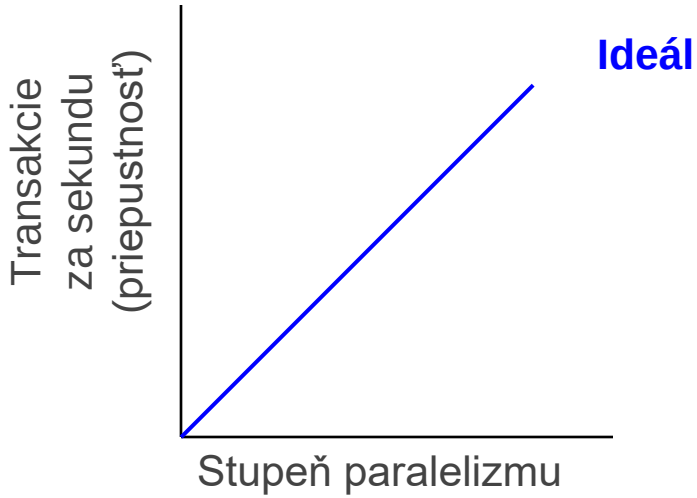
Paralelné DBMS

Paralelné vs. distribuované DBMS

- Paralelné DBMS
  - Jedna lokalita (Čas prístupu k dátam nezáleží od umiestnenia)
  - Jeden alebo viac počítačov
- Distribuované DBMS
  - Viacero lokalít kdekoľvek po svete
  - Hlavné obmedzenie: latencia

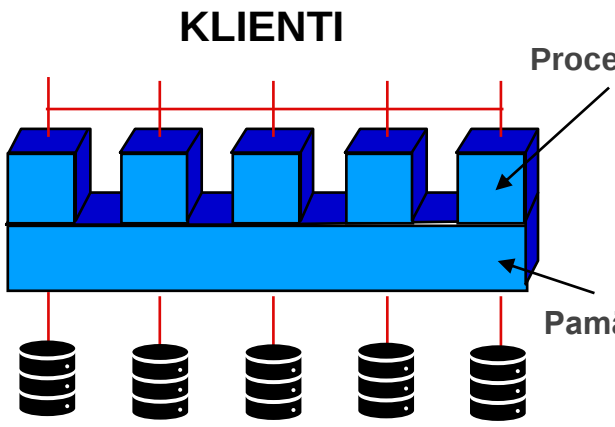
Výhody paralelného DBMS

- Zlepšenie výkonu
- O klientov sa postará viac počítačov – rozloží sa záťaž
- Ak niektorá časť systému spadne, ich úlohy môže prebrať iná časť systému
- Zlepšenie škálovateľnosti



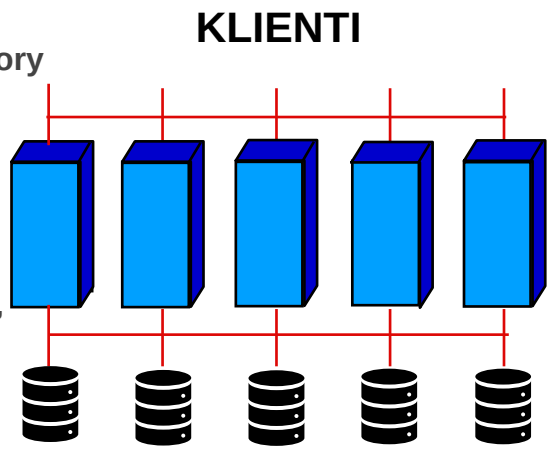
Architektúry paralelných DBMS

Zdieľaná pamäť (SMP)

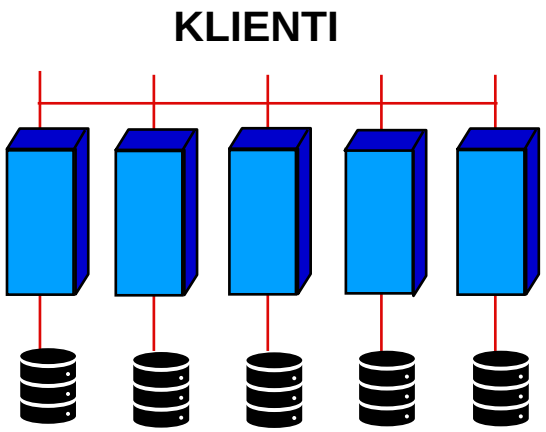


Ľahké programovať
Drahé postaviť
Náročné zvýšiť škálovateľnosť

Zdieľaný disk



Bez zdieľania



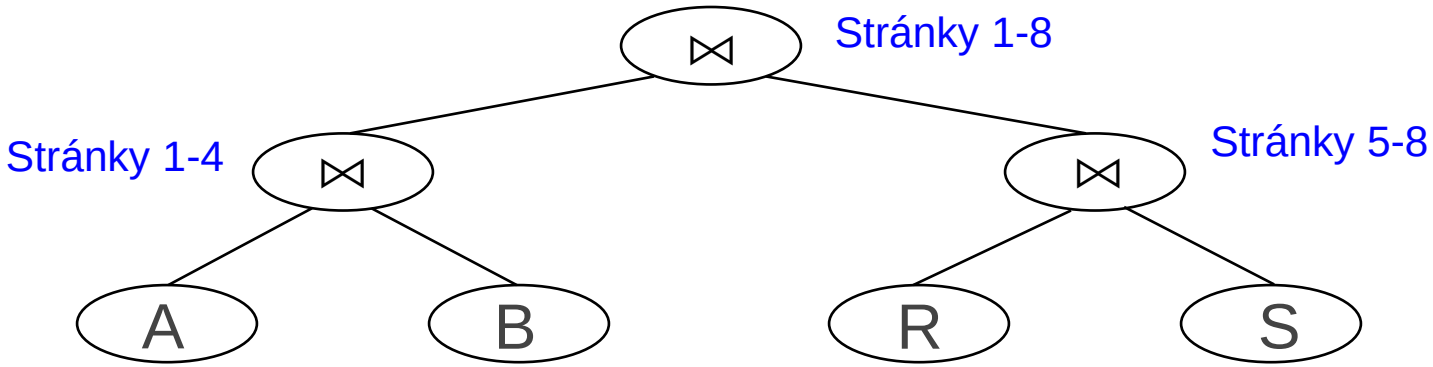
Ťažké programovať
Lacné postaviť
Ľahko škálovateľné

Rozličné typy paralelizmu databáz

- Vnútro-operátorový paralelizmus
  - Viacero uzlov pracuje na výpočte danej operácie (scan, sort, join)
- Medzi-operátorový paralelizmus
  - Každý operátor plánu dopytu môže bežať na inom uzle
- Medzi-dopytový paralelizmus
  - Rozdielne dopyty bežia na iných uzloch

Komplexné plány paralelných dopytov

- Komplexné dopyty: Medzi-operátorový paralelizmus
  - Left-deep plány je ťažké paralelizovať
    - Nevieme spustiť výpočet, keď čakáme na dáta z ľavého podstromu
  - Jediná pomoc: široké stromy



