TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: M 2612 – Elektrotechnika a informatika

Studijní obor: 2612R011 – Elektronické informační a řídící systémy

Distribuované databázové systémy

Bakalářská práce

Autor: **Pavel Novák**

Vedoucí práce: Mgr. Jiří Vraný

Zadání:

Název tématu: **Distribuované databázové systémy**

Zásady pro vypracování:

1. Seznamte se s problematikou distribuovaných databázových systémů a proveďte rešerši aktuálně používaných systémů i teoretických vývojových směrů.
2. Na několika zvolených nejpoužívanějších systémech proveďte srovnávací test výkonu, uživatelské přístupnosti a dalších parametrů. Pro účely testování můžete využít některou z metodik uveřejněných na internetu.

Rozsah grafických prací: dle potřeby dokumentace

Rozsah průvodní zprávy: cca 30

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Rád bych poděkoval firmě VS – Elektron s.r.o. za veškerou pomoc a poskytnutí výpočetních prostředků, Petru Jedinému za cenné rady a podmětné připomínky při práci s Linuxem, firmě Trustnet s.r.o. za poskytnutí připojení k internetu. Rovněž děkuji rodině za podporu po celou dobu studia.

Anotace

Bakalářská práce je zaměřena na srovnávání distribuovaných databázových systémů. Databázové systémy jsou systémy pro uchovávání třízených dat, které je potřebujeme rychle použít za různými účely. V této práci se zabývám různými aspekty distribuovaných systémů a jejich dalším vývojem.

Při sestavování testovacích modelů je nutné dbát na hardwarové nároky, licenční a tím i finanční možnosti a použitelnost jednotlivých systémů v praxi.

Obsah:

Úvod

Vzhledem k vývoji počítačové techniky za posledních několik desítek let a integrace do všech složek běžného života, nastává doba, kdy je zapotřebí třídit, vyhledávat a uchovávat velká množství dat. K tomuto účelu složí databázové systémy. Většina dat, které dnes člověk používá ke své činnosti, zabírá v tištěné formě mnoho místa a je nepraktické s nimi v této formě pracovat vzhledem k rychlosti vyhledávání informací, jejich doručování k dalšímu zpracování atd. Proto se vyvinuly systémy, které tyto nedostatky odstraňují a dávají tak možnost většího pohodlí koncovému uživateli.

Bakalářská práce bude zkoumat použitelnost databázových systémů vzhledem k výše uvedeným bodům. Práce je rozdělena na část teoretickou a praktickou (měření na vybraných databázích). První část je zaměřena na výklad pojmů, popis použitých systémů a budoucností a vývojovým směrem tohoto odvětví IT. V druhé části jsou provedena jednotlivá měření na vybraných systémech. Výběr byl omezen na několik produktů, protože většina systémů podléhá licenčnímu právu a nebylo možné je volně testovat. Pro tyto testy bylo nutné najít vhodnou formu testování, které jsou částečně popsány na internetu.

Cílem této práce je seznámit se distribuovanými databázovými systémy, zhodnotit jejich uživatelskou přístupnost, změřit rychlosti zápisu, přepisu a vyhledávání. Dalšími cíli je seznámení s dalším vývojovým směrem v této oblasti.

1. Teoretická část

V teoretické části jsou rozebrány základní pojmy popisující databázové systémy. Je zde popsán použitý hardware, vybraný operační systém a vybrané druhy databází.

1.1. Základní pojmy

Databáze je množina uspořádaných dat uložena na paměťovém mediu a softwarové prostředky pro jejich správu. V české literatuře se nazývají systémem řízení báze dat (SŘBD).

Jakým si předchůdcem databází byly papírové kartotéky, které za určitých předem daných podmínek (abecední seznam, číselný kód, atd..) umožňovaly uživateli vyhledávat informace. Výběr a manipulaci s těmito daty prováděl přímo člověk. Dalším krokem k dnešním databázím bylo automatické zpracování za pomocí strojů, které načítali děrné štítky a tím mohly pracovat s daty. Dalším vývojovým skokem bylo již ukládání dat na pevné médium a vývoj nových programovacích jazyků pro zefektivnění práce s daty.

Database management systém (DBMS) je software umožňující přímé řízení chodu distribuované databáze.

Počítačový klaster je seskupení několika počítačů spojených sítí, které má za úkol úzkou spolupráci na daném úkolu, tak že navenek pracují jako jeden počítač. Zvyšují výpočetní výkon, snižují finanční zatížení, zvýší spolehlivost a rychlost výpočtů.

Open Source jde o licenci, která umožňuje bezplatně použít daný software a umožní získat zdrojový kód, který může být upraven podle potřeb uživatele v jakémkoliv rozsahu.

B-tree je stromová datová struktura, používaná pro uchovávání a vyhledávání dat. Data jsou zpracovávána v logaritmicky omezeném čase. Používá se především pro databázové aplikace.

(Data) node (ndbd): proces, při kterém se uložená data kopírují do jednotlivých uzlů distribuované databáze. Každý uzel je umístěn na jednotlivém počítači.

Node Group – skupina uzlů je skupina jednoho nebo více uzlů a uložených dat nebo jejich kopií.

Partion - část dat uložených klastrem. Je zde mnoho klastrových uzlů spojující se v jeden klastr. Každý uzel je zodpovědný nejméně za jednu kopii dat uložených v dostupném klastru.

iSCSI – označení pro technologii, která zapouzdřuje SCSI komunikaci do protokolu IP.

SAN (Storage Arrea Network) – dedikovaná storage síť sloužící k ukládání a zálohování dat. Slouží k propojení storage zařízení s host systémy, jimiž jsou servery nebo servery v klastru.

Fibre Channel – technologie propojení výkonných úložných zařízení a serverů k ukládání a zálohování dat.

SQL je zkratka pro **Structured Query Language** - jazyk strukturovaných dotazů Jazyk SQL je specializovaný pro obsluhu databází a interaktivní práci sní.

API (Application Programming Interface) – sada rutin používaných aplikacemi k vytváření požadavků na služby nižší úrovně, které jsou zajišťovány operačním systémem počítače, a jejich využívání. Pomocí těchto rutin jsou obvykle prováděny činnosti údržby, například správa souborů a zobrazení informa.

1.2.Distribuovaný databázový systém

Distribuovaný databázový systém je množina uzlů počítačové sítě, navzájem propojených v komunikační síti přičemž:

- každý uzel je samostatný databázový systém
- tyto uzly navzájem spolupracují tak, že z každého uzlu je možné zpřístupnit údaje uložené na jiném uzlu přesně tak, jakoby byly umístěny ve vlastním uzlu

Samotná databáze je množina navzájem propojených databází, které jsou umístěny na různých uzlech tak ,že uživatel s nimi manipuluje jakoby byly umístěny v centrální databázi.

O celé řízení databází se stará Distribuovaný systém řízení báze dat (DSŘBD), který spolupracuje s lokálními SŘBD.

Distribuovaný databázový systém je množina logicky svázaných databází distribuovaných na jednotlivých uzlech propojených sítí. DSŘBD je softwarový systém umožňující z hlediska uživatele transparentní řízení distribuované báze dat.

Pod pojmem transparentní rozumíme, že pro uživatele se systém jeví jako by byl řízený jediným SŘBD.

Architektury distribuovaných databázových systémů se od sebe odlišují hlavně autonomií jednotlivých lokálních databází a integrací dat. Podle těchto kritérii potom rozlišujeme systémy se samostatnými bázemi dat, kde přístup k jednotlivým datům a operacím s nimi je možné vykonávat pomocí systému vzdáleného přístupu RDA – Remote Database Access. Další skupinou jsou systémy se spojenými databázemi, založené na spolupráci nezávislých databází umístěných v jednotlivých uzlech.

1.2.1. Databázové modely

Modelem v běžné řeči bývá označován jakýsi objekt, který byl vymodelován na základě nějakých změřených nebo vypočtených údajů. U databází se ale jedná o soubor pojmů, pomocí kterých vytváříme struktury. Pro objekty se u databází používá pojem databázové schéma.

Pro komunikaci s databázovým modelem potřebujeme jazyk pro definici dat (JDD) a jazyk pro manipulaci s daty, jehož podmnožinou je i dotazovací jazyk.

U databází rozlišujeme tři základní modely. Síťový a hierarchický umožňují vytvářet grafové struktury se systémovou podporou realizující vazby mezi daty. Relační model nemá žádné fyzické vazby, ale má silné prostředky pro

logickou specifikaci vazeb. Dále se budeme věnovat jen síťovému a relačnímu modelu, vzhledem k jejich četnosti.

Síťový databázový model

Síťový model je založen na 2 pojmech pro vyjádření typu dat. Typ *záznam* je určen pomocí jména a atributů. Instancí typu záznam bude datová struktura zvaná záznam. Vazby se vyjadřují strukturou nazývanou DBTG-set. DBTG-set je uspořádaná n-tice, kde R je jméno typu DBTG-set a je dán spořádanou n-ticí S_1 až S_n. Naa úrovni instance typu hovoříme pak o DBTG-set množinách skládajících se z jednoho záznamu S_1 (vlastnický záznam) a množiny záznamů S_n (členské záznamy).

Databázové schéma se znázorňuje pomocí diagramů ve kterých šipkami ukazujeme orientaci. Výsledkem je potom orientovaný graf s ohodnocenými hranami.

Na množiny c síťovém databázovém modelu jsou kladena omezení. Nelze přímo modelovat vztahy s charakteristikou M:N, nelze vytvořit typ množiny pomocí jednoho typu záznam.

O záznamech v DBTG-set budeme předpokládat kruhovou implementaci, která umožňuje určit 1. členský záznam v DBTG-set, následníka zpracovaného záznamu, případně nalézt k členskému záznamu přímo vlastníka v rámci daného typu.

Dotazy se v síťovém modelu zpracovávají procedurálně. Jedná se o procházení grafu pomocí konstruktorů jako jsou *get first member of*, *get owner of*, a podobně.

Relační model dat

Relační model představuje jednoduchou abstrakci souborů. Místo o souboru hovoříme o relaci, místo o záznamech hovoříme o prvcích relace nebo o n-ticích relace. Každá komponenta n-tice je atomická hodnota. To znamená, že je-li komponenta např. množinou adres dodavatelů, bere se jako jeden řetězec znaků. Relace obsahuje dále skupinu jmen, která umožňuje se přímo odvolávat na komponenty n-tice jménem. Jména odpovídají přímo položkám ze souboru, které nazýváme jména atributů, příslušnou komponentu pak hodnotou atributu.

Prezentace relací na monitoru nebo na papíře se znázorňuje pomocí dvojrozměrných tabulek s pojmenovanými sloupci. Vzhledem k matematickým vlastnostem relací a atributům nerozhoduje pořadí sloupců ani řádků. Pojem relace se u RMD nahrazuje většinou nahrazuje pojmem tabulka. V praxi se tohoto používá pro odstínění od matematické stránky věci a tabulková terminologie se ujala hlavně v komerčních produktech. Formálněji je relace R nad množinou atributů $A = \{A_1 : D_1, \ldots, A_n : D_n\}$ podmnožinou kartézského součinu množin hodnot $D_1 \times \ldots \times D_n$. Di nazýváme domény, Ai jsou jména atributů, dvojice $A_i : D_i$ je atribut. Jména atributů jsou pro jednu relaci navzájem různá, domény se mohou opakovat. R je jméno relace. Relaci popisujeme schématem relace $R(A_1 : D_1, A_n : D_n)$.

