenoZ, PrijmeniZ, CisloP, Hodiny

FROM ZAMĚSTNANEC, PROJEKT, PRACUJE_NA

WHERE OddP = 30 AND CislozZ = Zam AND CisloP= Proj

Předpokládejme, že dotaz je zpracováván v uzlu 3, kde je rovněž požadován výsledek. DSŘBD podle strážných podmínek stanovených pro PROJ30 a PRAC30 zjistí, že veškeré záznamy vyhovující podmínce (OddP = 30 AND CisloP= Proj) jsou k dispozici v uzlu 3, takže může provést následující dekompozici:

T1 ← ¶ $_{Zam}$ (PROJ30 |x| $_{CisloP=Proj}$ PRAC30)

T2 ← ¶ $_{CislozZ, JmenoZ, PrijmeniZ}$ (T1 |x| $_{Zam=CislozZ}$ ZAMĚSTNANEC)

VYSLEDEK ← ¶ $_{JmenoZ, PrijmeniZ, Proj, Hodiny}$ (T2 * PRAC30)

Poddotaz T1 se provádí v uzlu 3, neboť PROJ30 obsahuje právě ty záznamy, které vyhovují podmínce OddP=30, a PRAC30 obsahuje právě všechny záznamy, které se dají propojit s PROJ30. Poddotaz T2 se provede v uzlu 1 a výsledek se pošle do uzlu 3, kde se provede přirozené spojení. Jiná varianta by byla poslat dotaz Q do uzlu 1, kde je replikována celá databáze, dotaz provést a výsledek poslat zpět do uzlu 3. Optimalizátor dotazů zjistí cenu obou strategií a zvolí výhodnější z nich.

12.7. Řízení víceuživatelského přístupu a obnovovací systém v distribuovaných databázích

V distribuovaných databázích oproti centralizovaným databázím vyvstávají následující problémy:

- *manipulace s* **vícenásobnými kopiemi** *datových položek*. Metody pro řízení konkurenčního přístupu jsou zodpovědné za udržení konzistence těchto kopií. Rovněž tak obnovovací systém musí zajistit konzistenci všech kopií, pokud např. místo, na něž se kopie ukládá, selže.
- *selhání jednotlivých míst*. Pokud jedno či více míst selže, DSŘBD by měl pokračovat v provádění operací (je-li to možné). Po obnovení havarovaného místa by se odpovídající lokální databáze měla synchronizovat s ostatními, dříve než dojde k opětovnému připojení satnice do systému.
- *selhání komunikačních spojení (sítě)*. Systém musí být schopen vypořádat se s problémy způsobenými selháním spojení mezi uzly sítě. V extrémním případě dojde k rozdělení sítě na dvě či více částí, kdy místa v každé části spolu mohou komunikovat, ale navzájem o sobě nevědí.
- *distribuované potvrzování*. Tento problém se může objevit při potvrzování transakce přistupující k datům uloženým na více místech, pokud některá z nich selže v průběhu potvrzovacího procesu. Tento problém se často řeší tzv. **dvoufázovým potvrzovacím protokolem** (viz dále).
- *distribuovaný deadlock*. Deadlock se může vyskytnout mezi více uzly, takže je třeba rozšířit techniky zabraňující deadlocku o tuto možnost.

12.7.1. Řízení konkurenčního přístupu založené na primární kopii datové položky

Ve snaze vypořádat se s problémy při řízení konkurenčního přístup v distribuovaných databázích byla navržena řada metod rozšiřujících příslušné techniky v centralizovaných databázích. Proberme nyní některé z nich v kontextu rozšířeného centralizovaného *zamykání.* Podobné rozšíření aplikujeme na další konkurenční techniky.

Základní myšlenka spočívá v tom, že určíme jednu specifickou kopii každé datové položky jakožto **primární kopii**. Zámky nad určitou datovou položkou jsou pak spojeny (prováděny) s touto primární kopií a veškeré požadavky na zamykání a odmykání jsou posílány do uzlu, kde je tato primární kopie uložena.