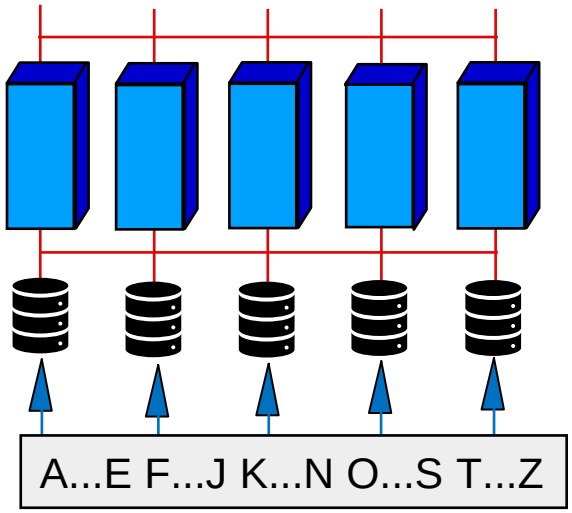
Komplexné plány paralelných dopytov

- Je pomerne jednoduché vybudovať spúšťač paralelných dopytov
  - S.M.O.P. (Simple Mather or Programming)
- Je ťažké napísať robustný optimalizátor pre paralelné dopyty
  - Je tu mnoho problémov
    - Rôzne vyťaženie uzlov môže ovplyvniť úspešnosť plánu
  - Výskum stále prebieha!
    - Komerčné databázy to zatiaľ nerobia

Automatické rozdeľovanie dát

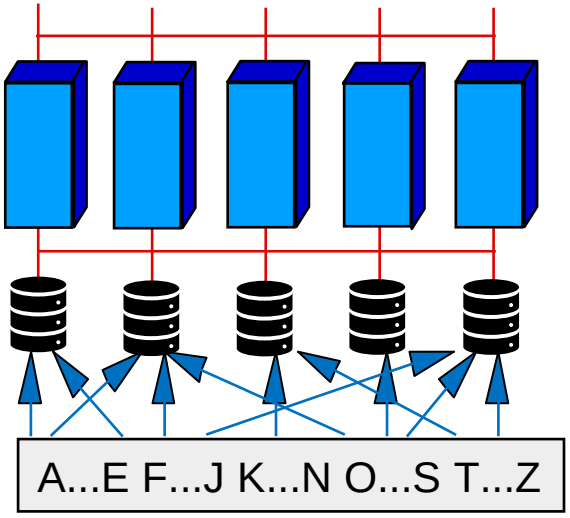
Rozdeľovanie tabuľky:

Rozsah



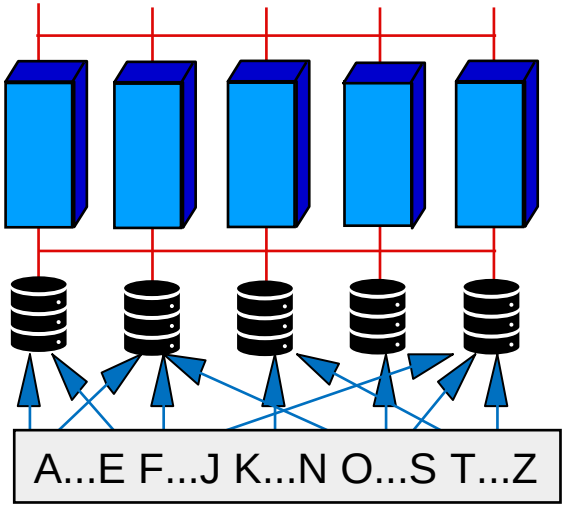
Dobré pre equijoiny, rozsahové dopyty group-by

Hash



Dobré pre equijoiny

Round robin



Dobré na šírenie záťaže (rovnako veľké dáta)

Zdieľanie disku a pamäte je menej citlivé na delenie,
Bez zdieľania ťaží z „dobrého" rozdelenia

Paralelizovanie operácií

- Skenovanie, selekcia, Bulk loading, agregačné operácie sú jednoducho paralelizovateľné.
  - Načítanie môže prebiehať paralelne a potom sa spoja získané riadky
  - Indexy môžu byť vybudované nad každou časťou dát
- Ak sa použije hashovanie alebo partície, stačí pracovať iba s tými uzlami, ktoré obsahujú dané riadky.
- Sortovanie
  - Môže každý utriediť tú časť, ktorú vlastní a výsledok sa spojí (niekto môže mať veľa, niekto nič)
  - Dáta sa spoja, rovnomerne rozdelia, utriedia sa a tak sa spoja
- Join - najlepšie paralelizovateľné sú hash join a sort-merge join

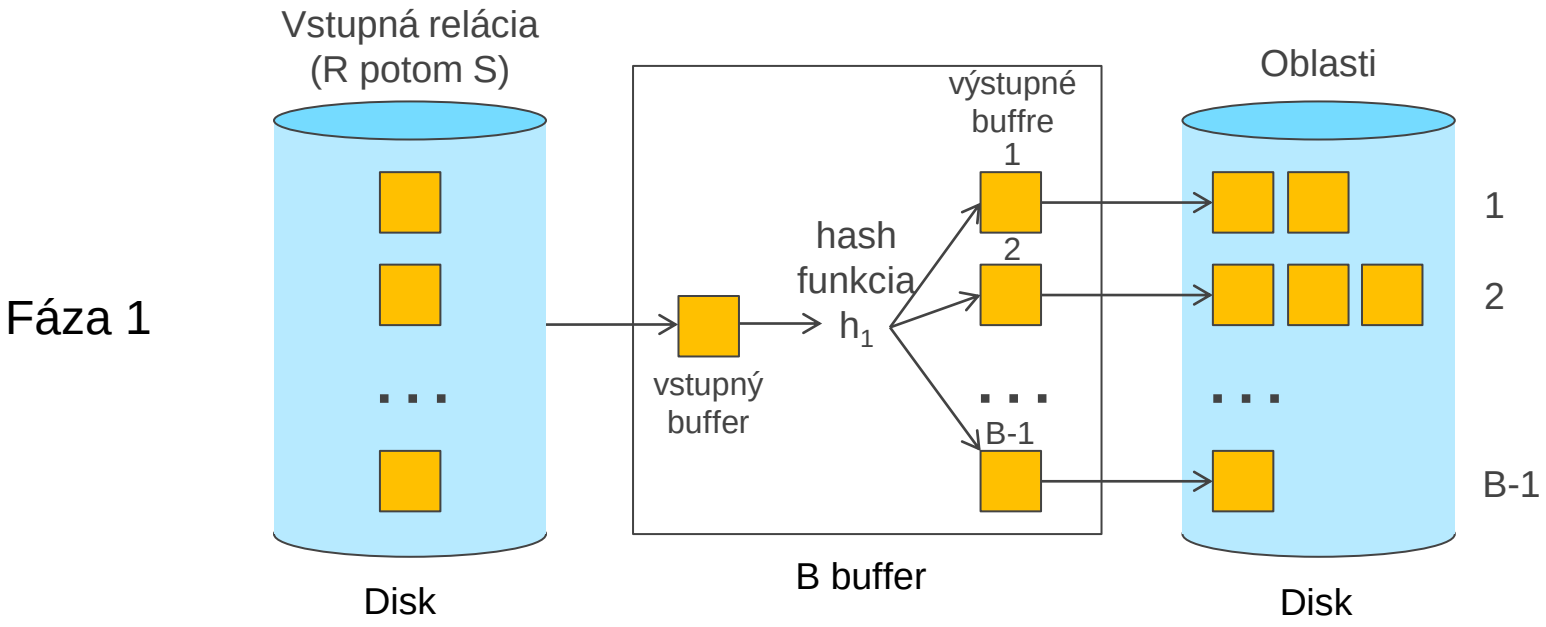
Paralelné triedenie

- Myšlienka:
  - 1. krok: Paralelné skenovanie, a rozsahové rozdelenie
  - 2. krok: Ako n-tice prichádzajú, začneme “lokálne” triedenie na každej časti
  - Výsledné dáta sú usporiadané, a rozsahovo rozdelené
  - Problém: šikmé dáta!
    - Riešenie: vezmeme vzorku dát a stanovíme deliace body z nej