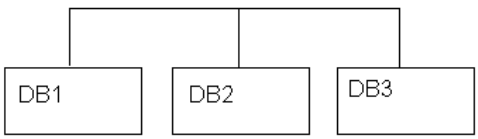
Významným prvkem IO na relaci R(A) je primární klíč. Primární klíč je množina prvků atributů z A, jejichž hodnoty jednoznačně určují n-tice relace R.

Nejdůležitějším přínosem RMD je relační kalkul a relační algebra. Relační SŘBD používají tyto prostředky jako zaklad dotazovacích jazyků. Relační algebra je dána operacemi, které mají za argumenty relace, jejichž výsledky jsou další relace. Relační kalkul je založen na logice 1. řádu.

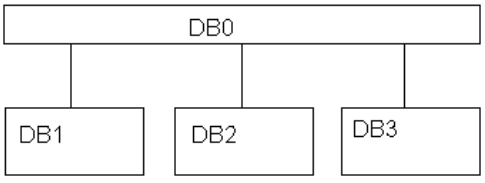
1.2.2. Klasifikace distribuovaných databázových systémů

Distribuovaný systém v zakladu klasifikujeme podle vlastní distribuce dat. Jedná-li se o několik databází pospojovaných dohromady, jedna se o tzv. „nepravou distribuci“ (obrázek 1.2.1.a). Jsou-li databáze organizovány společným organizátorem, hovoříme o „skutečné distribuci“ (obrázek 1.2.1.b). Mohou však existovat systémy, které jsou na rozhraní těchto forem.

Obr.1.2.1.a



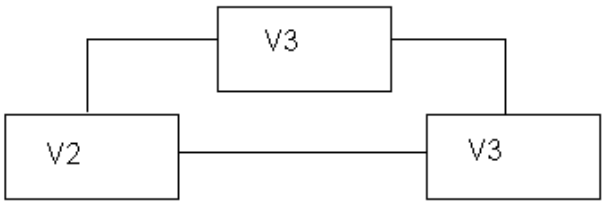
Obr.1.2.1.b



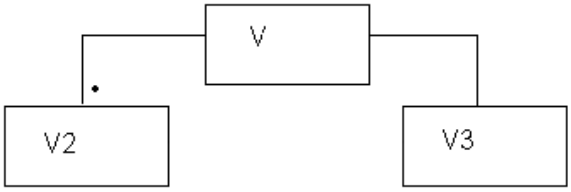
U distribuovaných systémů hovoříme také o oddělení a transparentnosti. Oddělenost jednotlivých komponent umožňuje dosáhnout paralelního provádění programů. Umožňuje také snazší zotavení po chybách a poruch komponent bez narušení systému. Transparence je potom záměrné zneviditelnění oddělenosti před uživatelem. Tomu je předložen celistvý systém namísto soustavy propojených komponent. Rozlišuje se celkem 8 druhů transparentnosti:
transparentnost přístupu – v distribuovaném systému by se mělo přistupovat ke všem souborům i jiným programovým objektům stejným způsobem a pomocí stejných operací bez hledu na to, zda se nalézají na „místním“ nebo na kterémkoliv „vzdáleném“ počítači.
transparence umístění – nejrůznější objekty by měly být přístupné bez toho, že by bylo nutné znát jejich konkrétní umístění v rámci distribuovaného systému
transparentnost souběžného přístupu – jednotliví uživatelé by měli mít možnost pracovat souběžně nad sdílenými daty bez vzájemného rušení
transparence replikací – pro zvýšení výkonu a spolehlivosti systému by měli být od všech datových objektů existovat více jejich kopií. Uživatel má potom možnost pracovat s kopiemi bez toho, že by museli vědět, že jde pouze o repliky
transparence výpadků – distribuovaný systém umožňuje odstínění uživatele od výpadku hardwaru a softwaru a umožní mu dokončit práci
transparence přesunů – v rámci systému by mělo být možné přesouvat objekty aniž by to bylo viditelné pro uživatele
transparence výkonu – systém by měl být rekonfigurovaný tak, aby se v případě vyšší zátěže dala tato zátěž rozložit a systém si zachoval svoji výkonnost
transparence růstu – systém a jeho aplikační programy by mělo být možné rozšiřovat bez změny celkové struktury systému

Dalším hlediskem podle kterého se dělí databázové systémy je rovnocennost uzlů. Pokud mají stejné programové vybavení, tzn. vyskytuje se úplné programové vybavení na každém uzlu, hovoříme o rovnocenném partnerovi (obr 1.2.1.c). V případě, že jeden uzel má vedoucí úlohu a obsahuje komponenty programového vybavení i dat, které se nevyskytují na žádném z ostatních uzlů, hovoříme o nerovnocenném partnerství (obr 1.2.1.d). Druhý typ architektury je stanice-server. Na pracovní stanice se používají data stažená ze serveru a uživatel je pak může využívat pro svoje transakce.

Obr.1.2.1.c



Obr.1.2.1.d



Původní SŘBD byli vyvinuty jako dávkově spouštěné programy, v současné době se používá transakční zpracování. Transakce je posloupnost logicky souvisejících akcí, jejichž provedením přecházíme z jedné přípustné databáze do druhé.

Distribuované architektuře SŘBD se vyskytují dva moduly: modul řízení transakcí a modul řízní dat. MŘT je část, která zodpovídá za aspekty vlastní distribuce dat. Má za úkol lokalizaci všech redundantních dat a zpracování uživatelských transakcí. Pokud systém obsahuje více MŘT používá transakce jenom jeden z nich. MŘT se dále dělí na transakční monitor a koordinátor transakcí. MŘD je část SŘBD, která pro lokální databáze plní běžné funkce centralizované SŘBD. Tento modul obsahuj také specifické akce jako jsou čtení hodnot logických objektů z dané lokální databáze, odsílání hodnot z pracovního prostoru MŘT, provádění operací read a write hodnot objektů a zápis hodnot logických objektů z pracovního prostoru do dané lokální databáze. Pro dvoucestné spojení mezi MŘT a MŘD se používá síťový modul. Modul také zajišťuje provedení změn ve všech uzlech, kde se vyskytují měněná data.

Podle používaných MŘD počítačových systémů se distribuované databázové systémy dál dělí na homogenní a heterogenní. V případě, že v každém uzlu jsou identické MŘD a počítačové systémy jedná s o plně homogenní systém. Heterogenní je takový systém, ve kterém jsou použity různé MŘD. U heterogenních systémů se spojení mezi MŘD a komunikačním systémem uskutečňuje pomocí translátoru, aby byla zajištěno, že bud vše jednotný datový model.

Distribuovaný systém má mnoho koordinačních problémů. Lokalizace dat a pravidla pro redundance představují pouze jeden z nich. Dále je potom nutné řešit problém s výpadkem jednoho uzlu, jak zařadit nový uzel, zjištění rozporu mezi dvěma uzly obsahující redundantní data, jeden uzel nebo celá autonomní část sítě musí pracovat samostatně, došlo-li k výpadku části (nebo celého) komunikačního

sytému. Jako další problém se jeví transparence umístění. Pro uživatele se distribuovaná databáze má jevit jako centralizovaná databáze. Transparentnost umístění je tedy stupeň, ve kterém SŘBD umožní vidět uživateli data nezávisle na umístění v uzlech sítě. S tímto souvisí pojem lokální autonomie. Jedná se o stupeň distribuovaný SŘBD, který umožní projektantovi uzlu nezávislost na zbytku systému. S těmito pojmy samozřejmě vyvstávají i další problémy jako pojmenování logických a fyzických objektů, údržba fyzických kopii jednoho logického objektu, fragmentace logických objektů a alokace fyzických reprezentantů ramenátů a jejich kopií.

1.3.Distribuce globálních relací

1.3.1. Fragmentace a alokace

Protože uživatel vnímá data ve formě globálních relací, ke kterým se vztahují jeho dotazy, musí se docílit co největší nezávislosti dat na aplikačních programech. V dotazech by se neměli vyskytovat aspekty uložení dat, způsob distribuce globálních dat do dílčích relací nebo redundantní uložení.

Distribuovaná data popisujeme takto:

1) určíme rozložení globálních dat do logických dílčích relací, potom můžeme toto rozložení popsat jako fragmentaci a části relace jako fragmenty
2) danému rozdělení přiřadíme místo uložení a případně místo redundantních uložení dalších částí

Tento postup definujeme pomocí zobrazení a označíme ho jako alokaci fragmentů.

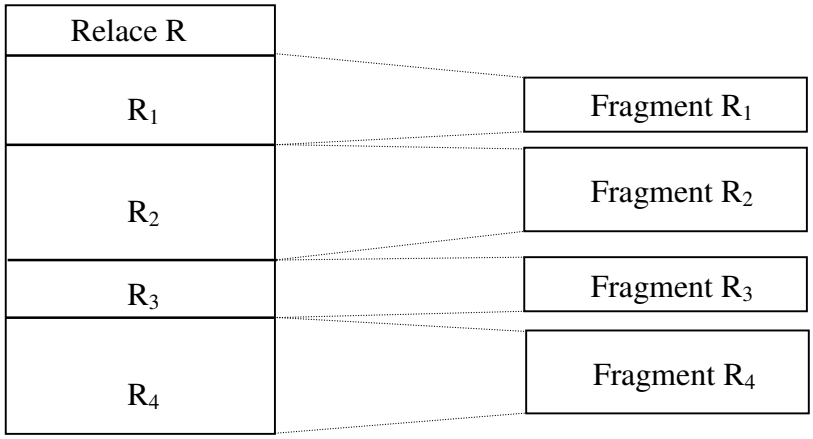
Rozdělení globálních relací:

- nefragmentované uložení
- horizontálně fragmentované
- vertikálně fragmentované
- kombinace předchozích metod

Horizontální fragmentace

Horizontální fragmentace je provedena pomocí operace relační algebry Výběr(Selection) na globální relaci podle databázového schématu. Umožňuje rozdělit relaci na několik fragmentů s totožným schématem relace tak, aby se

pomocí operace sjednocení dokázali zrekonstruovat původní relaci. Každá z fragmentů má takovou svoji kardinální relaci, že součet všech kardinálních fragmentů dává kardinalitu původní relace.



Relace R je horizontálně dekomponovaná na fragmenty $R_1, R_2, \ldots, R_n$, když

$$R_i = \sigma_{C_i}(R) \qquad i = 1..n$$

a pro rekonstrukci relace platí

$$R = \Upsilon(R_i) \qquad i = 1..n$$

Kde

σ - operace relační algebry Výběr s podmínkou C_i v konjunktivní normální formě pro daný fragment R_i.

