VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

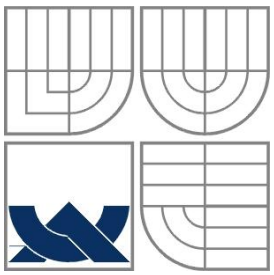
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

LADĚNÍ VÝKONNOSTI DATABÁZÍ

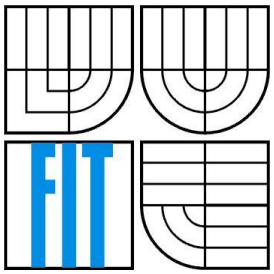
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Martin Paulíček
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

LADĚNÍ VÝKONNOSTI DATABÁZÍ

DATABASE PERFORMANCE TUNING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Martin Paulíček

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Petr Chmelař

BRNO 2011

Abstrakt

Úkolem této práce bylo seznámení se s nedostatečným výkonem databází a jejich možným zrychlením pomocí optimalizace konfigurace databáze, použití výkonnějšího hardwaru a paralelizace. Diplomová práce obsahuje popis relačních databází, úložných médií a různých forem paralelizace s jejich využitím v databázových systémech. Následuje popis implementace vytvořeného testovacího programu k ověření výkonnosti databáze. Program byl následně použit pro změření výkonu databáze PostgreSQL v jednotlivých testech. Testy byly zaměřeny na optimalizaci konfigurace databáze, porovnání více typů hardwarových konfigurací, porovnání různých databázových systémů (PostgreSQL, Oracle) a zrychlení databáze při použití paralelní metody „partitioning“. Jejich vyhodnocení je obsahem poslední části práce.

Abstract

The objective of this thesis was to study problems of an insufficient database processing performance and possibilities how to improve the performance with database configuration file optimizations, more powerful hardware and parallel processing. The master thesis contains a description of relational databases, storage media and different forms of parallelism with its use in database systems. There is a description of the developed software for testing database performance. The program was used for testing several database configuration files, various hardware, different database systems (PostgreSQL, Oracle) and advantages of parallel method “partitioning”. Test reports and evaluation results are described at the end of the thesis.

Klíčová slova

Relační databáze, ladění výkonu, optimalizace konfiguračního souboru PostgreSQL, pgbench zrychlení, škálovatelnost, paralelizace, paralelní zpracování, paralelní operace, paralelní architektury, úložná média, RAID

Keywords

Relational database, performance tuning, PostgreSQL database configuration file optimization, pgbench, speed up, scale up, parallelism, parallel database, parallel processing, parallel operations, parallel architectures, storage media, RAID

Citace

Paulíček Martin: Ladění výkonnosti databází, diplomová práce, Brno, FIT VUT v Brně, 2011

Ladění výkonnosti databází

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Chmelaře. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Bc. Martin Paulíček
18. 5. 2011

Poděkování

Rád bych zde poděkoval Ing. Petru Chmelaři za konzultace a odborné vedení při tvorbě této práce.

© Martin Paulíček, 2011
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Obsah 1
1 Úvod 3
2 Struktura relační databáze 4
2.1 Relace 4
2.2 Databázové schéma 4
2.3 Klíče 5
2.4 Základní operace 5
3 Ukládání dat 7
3.1 Přehled úložných médií 7
3.2 Hlavní paměť RAM 8
3.3 Magnetický disk 9
3.3.1 Fyzická charakteristika 9
3.3.2 Měření výkonu 10
3.4 RAID 11
3.4.1 Úrovně RAID 12
4 Měření paralelního zpracování 14
4.1 Zrychlení 14
4.2 Škálovatelnost 15
4.3 Problémy spojené s paralelním zpracováním 16
5 Formy paralelního zpracovaní 18
5.1 Interquery 18
5.2 Intraquery 18
5.3 Intraoperation 19
5.4 Interoperation 20
5.5 Smíšená paralelizace 20
6 Architektury paralelních databází 22
6.1 Architektura propojení sítě 22
6.2 Sdílení paměti 23
6.3 Sdílení disků 24
6.4 Bez sdílení 25
6.5 Hierarchická architektura 25
6.6 Distribuované databáze 26
6.7 Příklad paralelní databáze 27
7 Používané databázové systémy 29

7.1 PostgreSQL 29
7.2 Oracle 29
7.3 DB2 30
7.4 Microsoft SQL Server 30
7.5 MySQL 30
8 Měření výkonnosti databáze 31
8.1 TPS 31
8.2 TPC-B 31
8.3 JDBC TPC-B benchmark 32
8.4 pgbench 32
8.5 pgFouine 33
9 Testovací a vyhodnocovací program 35
9.1 Instalace a příprava testů 35
9.2 Testování 36
9.3 Zpracování výsledků 40
10 Testy a ladění výkonu 44
10.1 Testy a ladění konfigurace PostgreSQL 44
10.1.1 Konfigurační soubor PostgreSQL 44
10.1.2 Testování 46
10.2 Porovnání hardwarových konfigurací 48
10.2.1 Hardwarové konfigurace 48
10.2.2 Porovnání hardwarových konfigurací 49
10.3 Porovnání výkonu PostgreSQL a Oracle 51
10.4 Partitioning 52
11 Závěr 55
Literatura 56
Příloha A. Výsledky testů optimalizace serveru Minerva3 59
Příloha B. Výsledky testů srovnání hardwarových konfigurací 62
Příloha C. Vygenerovaný záznam výsledků testů serveru Minerva2 67
Příloha D. Instrukce k instalaci, nastavení a spuštění testů 70
Příloha E. CD s programem a výsledky testů 73

1 Úvod

Tato diplomová práce se zabývá tématikou rychlostí databází. Identifikuje problémy spojené s nedostatečným výkonem databáze a navrhuje možné řešení pomocí zrychlení hardwaru, optimalizace nastavení databáze a zavedení paralelizace. Dále popisuje výhody a problémy spojené s použitím těchto přístupů.

Vývoj počítačů a databázových systémů se od jejich vzniku nezastavil. Počítače od počátku již několikanásobně zrychlily výkon a tato tendence bude i nadále pokračovat v budoucnu. V roce 1965 definoval spoluzakladatel společnosti Intel Gordon Moore (Moorův) zákon o zvětšování počtu tranzistorů v závislosti na čase. Přesné znění jeho upraveného zákona říká, že počet tranzistorů v čipu se zdvojnásobí za každých 24 měsíců [3]. Jiná formulace tohoto zákona ukazuje, že disková kapacita se zvětšuje exponenciálně v závislosti na čase. I když se rychlost procesoru zvyšuje přibližně o 50% za rok, rychlost disků se zvětšuje jen o 8-10% [2]. Tento jev způsobuje vytváření tzv. úzkého hrdla (bottleneck) mezi vstupními a výstupními operacemi disku a rychlostí procesoru. Rychlost zpracování instrukcí v procesoru převyšuje rychlost operací s diskem, a tím vzniká problém s optimálním využitím diskového prostoru. Tato skutečnost se stala jednou z hlavních příčin výzkumu paralelních databází a technik propojení diskových polí. Dalšími důvody bylo například ukládání velkých objemů dat, získání rychlé doby odezvy, škálovatelnost, spolehlivost, vyvážení zatížení a dostupnost dat.

Od prvního nasazení databázových systémů se jejich složitost a objem nepřestal zvyšovat. Jedna z hlavních úloh databází je udržovat nízkou dobu odezvy na dotazy a operace s ní. Díky rostoucím databázím je tato úloha čím dál těžší a někdy se stává i nereálnou při použití jednoduchých databázových systémů. Proto vznikají nové metody, jak urychlit základní databázové systémy. Tyto metody se dělí na dvě hlavní oblasti podle typu urychlení databáze. První typ se zabývá zrychlováním velkoobjemových databází, a tedy pracuje s jednoduššími dotazy nad velkými objemy dat. Druhý typ se zaměřuje na zrychlení složitých dotazů, které jsou vykonávány nad menšími databázemi. Hlavními prostředky k dosažení vyšší rychlosti obou typů je použití výkonnějšího hardwaru, optimalizované databáze a paralelního zpracování. Kromě zrychlování výpočtu se vyvíjí lepší způsoby uložení dat, jak po fyzické stránce disku (kolmý zápis u magnetického disku, SSD disk), tak po technické stránce propojení více disků. Mimo rychlosti čtecích a zapisovacích operací je také nutné myslet na zabezpečení dat proti jejich ztrátě například poruchou hardwaru.

Diplomová práce začíná krátkým popisem základních principů relačních databází. Poté následuje kapitola zabývající se uložením dat. V kapitole jsou probrány různé typy úložných médií a také možnosti zrychlení a zabezpečení dat pomocí technologie RAID. Další kapitoly popisují možnosti paralelního zpracování v databázových systémech. Bude probráno měření dosaženého zrychlení, formy paralelních operací a různé paralelní architektury. Poté následuje představení dnešních nejpoužívanějších komerčních i open-source databází. Druhá část diplomové práce se věnuje testování a zrychlování operací na databázovém systému PostgreSQL. Nejdříve budou vysvětleny techniky k měření výkonnosti databáze. Poté bude popsán vytvořený testovací program pro testování a vyhodnocování rychlosti libovolné PostgreSQL databáze. Nakonec budou představeny provedené testy zabývající se zvýšením výkonu databáze pomocí různých technik. První z nich se zaměřují optimalizaci konfiguračního souboru databáze. Dále následuje srovnání výkonu rozdílných hardwarových konfigurací. Předposlední test ověřuje podobnost výkonnosti databází PostgreSQL a Oracle. Mezi další provedené testy patří zrychlení databázových operací pomocí implementace paralelního zpracování.

2 Struktura relační databáze

Pro správné pochopení problematiky optimalizace výkonů databázových systému je nutné popsat základní principy relačních databází. Právě na tento typ databází se zaměřuje tato diplomová práce. Následující části kapitoly popisují podle [4] základní strukturu relačních databází a schéma databáze. Dále jsou vysvětleny operace výběru (select), vkládání (insert), úpravy (update) a odstraňování (delete) dat spolu s principy transakčního zpracování.

2.1 Relace

Podle [4] je relační databáze tvořena kolekcí tabulek označovaných unikátním jménem. Řádek každé tabulky obsahuje vztah mezi množinou hodnot. Jelikož tabulka je kolekcí těchto vztahů, připomíná matematický koncept relace. Právě proto byl tento typ databáze pojmenován jako relační databáze. Každá tabulka obsahuje sloupce zvané atributy. Pro každý atribut existuje množina povolených hodnot zvaná doména atributu. Řádek tabulky poté reprezentuje vztah (relaci) určitých hodnot mezi všemi doménami. Řádek tvoří n-tici, kdy n je počet atributů tabulky. Prvky relace jsou tedy n-tice kartézského součinu všech domén. Jako příklad si uvedeme tabulku studentů s informacemi o jejich jménech, příjmeních, adrese a přihlašovacím jméně. Pro každý atribut si zavedeme doménu $D_{atribut}$ určující jeho přípustné hodnoty (vytvoříme tedy čtyři domény $D_{jméno}$, $D_{přijmení}$, D_{adresa} a $D_{přihlašovací\_jméno}$). Každý prvek relace bude čtveřice obsahující konkrétní hodnoty z těchto čtyř domén. Výsledná tabulka bude tvořit podmnožinu kartézského součinu všech domén: $Relace_{studenti} \subseteq D_{\text{jméno}} \text{ x } D_{\text{přijmení}} \text{ x } D_{\text{adresa}} \text{ x } D_{\text{přihlašovací\_jméno}}$. Obecně každá relace v relační databázi musí tvořit podmnožinu kartézského součinu domén: $Relace \subseteq D_1 \; x \; D_2 \; x \; \dots x \; D_{n-1} \; x \; D_n$ [4].