Paralelný join

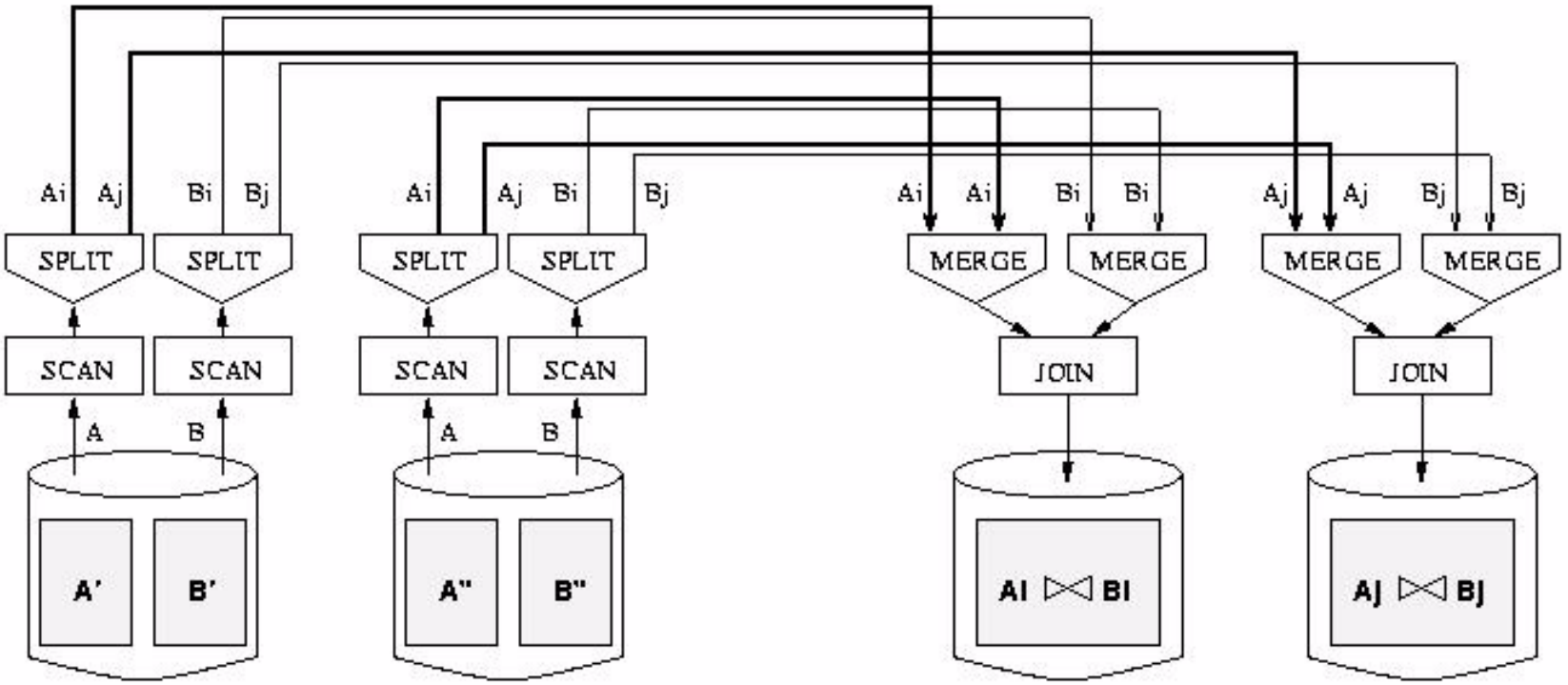
- Nested loops:
  - Každá vonkajšia n-tica musí byť porovnávaná s každou vnútornou n-ticou, s ktorou by mohla byť spojená.
  - Celá vnútorná tabuľka musí byť nakopírovaná k všetkým častiam vonkajšej tabuľky
  - Ak máme málo miesta na disku, robíme veľa opakovaného kopírovania
- Sort merge join:
  - Utriedime obe tabuľky paralelným triedením, ak už tak utriedené neboli pri delení dát
    - musíme mať rovnaké hraničné hodnoty delenia
  - Spájanie sa už deje nad lokálnymi dátami

Paralelný hash join

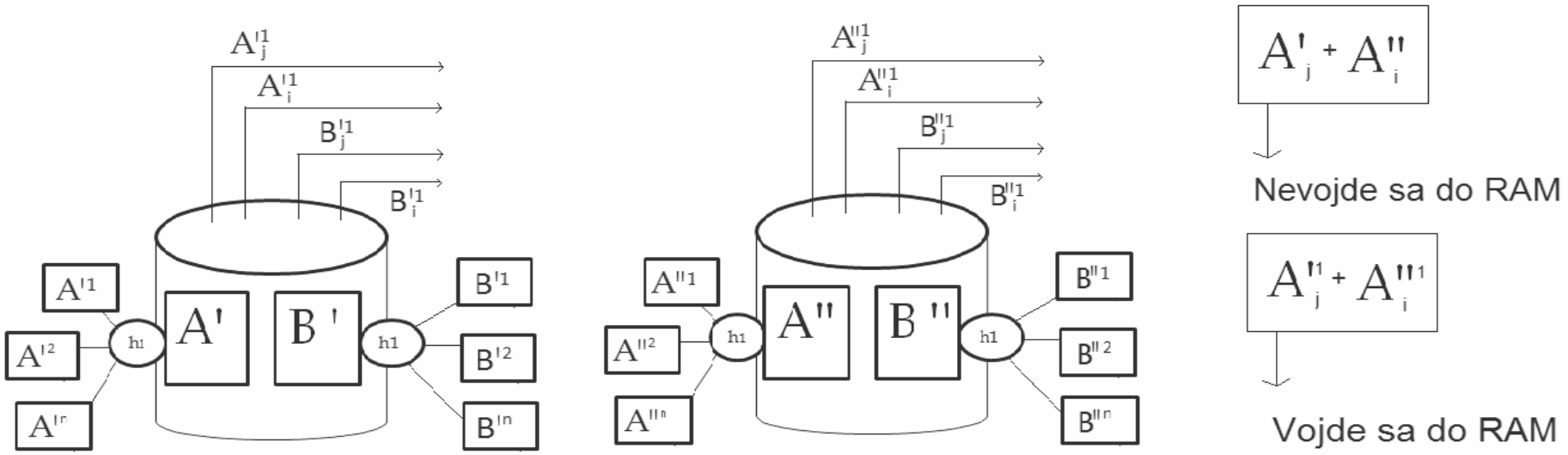


- Celkový join sa rozdelí na menšie joiny
- V prvej fáze, i-ta oblasť sa posiela na i-ty uzol:
  - Rovnaká hash funkcia na všetkých uzloch
  - Rovnaké oblasti sa zhromaždia na jednom mieste
- Druhá fáza je už join prijatých dát
  - Spojenie je možné iba s dátami v rovnakej oblasti delenia