Υ - operace relační algebry Sjednocení.

Pro fragmentování relací, které neobsahují atributy, podle kterých by bylo možné fragmentovat, se používá odvozená fragmentace. Fragmentace spočívá v myšlence, kde fragmentovaná relace se spojí pomocí operace polosjednocení (SemiJoin) s relací, kterou umíme fragmentovat. Relace, která vznikne spojením, fragmentujeme potm stejně jako u klasické horizontální fragmentace.

Odvozená horizontální fragmentace

Relace R' je horizontálně dekomponována na fragmenty $R_1, R_2, \ldots, R_n$, které vznikne z relace R, pro kterou platí, že ji získáme projekcí a polospojením relace R', kterou fragmentujeme a relací S, o které víme, že obsahuje atributy podle kterých je možné fragmentaci vykonat.

$$R = \underset{R'}{\pi}(R' \underset{K}{\otimes} S) \qquad i = 1..n$$

kde

π - je operace relační algebry Projekce.

$\otimes_{\mathbf{K}}$ - operace relační algebry Spojení přes množinu atributů *K*.

$$R_i = \sigma_{C_i}(R) \qquad i = 1..n$$

nebo

$$R_i = \sigma_{C_i}\left(\pi_{R'}\left(R' \otimes_{K} S\right)\right) \qquad i = 1..n$$

a pro rekonstrukci relace platí

$$R' = \Upsilon\,(R_i) \qquad i = 1..n$$

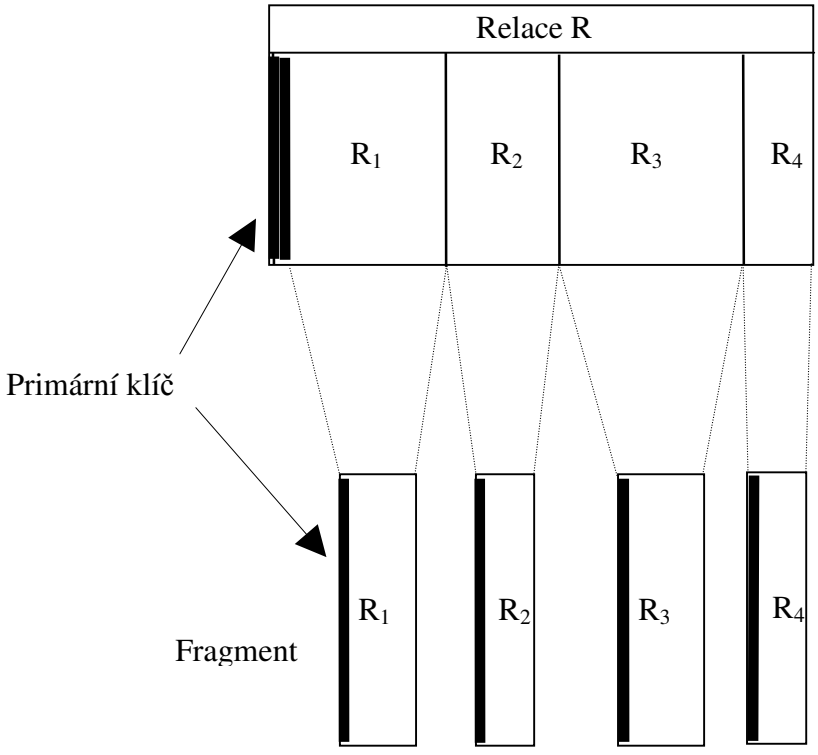
kde

σ - je operace relační algebry Výběr s podmínkou C_i v konjunktivní normální formě pro daný fragment R_i.

Υ - je operace relační algebry Sjednocení.

Vertikální fragmentace

Vertikální fragmentace se vykonává pomocí operace relační algebry Projekce (Projection) na globální relaci podle databázového schématu. Každý z fragmentů má kardinalitu původní relace. Množina atributů schématu fragmentu je podmnožinou atributů schématu původní relace, přičemž každý fragment obsahuje primární klíč původní relace. To nám umožní zrekonstruovat původní relaci pomocí operace relační algebry Spojení.



Relace R je vertikálně dekomponovaná na fragmenty $R_1, R_2, \ldots, R_n$, kde

$$R_i = \pi(R) \qquad i = 1..n$$

a pro rekonstrukci relace platí

$$R = \otimes_K(R_i) \qquad i = 1..n$$

kde
π - je operace relační algebry Projekce.
$\otimes_K$ - operace relační algebry Spojení přes množinu atributů K.

Hybridní fragmentace

V mnohých praktických situacích je výhodné použít vertikální a horizontální fragmentaci v kombinaci. Toto spojení umožňuje dekomponovat model na potřebné fragmenty při splnění požadovaných vlastností datového modelu. Podle toho, v jakém pořadí vykonáváme jednotlivé způsoby fragmentace se hybridní fragmentace dělí na:

- Horizontálně vertikální fragmentaci
- Vertikálně horizontální fragmentaci

1.4.Alokace fragmentů

U distribuovaných databázových systémů sledujeme dvě hlediska pro ukládání dat. První je efektivita při zpracování dotazu a druhé hledisko se vztahuje k zajištění vysoké dostupnosti dat. K prvnímu hledisku se vztahuje požadavek na co nejnižší cenu dotazu. Cena dotazu je pojem, kterým rozumíme čas přenosu, čas na vyhodnocení odpovědi a náklady na komunikaci v síti. Druhé kritérium splníme, pokud budeme moci zajistit v případě výpadku jednoho uzlu nerušený běh zbytku systému. Z toho vyplývá požadavek na redundantní uložení dat.

1.4.1. Optimální distribuce dat

Při optimalizaci distribuce musíme vědět, že se dají distribuovat jednak data, ale i funkce. Distribucí funkce znamená, že distribuujeme jisté transakce, které se týkají řízení přístupu, řízení jiných transakcí nebo zpracování dotazu a jsou přiřazeny jistým uživatelům. Zde se zavádí pojem transakční monitor. Jde o

monitor transakcí, který je uložen na některém uzlu a přebírá určitou část, přesně specifikovanou, dotazů a zpracovává je. Výhoda je v tom, že nepotřebuje, aby data ke kterým přistupuje by la uložena v daném uzlu. Na každém uzlu může běžet zároveň několik monitorů.

Zpracování dotazu potom bude vypadat takto:

- na libovolném uzlu i bude inicializován dotaz nebo aktualizační příkaz
- příkaz bude předán příslušnému uzlu j majícímu roli transakčního monitoru
- v případě dotazu vybere uzel j ten vhodný datový uzel k , kterým je při zpracování dotazu zapotřebí. V případě změny posílá uzel příkaz změny na všechny uzly, které obsahují kopie pozměněných dat.

1.5.Architektury schémat

Pod pojmem architektura schémat distribuovaného SŘBD se specifikují vazby, které existují mezi prvky systému nepovycházejí od uživatele.

1.5.1. Uzavřené a otevřené architektury

První architekturou, která se prosazovala v 70 letech minulého století byla uzavřená architektura. Jednalo se o koncept přístupu „shora dolu“., tj., kdy se vytvoří koncepce jednotlivých programových modulů a následně vše až po globální koncept schématu. Distribuované SŘBD se stará o řízní distribuované báze v uzlech sítě a s centrální autoritou se stará o logickou strukturu lokáních bází a sjednocuje je v jeden celek. Rozkladem těchto schémat vznikají lokální schémata pro jednotlivé uzly, řeší se kritéria pro fragmentaci a alokaci, která mají za úkol zefektivnit provoz celého systému.

Pro uzavřenou architekturu je typická homogennost programových prostředků, jednotný databázový model a pevná struktura celého systému.

Dalším přístupem, který se objevil v 80 letech minulého století, je „zdola nahoru“. Zde s vychází z existujícího fungujícího lokálního databázového systému a vytváří se integrovaný distribuovaný databázový systém. Usiluje se o unifikaci přístupu uživatele přes globální schéma a o autonomitu lokálních databázových aplikací. Při použití této architektury nastávají případy, kdy se

systém nemusí být konzistentní. Přístup „zdola nahoru“ představuje otevřenou architekturu, v které se uplatňuje nehomogennost použitých programových prostředků a databázových modelů.

Obě tyto metody ovšem vedou k centralizaci a přinášejí řadu nevýhod. Například obtížnou integraci existujícího programového vybavení , vznikají problémy vztahu správce dat a uživatelů, změny v uživatelské sféře nemusí být správcem dat správně oceněny, mnohdy se ztrácí lokální optimalizace systému ve prospěch globální optimalizace.

K vyřešení těchto problémů vedli logicky decentralizované architektury DDBS s více konceptuálními schématy anebo logické sítě vysoce autonomních databází. Systémy se potom dělí na:

- logicky centralizované a fyzicky centralizované
- logicky centralizované a fyzicky decentralizované
- logicky decentralizované a fyzicky centralizované
- logicky decentralizované a fyzicky decentralizované

Do první kategorie spadají klasické SŘBD. Většina distribuovaných SŘBD spadá o druhé kategorie. Třetí a čtvrtou kategorii většinou spojujme do jedné, jako logický decentralizovaný systém. Do této kategorie patří federativní architektura, která má jednotlivé komponenty rozdělné na lokální schémata, jež se dál dělí na cháma exportu (sdílní s ostatním komponentami) schéma importu (co používá z jiných komponent) a privátního schématu. Federativní katalog dat je komponenta, jenž obsahuje informace o ostatních komponentách a je zároveň nedostatkem této architektury, protože vede k centralizaci. Jemnější klasifikaci lze získat takto:

- autonomie
    (T) těsná integrace
    (P) poloautonomie
    (Ú) úplná autonomie
-fyzické distribuce
    ANO
    NE
-heterogenita
    ANO
    NE

1	Distribuované homogenní SŘBD	T	A	N
2	Distribuované homogenní federované SŘBD	P	A	N
3	Distribuované homogenní multidatabázové systémy	Ú	A	N
4	Distribuované heterogenní SŘBD	T	A	A
5	Distribuované heterogenní federované SŘBD	P	A	A
6	Distribuované heterogenní multidatabázové systémy	Ú	A	A
7	Logicky integrované homogenní vícenásobné SŘBD	T	N	N
8	Homogenní federované SŘBD v jednom místě	P	N	N
9	Multidatbázové systémy	Ú	N	N
10	Heterogenní integrované SŘBD	T	N	A
11	Heterogenní federované SŘBD v jednom místě	P	N	A
12	Heterogenní multidatabázové systémy	Ú	N	A

1.5.2. Obecná architektura

V obecné architektuře rozlišujeme schémata dle různých kritérii.