Prozatím nebylo specifikováno, jaké hodnoty mohou být prvky domén. V relačním modelu je specifikováno, že každá hodnota domény musí být atomická (setkáme se i s pojmem skalární). Tato podmínka zajišťuje, že hodnota vytváří nedělitelný celek. Pokud se vrátíme zpět k předešlému příkladu tabulky studentů a jejího atributu adresy, tak by ulice s popisným číslem a městem tvořilo již dále nedělitelnou hodnotu. Jestliže bychom chtěli adresu dále dělit, znamenalo by to, že se jedná o složený atribut. Tento atribut bychom mohli nahradit za atomické tři atributy adresy, popisného čísla a města. Pokud by student mohl mít více než jednu adresu, obsahem tohoto atributu by byly trojice (adresa, popisné číslo a město). Tyto trojice by tvořily zanořenou relaci. Takovýto atribut se poté nazývá vícehodnotový. Jestliže hodnoty všech domén relace jsou atomické, jedná se o normalizovanou relaci. Pokud při návrhu vzniknou relace se složenými nebo vícehodnotovými atributy, dostáváme relace nenormalizovanou. Tu poté převádíme pomocí pravidel na normalizovanou. Důvodem převodu je jednodušší práce s normalizovanými relacemi.

2.2 Databázové schéma

Když mluvíme o relačních databázích a relacích, musíme rozlišovat mezi databázovým schématem a instancí databáze. Databázové schéma popisuje logický návrh databáze. Instance reprezentuje databázi s relacemi a daty v určitý časový moment. Obecně se relační schéma skládá ze seznamu atributů a jejich korespondujících domén: $R(A_1 : D_1, A_2 : D_2, \dots, A_n : D_n), kde \; A_i(A_i \neq A_j \; pro \; i \neq$

$j)$ $je\ jméno\ atributu\ a\ D_i je\ jeho\ doména.$ Z definice vidíme, že atributy mohou mít stejnou doménu, ale vždy musí mít jednoznačné jméno. Při použití předchozího příkladu by schéma relace studenti vypadalo takto: $Studenti\_schéma = R(jméno, příjmení, adresa, přihlašovací\_jméno)$. Protože obsahem relace je množina n-tic, je relace neuspořádaná. Díky tomu také platí, že nejsou možné n-tice (řádky) se stejnými hodnotami v jedné relaci [4].

2.3 Klíče

Při práci s tabulkami potřebujeme způsob, kterým můžeme rozlišit prvky relace. Jak již bylo zmíněno v minulé podkapitole, žádná relace nesmí mít duplicitní n-tice. Znamená to tedy, že můžeme prvky rozlišit podle porovnání všech atributů. V praxi se ovšem setkáváme s atomickými či složenými atributy, které dokážou také jednoznačně rozlišit dvě n-tice. Takovýto atribut se nazývá superklíč. Všechny atributy spojené s tímto atributem se také stávají superklíčem. V relačních databázích se snažíme najít takové superklíče, pro které platí, že žádný z jejich podmnožin již není superklíč. Tyto klíče jsou poté nazývány kandidátními klíči. Příkladem kandidátního klíče relace studentů může být přihlašovací jméno nebo trojice (jméno, příjmení, adresa). Pro druhý kandidátní klíč předpokládáme, že dva studenti nemají zároveň stejné jméno a adresu.

Termínem primární klíč označujeme kandidátní klíč zvolený při návrhu relace jako atribut, který bude sloužit k jednoznačné identifikaci n-tice. Tento klíč nesmí být nikdy prázdný. Dále se také setkáme s pojmem cizí klíč. Jedná se o kandidátní klíč relace 1 uložený jako atribut relace 2. Tento atribut poté slouží k odkazu z relace 2 do relace 1 a musí splňovat následující dvě podmínky: Cizí klíč musí být buď plně zadán (každý atribut musí být vyplněn hodnotou) nebo úplně prázdný (každý atribut musí být prázdný). Dále je také nutné, aby každý cizí klíč v relaci 2 měl identické hodnoty v některé n-tici relace 1.

2.4 Základní operace

Při práci s relačními databázemi potřebujeme mít způsob na definování, manipulaci a dotazování relací. Pomocí operací definice spravujeme strukturu databáze. Mezi ty nejzákladnější operace pro manipulaci a dotazování patří vkládání, úprava, mazání n-tice a výběr n-tic podle určitých kritérií. V jazyce SQL jsou tyto operace nazývány INSERT pro vkládání, UPDATE pro změnu, DELETE pro mazání a SELECT pro výběr n-tic. Všechny operace dohromady jsou označovány zkratkou CRUD („Create Read Update Delete“).

Při práci s databází potřebujeme ve většině případů vykonat více základních operací najednou. Dále je nutné, aby se tato množina operací tvářila navenek jako jedna atomická operace. Toto seskupení operací se nazývá transakce. Pro zajištění integrity dat je důležité, aby transakce splňovala určité vlastnosti. Tyto vlastnosti se často označují zkratkou ACID podle prvních písmen čtyř vlastností: atomičnost („atomicity“), konzistence („consistency“), izolace („isolation“), trvalost („durability“).

Atomičnost zajišťuje, aby transakce byla vždy vykonána jako jedna jediná operace. Znamená to tedy, že všechny operace v transakci jsou buď úspěšně provedeny, nebo se žádná z nich neprovede. Tato vlastnost zabraňuje, aby při výskytu poruchy databázového systému se stav databáze dostal do nekonzistentního stavu. Konzistence zajištuje, aby databáze zůstala konzistentní po úspěšném provedení transakce. Tedy pokud je spuštěna právě jedna transakce a stav databáze byl před jejím

spuštěním v konzistentním stavu, musí byt dosažen tento stav i po dokončení této transakce. Hlavním faktorem ovlivňující tuto vlastnost je programátor implementující tuto transakci. On musí správně definovat všechny operace, aby jejich výsledkem byl opět konzistentní stav. Izolace je vlastnost definující možnost vykonávat více transakcí paralelně, tak aby byl dosažen stejný výsledek jako při provedení transakcí sekvenčně. Pokud by toto nebylo zajištěno, může se paralelním čtením a upravováním způsobit nekonzistence databáze. Trvalost zaručí, že po bezchybném dokončení transakce budou všechny změny provedené v databázi trvalé. Tedy i při chybě a pádu databázového systému nebudou tyto změny ztraceny.

V jazyce SQL jsou použity následující operace k dosažení transakčního zpracování. Pokud chceme transakci potvrdit, použijeme operaci „COMMIT“. Jestliže chceme operace provedené v rámci jedné transakce zrušit, zavoláme operaci „ROLLBACK“. Dále existuje pomocná operace „SAVEPOINT“, která slouží k relativnímu uložení pozice v rámci transakce. K této pozici je možné se poté vrátit pomocí operace „rollback“ (jedná se o částečný „rollback“). Pokud kdykoli při vykonávání transakce dojde k chybě, je automaticky použita operace „rollback“ k odstranění provedených změn a vrácení se k poslednímu konzistentnímu stavu databáze.

Při zpracování transakcí může dojít k jevu, kdy dvě transakce zároveň přistupují ke stejným datům. V tento moment musí existovat vykonávací plán, který je zpracován pomocí „řízení souběžného přístupu“ [4]. Takovýto plán zajišťuje, aby nebyla porušena konzistence databáze při vykonávání souběžných transakcí. Nejjednodušší plán vykonává transakce sekvenčně, a tedy nikdy nemůže dojít k porušení konzistence, pokud transakce splňují vlastnosti ACID. Bohužel sekvenční zpracování je velice časově náročné v porovnání s paralelním zpracováním, a proto vznikly techniky, které jsou schopny bezpečně zpracovávat paralelní transakce. Jakákoli operace v transakci vykonává buď zápis, nebo čtení v databázi. Existují jen dvě bezpečné kombinace těchto operací, aby nebyla porušena integrita dat. Bezpečné kombinace jsou, když obě transakce provádějí čtení nebo první transakce provede zápis a druhá poté čtení. Ostatní kombinace můžou vést k nekonzistenci databáze. Díky tomu je potřeba zavést techniky, které budou chránit před nedovolenými kombinacemi. Jednou z možností je technika uzamykání. Systém řízení báze dat uzamyká objekt ještě před tím, než k němu přistoupí transakce. Pokud je objekt zamčen, druhá transakce čeká na jeho odemčení, než bude moci k objektu přistoupit. Standard SQL-92 určuje tři možnosti porušení izolovanosti transakce. Následující obrázek 2.4.1 zobrazuje tyto možnosti a spolu s nimi definuje izolační úrovně, které lze v databázi zvolit. „Čtení nepotvrzené hodnoty“ znamená, že transakce má možnost přečíst nepotvrzenou hodnotu (hodnota, která nebyla potvrzena operací „commit“). „Neopakovatelné čtení“ umožnuje transakci po opakovaném čtení stejných dat načíst jinou hodnotu. Znamená to, že transakce uvidí hodnoty, které byly potvrzené mezi dvěma následnými operacemi čtení. „Fantomy“ značí, že transakce uvidí po opakovaném čtení více řádků (nové nebo zrušené řádky), které nebyly (byly) v prvním čtení. Z tabulky je patrné, že úroveň „READ UNCOMMITTED“ umožňuje porušení všech tří pravidel. Naproti tomu „SERIALIZABLE“ zabraňuje všem třem.

	čtení nepotvrzené hodnoty (dirty read)	neopakovatelné čtení (nonrepeatable read)	fantomy (phantoms)
READ UNCOMMITTED	ano	ano	ano
READ COMMITTED	ne	ano	ano
REPEATABLE READ	ne	ne	ano
SERIALIZABLE	ne	ne	ne

Obrázek 2.4.1 - Izolační úrovně v SQL [4]

3 Ukládání dat

Dnešní databázové systémy poskytují vysokou abstrakci při práci s daty, nicméně i tak se tato data musí fyzicky skladovat na jednom či více úložných zařízeních. Velká většina databází používá k uložení svých dat magnetické disky. Kromě nich nahrává data do hlavní paměti pro zpracování a zálohuje data překopírováním na pásky nebo jiné zálohovací zařízení. Fyzické vlastnosti úložných médií hrají při práci s daty velkou roli. Například náhodný přístup na disk je několikanásobně pomalejší než přístup do paměti. I různé typy disků mají odlišné rychlosti a poskytují jinou odolnost vůči chybám. Proto při návrhu databázového systému je velmi důležité správně zvolit paměťová média, na kterých bude systém fungovat.

V první části kapitoly budou představeny typy úložných médií. Následně bude probrána architektura RAID. Obě podkapitoly vychází z [1].

3.1 Přehled úložných médií

Ve většině počítačových systémů existuje několik typů úložných médií. Tyto úložiště jsou klasifikovány podle rychlosti přístupu k datům, podle ceny za datovou jednotku a podle spolehlivosti zařízení. Nyní budou uvedeny nejčastější typy datových médií seřazeny od nejrychlejších po nejpomalejší. Pro datová média také platí, že čím rychlejší médium je, tím má menší kapacitu. Seznam je proto seřazen i od nejmenší kapacity po největší. Paměti budou předvedeny v následujícím sledu: rychlá vyrovnávací paměť („cache“), hlavní paměť („main memory“), flash paměť („flash memory“), magnetický disk („magnetic-disk storage“) a páskové úložiště („tape storage“).

Systémová vyrovnávací paměť je nejrychlejším a zároveň nejdražším typem úložiště. Paměť je malá a její použití je řízeno hardwarem počítače. Je umístěna mezi procesorem a pamětí RAM, kde napomáhá rychlému přenosu dat. Vyrovnávací paměť je volatilním typem paměti. Znamená to, že po přerušení napájení je paměť smazána. Další typy vyrovnávacích pamětí najdeme například u pevných disků nebo řadiče diskových polí RAID. Tyto paměti lze již řídit pomocí softwaru.