Dátový tok siete pri paralelnom hash join-e



Vylepšený paralelný hash join



- Rozdelíme na toľko oblastí, aby oblasť ľavej relácie vošla do RAM prijímajúceho uzla
- Prichádzajúce dáta z pravej relácie už rovno spájame a posielame do výsledku bez ukladania na disk

Distribuované Databázy

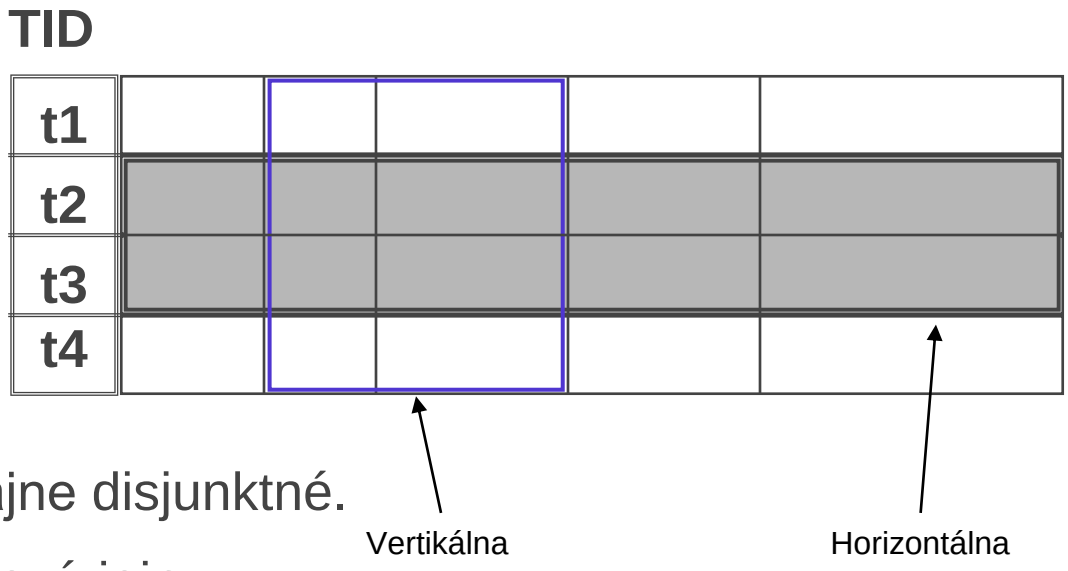
Typy distribuovaných databáz

- Homogénne: Vo všetkých lokalitách beží rovnaký DBMS.
- Heterogénne: V rôznych lokalitách bežia rôzne DBMS (rôzne RDBMS alebo dokonca ne-relačné DBMS).
- Základné požiadavky:
  - Nezávislosť od distribúcie dát
    - Používatelia by nemali vedieť kde sa dáta nachádzajú
    - Nechceme špecifikovať, čo sa má kde počítať
  - Atomickosť transakcií
    - Ak je zmena úspešná, ostane zachovaná
    - Ak sa zmena nepodarila, na žiadnom serveri nesmú ostať medzivýsledky

Architektúra distribuovaných DBMS

- Klient – server
  - Servery navzájom nespolupracujú
  - Klienti musia vedieť, ktoré dáta sú kde
  - Prepájanie dát z viacerých serverov iba v klientoch
- Middleware systémy
  - Manažér iba rozvrhuje prácu serverom, nedrží dáta
  - Výhodné, ak máme heterogénne databázy
- Spolupracujúce servery
  - Klient vysiela požiadavku na ľubovoľný server
  - Ak sú potrebné aj vzdialené dáta, server si ich sám popýta od toho servera, ktorý ich má

Uloženie dát



- Fragmentácia
  - Horizontálna: Zvyčajne disjunktné.
  - Vertikálna: Bezstratový-join
  - Najbežnejšia stratégia delenia: podľa miesta vzniku
- Replikácia
  - Zvyšuje dostupnosť
  - Rýchlejšie vyhodnotenie dotazu
  - Synchrónna vs. Asynchrónna.
    - Rozdiel je v tom, či sa môžu líšiť v lokálnych kópiách

Spôsoby využitia redundancie

- Replikované tabuľky
  - Pri výpočte nám netreba presun cez sieť, lebo máme všetko u seba na disku.
  - Použiteľné na malé, málo sa meniace tabuľky
- Replikované materializované pohľady
  - Máme iba tie dáta, ktoré naozaj potrebujeme
    - šetríme disk
    - dá sa použiť aj na väčšie tabuľky
    - môžeme spájať stĺpce rozdelené vertikálnym delením
- Replikované indexy
- Globálne join indexy
  - Indexuje, kde sú ktoré dáta uložené, ale nedrží ich
  - Keď chcem robiť join podľa daného stĺpca, viem presne povedať, aké dáta mi má vzdialený server poslať

Manažment distribuovaného katalógu

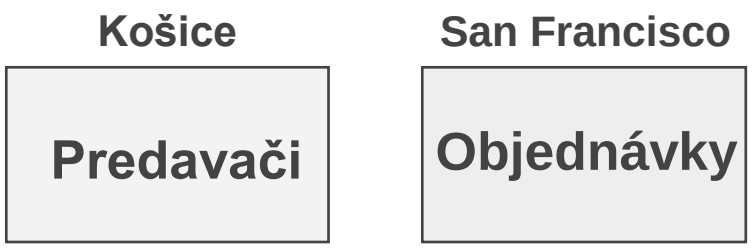
- Musíme sledovať, ako sú dáta distribuované až na úroveň stránok.
- Musíme vedieť pomenovať každú repliku a každý fragment. K zabezpečeniu lokálnej autonómie:
  - <**local-name, birth-site**>
- Katalóg stránok: Popisuje všetky objekty(fragmenty, repliky) lokality + zaznamenáva kde sa nachádzajú repliky dát vytvorených na tomto mieste.
  - Ak chceme nájsť nejakú reláciu, pozrieme sa na miesto vzniku v katalógu.
  - Miesto vzniku sa nikdy nezmení, aj keď je presunutý vzťah.