Lokální interní schéma (LIS) – schéma odpovídající centralizovanému databázovému systému

Lokální konceptuální schéma (LKS) – konceptuální schéma v centralizované architektuře,

každý uzel má lokální konceptuální schéma

Schéma lokální reprezentace (SLR) – schéma reprezentuje databázi „navenek“

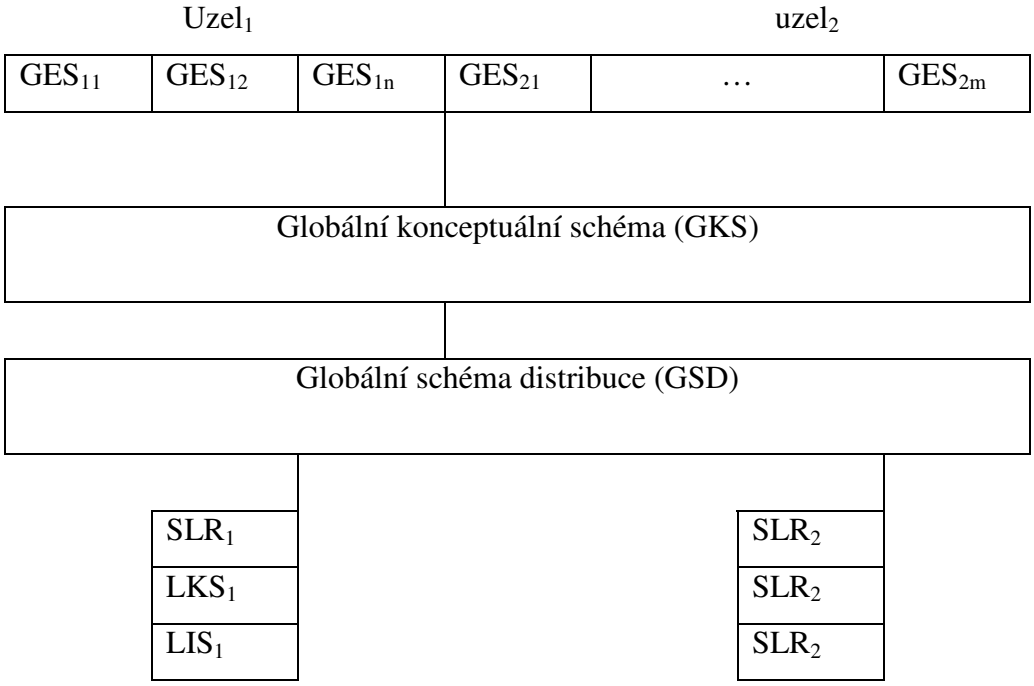
Globální schéma distribuce (GSD) – schéma popisující kritéria rozdělování dat na uzly a zda

z nich budou uloženy kopie

Globální konceptuální schéma (GKS) – umožňuje uživateli pohled na distribuovanou databázi

jako na jednu logickou jednotku

Globální externí schéma (GES) – umožňuje externí pohled na globální data



1.5.3. Realizace globálního katalogu

Globální katalog je katalog, který obsahuje všechna data potřebná pro ukládání a čtení dat z databází. V jednotlivých případech architektur se tento katalog a jeho umístění liší. V případech LIS, LKS, SLR a GES je uložen na svém „rodném“ uzlu. U GSD a GKS jsou zpravidla kladeny globální dotazy. Potřebujeme tedy globální katalog z každého uzlu, kde se zjistí, zda jsou názvy relací a atributů i jejich obory hodnot v globálním dotazu správné a jak jsou tyto relace realizovány logicky a fyzicky. Pro toto porovnání lz použít několik metod.

Centralizovaný katalog

Na centrálním počítači je umístěn celý katalog, ve kterém jsou uložena data o globálních relací. Jednoduchý systém s jednoduchým zabezpečením katalogu. Nevýhodou j vysoká cena komunikace, protože veškeré dotazy jdou přes tento uzel.

Plně redundantní katalog

Na každém uzlu je kompletní katalog. Řešení je vhodné pro případy, kde nedochází často ke změnám katalogu.

Několikanásobný katalog

Kombinace dvou předchozích řešení. Používá se pro případy, kde jsou spojeny sítě. V tom případě má každá síť uloženu jednu kopii katalogu.

Lokální katalog

Počítač má jen katalogové informace o lokálně uložených datech. Veškeré globální dotazy znamenají komunikaci s více uzly. U neredundantně uložených dat se dotazy zodpovědí bez další komunikace. U redundantně uložených dat je potřebná aktualizace kopií na odpovídajících uzlech.

1.6. Správa transakcí

Pojem transakce vyjadřuje spojení logicky sobě patřících jednotlivých operací, které mají své požadované vlastnosti jako jsou atomičnost, izolovaná atomičnost, permanentnost a uspořádatelnost. Atomičnost znamená vykonatelnost celé již dále nedělitelné transakce. Pokud nemůže být vykonána celá, vrátí se vše do původního stavu. Zde hovoříme o tzv. vrcení (rollback). Izolovaná vratnost j vrácení jedné transakce tak, že tím není postižena žádná jiná transakce. Permanentnost zaručuje, že všechny provedené změny budou uloženy v bázi dat a nemohou být ztraceny. Uspořádatelností se rozumí, že každý výsledek vznikne distribuovaným provedením určitého množství transakcí s musí dát získat i sériovým provedením posloupnosti těchto transakcí.

1.6.1. Globální a lokální transakce

Rozdíl mezi nimi je ten, že lokální transakce nepotřebují ke svému provedení přístup k datům z jiného uzlu. Globální přesahují svým rozsahem jeden uzel.

Globální transakce se při vykonávání rozpadají na podtransakce, které jsou řízeny primární transakcí. Jednotlivé transakce se vykonávají na různých uzlech distribuovaného systému. Pokud dílčí transakce T vytvoří další dílčí transakci T', označíme ji jako podtransakci transakce T.

V jednoduchém modelu při použití pouze transakce read a write znamená, že transakce T čte pouze jednu kopii logického objektu X a transakce T zapisuj změnu na všechny kopie logického objetu X.

1.6.2. Výpadky uzlů a přerušení řízení

Distribuované SŘBD jsou navrhovány pro co největší spolehlivost a dostupnost. Systém se musí umět vyrovnat s výpadkem uzlu nebo jednotlivá přerušení vedení. Proto je potřeba redundantně ukládat buď všechna nebo část dat. Dalším předpokladem je použití synchronizačních a koordinačních protokolů.

Mohou nastat tři základní typy výpadků.

1) Výpadek nastane v první fázi. Podtransakci není možno inicializovat a primární transakce nedostane zpětnou zprávu o akceptaci podtransakce.
2) Výpadek nastane v druhé fázi. Primární transakce nedostává zprávu o dosažení RC-bodu.
3) Výpadek nastane v třetí fázi. Primární transakce dostala zprávu RC nebo Abort, ale nemůže zaslat globální zprávu Commit nebo Abort.

1.6.3. Chyby u podtransakcí

Při výpadku v první fázi se primární transakce, pokud má možnost, pokusí inicializovat transakci na jiném uzlu a zaznamená časovou podmínku k původnímu uzlu. Pokud se mezi vykonávanou transakcí na jiném uzlu zpětně ozve první použitý uzel, primární transakce odešle na tento uzel příkaz Abort.

Pokud výpadek nastane v druhé fázi, primární transakce neví nic o podtransakci a proto odesílá po vypršení času na uzel příkaz Abort.

Systém se v případě chybových stavů snaží získávat informace i u sesterských podtransakcí. Tzn. snaží se zjistit, jestli jektá z nich neobdržela příkaz Abort nebo RC a v případě, že ano, pokračuje jako by zpráva došla přímo do primárního uzlu. Nastane-li výpadek v třetí fázi, lokální systém už přijal RC a zajistil změny. Zde však není jisté jestli přijal Commit nebo Abort z primárního

uzlu. Primární transakce proto zůstává i nadále aktivní do té doby, než zprávu doručí.

1.6.4. Chyby u primárních transakcí

Primární transakce řídí všechny podtransakce a proto její výpadek je zvláště kritický. Při výpadku v první fázi nastávají tyto případy: všechny podtransakce už ohlásily RC-bod, alespoň jedna podtransakce zahlásila Abort nebo žádná podtransakce dosud nehlásila zprávu Abort, ale také nehlásily RC-bod.

První případ znamená, že nelze spustit globální Abort a výsledkem je obecné čekání.

V druhém případě byl zahlášen transakcí Abort a proto nemůže být již proveden Commit a vyhlásí se globální Abort.

Pro třetí případ platí, buď podtransakce budou nadále čekat nebo celou transakci vrátí zpět a v tom případě vyhlásí zbyle transakce které ještě nic nehlásily Abort.

Při výpadku ve druhé fázi nastávají tyto případy: primární transakce odeslala Commit nebo Abort. V tomto případě se provede u podtransakcí Commit nebo Abort. Druhém případě se neodeslala žádná zpráva. Zahlásí-li podtransakce RC, pak se provede první případ. Pokud jedna transakce ohlásí Abort, primární transakce provede také Abort.

1.6.5. Zotavení po chybě

Krátkodobé zotavení

Při zotavování se provede:

- vrácení přerušených transakcí
- opětné zprovoznění lokální SŘBD po spadnutí systému
- zaznamenání změn uzavřených transakcí ze žurnálu (log file)
- vrácení neuzavřených transakcí
- automatické znovuspuštění vracených transakcí
- zajištění aktualizovaného a neaktualizovaného stavu dat
- ošetření jednoduchých chyb vzniklých při zápisu nebo čtení

Dlouhodobé zotavení

Konzistence databáze je založena na tom, že dokončená a korektně synchronizovaná transakce vytvoří zase jenom konzistentní databázi. Bohužel to neplatí vždy. Pokud nastane programátorská chyby nebo se například vyskytne hardwarový problém, nastane stav, kdy bude výsledek neodpovídat zadaným příkazům.

1.7. Synchronizace

U distribuovaných SŘBD vychází synchronizace obvykle z protokolů, které umožňují uzamykat a odemykat jednotlivé logické objekty. Pokud existuje přímé propojení mezi logickými a fyzickými objekty (bez kopií) lze se použít dvoufázový uzamykací protokol. Pro obecnější případ, je ale potřeba tyto protokoly příslušně pozměnit. V distribuovaném prostředí na rozdíl od centralizovaného většinou nestačí, aby se transakce prováděli organizovaně pomocí dvoufázové uzamykatelného protokolu. Zde je potřeba navíc uvažovat globální rozvrh transakcí.

1.7.1. Centrální a decentralizované uzamykání

Každý objekt ke kterému se snažíme přistoupit musí být na začátku uzamčen. Uzamykací modul nejdříve překontroluje v tabulce zámků, zda je daný objekt uzamčen a zda nehrozí žádný konflikt. Zámky jsou platné do té doby, než systém zaznamená RC-bod, Commit nebo Abort nebo pokud se transakce nezruší a nevrátí se vše zpět.

Centrální uzamykání spočívá v tom, že každý uzel obsahuje centrální tabulku zámků pro všechny uzly. Tato metoda přispívá k jednoduchému rozeznávání uváznutí a jednoduché implementaci. Pokud této tabulky použijeme při uzamykání globálních objektů a lokální se budou uzamykat přes lokální uzly, musíme rozeznávání a ošetření uváznutí řešit jako u decentralizovaných zamykání. Jednou z nevýhod je také velmi vysoké zatížení sítě a zranitelnost celého systému výpadkem centrálního uzlu.