Hlavní paměť slouží jako médium pro data, s kterými se bude pracovat. Pro optimální výkon systému je nutné tuto paměť dobře spravovat. Pokud se již data v paměti nebudou používat, je nutné paměť uvolnit pro budoucí použití. Správu hlavní paměti má za úkol operační systém počítače. V dnešní době může hlavní paměť dosahovat mnoha gigabajtů (dokonce až stovky GB pro velké serverové systémy), ale i tak je to ve většině případů málo pro uložení celé databáze. Hlavní paměť patří do typu volatilních pamětí, a proto po přerušení napájení nebo po pádu systému je obsah paměti ztracen. Podrobněji bude popsána v kapitole 3.2.

Oproti hlavní paměti flash paměť udržuje data i po vypnutí napájení. Jedná se tedy o nevolatilní paměť. Čtení dat z paměti trvá méně než 100 nanosekund, což je přibližně stejně jako rychlost hlavní paměti. Bohužel zápis do flash paměti je daleko komplikovanější. Data mohou být zapsána do paměti jen jednou, kdy doba zápisu se pohybuje od 4 do 10 mikrosekund. Tato data není možné hned přepsat, je nutné nejdříve smazat část paměti, v které jsou data uložena a poté je možné zapsat na toto místo znovu. Nevýhodou flash pamětí je omezený počet mazání a tedy i zápisů do paměti sahající od deseti tisíc k jednomu milionu [1]. Flash paměť je formou elektricky mazatelné nevolatilní pamětí (EEPROM). Flash paměti našly využití v nahrazení magnetických disku pro skladování menších objemů dat v řádu jednotek až desítek gigabajtů. Dále se rozšířily do ostatních zařízení, jako jsou fotoaparáty, telefony a další. V dnešní době se začínají využívat flash paměti jako

úplná náhrada magnetických disků. Tyto disky se nazývají SSD (Solid State Drive) a na rozdíl od klasických magnetických disků neobsahují žádné mechanické části a mají mnohem menší spotřebu. Největší výhodou těchto disků je rychlost čtení, která výrazně přesahuje rychlosti magnetických disků. Nevýhodou je jejich vysoká cena a omezený počet mazání datových bloků potřebných k jakémukoli zápisu dat.

Magnetické disky slouží jako primární médium spolu se SSD pro dlouhodobé uložení dat. Většinou celá databáze systému je uložena na jednom nebo více těchto disků. Pro operace s daty uloženými na tomto médiu je nutné data nejdříve zkopírovat do hlavní paměti. Po dokončení jsou data opět zapsána na disk. Velikost dnešních disků se pohybuje od 250 GB do 4 TB (platné pro rok 2011 [9]). Každý rok se zvětší jejich velikost zhruba o 50 procent. Disky jsou odolné vůči výpadkům napětí nebo selhání systému. Pokud nastane chyba v samotném disku, znamená to ve většině případů ztrátu veškerých dat. Charakteristiky pevného disk jsou podrobněji popsány v kapitole 3.3. Možné zajištění odolnosti vůči chybám je uvedeno v kapitole 3.4.

Posledním typem jsou páskové úložiště. Jejich hlavním účelem je zálohování velkých objemů dat. Magnetické pásky jsou levnější než magnetické disky, bohužel jejich doba přístupu k datům je velice pomalá. Důvodem je sekvenční přístup, díky kterému veškeré nesekvenční operace s daty jsou hodně zdlouhavé. Pásky nabízí největší možný úložný prostor ze všech typů médií. Většina velkých databázových systémů používá pásky jako zálohovací zařízení.

V kapitole nebyla zmíněna optická úložiště dat, jako jsou CD, DVD nebo BD. Tyto typy paměti nejsou využívány v databázových systémech. Většinou jde o paměti s jediným možným zápisem, a tedy jejich využití by mohlo být jen formou zálohy. Na to ovšem mají optické disky nedostatečnou kapacitu. V budoucnosti je možné nasazení optických médií jako zálohovacích zařízení díky holografické technologii zápisu dat. Ta již nezapisuje informace jen na povrch média, ale používá více vrstev a tím výrazně zvětšuje kapacitu úložiště.

3.2 Hlavní paměť RAM

Jak už bylo výše zmíněno, hlavní paměť slouží k uchovávání dat, se kterými se bude pracovat. Pokud se zaměříme na databázové systémy, tak systém řízení báze dat nikdy nepřistupuje přímo na disk při práci s daty. Využívá se vyrovnávací paměti, do které se krátkodobě ukládají bloky dat. Databázový systém využívá jak vyrovnávací paměť operačního systému, tak si vytváří a spravuje vlastní vyrovnávací paměť. Vyrovnávací paměť operačního systému je použita při čtecích a zapisovacích diskových operacích. Důvodem vytváření vlastní paměti jsou specifické nároky, které musí paměť splňovat. Mezi ně patří podle [4] „co nejefektivnější algoritmus uvolňování prostoru ve vyrovnávací paměti“, „omezení času, kdy lze zapsat blok z vyrovnávací paměti na disk“, „možnost vynuceného zápisu všech modifikovaných bloků na disk“ a „možnost označit některé bloky, které by pokud možno měly být trvale ve vyrovnávací paměti“. Ve většině případů není možné nahrát do paměti všechny bloky dat, se kterými bude databáze pracovat. Právě proto je nutné používat efektivní uvolňování paměti s nepoužívanými bloky dat. Z důvodu konzistence dat je nutné při některých operacích (jedná se o operace potvrzení nebo zrušení transakce popsané v kapitole 2.4) provést následný zápis na disk. Tato vlastnost chrání databázi při výskytu chyb a závad před porušením konzistentního stavu a integrity dat. Pro zrychlení vykonávání databázových operací se také často využívá trvalé nahrání bloků dat do paměti. Jde například o indexy nebo systémové katalogy. Databázový systém si tedy primárně vytváří vlastní vyrovnávací paměť pro urychlení vykonávaných operací, udržení konzistentního stavu databáze a uchování potvrzených dat i při výskytu poruchy.

3.3 Magnetický disk

Magnetické disky zajišťují většinu dnešních sekundárních úložišť databázových systémů. Jejich kapacita roste rok od roku. Spolu s tímto růstem se zvětšují prostorové požadavky velkých aplikací. Některým z nich roste potřeba většího úložiště ještě rychleji, než je růst kapacity magnetických disků. Aplikace s rozsáhlými databázemi mohou obsahovat až stovky propojených disků.

Magnetické disky a jejich vlastnosti jsou velice důležité pro databázové systémy, právě proto je kladen velký důraz na jejich popis a správné pochopení. Správným databázovým nastavením magnetických disků můžeme velice ovlivnit jejich výkon jak pozitivně tak negativně. Nyní bude přesněji popsána podle [1] struktura magnetického disku a prvky určující jeho výkon.

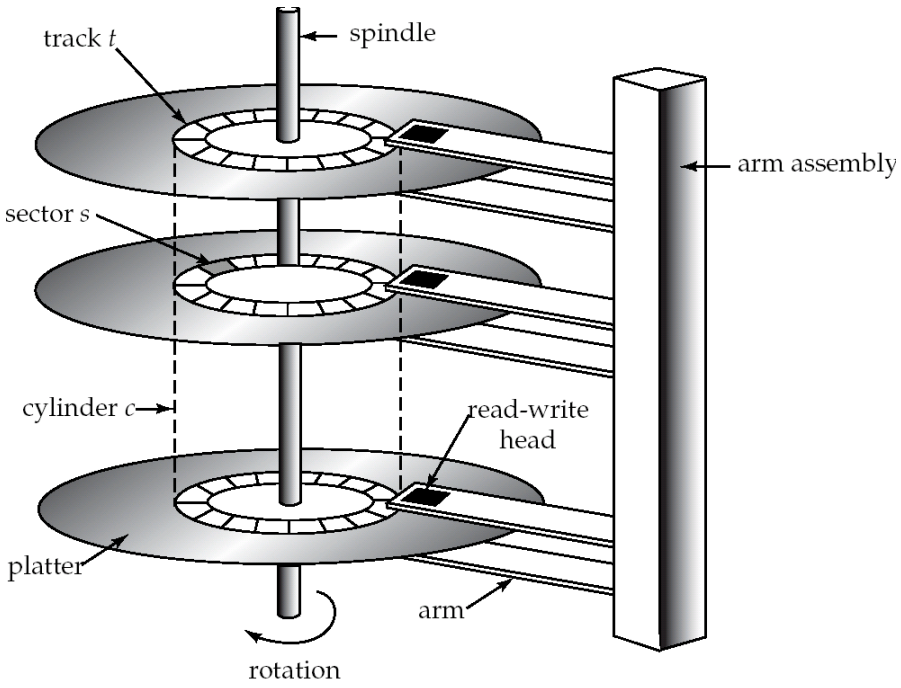
3.3.1 Fyzická charakteristika

Po fyzické stránce jsou disky relativně jednoduché. Každý disk se skládá z ploten („platter“) a čtecích a zapisovacích hlav („read-write head“). Každá plotna má plochý kruhový tvar a na svém povrchu obsahuje magnetický materiál. Ten slouží k zápisu a uchování informací. Magnetický disk obvykle obsahuje jednu až pět ploten uložených nad sebou do válce. Při spuštění disku se plotny automaticky roztočí na konstantní rychlost (nejčastěji 5200, 7200 a u serverových disků až 15000 otáček/minutu [1]). Rychlost otáčení z velké části ovlivňuje rychlost celého disku. Plotna obsahuje stopy („track“), které se dělí na sektory („sector“). Díky kruhovitému tvaru se délka stop (i počet sektorů) zvětšuje od středu po okraj. Sektor tvoří nejmenší datovou jednotku disku, jeho velikost je typicky 512 bytů [1]. Každá disková hlava zapisuje data do sektoru pomocí změny magnetické polarizace. Každá plotna obsahuje z obou stran čtecí i zapisovací hlavu, ty jsou připevněny na společné rameno disku. Toto rameno tedy pohybuje všemi hlavami najednou. Disky se dále dělí podle poloměru délky plotny (nejčastěji se setkáme s disky velikosti 3,5 palce pro stanice a 2,5 palce pro přenosné počítače [1]).

Celé schéma disku je možné vidět na následujícím obrázku 3.3.1. Uprostřed ploten je společná hřídel („spindle“). U středu plotny je zobrazena stopa („track“). Stopa je rozdělena na sektory („sector“). Všechny data, která jsou přístupná bez pohybu hlavy, se nazývají válec („cylinder“). Napravo je zakresleno společné rameno („arm assembly“) nesoucí čtecí a zapisovací hlavy („read-write head“).

Pevné disky jsou propojeny s počítačem přes vysokorychlostní sběrnice. V dnešní době existuje více typů sběrnic, běžně používané rozhraní pro interní disky jsou ATA („AT attachment“), SATA („serial ATA“) a SCSI („Small Computer System Interface“). Pro připojení externího disku se používá rozhraní USB a FireWire. Rychlost rozhraní podporují přenos až 133 MB/s pro rozhraní ATA, až 300 MB/s pro SATA II, až 640 MB/s pro Ultra640 SCSI, až 625 MB/s pro USB 3.0 a až 393 MB/s pro FireWire 3200 [7]. Podle [8] vlastnosti rozhraní ovlivňují rychlosti disků jen z velice malé části. Hlavním faktorem je mechanika, materiály, elektronika a firmware pevných disků. Právě proto se můžeme setkat s použitím různých rozhraní jak pro osobní tak i firemní účely.

Databázové systémy mají velké prostorové požadavky, kdy nestačí propojení několika pevných disků. Často se proto setkáme s použitím diskových polí. Pro lokální disková pole se využívá technologie RAID (tato technologie bude popsána v kapitole 3.4). Kromě lokálních disků můžeme využít disků přístupných přes vysokorychlostní síťové rozhraní pomocí architektury SAN („storage area network“). SAN zobrazuje diskové pole jako jeden velký disk. Alternativou k SAN architektuře je NAS („Network attached storage“). NAS zprostředkovává oproti SAN síťový souborový systém NFS nebo CIFS.