Distribuované dopyty

```
SELECT AVG(P.vek)
FROM Predavači P
WHERE P.rating > 3
    AND P.rating < 7
```

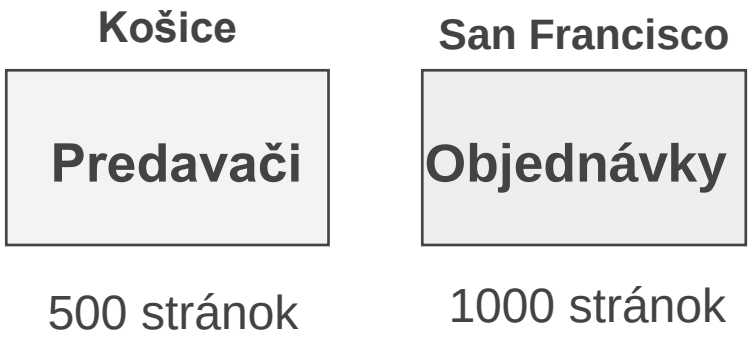
- Horizontálne fragmentované: riadky s rating < 5 v Shanghai, >= 5 v Košiciach.
  - Musíme počítať SUM (vek), COUNT (vek) na oboch miestach.
  - Ak WHERE obsahuje len P.rating>6, stačí na jednom mieste
- Vertikálne fragmentované: *rating* v Shanghai, *meno* a *vek* v Košicach, *id* v oboch.
  - Musíme rekonštruovať vzťah tým, že sa pripojíme cez *id*, a potom vyhodnotíme dopyt.
- Replikácia: Predavači na oboch stranách.
  - Vyberieme miesto na základe toho, kde tie dáta potrebujeme, alebo na uzle, ktorý je najmenej vyťažený

Distribuovaný join



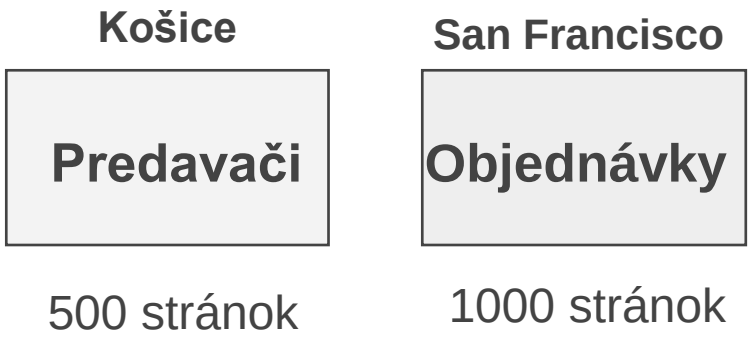
- Nested Loops je nereálny:
  - **Cena:** 500 D + 500 * 1000 (D+S)
  - **D** je cena čítania/zápisu stránky; **S** je cena dopravy stránky.
  - S je oveľa väčšie ako D
- Ak dopyt nebol zadaný v Košiciach, treba pridať náklady na dopravu výsledku dopytu
- Index nested loops môže byť OK, ak ľavá relácia je veľmi malá (požiadavky na riadky, však treba pre zvýšenie efektivity spájať do balíčka požiadaviek)
- Poslanie na jedno miesto: Pošleme Objednávky do Košíc.
  - Cena: 1000 S + 4500 D (cena lokálneho joinu obvykle 3*(500+1000))
  - Ak je výsledok veľmi veľký, môže byť lepšie poslať obe relácie na miesto použitia a spojiť ich na cieľovom mieste

Semi join



- Vyrobíme projekciu na joinovaný stĺpec v tabuľke Predavači v Košiciach a pošleme ju do San Francisca
    - so selekciou na riadky, ak je nejaká vo WHERE podmienke týkajúca sa iba predavačov – ale to je operácia v pláne dopytov pred tým
- V San Franciscu vyrobíme selekciu tých riadkov, ktoré budú vo výsledku + projekcia na stĺpce výsledku a to sa pošle do Košíc
- V Košiciach urobíme lokálny join tabuľky Predavači s tým čo prišlo

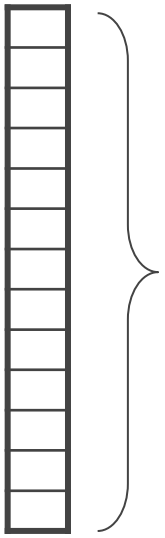
Semi join



- Cena výpočtu
  - Nech mám aj podmienku p.rating>8, ktorá vyselektuje iba 20% riadkov: 100 stránok
  - Nech joinovaný stĺpec je 20 % veľkosti tabuľky : posielam 20 S
  - Ak to nie je unique stĺpec, pred odoslaním odstránime duplicity
    - Ak by boli väčšie dáta, triedenie môže vyžiadať ďalšie diskové prístupy
  - V San Franciscu joinujem prijaté dáta s Objednávkami 1000 D
  - Povedzme, že vo výsledku bude 20% riadkov Objednávok (200 S) a v Košiciach spravím join 3*(100+200) D
  - Celkovo 500 D + 20 S + 1000 D + 200 S + 900 D = 2400D + 220S

Bloom join

- Zvolíme si počet oblastí (násobok toho, koľko vojde do paketu) ktoré sú cieľom hashovacej funkcie

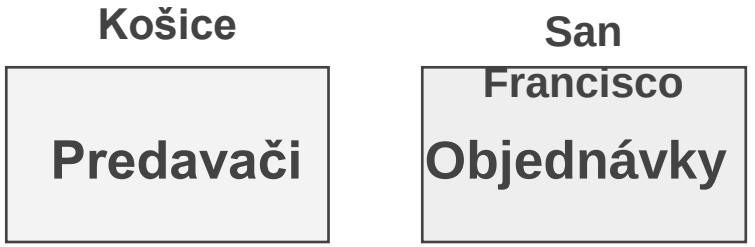


Bitové pole

n = x.1460 B
= x.11680 b

- V Košiciach počítame hash hodnotu z každej hodnoty joinovaného stĺpca do intervalu [0, n-1] a napíšeme „1“ do tých políčok ktoré nám vyjdu a pošleme
- V San Franciscu prechádzame všetky hodnoty joinovaného stĺpca a počítame hash. Ak pre danú hodnotu nájdeme v poli „1“ tak daný riadok budeme posielať
  - Môžeme zaslať aj riadky, ktoré sa nebudú dať spojiť!
- V Košiciach urobíme join

Bloom join



500 stránok

1000 stránok

- Cena výpočtu
  - Nech mám aj podmienku p.rating>8, ktorá vyselektuje iba 20% riadkov: 100 stránok
  - Pošlem bitové pole veľkosti menšej ako 1 stránka (2-3 pakety): 1 S
  - V San Franciscu prechádzam Objednávky a selektujem riadky podľa hashu 1000 D
  - Povedzme, že vo výsledku bude 21% riadkov Objednávok (210 S) a v Košiciach spravím join 3*(100+210) D
  - Celkovo 500 D + 1 S + 1000 D + 210 S + 930 D = 2430D + 211S

Zmena distribuovaných dát

- Synchrónna replikácia: Všetky kópie modifikovanej relácie musia byť zmenené pred tým, ako je transakcia komitnutá
  - Používatelia nevnímajú, že dáta sú distribuované
- Asynchrónna replikácia: Kópie modifikovanej relácie sú periodicky obnovované. Niektoré kópie môžu mať dočasne neaktuálne dáta.
  - Používatelia si uvedomujú, že dáta sú distribuované
  - Väčšina prístupov

Synchrónna replikácia

- Voľby: Transakcia musí prepísať väčšinu kópií, aby objekt bol považovaný za zmenený; musí prečítať dostatok kópií, aby si bola istá, že má aktuálne dáta.
  - Napr. 10 kópií; 7 sa prepisuje pri zmene; 4 kópie sa čítajú.
  - Každá kópia má číslo verzie
  - Zaujímavé iba keď je málo čítaní.
- Read-any Write-all: Zápisy sú pomalé, čítania rýchle.
  - Najčastejší prístup
- Výber techniky určuje, čo je potrebné zamykať.