Na této myšlence je založena řada různých metod lišících se způsobem stanovení primární kopie. Například technika **primárního místa** uchovává všechny jedinečné kopie v jednom (primárním) uzlu. Modifikací právě zmíněné metody je metoda **primárního místa se záložním místem**. Jiná technika je metoda **primární kopie**, které umožňuje ukládat jedinečné kopie na více různých míst s tím, že místo obsahující jedinečnou kopii nějaké datové položky pak figuruje jako **koordinační místo** pro řízení konkurenčního přístupu k této položce.

Technika primárního místa. Jak už bylo řečeno, v tomto případě je určeno jediné **primární místo** k uchování všech primárních kopií a působí tedy jako **koordinační místo** *pro všechny datové položky*. (Tj. veškeré zámky jsou spravovány tímto místem.) Tato metoda je

rozšířením zamykacích postupů v centralizovaných databázových systémech. Pokud veškeré transakce používají dvoufázový zamykací protokol, je zaručena sériovost. Výhodou této metody je právě ono rozšíření původních centralizovaných zamykacích technik, takže není příliš složitá. Na druhé straně vzhledem k tomu, že veškeré požadavky ohledně zámků jsou posílány na jediné místo, může dojít k přetížení místa a zahlcení systému. Mimoto pokud toto primární místo zhavaruje, dojde k ochromení celého systému.

I když jsou veškeré zamykací a odemykací operace prováděny na primární místě, přesto je možné přistupovat ke konkrétní datové položce na libovolném místě, obsahující kopii zmíněné položky. Pokud např. transakce obdrží od primárního místa na určitou položku potvrzený *read-lock*, může použít libovolnou kopii této položky. Pokud ale transakce obdrží požadovaný *write-lock* a provede aktualizaci položky, pak je DSŘBD zodpovědný za aktualizaci všech dalších kopií a musí to provést ještě dříve, než je uvolněn zámek nad touto položkou.

Primární místo se záložním místem. Tato technika používá kromě primárního místa ještě jedno místo jakožto **záložní místo**. Veškeré zamykací a odemykací operace jsou prováděny jak v primárním místě, tak i v záložním místě. V případě, že selže primární místo, zbývá ještě záložní místo, které převezme úlohu primárního místa a je určeno nové záložní místo. Tento postup sice řeší problém s havárií primárního místa, ale na druhé straně zpomaluje zamykací procesy, protože veškeré požadavky na zamykání a jejich potvrzení se odehrávají na dvou místech (primární a záložní).

Technika primárních kopií. Tato metoda se pokouší distribuovat požadavky na zámky na ta místa, která obsahují primární kopie požadovaných datových položek – připouští se, že primární kopie jsou obecně uloženy na různých místech. Selhání jednoho místa má vliv pouze na ty transakce, které požadují zámky na položkách, jejichž primární kopie jsou umístěny v tomto místě, přičemž ostatních transakcí se to netýká. Tato metod může také využívat záložní místo ke zvýšení dostupnosti a spolehlivosti.

Výběr nového koordinačního místa v případě selhání. Kdykoliv v některé z předchozích technik selže koordinační místo, ostatní fungující místa musejí vybrat nového koordinátora. V případě *techniky primárního místa bez záložního místa* musejí být všechny prováděné transakce přerušeny a restartovány, přičemž obnovovací proces je poněkud úmorný. Část obnovovacího procesu zahrnuje i výběr nového primárního místa a spuštění zamykacího procesu (manažer zámků) a ukládání veškerých informací o zamykání v tomto místě. U *metod se záložním místem* jsou transakce pozastaveny, dokud záložní místo nezaujme roli primárního místa a není určeno nové záložní místo, kam je třeba nakopírovat všechny zamykací informace z nového primárního místa.