Decentralizovaný systém má v každém uzlu vlastní tabulku. Používá se více pro jeho menší komunikační náklady a vyšší nezávislosti uzlů. Nevýhodou zůstává problém s rozpoznání globálního uváznutí.

1.7.2. Globální uváznutí

Uváznutí znamená, že transakce cyklicky čekají na uvolnění uzamčených objektů nebo jiných zdrojů. U distribuovaných systémů se objevují jednak stavy „čekání na něco“, ale také čekání na zprávy primární transakce či podtransakce. Možné uváznutí se zakreslují pomocí čekacích grafů, kde vrcholy jsou transakce a „čekání“ jsou orientované hrany. Uváznutí je v tomto případě každá kruhová oblast v grafu.

Dalším metodou je metoda časových razítek. Všechny podtransakce jsou opatřeny časovým razítkem primární transakce a tak nemůže dojít k zacyklení. Tato metoda využívá dvou příkazů, jež mají za úkol přerušení a vrácení zpět. První je Die, který používá časového razítka mladší transakce. Pokud se pokusí nějaký objekt uzavřít starší transakci, způsobí tím vlastní přerušení. Druhý příkaz je wound, jež aktivuje starší transakce. Při uzavírání objektu s mladším časovým razítkem, pak se tuto transakci pokusí přerušit.

Globální uváznutí se může rozpoznávat centralizovaně, decentralizovaně nebo jejich kombinací. Podnět k rozpoznání uváznutí vychází buď z překročení čekací doby nebo z neposkytnutého požadavku na uzamčení. Někdy se používá periodické spouštění rozpoznávání.

Centralizované rozpoznávání používá lokální grafy, které skládá do globálního v primárním uzlu a v něm hledá cykly. Decentralizované rozpoznávání používá přímo všechny uzly. Problémem u tohoto rozpoznávání je rozpoznání tzv. nepravých uváznutí a tudíž zbytečné vrácení zpět. Každý lokální uzel u této metody obsahuje externí vstup/výstup, který modeluje vztahy ke vnějšímu světu. Algoritmus je založen na elementárních cyklech, což znamená, že každý uzel kromě počátečního a koncového se vyskytuje pouze jednou a složitější cykly se sestavují z více elementárních.

1.8.Použitý hardware

Pro sestavení měřícího systému byly použity 4 počítače spojené 100Mb sítí. Hardwarová konfigurace počítačů: AMD Sempron 2800+ AM2, základní deska MSI K9 VGM-V, 1GB operační paměti, pevný disk Seagate 80GB IDE. Pro vytvoření sítě byl použit router WA-2204A. Jako připojení k internet jsem zvolil wi-fi od společnosti Trustnet s.r.o., která poskytla vysoce rychlostní připojení na dobu testu.

1.9. Použitý software

1.9.1. Operační systém

Při výběru OS byla limitujícím problémem finanční zátěž. Na OS použitelný v tomto případě bylo kladeno několik podmínek a to, musí být síťový, jednoduchá obsluha, možnost výběru a útlumu nežádoucích jevů (běh jiných programů při testu). V úvahu proto přicházeli pouze dva systémy a to Linux a Windows Server 2003. Vzhledem k finanční náročnosti při pořizování licence pro Windows Server 2003, která se musí koupit pro každý počítač, bylo jednoduší zvolit některou z distribucí Linuxu.

Distribuci Linuxu jsem zvolil Ubuntu Linux 6.10. Volba byla ovlivněna několika fóry o Linuxu. Dalším důvodem byla jeho jednoduchá instalace, nevelké hardwarové nároky a jednoduchost při instalování dalších součástí potřebných pro test.

1.9.2. Použité databázové systémy

Open Source databáze

MySQL

Relační databáze ukládající data do oddělených tabulek. To dodává rychlost a flexibilitu. SQL je nejběžnější standardizovaný jazyk k zpřístupnění databází a je definován ANSI/ISO SQL Standard. SQL Standard je vyvíjen od roku 1986 a v dnešní době existuje několik verzí. MySQL je Open source

software pro nekomerční využití v opačném případě je možné zakoupit licencovanou verzi u které je možné též měnit zdrojový kód. Databázový systém vyvíjí společnost MySQL AB, jenž založili původní vývojáři tohoto systému.

MySQL server byl původně vyvinut, aby zvládal větší databáze mnohem rychleji než existující řešení. Pro svoji universálnost byl úspěšně využíván ve vysoce náročném prostředí po několik let. Ačkoli pod neustálým vývojem, MySQL server dnes nabízí bohatou a užitečnou sadu funkcí. Jeho konektivita, rychlost, a zabezpečení dělá z MySQL server vysoce vyhovující pro zpřístupnění databází na internetu.

MySQL Database Software je klient/server systém, který obsahuje multi-threaded SQL server podporující různé backend, několik různých klientských programů a knihoven, administrativních nástrojů a širokou škálu aplikací programování rozhraní (API).

Hlavní vlastnosti MySQL:

- Psáno v C a C++, pro programovaní lze využít mnoho dalších jazyků (Perl, PHP, Jawa, atd.)
- Podporuje širokou škálou překladačů
- Pracuje na mnoha různých operačních systémech
- Užívá GNU Automate, Autoconf a Libtool pro přenositelnost na ostatní databáze
- MySQL server design je vícevrstvý s nezávislými moduly
- Plně multi-threaded využívá více vláken jádra. Snadněji využije více procesorů, pokud jsou k dispozici.
- Používá velice rychle B-tree diskové tabulky (MYISAM) s indexovou kompresí.

Zabezpečení:

Práva a hesla k systému jsou velmi flexibilní a bezpečné a umožňují správci dobrou kontrolu nad systémem. Hesla jsou bezpečná, protože veškerý přenos hesel je šifrován, jakmile se klient připojí k serveru.

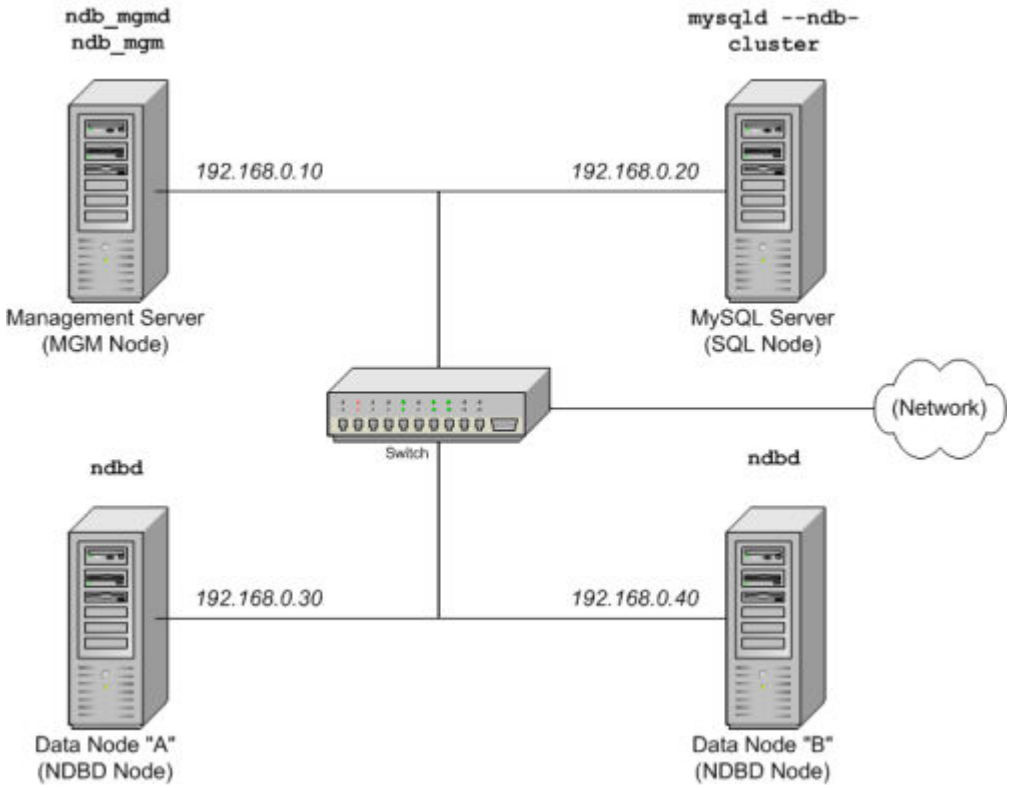
MySQL Cluster

MySQL Cluster je technologie umožňující použít databáze v „shared-nothing sytem“. Tato technologie dovoluje systému pracovat s velmi levným hardwarem a s minimálními požadavky na hardware a software. Klastr je navrhnut tak, aby nemohlo dojít k jednobodové chybě. Pro toto řešení je potřeba, aby každý použitý počítač měl svoji operační paměť a pevný disk. Shared-nothing sytem používá mechanismy sdílení

(síťové sdílení, síťový souborový systém). SAN není doporučován ani podporován. Jako minimální konfigurace sítě je vyžadován 100Mb Ethernet.

Vytvoření klastru

Pro vytvoření minimálního klastru je zapotřebí 4 počítačů. Tento počet se dá uměle snížit spojením Data Node „A“ s Management server a Data Node „B“ s MySQL sever. Ušetří hardwarové prostředky, ale zpomalí veškeré operace. Tato konfigurace je vhodná pro zálohu používaných dat u malých systémů. Pro běžné použití je vhodné mít minimálně 4 systémy. Použijeme-li více Data Node, celý systém je stabilnější a rychlejší.



Konfigurace Storage a SQL Nodes

Pro vytvoření klastru je nutné vytvořit v souboru my.cnf (konfigurační soubor) propojení s ostatními počítači. V souboru my.cnf se uvádí umístění mysqld a ndbd.

```
# Options for mysqld process:
[MYSQLD]
ndbcluster                      # run NDB storage engine
ndb-connectstring=192.168.0.10  # umístění management serveru
```

```
# Options for ndbd process:
[MYSQL_CLUSTER]
ndb-connectstring=192.168.0.10  # umístění management deamon serveru
```

Konfigurace management node

Zdrojový kód pro management node:

```
# Options affecting ndbd processes on all data nodes:
[NDBD DEFAULT]
NoOfReplicas=2    # Number of replicas
DataMemory=80M    # How much memory to allocate for data storage
IndexMemory=18M   # How much memory to allocate for index storage
                  # For DataMemory and IndexMemory, we have used the
                  # default values. Since the "world" database takes
up
                  # only about 500KB, this should be more than enough
for
                  # this example Cluster setup.

# TCP/IP options:
[TCP DEFAULT]
portnumber=2202   # This the default; however, you can use any
                  # port that is free for all the hosts in cluster
                  # Note: It is recommended beginning with MySQL 5.0
that
                  # you do not specify the portnumber at all and
simply allow
                  # the default value to be used instead

# Management process options:
[NDB_MGMD]
hostname=192.168.0.10           # Hostname or IP address of MGM node
datadir=/var/lib/mysql-cluster  # Directory for MGM node log files

# Options for data node "A":
[NDBD]
                                # (one [NDBD] section per data node)
hostname=192.168.0.30           # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node's data
files

# Options for data node "B":
[NDBD]
hostname=192.168.0.40           # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node's data
files

# SQL node options:
[MYSQLD]
hostname=192.168.0.20           # Hostname or IP address
                                # (additional mysqld connections can
be
                                # specified for this node for various
                                # purposes such as running
ndb_restore)
```

Po této konfiguraci není složité spustit klastr. Každý uzel se musí spustit samostatně. Management node musí být spuštěn jako první, následuje spuštění data node a jako poslední se spouští Sql node. Po této operaci je klastr plně funkční a připraven k použití.