Obrázek 3.3.1 - Schéma magnetického disku [1]

3.3.2 Měření výkonu

Hlavními parametry magnetického disku určující velikost a výkon jsou kapacita, přístupová doba, přenosová rychlost a spolehlivost. Kapacita disků se každý rok zvětšuje, tento rok se můžeme setkat s velikostmi až 4 TB [9].

Přístupová doba je čas od obdržení dotazu (čtení nebo zápisu) po přístup k datům a počátek datového přenosu. Pro přístup k určitému sektoru je potřeba nejdříve nastavit rameno s čtecí hlavou nad požadovanou stopu a poté počkat na otočení plotny na místo se sektorem. Doba pro přesunutí ramena se nazývá doba vystavení („seek time“). Obvyklá hodnota se pohybuje mezi 2 až 30 milisekundami. Doba otočení plotny na požadovanou pozici se nazývá doba čekání („rotational latency time“). Tato doba závisí na rychlosti otáčení disku. Například pro disk s 5400 otáčkami za minutu trvá jedno celé otočení 11,1 milisekund. Doba otočení se poté pohybuje podle relativní pozice plotny od 0 po 11,1 milisekund. Přístupová doba se tedy vypočítá sečtením doby vystavení a doby čekání. Obvykle se pohybuje od 8 do 20 milisekund [1].

Po přístupu k prvnímu sektoru s daty začíná jejich přenos (na disku se zapisuje nebo čte podle požadované operace). Rychlost přenosu („data-transfer rate“) je poměr množství zapsaných nebo přečtených dat za určitý časový interval. Dnešní pevné disky nabízejí maximální rychlost přenosu mezi 25 až 100 MB za sekundu. Průměrná rychlost přenosu je ovšem daleko nižší.

Posledním a velice důležitým parametrem je spolehlivost disku. Ta je uvedena v průměrné době do poruchy („mean time to failure“). Tato hodnota uvádí průměrnou dobu, po kterou můžeme očekávat, že disk bude fungovat bezchybně. U dnešních disků se doba pohybuje od 500 po 1200 tisíc hodin (odpovídá 57 až 136 let). Pokud máme ovšem 1000 nových disků, pravděpodobnost, že se jeden pokazí, je již jen 1200 hodin. Předpokládaná doba života většiny disků se pohybuje okolo 5 let. Po této době se velice zvyšuje riziko jejich selhání. K dosažení vyšší spolehlivosti se používají různé typy zapojení disků pomocí technologie RAID (podrobněji popsáno v následující kapitole 3.4).

3.4 RAID

V dnešní době je nutné použít více disků pro uložení aplikací, jako jsou například webové aplikace, databáze a multimediální aplikace. I přes růst kapacit disků již není možné používat jeden disk na skladování všech dat. Díky většímu počtu disků je možné dosáhnout lepšího výkonu zápisu nebo čtení pomocí zavedení paralelních operací. Při použití jednoho disku je nutné provádět operace sekvenčně, pokud máme disků více, je možné, aby čtení nebo zápis byl prováděn paralelně. Větší množství disků nenabízí jen možnosti urychlení jejich operací, ale také představuje možnost zlepšení spolehlivosti při zavedení redundantních dat. Ty poté chrání data před jejich zničením při selhání některého disku. Různé možnosti zapojení více disků dohromady se nazývá „vícenásobné pole nezávislých disků“ („Redundant Array of Independent Disks“). V minulosti sloužilo spojování více menších levných disků k dosažení nižší koncové ceny. Ceny velkých disků převyšovaly ceny spojených kapacit menších disků. Dnes jsou již disky levné a technologie RAID se používá výhradně k zvýšení spolehlivosti a rychlosti.

Pokud se budeme zabývat spolehlivostí disků, je nutné si spočítat, jak pravděpodobné je jejich selhání. Pravděpodobnost, že selže jeden disk z N použitých, je daleko větší než, že selže jeden předem specifikovaný disk. Předpokládejme, že průměrná doba do selhání jednoho disku je 100 000 hodin od jeho spuštění. Potom průměrný čas do selhání disku v poli 100 disků bude 100 000/100, což je rovno 1 000 hodinám (1 000 hodin odpovídá přibližně 42 dnům) [1]. Takto velká frekvence selhání disků je neúnosná, a proto jsou zavedena opatření, která zabraňují před ztrátou dat při poruše. Řešením problému je zavedení redundance. To znamená, že zavedeme více informací, které budou při chybě sloužit k obnovení dat. Nejjednodušší a zároveň nejdražší metodou je zrcadlení („mirroring“), kdy se vytváří přesná kopie disků. Logický disk obsahuje dva fyzické, zápis na něj vytvoří zápis na oba fyzické disky. Pokud jeden disk selže, může se pokračovat s prací na druhém disku. Data jsou ztracena jen v případě, pokud nastane chyba u obou disků zároveň. Zrcadlené disky poskytují daleko větší spolehlivost než použití jen jednoho disku (průměrná doba do poruchy se pohybuje od 500 000 do 1 000 000 hodin). Dalším typem redundance je zavedení paritních bitů nebo bloků dat. Ty se využívají při použití více disků. Jejich výhodou je menší potřebný prostor oproti metodě zrcadlení. Naproti tomu mají nevýhodu při poruše, kdy je nutné vyměnit disk a znovu dopočítat data, než bude možné s disky znovu pracovat.

Dále budou popsány výhody paralelního přístupu k více diskům. Při zrcadlení disků dosahujeme dvojnásobné rychlosti u zpracování požadavků ke čtení. Je to dáno zasíláním dotazů buď na jeden, nebo na druhý disk. Rychlost čtení z disků je stejná jako u systému s jedním diskem, ale počet čtení za časový interval se zdvojnásobí. S přidáním více disků můžeme vylepšit i rychlost přenosu dat pomocí zavedení prokládání dat („striping“) mezi více disků. Nejjednodušší forma prokládání dat je založena na rozdělení bitů na jednotlivé disky („bit-level striping“). Tento typ prokládání vyžaduje, aby počet disků byl násobek osmi nebo faktorem čísla osm. Pokud použijeme pole o čtyřech discích, budou bity *i* a *4 + i* každého bajtu uloženy na stejný disk. Složitější variantou prokládání je prokládání po blocích („block-level striping“). Pole n disků je chápáno jako jeden velký logický disk. Logický blok *i* bude uložen na disk *(n mod i) + 1*. Zrychlení čtení velkého objemu dat je n-násobné. Při čtení jednoho bloku zůstává rychlost čtení stejná jako u systému s jedním diskem, ale ostatní disky můžou již vykonávat jiné akce. Prokládání po blocích je nejčastěji používaným typem. Ostatní typy prokládání po bajtech, sektorech nebo blocích sektorů jsou také možné. Zrychlení disků se zabývá dvěma hlavními cíli. Prvním je účinné rozložení zatížení při více přístupech k menším

objemům dat, tak aby se zvýšila propustnost. Jako druhý cíl je paralelizace přístupů k velkým objemům dat, tak aby se snížila doba odezvy.

3.4.1 Úrovně RAID

Zrcadlení nabízí vysokou spolehlivost s nevýhodou vysoké ceny. Naproti tomu prokládání dat zajišťuje vysokou přenosovou rychlost, ale neovlivňuje spolehlivost. Existuje mnoho alternativních schémat, které kombinují obě techniky k získání různých poměrů ceny, spolehlivosti a rychlosti. Tyto konfigurace se nazývají RAID úrovně. V dnešní době se můžeme setkat s následujícími úrovněmi RAID:

- **RAID 0** – Diskové pole s prokládáním bloků dat. Zápis bloků se střídá mezi disky. Není zajištěna žádná redundance. Schéma je zobrazeno na obrázku 3.4.1.

Obrázek 3.4.1 - RAID 0 [1]

- **RAID 1** – Diskové pole zrcadlí bloky dat. Disky jsou identické. Schéma je zobrazeno na obrázku 3.4.2.

Obrázek 3.4.2 - RAID 1 [1]

- **RAID 2** – Diskové pole s prokládáním bitů a s přidáním redundance v podobě ECC („error correction code“). Vychází z opravného kódu ECC v paměťových systémech. Data jsou postupně ukládána po bitech na disky. Vždy jsou také vypočítány paritní bity, které se ukládají na oddělené disky. Pokud disk selže, je možné dopočítat data chybného disku a chybu opravit. Obrázek 3.4.3 zobrazuje čtyři disky s daty a tři disky s paritou (disky označené písmenem P).

Obrázek 3.4.3 - RAID 2 [1]

- **RAID 3** – Diskové pole podobné úrovni RAID 2. Zde se již používá jen jeden fyzický disk na ukládání paritních bitů. Tato úroveň je stejně dobrá jako RAID 2 a zároveň levnější díky použití jen jednoho paritního disku. Obrázek 3.4.4 zobrazuje čtyři disky s daty a jeden disk s paritou.

Obrázek 3.4.4 - RAID 3 [1]

- **RAID 4** – Diskové pole s prokládáním bloků dat s použitím paritního bloku. Opět je parita ukládána na zvláštním disku a při chybě je možné data dopočítat. Čtení bloku je pomalejší, protože se čte vždy z jednoho disku. Celkové čtení velkých dat je daleko rychlejší, protože dochází k paralelnímu čtení více disků zároveň. Oproti tomu zápis jednoho bloku nemůže být

prováděn paralelně kvůli výpočtu a zápisu paritního bloku. Obrázek 3.4.5 zobrazuje čtyři disky s daty a jeden disk s paritou.

Obrázek 3.4.5 - RAID 4 [1]

- **RAID 5** – Diskové pole s prokládáním bloků dat s použitím paritního bloku. Tato úroveň vylepšuje RAID 4, protože paritní blok se již neukládá na zvláštní fyzický disk, ale postupně se ukládá na všechny datové disky. Oproti RAID 4 všechny disky mohou zpracovávat požadavky ke čtení a zápisu. RAID 5 je tedy výhodnější než RAID 4 a přitom má stejnou cenu. Díky tomu se v reálném prostředí RAID 4 nevyužívá. Obrázek 3.4.6 zobrazuje pět disků s paritou na různých discích.

Obrázek 3.4.6 - RAID 5 [1]

- **RAID 6** – Diskové pole s prokládáním bloků dat s použitím dvou paritních bloků. Úroveň přidává jeden paritní blok navíc oproti RAID 5, aby zabránila ztrátě dat při chybě dvou disků najednou. Obrázek 3.4.7 zobrazuje šest disků. Při zápisu se vždy na dva disky uloží paritní bloky.

Obrázek 3.4.7 - RAID 6 [1]

4 Měření paralelního zpracování

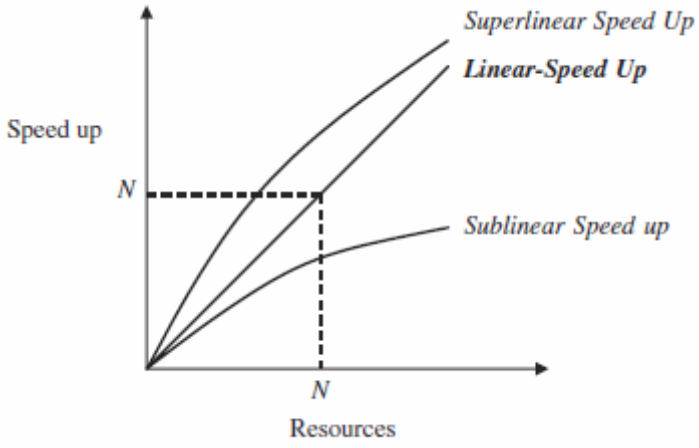
Hlavním cílem paralelních databází je zrychlení jejich výkonu. K jejímu měření slouží dvě metody. První se jmenuje „zrychlení“ (tzv. „speed up“) a měří propustnost. To je stanoveno pomocí poměru počtu dokončených úloh k určitému časovému intervalu. Druhá metoda je pojmenována „škálovatelnost“ (tzv. „scale up“) a měří dobu odezvy. Ta odpovídá délce času, která je zapotřebí k dokončení jediné úlohy. V následujících dvou podkapitolách budou podrobněji popsány obě metody podle [2]. Poté bude následovat shrnutí problémů spojených s použitím paralelního přístupu.