Záložní místo je vybráno z funkčních míst primárním místem (původním nebo nově určeným). Pokud ovšem neexistuje záložní místo nebo selhalo jak primární, tak i záložní místo, dochází k procesu označenému jako **volba**. Každé místo M, které po opakovaných a bezvýsledných pokusech o spojení s koordinačním místem usoudí, že koordinátor selhal, může zahájit volební proces tím, že pošle ostatním místům návrh, že M bude novým koordinátorem. Jakmile místo M obdrží od ostatních míst většinovou podporu (většina míst souhlasí), prohlásí se za nového koordinátora. Samotný volební algoritmus je poněkud složitý, neboť je mimo jiné brát v úvahu i to, že se současně více míst ve stejném čase snaží být koordinátorem.

12.7.2. Řízení konkurenčního přístupu založené na hlasování

V případě této metody neexistuje cosi jako primární kopie každé datové položky, nýbrž se posílá zamykací požadavek do každého místa, které obsahuje kopii zmíněné položky. Každá kopie si spravuje své vlastní zamykací procesy, takže může přijmout nebo odmítnout požadavek na uzamčení. Pokud transakce požadující uzamčení určité položky obdrží od většiny kopií potvrzení zámku, pak je zámek nad položkou potvrzen a všechny kopie obdrží informaci o uzamčení. Je-li naopak požadavek většinou kopií odmítnut během určitého časového intervalu, pak transakce svůj požadavek zruší a informuje o zrušení příslušná místa. Tato metoda je považována za skutečnou distribuovanou metodu řízení konkurenčního přístupu, neboť odpovědnost za rozhodnutí o přidělování (odmítnutí) zámků leží na všech místech. Simulací těchto postupů bylo prokázáno, že právě popsaná metoda znamená ovšem vyšší objem zpráv posílaných mezi jednotlivými místy než ostatní metody. Pokud bychom brali v úvahu i fakt, že by během hlasovacího procesu více míst selhalo, stal by se tento postup neobyčejně složitým.

12.7.3. Distribuované obnovování

Distribuované obnovování je nezbytnou součástí distribuovaných databázových systémů. V mnoha případech je obtížné zjistit, zda určité místo selhalo, aniž si ostatní místa vymění mezi sebou řadu zpráv. Předpokládejme, že místo X pošle místu Y zprávu a očekává odpověď, ale neobdrží nic. Vysvětlení může být následující:

- zpráva k místu Y vůbec nedorazila, protože je problém s komunikačním vedením
- místo Y je mimo provoz a nemůže odpovědět
- místo Y obdrželo zprávu od X, poslalo odpověď, ale ta nedorazila

Pokud nemáme k dispozici další informace, je obtížné zjistit, co se vlastně stalo. (Je třeba např. vědět, zda ostatní místa se mohou úspěšně spojit s místem Y či nikoliv, v extrémním případě může být místo Y mimo provoz a ještě nastat i problém s komunikačním vedením.)

Jiným problémem distribuovaného obnovování je distribuované potvrzování. Pokud transakce aktualizuje data na několika různých místech, nemůže být potvrzena (odsouhlasena), dokud není jisté, že na žádném z míst nemůže dojít ke ztrátě aktualizací. To znamená, že každá místo musí nejprve trvale zapsat transakční aktualizace do lokálního protokolu (logu) na disk. K zajištění korektnosti distribuovaného potvrzování se používá dvoufázový potvrzovací protokol.

Literatura:

[1] ELMASRI, R., NAVATHE, S., B. *Fundamentals of Database Systems*, 5th edition. Addison-Wesley, 2007. ISBN 978-03-213-6957-4.

[2] SILBERSCHATZ, A., KORTH H. F., SUDARSHAN S. *Database System Concepts, 5th edition*, New York: McGraw-Hill, 2006. ISBN 978-0-07-295886-7

[3] POKORNÝ, J. *Databázová abeceda*. Veletiny: Science, 1998, ISBN 80-86083-02-2.

[4] POKORNÝ, J., HALAŠKA, I. *Databázové systémy*, 2. vydání. Praha Vydavatelství ČVUT, 2003, ISBN 80-01-02789-9.