PostgreSQL

PostgreSQL vychází z jazyka Postgres. Z prostředí databáze byl převzat datový model a datové typy. Jazyk PostQuel byl nahrazen jazykem SQL. Jedná se o Open Source software. Na vývoji se podílí mnoho vývojářů, kteří se do tohoto projektu zapojili přes vývojářskou konferenci PostgreSQL a jsou koordinovány Marcem G. Fournierem. Systém je multiplatformní.

Vytvoření klastru

Vytváření klastru začneme inicializací database storage area na disku, voláním databázového klastru. Databázový klastr je kolekce databází, které jsou zpravovány jedním databázovým serverem. Po inicializaci obsahuje klastr databázové jméno *postgres*, které je nastaveno jako základní databáze pro použití pomůcek a uživatelů. Databázový server nepotřebuje, aby *postgres* databáze existovala, ale mnoho externích programů předpokládá její existenci (např. Slony-I).

V souborovém systému, budou databázové klastry jednotlivými složkami pod kterými budou všechna data uložena. Nazýváme je *data direktory* nebo *data area*. Data nemají standardně nastavenou cestu a je jen na uživateli kam je umístí(např. /usr/local/pgsql/data nebo /var/lib/ pgsql/data). Pro inicializaci klastru použij příkaz *initdb,* který je instalován s PostgreSQL.

root# **mkdir /usr/local/pgsql/data**
root# **chown postgres /usr/local/pgsql/data**
root# **su postgres**
postgres$ **initdb -D /usr/local/pgsql/data**

Tento klastr neumí předávat informace, pouze je rozkládá na jednotlivé počítače a následně je z nich čte. Pro „master-multi slaves system“ je zapotřebí nástavbový software s názvem Slony-I.

Slony-I

Slony-I je systém určen pro datová centra a zálohovací sítě, kde normální módy operací a uzly jsou dostupné po celou dobu a kde všechny uzly mhou být chráněny. Pokud se ovšem uzly pravidelně vypínají nebo nejsou chráněny, Slony-I není ideální řešení zálohování a distribuce.

Slony-I:

- kopírování dat do uzlů je neočekávané spojení
- kopírování databází z hlavního serveru do uzlů probíhá periodicky
- konfigurace změn probíhá v libovolném směru

Slony-I není:

- síťový management systém
- neumí detekovat automaticky chyby, neumí automaticky vytvořit z uzlu mástra nebo jiná data
- není multi-master zálohovací systém

Komerční databáze

V sekci komerční databáze jsou popsány 2 největší hráči na poli komerčních, běžně používaných databázových systémů. Jedná se o Microsoft SQL Server a Oracle. Tento software nebyl dále zkoušen v Praktické části z licenčních důvodů.

Microsoft SQL Server

Relační DBMS produkovaný firmou Microsoft. SQL server je vhodný pro malé nebo středě velké databáze. Jedná se o komerční produkt licencovaný firmou Microsoft.

Historie

Základní kód MS SQL Sever je v originále Sybase SQL Sever a byl Microsoftem uveden na trh jako protipól konkurenčnímu Oraclu, IBM a dalším velkým hráčům na tomto poli.

První verzí byl v roce 1989 SQL Server 1.0 na platformě OS/2, který byl následován systémem Sybase SQL Server 3.0 pro Unix, VMS atd.

V roce 1992 Microsoft vydal Microsoft SQL Server 4.21 pro Windows NT a od verze 6.0 se software plně osamostatnil a přestal používat základy ze Sybase. Nejnovější verzí je Microsoft SQL Server 2005, která ve verzi SQL Server 2005 Express Edition je volně ke stažení.

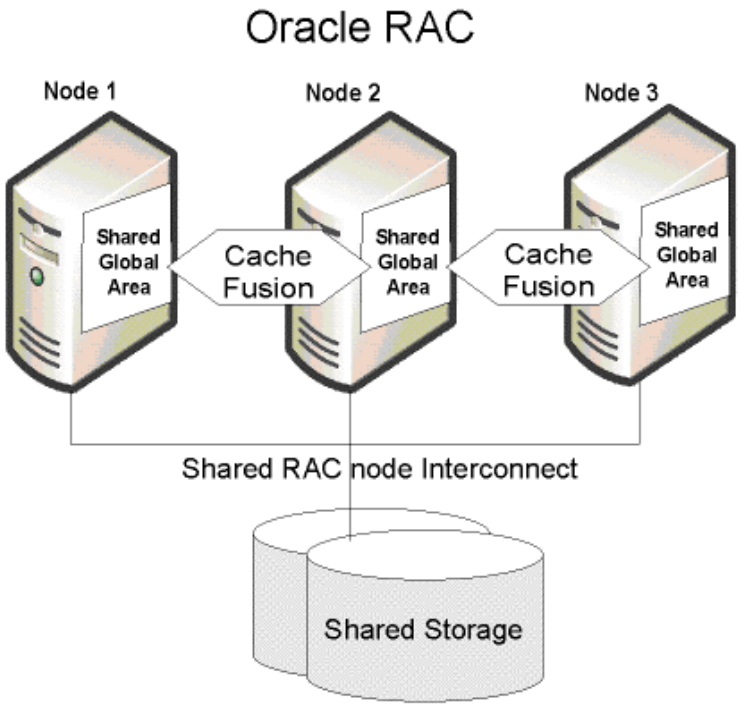
Databázový systém obsahuje několik jazyků pro programování (.NET, Visual Basic). Do nejnovější verze byli naimplementovány služby pro rychlou replikaci dat, podpory pro SAP produktů atd.

Oracle

Moderní multiplatformní databázový systém s pokročilými možnostmi zpracování dat, s vysokým výkonem a snadnou škálovatelností. Systém je vyvíjen firmou Oracle Corporation a aktuální verzí je Oracle Database 10g. Jedná se o relační databázi s implementovaným jazykem SQL, který je rozšířen o Oracle funkce. Z programovacích jazyků obsahuje např. PL/SQL, XML, XSQL a další.

Distribuovatelnost systému je zde vyřešena pomocí nástavbového programu Oracle NET. Oracle NET usnadňuje sdílení dat mezi databázemi i v případě, že tyto databáze jsou uloženy na různých typech serverů pracujících v různých operačních systémech a s různými komunikačními protokoly.Pro konfiguraci se používá nástroj Oracle NET Configuration Assistant, který po instalaci provede počáteční kroky konfigurace sítě a automaticky vytvoří výchozí, základní konfigurační soubory. Nástroj pod Windows obsahuje grafické rozhraní pro pohodlnější konfiguraci. Pro další práci se systémem a jeho správu se používá Oracle NET Manager, který má za úkol monitorovat veškeré dění v síti.

Příklad zapojení databáze Oracle



2. Praktická část

2.1. Metodika testování databází

V tomto testu se budeme zabývat několika metodami jak otestovat databáze. Pro určení, která z databází je lepší, je zapotřebí určit které atributy databází je možné otestovat a které jsou pro užívání zásadní.

Testovat se dá vytvoření databáze, přepis, čtení dat které databáze obsahuje a čtení dat které v databázi nejsou. Dále je zapotřebí zhodnotit kolik chyb se vyskytne pokud, je část systému mimo provoz.

U těchto testů jsou různé metodiky a na internetu se dají vyhledat programy, které tyto testy umí provést. Databázové programy mají testy v sobě implementovány, ale u různých systémů jsou tyto testy různé a může se stát, že nebude vždy testováno vše co potřebujeme. V každém případě u těchto testů nevidíme postup zpracování. Nevíme jakým způsobem se test provádí, jakým způsobem přistupuje k databázi, jak zní čte a jak do ní data zapisuje. Je několik možností jak tento problém odstranit. Jedna z možností je vytvořit externí benchmarkový program pro tyto testy. V několika diskuzích zabývající se databázovými systémy jsem našel program, který vyhovoval naším potřebám. Testoval výše uvedené problémové stavy. Program vypisoval pouze čas potřebný pro jednotlivé procesy. U testu výskytu chyb jsem uměle navodil situaci, kdy jsem záměrně odpojil jeden z data node a testoval jak se databáze zachová.

2.2. Test databází

2.2.1. Podmínky testů

Každý výsledek z níže prováděných testů je hodnota aritmetického průměru z 10 provedených testů. Testy jsou prováděny v laboratorních podmínkách, na počítačích, na kterých byly spuštěny pouze databázové systémy a vše ostatní bylo vypnuto.

2.2.2. Použitá data

Pro praktickou část jsem použil generovaná data pro testy čtení, zápisu a přepisu (obr 2.2.2). Pro test chybovosti a čtení dat, které v databázi nejsou obsaženy byli použita data z databáze logů firmy Trustnet s. r. o.

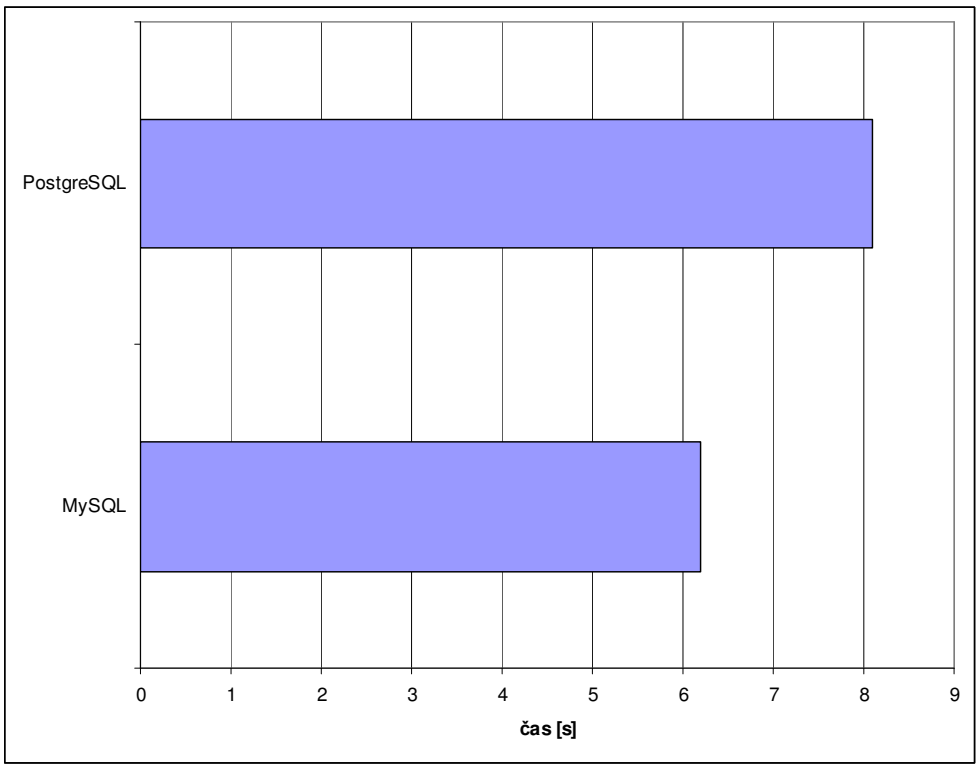
Jméno	Přijmení	Město	Pozice
. . .	. . .	. . .	. . .