4.1 Zrychlení

Zrychlení („speed up“) odpovídá zkrácení času vykonávání úlohy přidáním větší výpočetní síly. Při použití paralelismu to znamená, že určitá úloha se bude zpracovávat o to rychleji, o kolik se zvedne stupeň paralelismu. Zrychlení slouží typicky pro měření dotazů, které jsou jen pro čtení („read-only“). Hodnotu zrychlení můžeme vypočítat pomocí následujícího vzorce:

$$Zrychlení = \frac{čas\ výpočtu\ na\ jednom\ procesoru}{čas\ výpočtu\ na\ více\ procesorech}$$ [2]

Existují tři typy zrychlení podle poměru zrychlování k zvětšování systému. První typ lineární zrychlení odpovídá lineárnímu zlepšení výkonu při stejném zvětšování sytému. Pokud se systém rozšíří o N, musíme zrychlit jeho zpracování také o N, abychom dosáhli lineárního zrychlení. Tohoto typu zrychlení se snažíme dosáhnout, protože potřebujeme nejméně stejnou výkonnost, jako při počáteční velikosti databáze. Pokud zrychlení bude menší než N, bude se jednat o sublineární zrychlení (sublinear speed up). Nejlepší varianta vzniká při zrychlení větším než N, pak se dosahuje superlineární zrychlení (superlinear speed up). Tento typ je bohužel jen zřídka dosažitelný. Většinou k jejímu dosažení je využita speciální vlastnost paralelního zpracování (například větší paměť přístupná díky použití více procesorů). Na následujícím obrázku 4.1.1 je možné vidět všechny tři typy zrychlení v závislosti na velikosti systému (počet procesorů, disků, …) zobrazené na horizontální ose a zvýšení rychlosti na vertikální ose.



Obrázek 4.1.1 – Zrychlení (převzato z [2])

K lepšímu pochopení rozdělení těchto tří typů si uvedeme následující příklad. Předpokládejme, že vykonání úlohy na jednom procesoru zabere 100 sekund. Pokud k výpočtu použijeme 5 procesorů

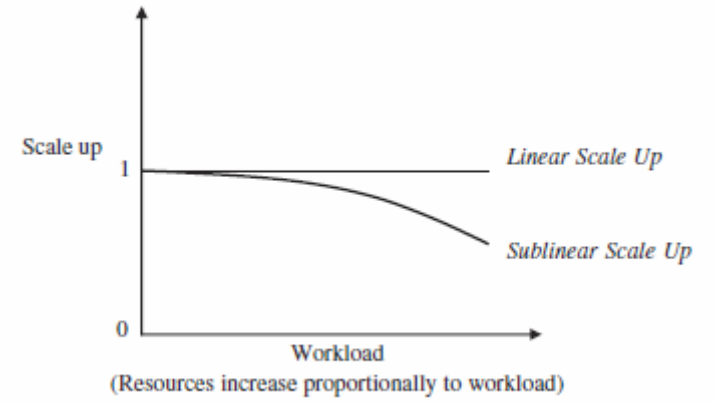
a výpočet bude trvat 20 sekund, zrychlení bude rovno 5. Protože jsme využili 5 procesorů a dosáhli zrychlení 5, jedná se o lineární zrychlení. Kdyby délka výpočtu trvala 33 sekund, zrychlení by bylo přibližně 3. Díky tomu by bylo zrychlení menší než je počet procesorů a dostali bychom sublineárnímu zrychlení („sublinear speed up“). Jestliže by ovšem výpočet trval méně než 20 sekund (například 16.5 sekund), pak bychom se dostali zrychlení 6. Díky tomu by bylo zrychlení větší než počet procesorů a tedy v grafu bychom se ocitli na superlineárním zrychlení [2].

4.2 Škálovatelnost

Pro vlastnost „Scale up“ platí, že výkon paralelní databáze je zachován, pokud počet zdrojů (procesorů, disků,…) je zvětšen stejnou mírou jako objem dat v databázi. Škálovatelnost určuje tedy schopnost zpracovávat větší úlohy pomocí větší úrovně paralelismu (přidáním více zdrojů) za stejný časový interval jako původní systém. U většiny systémů takto testujeme, zda po přidání výpočetní síly jsme schopni udržet původní výpočetní výkon s nárůstem objemu dat. Tento výpočet se často používá u transakčních systémů manipulujících s daty. Výpočet spočítáme podle následujícího vzorce:

$$Škálovatelnost = \frac{uplynulý\ čas\ při\ zpracování\ jedním\ procesorem\ na\ malém\ systému}{uplynulý\ čas\ při\ zpracování\ více\ procesory\ na\ větším\ systému}$$ [2]

Jako u zrychlení se i zde setkáme s lineární škálovatelností. Ta určuje schopnost zachovat stejný výkon, když je rovnoměrně zvýšena zátěž spolu s výpočetní silou. Podle výše uvedeného vzorce vidíme, že potřebujeme škálovatelnost rovnu jedné, pokud chceme dosáhnout lineární škálovatelnosti. V případě, že dostaneme číslo menší než jedna, bude se jednat o sublineární škálovatelnost. Superlineární škálovatelnost je velice vzácná, a proto ji není potřeba brát v úvahu. Dosáhnutí jen lineární představuje nejvyšší cíl v paralelních databázových systémech. Následující obrázek 4.2.1 zobrazuje lineární a sublineární škálovatelnost.



Obrázek 4.2.1 – škálovatelnost (převzato z [2])

Škálovatelnost můžeme dále rozdělit na dva typy podle zvoleného měření. Prvním typem je transakční škálovatelnost, kdy se počítá nárůst počtu transakcí za určitý časový interval. Druhým typem je datová škálovatelnost, která vychází ze zvětšení objemu databáze a tedy i velikosti zpracovávané úlohy.

Nyní jsou představeny příklady transakčního a poté i datového typu. Původní systém vykonává 100 instrukcí za 10 sekund. Pak jestliže by paralelní systém o dvojnásobném hardwaru zpracoval 200 instrukcí za stejnou dobu jako původní systém, byl by tento systém lineární. Kdyby byl uplynulý čas u paralelního systému delší, dostali bychom sublineární škálovatelnost. S datovou škálovatelností se

setkáváme u OLAP databází, kde tabulka informací je o mnoho rozsáhlejší než všechny ostatní tabulky obsahující dimenze dohromady. Pokud by úloha nad touto tabulkou zabrala 1 procesoru 1 hodinu, pak pro zachování lineární škálovatelnosti by musel systém zpracovat dvojnásobnou tabulku s 2 procesory za stejný čas. Při času přesahujícímu 1 hodinu bychom získali sublineární škálovatelnost [2].

4.3 Problémy spojené s paralelním zpracováním

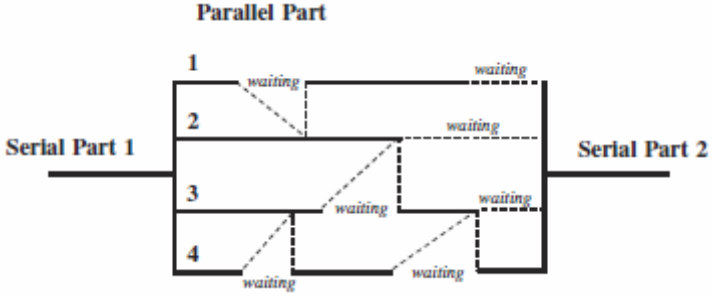
Při nasazení paralelizace se setkáme s řadou faktorů, které mohou negativně ovlivnit jak zrychlení, tak škálovatelnost. [2] udává následující typy problémů: cena spuštění a konsolidace výsledků, soupeření o přístup k datům, komunikace mezi procesy a rozložení zatížení.

Při spouštění více procesů u paralelních operací můžeme někdy čelit problému, kdy doba spuštění všech procesů převažuje nad samotnou dobou zpracování. Může se jednat například o spuštění tisíce procesů. Zpomalení může nastat i při malém počtu procesů, pokud výpočetní doba je dostatečně krátká. Tento jev ovlivňuje zejména „zrychlení“. Dalším z faktorů ovlivňující výsledný čas je cena konsolidace výsledků, která odpovídá době potřebné ke shromáždění všech výsledků z paralelních operací. Tato hodnota také často způsobuje překážku k dosažení lineárního zrychlení. Obě tyto části nelze vylepšit pomocí paralelizace, protože se jedná o úlohy, které vykonává vždy jeden, nejčastěji hostitelský, proces. Výpočet pomocí paralelního zpracování vždy začíná určitým procesem, který vytvoří dílčí podúlohy pro nově vznikající procesy. Po dokončení těchto výpočtů je zahájen sběr všech dat obvykle pomocí počátečního procesu. Díky tomu i tato část ovlivňuje výsledné zrychlení celého procesu. Z výše uvedeného popisu vyplývá, že u obou těchto úseků se jedná o sekvenční úlohy dokládající pravdivost Amdahlova zákona [5]. Ten uvádí, že výpočetní čas se dá rozdělit na sekvenční a paralelní část. I když budeme zvyšovat stupeň paralelizace, vždy bude výsledný čas shora omezen sekvenční částí vykonávanou jedním prvkem.

Jelikož paralelní systém často přistupuje ke sdíleným zdrojům, dochází v mnoha případech ke kolizi při přístupu ke stejným datům. V ten moment spolu začnou procesy soupeřit o právo k přístupu. Jev soupeření zpomaluje jak zrychlení tak škálovatelnost.

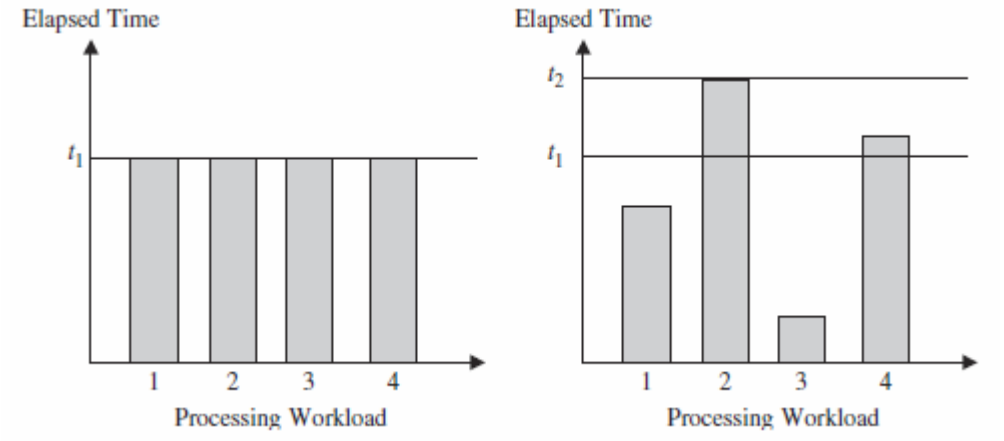
Častá je také vnitřní komunikace mezi procesy. Díky synchronizovanému prostředí je proces nucen čekat na stav, kdy budou ostatní procesy připravené ke komunikaci. Tento časový úsek může způsobovat nečinnost mnoha procesů a zpomalovat celkový čas výpočtu.