Cena synchrónnej replikácie

- Pred zahájením zmeny musí transakcia získať zámky na všetkých modifikovaných kópiách.
  - Zasiela požiadavky na zámky a pokiaľ čaká na zamknutie, drží všetky, ktoré už získala!
  - Ak je niektorý server dočasne vypnutý, je potrebné naňho čakať, pokiaľ sa nereštartuje.
  - Aj bez chýb komitovanie vyžaduje veľa správ zložitého komitovacieho protokolu.
- Hlavný dôvod pre používanie *asynchrónnej replikácie*

Asynchrónna replikácia

- Umožní transakcii komitnúť ešte pred zmenou dát vo všetkých kópiách
  - Klienti si musia byť vedomí toho, že akú kópiu čítajú, a že ich kópie môžu byť už staré.
- Dva hlavné prístupy: Primárny uzol a Peer-to-Peer replikácia.
  - Rozdiel je v tom, koľko kópií je primárnych alebo zmeniteľných

Peer-to-peer replikácia

- Primárnych kópií môže byť viacero
- Zmena primárnej kópie sa musí nejako dostať k ostatným kópiám
- Ak sa dve primárne kópie zmenia tak, že vznikne konflikt, je potrebné to riešiť (napr., v Košiciach nastavíme cenu výrobku 20 Eur a v Paríži v tom istom čase tomu istému výrobku cenu 25 Eur.)
- Najlepšie je takýmto konfliktom predchádzať
    - Každá primárna lokalita obsahuje rôznu časť dát.
    - Primárnosť kópie sa v čase mení

Primárny uzol

- Práve jedna kópia je prehlásená za primárnu kópiu.
- Ostatné kópie nemôžu byť menené priamo.
  - Sekundárne kópie
- Ako sa zmeny v primárnej kópii dostanú k sekundárnym?
  - Deje sa to v dvoch krokoch:
    - Zachytenie zmeny komitnutou transakciou
    - Aplikovanie tejto zmeny

Implementácia zachytenia zmeny

- Procedurálne odchytávanie: Trigger, ktorý typicky urobí fotku dát po každej komitnutej zmene
- Založená na logoch: Log sa použije okrem prípadného zotavenia aj na generovanie tabuľky zmenených dát
  - Ak sa tabuľka mení vždy keď sa chvost logu zapíše na disk, je potrebné aj odstraňovanie zmien abortnutých transakcií
  - Lacnejšie a rýchlejšie riešenie

Implementácia aplikácie zmeny

- Aplikujúci proces nad sekundárnymi dátami cyklicky získava tabuľku zmenených dát a mení lokálne dáta.
  - Perióda opakovania sa dá meniť
- Sekundárna kópia môže byť pohľad nad zmenenou tabuľkou
  - Materializovaný pohľad sleduje zmeny všetkých tabuliek, z ktorých je vytvorený

Data Warehousing replikácia

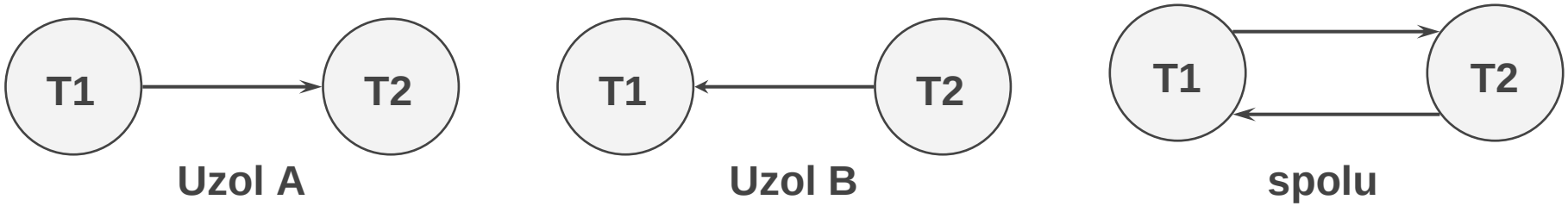
- Cieľ: Vytvorenie obrovských warehousov dát z rôznych lokalít
  - Umožňuje komplexné dopyty na rozhodovanie v rámci organizácie
- Warehousy je istá forma asynchrónnej replikácie.
  - Zdroje sú časti heterogénne a je tiež potrebná konverzia dát (napr. $ na €)
- Procedurálne odchytávanie je v tomto prípade lepšie, lebo cyklus odchytávania je dlhý

Distribuované zamykanie

- Centralizované: Jeden uzol zabezpečuje zamykanie všetkých.
  - Nebezpečné: single point of failure.
  - Najjednoduchšie na prácu s deadlockmi
- Primárna kópia: Zamykanie objektu sa deje tam, kde je jeho primárna kópia
  - Čítanie môže požadovať zamknutie v primárnej kópii aj v sekundárnej kópii, ktorú číta.
- Plne distribuované: Zamknutie každej kópie pri zápise, ale iba lokálnej pri čítaní.

Distribuované odhaľovanie deadlockov

- Každý uzol vytvára graf čakaní transakcií
- Globálny deadlock možný, aj keď lokálne grafy nemajú cyklus



- Tri riešenia
  - Centralizované – poslať všetky lokálne grafy na jedno miesto
  - Hierarchické – poslať lokálny graf rodičovi v hierarchií lokalít
  - Timeout - Abort transakcie, ktorá čaká pridlho

Distribuované zotavenie po páde

- Nové problémy:
  - Nové druhy pádov: spojenia alebo vzdialené uzly
  - Ak časti transakcií sa dejú v rôznych lokalitách, musia buď prejsť všetky alebo ani jedna - potrebujeme komitovací protokol
- Log lokálnych zmien sa udržiava na každom uzle, ale pribudnú nové typy záznamov

Dvojfázový commit (2PC)

- Uzol, na ktorom transakcia vznikne je **koordinátor** ostatné uzly zapojené do realizácie transakcie sú **podriadené**.
- Keď Transakcia chce komitnúť:
  - Koordinátor pošle všetkým podriadeným správu **prepare**
  - Podriadení zapíšu záznam **abort** alebo **prepare** do logu log a pošlú koordinátorovi správu **no** alebo **yes**
  - Ak koordinátor dostane od všetkých **yes**, zapíše **commit** do logu a pošle správu **commit** všetkým podriadeným. Inak zapíše do logu **abort** a pošle podriadeným správu **abort**
  - Podriadení zapíšu do logu záznam **abort/commit** podľa toho, čo im došlo a zašlú správu **ack** koordinátorovi
  - Koordinátor zapíše po prijatí všetkých **ack** správ záznam **end**

Dvojfázový commit (2PC)

- Dve fázy voľba a ukončenie, obe iniciované koordinátorom
- Ľubovoľný uzol sa môže rozhodnúť transakciu abortnúť
- Každá správa zodpovedá rozhodnutiu uzla, a pred zaslaním je zapísaná do logu, aby sme sa vedeli zotaviť
- Všetky záznamy logov transakcie obsahujú číslo transakcie a identifikátor koordinátora.
- Koordinátorov logovacie záznamy abort a commit obsahujú aj identifikátory všetkých podriadených

Reštart po páde uzla

- Ak máme záznam **commit** alebo **abort** transakcie T ale nemáme jej end, musíme urobiť redo/undo T.
  - Ak je to koordinátor, opakovane posiela správy **commit/abort** pokiaľ nedostane všetky **ack** správy
- Ak v podriadenom uzle máme záznam **prepare**, ale už nemáme **commit/abort**, opakovane sa pýta koordinátora, aby zistil stav transakcie a zapísal záznam **commit/abort**, potom zrealizoval redo/undo a zaslal **ack**
- Ak nemáme ani záznam **prepare**, sami začneme abort transakcie a vykonáme undo .
  - Ak je to koordinátor, musí predpokladať, že podriadení ešte môžu posielať správy **no / yes**, ktorý treba poslať **abort**

Blokovanie

- Ak koordinátor padne, podriadení, ktorí hlasovali yes, nevedia, či majú abortnúť a musia počkať na reštart koordinátora.
  - Koordinátor mohol stihnúť zapísať commit, ale už ho nestihol rozoslať
  - Transakcia je blokovaná
  - Aj keby podriadení vedeli ako hlasovali ostatní podriadení, sú blokovaní pokiaľ nik z nich nehlasoval no.