Obr 2.2.2

2.2.3. Test vytvoření databáze

Tento test prověří za jak dlouho se vytvoří databáze, která bude mít 100 000 záznamů. Záznamy budou automaticky vygenerovány.

databáze	Čas [s]
MySQL	6,2
PostgreSQL	8,1



Při testu vytvoření databáze jsem nechal testovací program vytvořit databázi o 100 000 položkách. Data se automaticky vygenerovaly a poté se přenesly najednou do databáze. V případě, že by se data generovaly a posílaly se postupně do databáze, vznikala by chyba spojená s rychlostí disků, kdy databáze

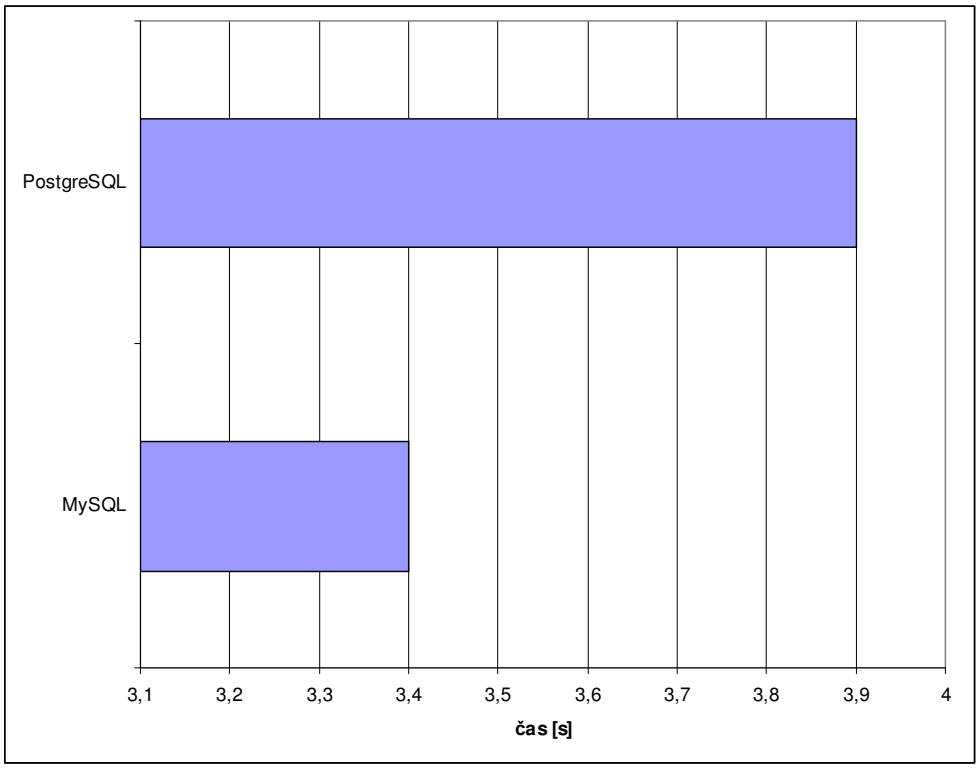
by pokaždé musela čekat než disk vykoná operaci. V tomto případě jsem se tomuto problému vyhnul.

Z testu vyplývá, že MySQL je při vytváření rychlejší než PostgreSQL o 1,9 sekundy.

2.2.4. Test přepisu databáze

Test prověří za jak dlouho se položky v databázi dokáží přepsat. Záznamy budou opět automaticky vygenerovány. Data budou mít stejnou velikost jako původní data. Nebudeme zvažovat přepis dat o větší nebo menší délce. V těchto případech by se měla zvětšit nebo zmenšit velikost databáze na disku. Vzhledem k velikosti testovaných databází je tento test nezajímavý, protože změny nejsou tak markantní.

databáze	Čas [s]
MySQL	3,4
PostgreSQL	3,9



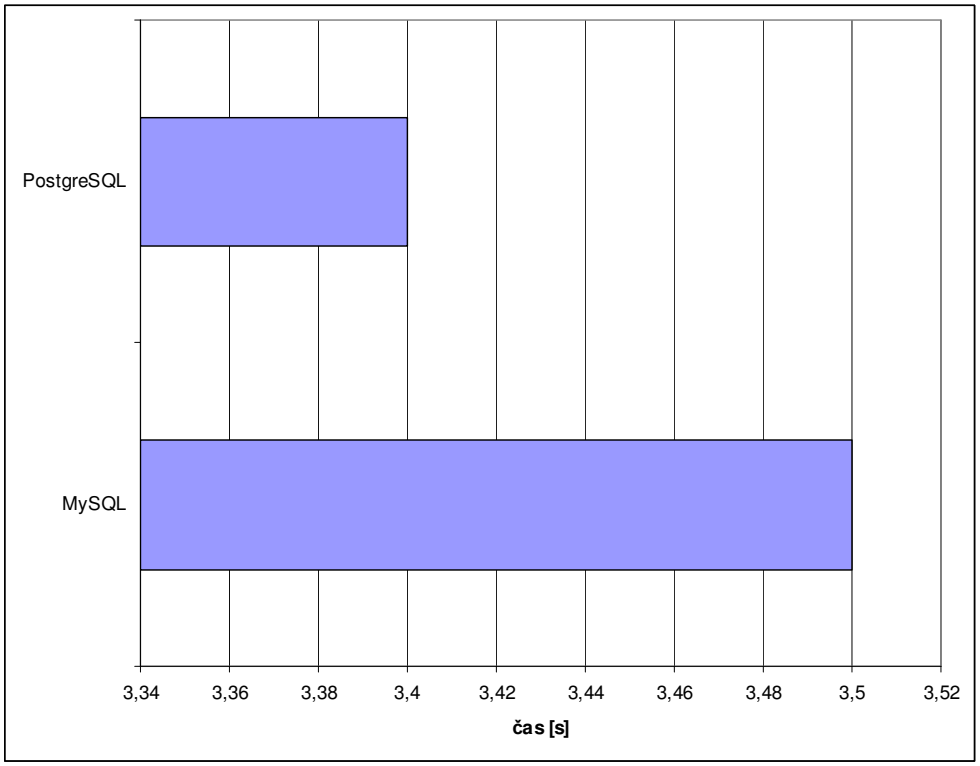
Testovaní přepisu probíhalo pomocí vygenerovaných dat, které se zapisovaly na místo původních. Data byla vygenerována ve stejné délce jako

původní. Při změně délky dat by mělo dojít také ke změně velikosti databáze na disku. Tento test nebyl proveden z důvodu velikosti původní databáze (změna by nebyla markantní) a hlavně protože testovací software tuto funkci nepodporoval. Test ukázal, že MySQL bylo opět rychlejší než PostgreSQL.

2.2.5. Test čtení z databáze

Tento test je velice důležitý, protože většina práce s databází je čtení. Rychlost čtení dat je proto velmi důležitý faktor.

databáze	Čas [s]
MySQL	3,5
PostgreSQL	3,4

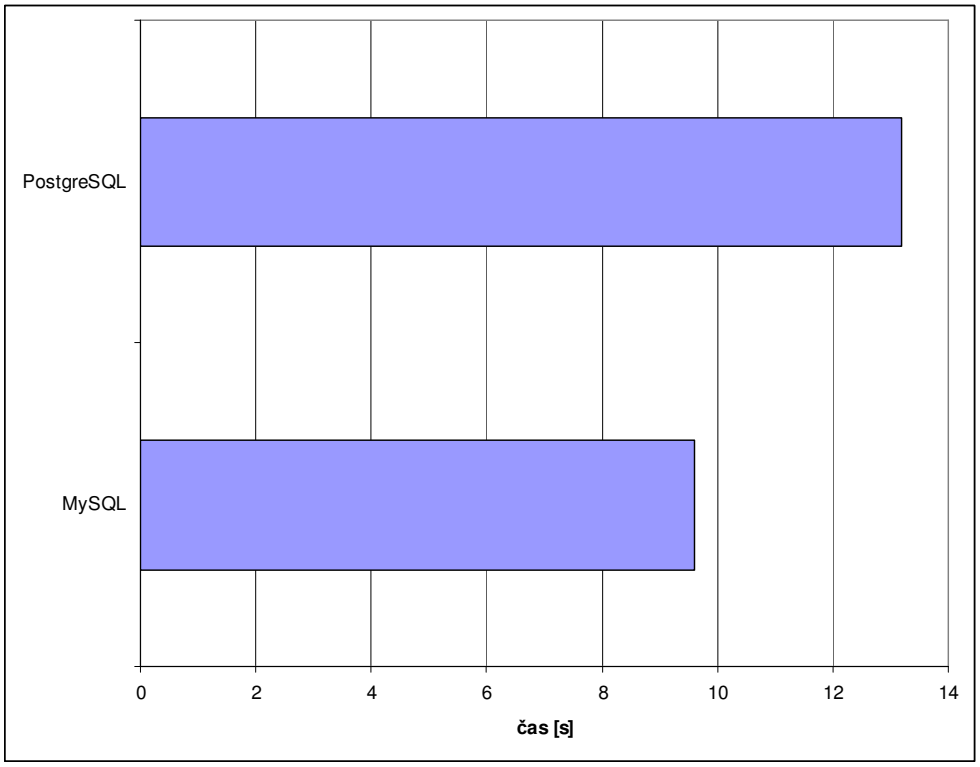


Čtení dat z databáze je nejdůležitější operací, která se z databázemi provádí. Při práci s databázemi je na prvním místě výskytu dotazů. Většina uživatelů v databázích pouze čte a proto je tato operace důležitá.

2.2.6. Test čtení dat, které v databázi nejsou obsaženy

Při vyhledávání dat se často stává, že vyhledáváme data, která nejsou v databázi obsažena. Test proběhne v poměru 80:20. Tzn. 80% ve prospěch existujících dat a 20% dat která v databázi nejsou.

databáze	Čas [s]
MySQL	9,6
PostgreSQL	13,2



Na předchozí test navazuje test čtení dat, která v databázi neexistují. Test spočívá v tom, že budeme načítat data, která v databázi nejsou, ale budou prokládána daty která v databázi existují v poměru 80:20 v prospěch existujících. V běžné praxi se tento jev často vyskytuje. Uživatelé buďto hledají neexistující data nebo svůj dotaz špatně zadají a systém tak nemůže daný dotaz zodpovědět. Podle statistik jsou tyto překlepy v 30% případů čtení. MySQL si s touto operací poradil za 9,6 s, PostgreSQL to trvalo 13,2 s. V testu čtení dat byl čas MySQL o 0,1 s pomalejší než u Postere, ale při čtení dat prokládané daty neexistujícími se role markantně obrátila a PostgreSQL zaostalo o 3,6 s.