Na následujícím obrázku 4.3.1 je zobrazeno vykonávání úlohy pomocí paralelního zpracování. Úloha je rozdělena na sériovou část („Serial Part 1 a 2“) a paralelní část („Parallel Part“). Paralelní část úlohy zobrazuje komunikaci a kolize mezi procesy. Čárkovaná čára představuje čekání („waiting“) jednoho procesu na dosažení určitého bodu dalšího procesu k možnému pokračování ve vykonávání. Tato čekací doba odpovídá čekání na komunikaci nebo na uvolnění sdílených dat.



Obrázek 4.3.1 – paralelní zpracování úlohy [2]

Posledním faktorem ovlivňujícím paralelní výkon je rozložení zatížení mezi procesy. Správné vyvážení je jedním z hlavních prostředků jak dosáhnout lineárního zrychlení. Kromě rovnoměrného rozložení zátěže je důležité mít také rovnoměrně rozložena data v databázi, pokud je použit partitioning. Partitioning je metoda rozdělující data do více oddílů, které poté můžou být použity k paralelnímu zpracování [2]. Podle [1] může vzniknout nerovnoměrné rozdělení jak v hodnotách atributů, tak i v celých oddílech při použití špatné rozdělovací techniky. Na obrázku 4.3.2 je zobrazen rozdíl mezi rovnoměrně (část vlevo) a nerovnoměrně (část vpravo) rozděleným zatížením. Vertikální osa určuje uplynutou dobu („Elapsed Time“). Horizontální osa znázorňuje zátěž každého procesu („Processing Workload“). Pokud bychom nahradili horizontální osu za četnost prvků v oddílu a horizontální osu za jednotlivé oddíly, dostali bychom grafy pro (ne)rovnoměrné rozložení dat v oddílech.



Obrázek 4.3.2 – Rovnoměrné a nerovnoměrné rozložení zatížení [2]

5 Formy paralelního zpracovaní

V databázích se podle [1] můžeme setkat se čtyřmi formami paralelizace podle toho, v jakém místě se využívá paralelního přístupu. Pokud paralelizujeme určitou úlohu tak, aby každý dotaz nebo transakce byla zpracovávána jedním procesem, získáme mezidotazový („interquery“) paralelismus. Když použijeme procesy pro paralelní řešení dílčích částí dotazu, dostaneme „intraquery paralelismus“. Obě zmíněné formy paralelizace můžeme zařadit mezi architekturu MIMD („Multiple Instruction Multiple Data“) Flynnovy klasifikace, kde různé instrukce pracují nad různými daty. Intraquery paralelismus je řešen pomocí posledních dvou forem. První z nich se jmenuje „interoperation paralelismus“ a využívá zrychlení pomocí paralelního vykonávání stejných částí podúloh na jiných datech. Podle Flynnovy klasifikace jde o SIMD („Single Instruction Multiple Data“) architekturu, kde stejná instrukce pracuje s odlišnými částmi dat. Jedná se například o operace třídění a hledání nad velkou tabulkou rozdělenou do oddílů („partitioned table“). Poslední formou je „intraoperation paralelismus“, který se vyskytuje při souběžném vykonávání různých operací v rámci jednoho dotazu nebo transakce. Tuto formu nelze jednoznačně zařadit podle Flynnovy klasifikace. V další části kapitoly budou přesněji popsány tyto formy paralelizace.

5.1 Interquery

V této formě paralelizace se vykonávají různé dotazy nebo transakce paralelně ve stejný čas jak již bylo zmíněno v úvodu této kapitoly. Díky této metodě se dá jednoduše zvýšit propustnost dotazů i transakcí. Tento typ paralelizace ale nijak nemění čas vykonávání jednotlivých dotazů nebo transakcí oproti sekvenčnímu zpracování. Právě proto je hlavním použitím tohoto typu škálovatelnost transakčních systémů, kdy pomocí paralelizace jsme schopni dosáhnout vykonávání velkého počtu transakcí za sekundu.

Při srovnání se sekvenčním zpracováním se interquery paralelismus jeví vždy jako rychlejší. Sekvenční vykonávání transakcí musí vytvořit frontu všech transakcí, kdy v jeden čas může být zpracovávána právě jedna transakce. Díky použití paralelizace jsme schopni výrazně zkrátit čekací dobu transakcí ve frontě a tím i získat lepší výsledný čas zpracování.

Interquery paralelismus je nejjednodušším typem paralelizace k nasazení do databázového sytému, zvláště pokud se jedná o paralelní systém pomocí sdílené paměti (tento typ architektury bude podrobněji popsán v kapitole 6.2). Ten můžeme vytvořit bez větších změn z normálního databázového systému díky podpoře souběžného zpracování transakcí. Provozování této formy paralelizace je ale naopak komplikovanější při použití architektury sdílených disků nebo architektury bez sdílení (podrobněji popsáno v kapitolách 6.3 a 6.4). U těchto architektur je nutné zajistit ještě další akce, jako je zamykání a logování, pro zabezpečení synchronního stavu databáze. Není možné, aby různé procesy upravovaly stejná data ve stejný čas. Dále je nezbytné zaručit přístup procesů k nejaktuálnějším datům.

5.2 Intraquery

Tato forma paralelizmu spočívá v paralelním vykonávání jednoho dotazu pomocí více procesů. Jelikož se jedná o paralelizaci jednoho dotazu, je tento typ důležitý pro zrychlení časově náročných

dotazů. Naopak pro urychlení zpracování většího počtu dotazů nemůže být použit, protože používá sekvenčního vykonávání dotazů.

Pro lepší pochopení, jak je možné paralelizovat vykonávání jednoho dotazu, si představme tabulku, kterou potřebujeme seřadit. Předpokládejme, že tato tabulka je rozdělena do oddílů pomocí partitioningu definovaného jedním atributem, podle kterého potřebujeme vše seřadit. Řadící funkce může být implementována jako paralelní seřazení všech oddílů s následným spojením výsledků.

Pokud chceme využít intraquery paralelizaci, nejsme omezeni jen na souběžné vykonávání stejných operací. Je možné paralelizovat jednotlivé operace při vykonávání dotazu, i když nejsou shodné. Každý dotaz je před vykonáváním rozložen na strom operátorů (operator tree) reprezentující atomické operace. Výsledkem vykonávání těchto operací směrem zdola nahoru dostaneme v kořeni stromu výsledek dotazu. Tento výpočet je možné paralelizovat současným vykonáváním některých operací. Přitom je nutné zajistit, aby operace vykonávané souběžně na sobě nezávisely. Dokonce můžeme vytvořit paralelní propojení pomocí zřetězeného zpracování (pipeline), kdy jeden proces generuje výstup a další ho hned načítá a zpracovává (podrobněji vysvětleno v kapitole 5.3).

Jak bylo výše popsáno, intraquery paralelizace může byt implementována pomocí dvou typů paralelního zpracování. První typ paralelizující stejný typ operace v rámci jednoho dotazu se nazývá intraoperation paralelismus. Druhý typ vykonává jeden dotaz pomocí souběžného řešení odlišných operací a je pojmenován interoperation paralelismus. Tyto dva typy jsou k sobě komplementární, můžeme je tedy využít zároveň při vykonávání jednoho dotazu.

Jelikož počet operací v jednom typickém dotazu není velký v porovnání s množstvím prvků, nad kterými se dotaz bude provádět, první typ paralelizace zrychluje více při zvyšování paralelizace. Protože k zpracování dotazů v dnešních paralelních databázových systémech je použito relativně malé množství procesorů, je důležité využití obou forem.

5.3 Intraoperation

Jelikož databáze pracují nad tabulkami s velkým množstvím záznamů, můžeme paralelizovat operace pomocí jejího paralelního spuštění nad více rozdílnými podmnožinami tabulky. Tato forma paralelizace je často nazývána paralelizací oddílů tabulek, protože využívá dat rozdělených do určitých množin. Protože počet záznamů v tabulce může být obrovský, tak i stupeň paralelizace je potenciálně velice vysoký. Právě proto se tento typ paralelizace často používá v databázových systémech.

Abychom mohli nasadit tento typ paralelizace do databázového systému, musíme umět vyřešit hlavní problémy, které jsou s ním spojené. Podle [2] jsou to tyto tři problémy. Jak je možné uspořádat operace, aby mohli být prováděny nad různými datovými množinami? Jak rozdělit data do oddílů, aby s nimi mohly operace pracovat? Jak lze převést základní databázové sekvenční operace jako je hledání, řadění, agregační funkce a spojování tabulek na paralelní operace? Jediným řešením všech těchto otázek je formulování nových paralelních verzí standardních databázových operací. Každá nově vytvořená operace musí umět odpovědět na tyto otázky, aby se mohla použít při paralelním zpracování.

5.4 Interoperation

Interoperation paralelismus pracuje s paralelním vykonáváním různých operací v rámci jednoho dotazu či transakce. Podle [2] lze rozlišovat dva podtypy této formy paralelizace. Prvním je paralelizace pomocí zřetězeného zpracování („pipelined parallelism“). Druhý podtyp se nazývá nezávislý paralelismus („independent paralelism“). Obě tyto formy budou podrobně popsány podle [2] v následující části kapitoly.

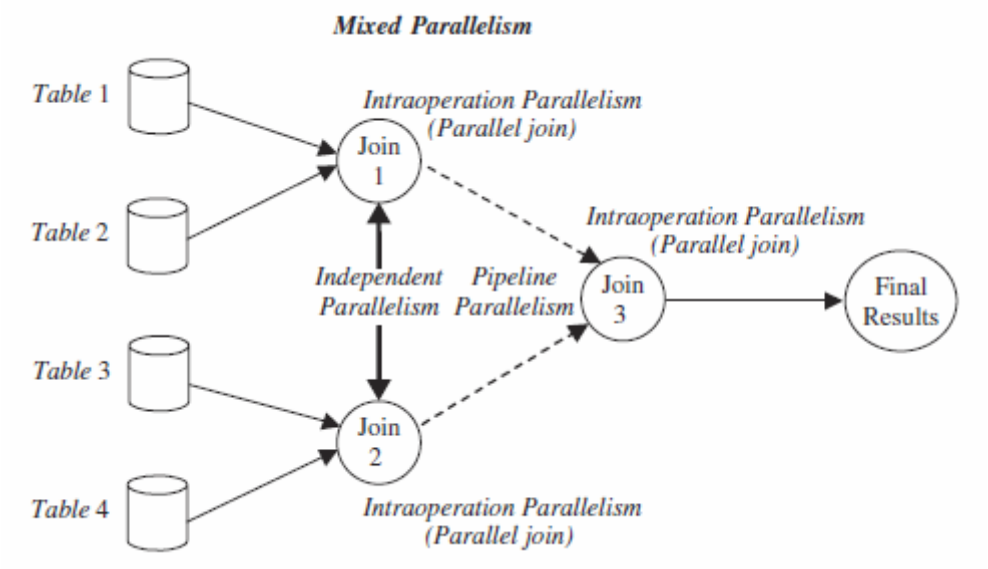
Pipeline slouží k zřetězení více procesů nebo operací za sebou. Při jeho použití se výsledky jedné operace průběžně zpracovávají pomocí operace další. Takto můžeme zřetězit nekonečně mnoho procesů. Zrychlení vzniká při překrývání spuštěných procesů. Není nutné čekat na úplné dokončení předchozí operace, ale stačí průběžně načítat a zpracovávat produkované výsledky. Pipeline je inspirován používáním montážních linek ve výrobních továrnách (například výroba aut). V databázovém paralelním zpracování pipeline představuje více operací tvořící montážní linku, jejímž výsledkem je odpověď na zadaný dotaz. Jedním z hlavních výhod tohoto typu zpracování je provádění operací bez potřeby zápisu pomocných výsledku na disk.