Po páde spojenia

- Ak vzdialený uzol neodpovedá počas commit protokolu, buď padlo spojenie, alebo uzol
  - Ak som koordinátor môžem abortnúť transakciu.
  - Ak som podriadený a ešte som nehlasoval yes, môžem abortnúť transakciu.
  - Ak som podriadený a hlasoval som yes, som blokovaný, pokiaľ koordinátor neodpovie.

Pozorovanie 2PC

- Správy ack slúžia na to, aby koordinátor vedel, kedy môže transakciu odstrániť z tabuľky transakcií
- Ak koordinátor padne po zaslaní správ prepare, ale pred zapísaním **commit/abort**, po zotavení abortuje transakciu.
- Ak podriadení nerobili žiadne zmeny dát, je jedno či transakciu komitli alebo abortli.

2PC s očakávaným abortom

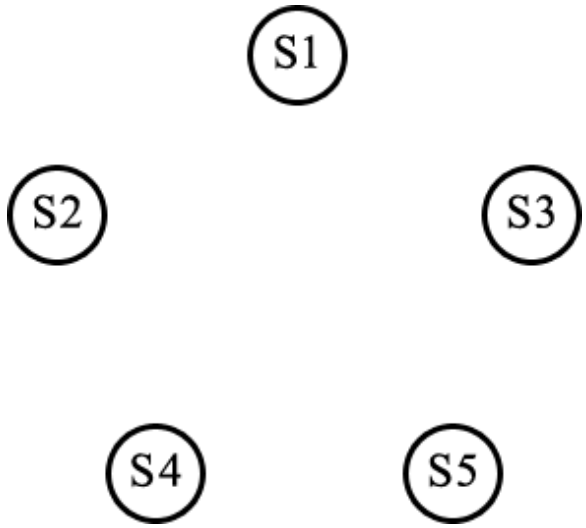
- Keď koordinátor abortne transakciu, urobí undo a ihneď odstráni transakciu z tabuľky transakcií
    - Nečaká na správy **ack**, “predpokladá abort” ak transakcia nie je v tabuľke transakcií. Identifikátory podriadených sa nelogujú v zázname **abort**
- Podriadení neposielajú ack ak nastal abort.
- Ak podriadený nerobí žiadne zmeny dát posiela po prijatí prepare namiesto yes/no správu reader
- Koordinátor reader-ov ignoruje.
- Ak všetci podriadení sú reader-i 2. fáza sa nerobí

Rozloženie dát s optimálnym rozdelením metóda “All-beneficial Sites”

- Umiestnime tabuľky a ich kópie na všetky uzly, kde ich umiestnenie je viac výhodné ako nevýhodné (Benefit prevyšuje náklady)

- Benefit $_{U,tab}$ = (čas na dopyt vzdialeného uzla – čas na lokálny dopyt) * Frekvencia dopytov na tabuľku tab uzla U.

- Náklady $_{U,tab}$ = (čas na lokálny update + čas na vzdialený update) * Frekvencia updateov.



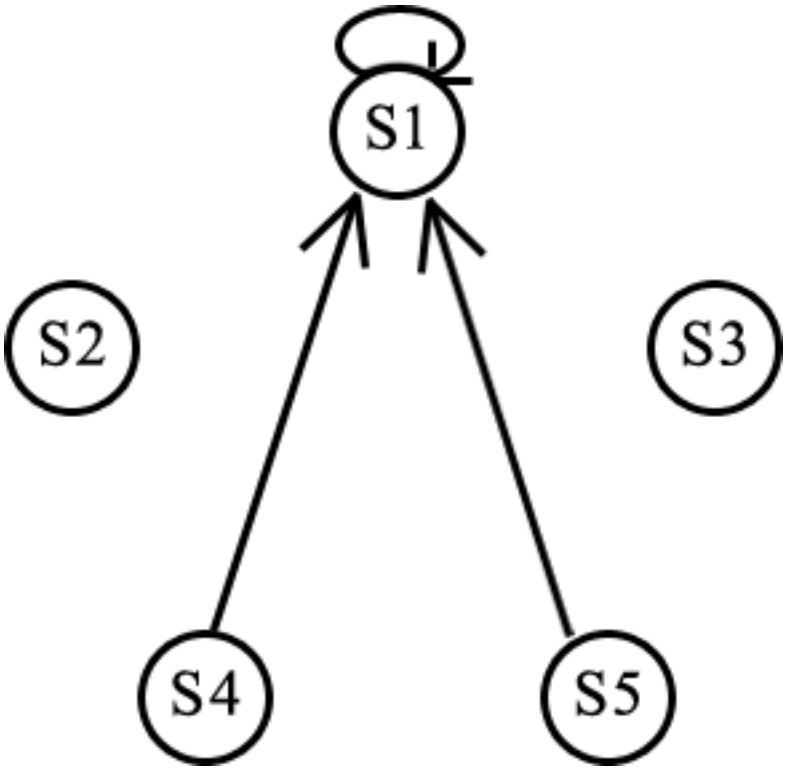
Tabuľka	Veľkosť	priem. čas dopytu	priem. Čas vzdial. Dopytu
Tab1	0,3 GB	100 (150)ms	500 (600)ms
Tab2	0,5 GB	150 (200)ms	650 (700)ms
Tab3	1 GB	200 (250)ms	1000 (1100)ms

Transakcia	Uzol dopytu	Frekvencia	Akcie
T1	S1,S4,S5	1	3*Read z Tab1, 1*Write do Tab1, 2* Read z Tab2
T2	S2,S4	2	2*Read z Tab1, 3*Read z Tab3, 1*Write do Tab3
T3	S3,S5	3	3*Read z Tab2 , 1*Write do Tab2, 2*Read z Tab3