2.2.7. Test chybovosti databáze

Úkolem testu je odhalit chování databáze při výpadku jednoho z uzlů.V testu jsem záměrně odpojil jeden z data node .Zkoumal jsem, jak se data po znovu připojení a navázání spojení budou chovat. Při výpadků uzlu by ideální databáze měla fungovat dále beze změn. Zátěž se musí automaticky rozdělit do zbylých uzlů.

databáze	Počet chyb
MySQL	0
PostgreSQL	0

My SQL zahlásí odpojení daného uzlu a nemožnost do něj zapisovat, ale po znovu připojení se data automaticky přehrají podle zbývajících uzlů. Pro tyto případy je vhodné mít více než dva uzly. V případě tohoto stavu se managment obrací na ostatní uzly, od kterých nedostal chybové hlášení a na základě toho se rozhoduje jak ušlá data nahradí.MySQL zazálohoval scházející data do znovu připojeného data node. Managment ohlásil jen ztrátu spojení a znovu jeho navázání. Zkopíroval data a systém byl opět funkční bez ztrát dat.

U PostgeSQL nastane hned v úvodu problém, protože databáze nemá chybová hlášení o odpojení uzlů. Chybová hlášení se neobjevují a data z hlavního serveru se automaticky bez kontroly nahrávají do zbylých větví. Kolizní stav by mohl nastat pokud by se databáze nestačila zaktualizovat dříve než k ní přistoupí některý z uživatelů.

Pro dlouhodobější přistup je vhodnější použít MySQL server. MySQL server dává díky své architektuře lepší přístup k aktuálním datům a v případě výpadku se data okamžitě aktualizují a během aktualizace se tento node nepoužívá k přístupu. Oba testy proběhly bez problémů a data se nahradila beze ztrát.

2.3. Budoucnost distribuovaných databází

Dnešní trendy v tomto směru vykazují potřebu větších a rychlejších databází. Pro toto odvětví je zapotřebí mnoho volného místa na ukládání a zálohování dat. Je potřeba též, aby servery mezi sebou mohly komunikovat co nejrychleji a proto je

zapotřebí rychlé páteřní spoje mezi jednotlivými uzly. Dále je požadováno, aby výpadek jednoho nebo více uzlů neohrozil další chod systému.

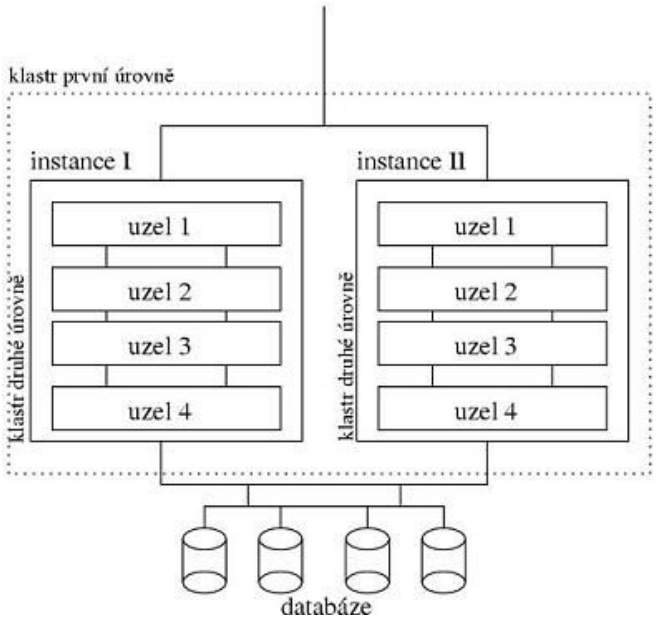
Jako jedna z alternativ těchto problémů se nabízí *Víceúrovňové databázové klastry*.

Návrh spočívá na principu rozdělení databázové instance na více vzájemně propojených serverů. Pod tímto pojmem rozumíme procesy v rámci jednoho běhu OS sloužící k ovládání databáze. Toto řešení transparentně přistupuje ke všem datům a neutralizuje případné výpadky v systému. Nastane-li výpadek, všechny aplikace směřují pouze na nedotčené instance. Nevýhodou tohoto řešení je, že nedochází k přímému distribuování zátěže, ale vše je směřováno pouze na primární instanci. Problém je v tom, že ne každá aplikace dokáže na softwarové vrstvě vyvolat takovou škálovatelnost, aby režie na provoz neovlivnila svoji vlastní výkonnost. Model se jakoby vrací k pseudo distribuovaným databázím, ale nabízí kompromis, kdy se hlavní provoz směřuje jen na jednu stanici, ale ostatní stanice mohou vykonávat předem jasně určené požadavky.

Systém zde naráží na druhou nevýhodu a tou je nesnadná rozšiřitelnost systému.

Řešení spočívá v rozdělení jedné instance na klastr druhé úrovně. Klastr musí zajistit jedno prostředí pro běh aplikace. Protože každá úroveň systému má na starosti pouze jednu vlastnost, může být vrstva lépe optimalizována a realizace je snadnější. Výsledkem je vysoce dostupný systém, který se dá dále jednoduše rozšiřovat.

Schéma modelu víceúrovňového klastru



3. Závěr

V závěru bych zhodnotil výsledky a průběh práce. V první etapě jsem prostudoval dostupný materiál k distribuovaným databázovým systémům a publikace k testům databází. Po pročtení a nasbírání dostatku informací o tomto problému jsem přistoupil k samotnému otestování.

V teoretické části jsem se zabýval principy funkčnosti distribuovaných databázových systémů. Jsou zde popsány základní principy a postupy při stavbě databázových systémů a základní hardware potřebný pro sestavení systému.

Při testech jsem narazil na několik problémů, jako byly licenční podmínky u jednotlivých databázových systémů. Pro samotný test bylo nutné nainstalovat tyto databáze a po spuštění distribuce je otestovat na výše uvedené vlastnosti. V testech se prokázalo, že pro běžné použití by je nejvhodnější MySQL server pro svoji výkonnostní nadvládu. Bohužel zde nemohly být použity komerční produkty, které by do celkových výsledků promluvily.

4. Přílohy

Při zpracování této bakalářské práce nastalo několik problémů. První z nich bylo shromáždit vhodnou literaturu k danému tématu. Na našem knižním trhu nejsou přímo knihy, které by se daným tématem zabývaly a proto bylo nutné použít anglicky psanou literaturu. Dalším místem, kde jsem čerpal informace byl internet a to hlavně z encyklopedie http://wikipedia.org. Jako další problém bylo vybrat vhodný operační systém pro přípravu testu. Požadavky na systém byly jednoduché, musel být v rámci finančních možností co nejlevnější a měl mít plnou podporu síťových aplikací. Tyto podmínky bezezbytku splňuje Linux a jeho distribuce Ubuntu. Hardwarové nároky byly splněny, protože jak OS, tak databázové systémy nejsou náročné v aplikacích, které se zde testovali. Pro běžnou praxi by, ale bylo zapotřebí přehodnotit dané parametry a některé části testovací sestavy pozměnit.

4.1. add 1.9.1. Ubuntu Linux 6.10

Ubuntu je distribuce Linuxu vyvíjená komunitou vývojářů za podpory firmy Canonical. Tato platforma je založena na spolehlivém základě Debian GNU, jedná se o jeho derivaci. Debian se snaží být universálním prostředím, kdežto Ubuntu se snaží přiblížit Linux běžným uživatelům. Ubuntu jako ostatně většina Linuxových platforem je šířena pod Open Source licencí.

Historie

První verze Ubuntu se objevila v roce 2004 s názvem Ubuntu 4.10. Tato číslovka znamená rok a měsíc vydání a stala se předlohou pro číslování dalších vývojových stupňů.

O rok později k nové verzi Ubuntu 5.04 vychází první vydání Kubuntu, které využívá KDE místo GNOME. Ve stejném roce vychází ještě verze 5.10 a sní Ebuntu, které mělo směřovat na trh vzdělání.

V roce 2006 vychází tento systém poprvé jako LST ve verzi 6.06. bylo vydáno na jednom „live“ CD a obsahovalo poprvé verzi pro server.

4.2. add 1.8 Jiné hardwarové řešení

V případě nasazení do praxe by dané řešení stačilo pro malou databázi. Limitující faktory u testovací sestavy PC je rychlost sítě, velikost disků a jejich rychlost. Pokud bude databáze využívána malým počtem uživatelů a uživatelé se budou připojovat pouze na několik málo operací, systém bude dostačující. V případě velkého množství uživatelů a větší pracovní vytíženosti se projeví nedostatky běžně dostupného hardwaru.

Jednou z možností jak tuto situaci vyřešit, je postavit páteřní síť na 1Gb Ethernetu. Toto řešení je nejlevnějším způsobem jak zrychlit práci s daty . Distribuovaný systém ve většině případů má na síť velké nároky. Management neustále kontroluje zda jsou ostatní části systému připraveny k použití a tím síť vytěžuje.

Dalším řešením pro zvýšení průchodnosti dat by bylo použití iSCSI, Fibre Channelu nebo SAN technologie. K těmto řešením je ale nutné podotknout, že jsou finančně náročné a je zapotřebí přizpůsobit těmto systémům i zbytek hardwaru, tzn. používat rychlejší disky nebo úložná externí centra.

V neposlední řadě je nutné zmínit se o procesorové náročnosti a náročnosti týkající se operační paměti. V prvním případě, za předpokladu, že jde o distribuovaný systém jsou požadavky rozloženy rovnoměrně mezi jednotlivé servery a jejich procesory a tím se problémy s nedostatky minimalizují. V druhém případě týkající se operační paměti je tento problém jednoduše a levně řešitelný. Systémy mají tendenci si načítat co nejvíce dat přímo do operační paměti, ale při jejím nedostatku systém vyhledává data v pevném úložišti. Je samozřejmě lepší, pokud má systém dostatek paměti na to, aby nejpoužívanější data měl přímo v paměti a nemusel tak prodlužovat dobu zpracování.

Použitá literatura

[1] Kolektiv autorů: Linux Dokumentační Projekt 3. vydání, Computer press Praha 2003

[2] Dokumentační projekt MySQL dostupný na http://www.mysql.com/

[3] Dokumentační projekt PostgreSQL dostupný na http://www.postgresql.org/

[4] http://wikipedia.org/ - přehled dostupných systémů

[5] Martin Kysela: Přecházíme na Linux, Computer press Praha

[6] Kevin Loney, Marlene Theriault: Mistrovství v Oracle, Computer press Praha 2002