Paralelizace pomocí pipeline je výhodná pro malý počet procesorů. Bohužel ale nenabízí velkou možnost škálovatelnosti kvůli následujícím důvodům. Zřetězení ve většině případů není tvořeno velkým počtem prvků a tak neposkytuje možnost pro velký stupeň paralelizace. Ten odpovídá množství operací ve zřetězeném zpracování. Dále není možné nasadit pipeline na operace, které neprodukují výsledky průběžně. Posledním důvodem je použití pipeline jen u operací s podobnou dobou zpracování. Pokud se využije zřetězeného zpracování u operací s různou dobou zpracování, nedochází k velkému zrychlení, protože malé operace jsou často nuceny čekat na ty delší. Kvůli těmto omezením je využívání zřetězeného zpracování jen pro malý stupeň paralelizace. Pokud je stupeň veliký, stává se nejdůležitější formou paralelizace partitioning.

Pod nezávislým paralelismem si můžeme představit operace v rámci jednoho dotazu, které na sobě vůbec nezávisí. Příkladem je propojení čtyř tabulek (Tx, kde x reprezentuje číslo tabulky) pomocí operace spojení („join“): T1 join T2 join T3 join T4. Pro získání výsledku lze paralelně vykonat operace T1 join T2 a T3 join T4. Koncový výsledek bychom dostali spojením pomocí operace join obou výsledků z předchozích operací. Stejně jako pipeline paralelismus, tak i nezávislý paralelismus neposkytuje velký stupeň paralelizace kvůli omezenému počtu operací, které jsou vzájemně nezávislé v rámci jednoho dotazu.

5.5 Smíšená paralelizace

V běžném prostředí se nesetkáváme jen s jednou formou paralelizace, ale velice běžné je použití více forem najednou. Využití více typů paralelizace uvidíme na příkladu spojení čtyř tabulek z minulé kapitoly. Úkolem je spojit čtyři tabulky (Table x, kde x reprezentuje číslo tabulky) pomocí operace spojení („join“). Řešení je znázorněno na obrázku 5.5.1, kde si můžeme všimnout použití tří forem paralelizace (intraoperation, pipeline a nezávislé). Nezávislá paralelizace je použita stejným způsobem jako v příkladu z minulé kapitoly. Tedy jedná se o paralelní join 1 a 3. Pipeline paralelizace využívá postupného vytváření výsledků, z předchozích operací join, které následně spojuje v poslední operaci join. Intraoperation paralelizace je použita v každé operaci join. Využívá se operace paralelního spojení („parallel join“), kdy operace spojení je vykonávána souběžně nad více oddíly tabulky.



Obrázek 5.5.1 – Smíšená paralelizace [2]

6 Architektury paralelních databází

Pod paralelními databázemi si představíme většinou jednu administrativní oblast s homogenním pracovním prostředím a uložením všech dat na stejném místě (ať už se jedná o stejnou místnost, či budovu). Nicméně architektur paralelních databází existuje více a uvedený příklad reprezentuje jen jednu z mnoha architektur centralizovaného systému.

V následující kapitole jsou uvedeny různé druhy architektur paralelních databází podle [1]. Nejdříve budou představeny různé topologie sítí pro komunikace mezi komponentami. Poté budou popsány architektury, kdy procesory sdílející určité prostředky (paměť, disky, bez sdílení). Následně bude vysvětlena hierarchická architektura typická smíšením více typů sdílených architektur. Nakonec budou představeny distribuované databáze.

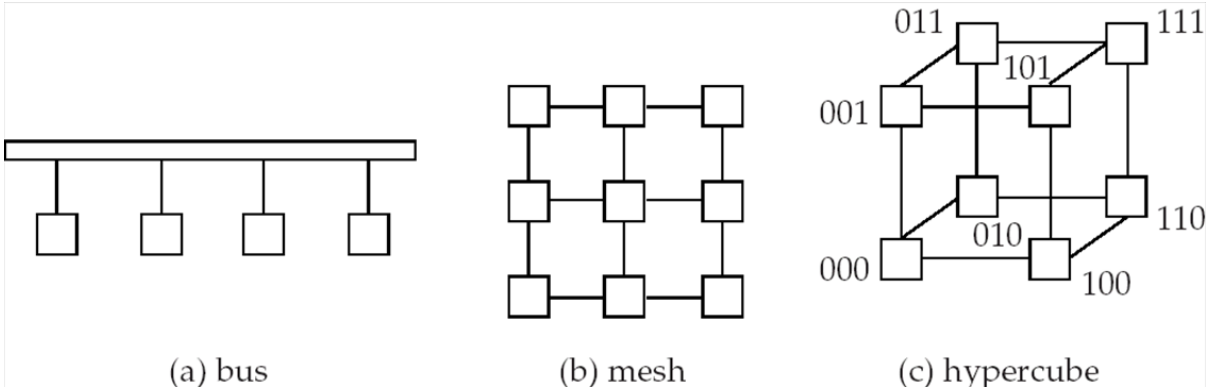
6.1 Architektura propojení sítě

Paralelní systémy se skládají ze sady komponent, jako jsou procesory, paměť a disk. Tyto komponenty spolu komunikují přes propojovací síť. Podle [1] rozeznáváme tři běžné typy propojení: sběrnice (bus), smyčková topologie (mesh), a hyperkostka (hypercube). Tyto tři typy budou následně popsány a jsou zobrazeny na obrázku 6.1.1.

Pro sběrnici platí, že všechny komponenty systému posílají a přijímají data na jedné komunikační sdílené sběrnici. Tento typ je vidět na obrázku 6.1.1a. Sběrnice může představovat síť ethernet nebo paralelní propojení. Tato architektura je výhodná pro malý počet procesorů. Díky omezení sdílené sběrnice, kdy může komunikovat v jeden čas jen jeden procesor, tento typ nelze efektivně zrychlovat se zvyšováním stupně paralelizace.

Smyčková topologie nebo také tzv. „mesh“ topologie využívá propojení uzlů do mřížky. Každý uzel je propojen se všemi sousedními uzly v mřížce. V dvoudimenzionální mřížce bude uzel spojen s čtyřmi přilehlými uzly (příklad je vidět na obrázku 6.1.1b). V třídimenzionální mřížce se počet spojení zvyšuje na šest. Pokud spolu chtějí komunikovat uzly, které nejsou přímo propojeny, využijí se prostřední přímo propojené uzly pro směrování jejich zpráv. Počet propojení roste s přidáváním nových uzly a tedy i přenosová kapacita sítě se zvětšuje se zvyšováním stupně paralelizace.

Pro hyperkostku platí, že její komponenty jsou binárně číslované. Komponenty, jejichž čísla se liší právě v jednom bitu, jsou propojeny. Díky tomu každá z n komponent je spojena s $\log n$ ostatními komponentami. Na obrázku 6.1.1c je vidět architekturu s osmi uzly. Toto propojení zaručuje, že zpráva mezi jakýmikoli dvěma uzly projde maximálně $\log n$ komponentami. Ve srovnání se smyčkovou topologií, kde cesta povede přes $2(\sqrt{n}-1)$ spojení ($\sqrt{n}$ spojení v případě, že se jde přes hranu topologie), komunikační zpoždění u hyperkostky je daleko menší než u smyčkové topologie.



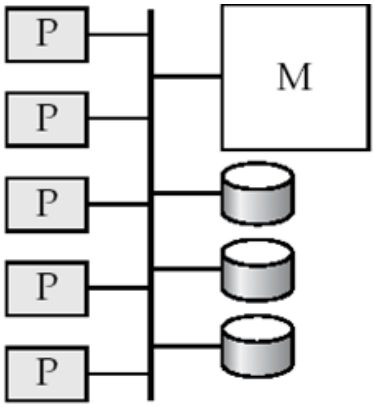
Obrázek 6.1.1 - Typy propojených sítí (a - sběrnice, b – smyčková topologie, c - hyperkostka) [1]

6.2 Sdílení paměti

V architektuře se sdílenou pamětí procesory a disky mají přístup ke společné paměti, nejčastěji pomocí sběrnice nebo propojené sítě. Výhodou této architektury je velmi vysoká efektivita komunikace mezi procesory. Data jsou totiž uložena na sdílené paměti, a tedy procesory k nim mohou přistupovat bez nutnosti jejich přesunu. Procesor může také komunikovat s okolními procesory daleko rychleji použitím zápisů do sdílené paměti než zasíláním zpráv přes komunikační mechanismus. Nevýhodou architektury sdílené paměti je nemožnost škálovatelnosti. Architektura umožňuje použití jen omezeného počtu procesorů, pro který platí, že rychlost sběrnice nebo propojené sítě nelimituje rychlosti procesorů a nestává se tak úzkým hrdlem architektury. Při přidávání více procesorů po dosažení maximálního počtu, se již rychlost nezvyšuje. Od té doby procesory stráví více času čekáním na právo k přístupu k sdílené paměti než samotným výpočtem [1].

Kvůli omezení počtu přístupů ke sdílené paměti mají většinou procesory velkou vyrovnávací paměť („cache“). Bohužel ne všechna data lze nahrát do vyrovnávací paměti, a proto se někdy přístupu na sdílenou paměť nevyhneme. Nejdůležitější vlastností vyrovnávací paměti je její synchronizace se sdílenou pamětí. Při používání vyrovnávací paměti musí být zaručen stejný stav všech dat v obou typech pamětí. Pokud procesor zapíše do sdílené paměti a tím upraví nebo smaže data, která byla v paměti před zápisem, je nutné provést úpravu ve všech vyrovnávacích pamětech obsahujících tato data. Režie udržování souvislého stavu vyrovnávacích pamětí se zvyšuje čím dál více s rostoucím počtem procesorů. Právě proto můžeme zvyšovat rychlost systému pomocí škálovatelnosti jen po nějaký limitní bod.

Na obrázku 6.2.1 je vidět schéma architektury se sdílenou pamětí. „P“ představuje procesory, „M“ zastupuje sdílenou paměť a válec zobrazuje disk. Všechny tyto části jsou propojeny sdíleným médiem, jako je paměťová sběrnice nebo propojená síť.

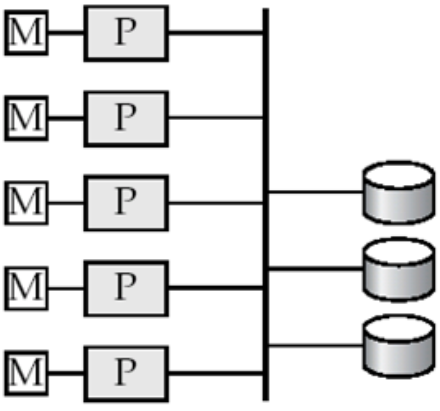


Obrázek 6.2.1 - Architektura se sdílenou pamětí [1]

6.3 Sdílení disků

V architektuře sdílených disků mají všechny procesory přístup ke všem diskům přes síť. Navíc má každý procesor vlastní paměť. Tento systém poskytuje dvě výhody oproti předchozí architektuře sdílené paměti. Zaprvé díky privátní paměti pro každý procesor je eliminován problém úzkého hrdla u paměťové sběrnice. Druhá výhoda je odolnost vůči chybám. Systém poskytuje levný způsob k dosažení lepšího stupně odolnosti vůči chybám. Ten je dosažen oddělením procesorů s jejich pamětí od databáze uložené na disku. Když procesor nebo jeho paměť selže, můžou ostatní procesory přebrat jeho úlohy. Toto je uskutečnitelné díky společné databázi přístupné ze všech procesorů. Kromě ochrany proti chybě procesoru či jeho paměti lze zabezpečit i databázi použitím RAID architektury (podrobněji popsána v kapitole 3.4).

I tato architektura se setkává s problémem škálovatelnosti. I když paměťová sběrnice již nepředstavuje problém, vzniká nové zpomalení při velkém dotazování procesorů na sdílené disky. Úzké hrdlo se přesunulo na sběrnici mezi diskem a procesory. Při srovnávání s architekturou sdílené paměti, lze zde zvětšovat počet procesorů, ale komunikace procesorů je pomalejší kvůli pomalejšímu komunikačnímu médiu.