Cena a benefit pre každú tabuľku umiestnenú na piatich možných miestach

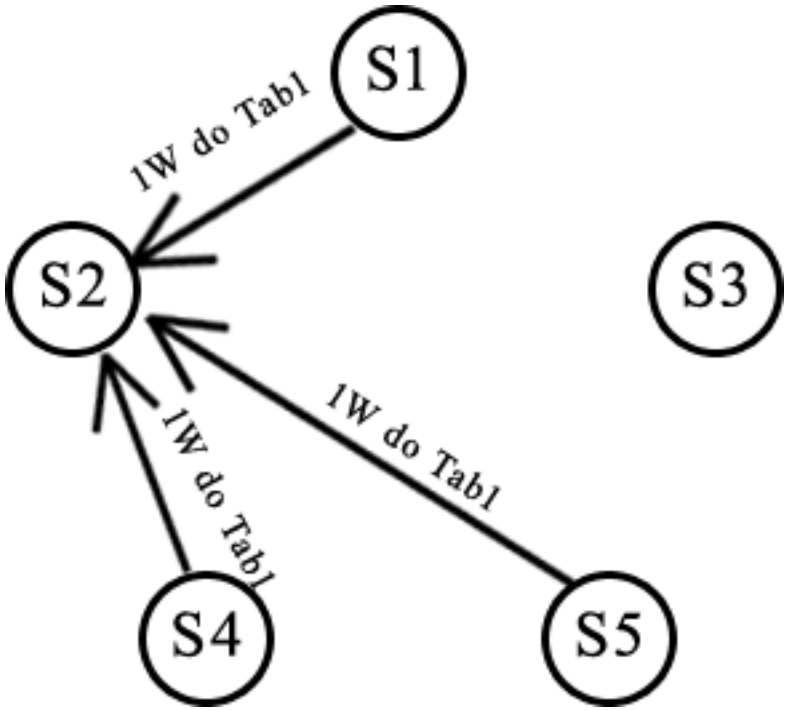
Table	Site	Remote Update (Local update) Transaction	No. Of Writes* Freq. *Time (Miliseconds)	Cost (milliseconds)
tab1	S1	T1 from S4 and S5 (T1 from S1)	2 * 1 * 600ms + 1*1*150ms	1350ms
	S2	T1 from S1,S4,S5	3 * 1 * 600ms	1800ms
	S3	T1 from S1,S4,S5	3 * 1 * 600ms	1800ms
	S4	T1 from S1 and S5 (T1 from S4)	2 * 1 * 600ms + 1*1*150ms	1350ms
	S5	T1 from S1 and S4 (T1 from S5)	2 * 1 * 600ms + 1*1*150ms	1350ms
Tab2	S1	T3 from S3 and S5	2 * 3 * 700ms	4200ms
	S2	T3 from S3 and S5	2 * 3 * 700ms	4200ms
	S3	T3 from S5 (T3 from S3)	1 * 3 * 700ms + 1*3*200ms	2700ms
	S4	T3 from S3 and S5	2 * 3 * 700ms	4200ms
	S5	T3 from S3 (T3 from S5)	1 * 3 * 700ms + 1*3*200ms	2700ms
tab3	S1	T2 from S2 and S4	2 * 2 * 1100ms	4400ms
	S2	T2 from S2 and S4	1 * 2 * 1100ms + 1*2*250ms	2700ms
	S3	T2 from S4 (T2 from S2)	2 * 2 * 1100ms	4400ms
	S4	T2 from S2 and S4	1 * 2 * 1100ms + 1*2*250ms	2700ms
	S5	T2 from S2 and S4	2 * 2 * 1100ms	4400ms

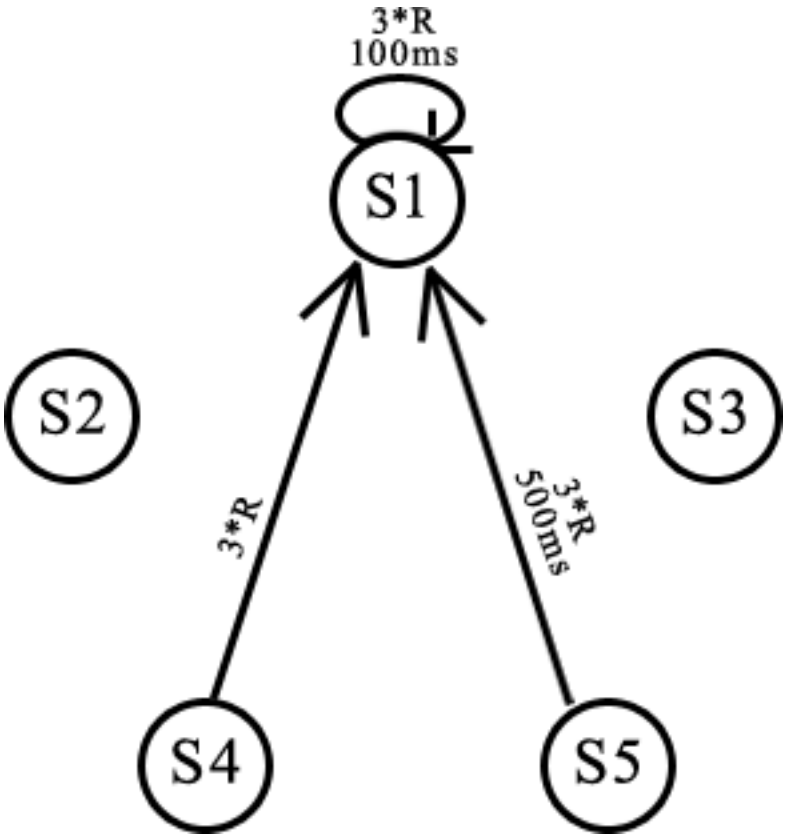
Table	Site	Query (Read) Source	No. of Reads*Freq. *Time (Remote-Local Time)	Benefit (milliseconds)
tab1	S1	T1 at S1	3*1*(500-100)	1200ms
	S2	T2 at S2	2*2*(500-100)	1600ms
	S3	None	0	0
	S4	T1 and T2 at S4	(3*1+2*2)*(500-100)	2800ms
	S5	T1 at S5	3*1*(500-100)	1200ms
tab2	S1	T1 at S1	2*1*(650-100)	1000ms
	S2	None	0	0
	S3	T3 at S3	3*3*(650-150)	4500ms
	S4	T1 at S4	2*1*(650-150)	1000ms
	S5	T1 at S3 at S5	(2*1+3*3)*(650-150)	5500ms
tab3	S1	None	0	0
	S2	T2 at S2	3*2*(1000-200)	4800ms
	S3	T3 at S3	2*3*(1000-200)	4800ms
	S4	T2 at S4	3*2*(1000-200)	4800ms
	S5	T3 at S5	2*3*(1000-200)	4800ms



T1: 3 * R Tab1
1 * W Tab1
2 * R Tab2

2.600ms + 1.150ms = 1350ms
S4,S5 S1


S1
1W do Tab1
S2
S3
1W do Tab1
1W do Tab1
S4
S5



Benefit: T1 v S1

3*(500 - 100)

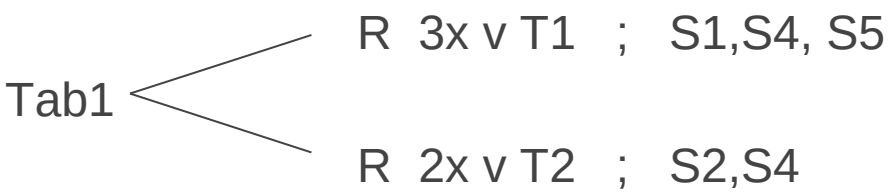
T1 v S2

žiaden

T1 v S4 alebo S5

žiadne

Benefity sa robia skrz všetky benefity



Tab1 z S4 R 3x v T1, 4x v T2

Tab1 čítame dokopy 7 krát z S4

Benefit: (500 - 100) = 7.400 = 2800ms

Tab1: S4

Tab2: S3,S5

Tab3: S2,S3,S4,S5