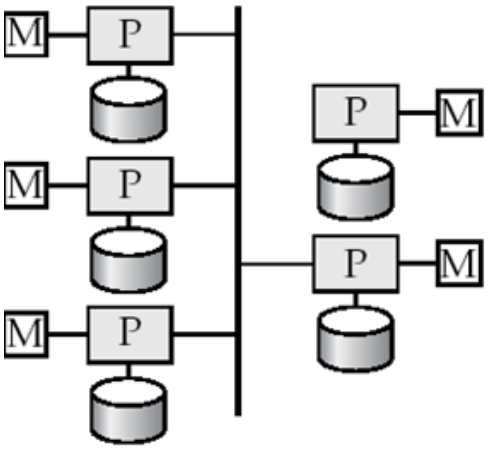
Obrázek 6.3.1– Architektura se sdílenými disky [1]

Architektura je zobrazena na obrázku 6.3.1. Opět „P“ představuje procesory, „M“ zastupuje sdílenou paměť a válec zobrazuje disk. Procesory a disky jsou propojeny pomalejší sběrnicí, než tomu bylo u architektury se sdílenou pamětí, kdy se komunikovalo přes paměťovou sběrnici nebo propojenou síť.

6.4 Bez sdílení

V architektuře bez sdílení každý uzel systému obsahuje procesor, paměť a jeden nebo více disků. Procesor z jednoho uzlu může komunikovat s ostatními procesory přes vysokorychlostní síť. Každý uzel představuje server pro svá data umístěná na vlastním jednom disku nebo discích. Zpracovávané úlohy odkazují vždy jen na lokální disky a tím odpadá nutnost velké komunikace přes síť. Tato architektura díky tomu překonala problémy předchozích architektur spojené s velkou zátěží při komunikaci přes jedno sdílené médium. Nyní se komunikuje přes síť pouze v případě, že procesor potřebuje přistoupit k cizím diskům nebo v případě zasílání výsledků dotazů.

Architektura bez sdílení podporuje větší možnosti škálovatelnosti než ostatní architektury. Toho je také dosaženo pomocí sítě propojující uzly navrhované pro možné zvětšování. Čím více je přidáno uzlů, tím se zvedá propustnost sítě. Mezi hlavní nedostatek architektury patří cena komunikace a přístupu k cizím diskům. Ta je větší než u předchozích architektur sdílení paměti a disků.



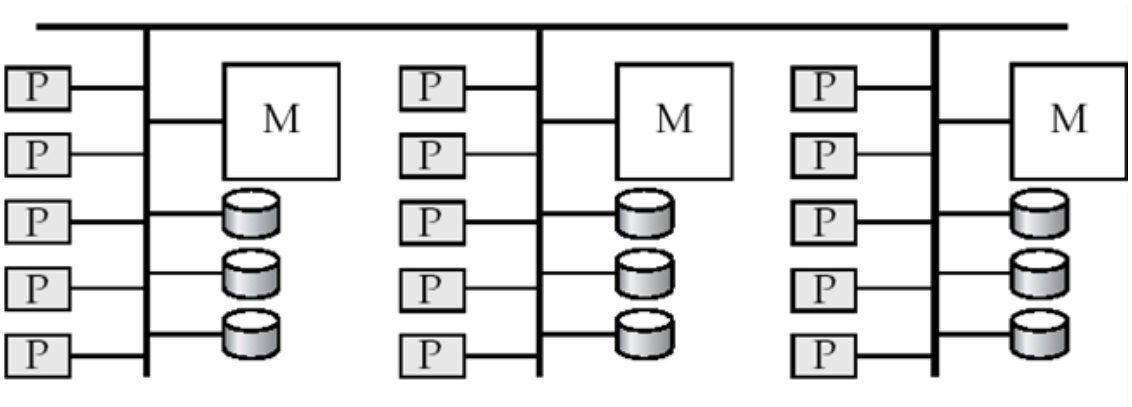
Obrázek 6.4.1 - Architektura bez sdílení [1]

Obrázek 6.4.1 zachycuje architekturu bez sdílení. Znovu „P“ představuje procesory, „M“ zastupuje sdílenou paměť a válec zobrazuje disk. Komunikace mezi uzly probíhá přes vysokorychlostní síť.

6.5 Hierarchická architektura

Hierarchická architektura spojuje vlastnosti předchozích architektur se sdílenou pamětí, sdílenými disky a bez sdílení. Na nejvyšší úrovni se nacházejí uzly propojené sítí, které nesdílí ani paměť, ani disky. Jedná se tedy o použití architektury bez sdílení. V každém uzlu může být architektura se sdílenou pamětí a několika procesory. Jako alternativu lze sestavit v každém uzlu systém se sdílenými disky, kdy každý systém sdílející sadu disků by mohl být systémem se sdílenou pamětí. Tímto by byl vytvořen hierarchický systém se sdílenou pamětí a několika procesory na nejnižší úrovni. Na nejvyšší úrovni by byla použita architektura bez sdílení. A mezi oběma vrstvami by bylo možné nasadit architekturu se sdílenými disky.

Obrázek 6.5.1 zobrazuje hierarchickou architekturu složenou z uzlů se sdílenou pamětí. Všechny uzly jsou poté propojeny architekturou bez sdílení. I zde „P“ představuje procesory, „M“ zastupuje sdílenou paměť a válec zobrazuje disk.

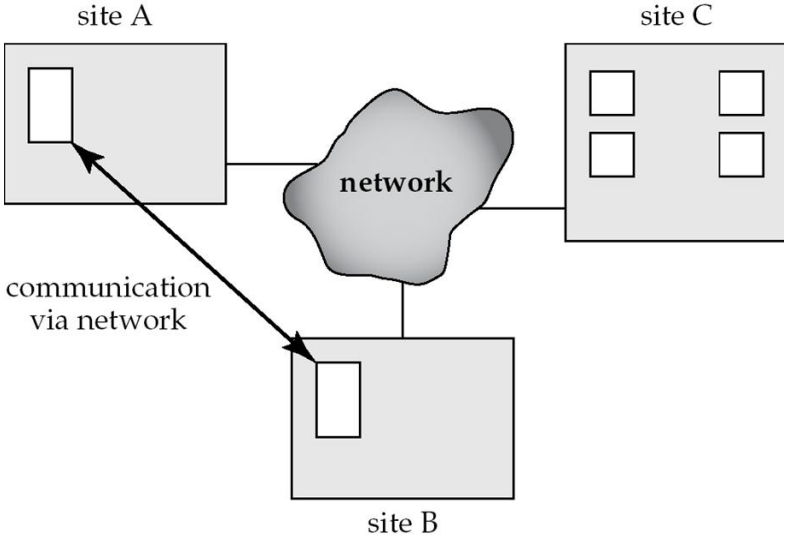


Obrázek 6.5.1 - Hierarchická architektura [1]

Složitost hierarchické architektury se projevila i na programování těchto systému. Proto vznikla architektura distribuované virtuální paměti, která logicky představuje jednu sdílenou paměť, ale fyzicky se jedná o více oddělených paměťových systémů. Díky tomu můžou procesory přistupovat k odděleným částem pamětí jako k jedné spojité virtuální paměti. Jelikož rychlost přístupu k datům se liší podle toho, jestli se jedná o lokální nebo vzdálené disky, je tato architektura pojmenována jako neuniformní paměťová architektura.

6.6 Distribuované databáze

V distribuovaném databázovém systému je databáze uložena na více počítačích. Tyto počítače spolu komunikují pomocí různých komunikačních médií, jako jsou vysokorychlostní sítě nebo telefonní linky. Oproti předchozím architekturám distribuovaný systém nesdílí hlavní paměť ani disky. Počítače v tomto systému se můžou velice lišit svou velikostí a funkcionalitou. Oproti paralelním databázím, kde jsou procesory úzce propojeny a společně představují jeden databázový systém, distribuované databáze jsou tvořeny volně vázanými oblastmi, které nesdílí žádné fyzické komponenty. Architektura je zobrazena na obrázku 6.6.1. Obrázek představuje obecnou strukturu distribuovaného systému rozděleného na tři části („site“) A, B, C, které reprezentují jednu administrativní oblast. V každé oblasti může být jeden a více podsystémů, ty jsou zobrazeny prázdným obdélníkem. Komunikace mezi podsystémy probíhá přes síť (network).



Obrázek 6.6.1– Architektura distribuovaného systému

Hlavní rozdíl mezi architekturou bez sdílení a distribuovaným systémem je geografické umístění obou architektur. U prvního typu bez sdílení je celý systém umístěn v jedné geografické lokalitě. Naproti tomu u distribuované databáze jsou uzly často geograficky odděleny. Díky větším vzdálenostem mezi částmi systému jsou uzly spravovány odděleně a mají pomalejší spojení. Mezi další důležité rozdíly patří zavedení globálních a lokálních transakcí. Lokální transakce přistupuje k datům jen v uzlu, kde byla transakce zavolána. Globální transakce buď přistupuje k datům z jiného uzlu, než z kterého byla transakce spuštěna nebo se jedná o transakci, která přistupuje k datům z více částí systému. Existuje mnoho důvodů k použití distribuovaných databázových systému v reálném prostředí. Mezi hlavní patří možnost sdílení dat, autonomie částí systému a dostupnost systému.

Právě sdílení dat je jednou z nejžádanějších vlastností, kdy uživatel jedné oblasti může přistoupit k datům z oblasti jiné. Například v distribuovaném bankovním systému, kde si každá pobočka ukládá vlastní data u sebe, je možné, aby uživatel z jedné pobočky přistoupil k datům z odlišné pobočky. Bez této vlastnosti by uživatel, přející si převést peníze z jedné pobočky do jiné, musel použít nějakého jiného externího mechanismu spojujícího obě pobočky [1].

Výhodou sdílení dat pomocí distribuovaného systému je možnost určitého stupně kontroly nad daty, které jsou uloženy lokálně. U centralizovaného systému existuje databázový administrátor, který spravuje centrálně celou databázi. U distribuovaného systému také existuje globální administrátor, který je odpovědný za celý systém. Kromě něho ale existují ještě administrátoři lokálních databází pro každou oblast. Podle návrhu distribuovaného systému existují administrátoři s různými stupni lokální samosprávy. Možnost lokální správy je mnohdy hlavní výhodou distribuovaného systému.

Dostupnost částí systému zaručuje určitou odolnost vůči chybám. Pokud jedna část distribuovaného systému selže, zbývající části můžou fungovat nadále. Pokud jsou data z podsystémů replikována do více oblastí systému, je možné úplně nahradit funkcionalitu a data při výpadku nějaké části. Selhání musí být vždy detekováno systémem a vhodné opatření musí být spuštěno pro zotavení se z chyby. Systém musí přestat používat část systému, která již nefunguje a popřípadě přesměrovat požadavky na systémy obsahující replikovaná data. V momentě, kdy je část systému opravena a opět funkční musí systém zajistit opětovné zapojení této části. Zotavení z chyby je daleko složitější u distribuovaných systémů než u centralizovaných. Jejich schopnost pokračovat v chodu navzdory selhání jedné části zvyšuje spolehlivost systému. Nepřetržitý chod databázových systémů s realtime aplikacemi je velice důležitý. Nedostupnost aplikace může způsobit ztrátu části zákazníků. Například u letecké společnosti by to znamenalo ztrátu potenciálních zákazníků ve prospěch konkurenčních společností [1].

6.7 Příklad paralelní databáze

Příkladem databáze s paralelní architekturou je open-source databáze PostgreSQL verze 9.03. Podle [10] PostgreSQL podporuje spolupráci více databázových serverů najednou. Díky spolupráci dosahuje vlastnosti vysoké dostupnosti („high availability“) a vyváženého zatížení („load balancing“). Vysoká dostupnost slouží k ochraně proti pádu serveru a nedostupnosti aplikací spolupracující s tímto serverem. Jedná se o použití dvou a více serverů najednou, kdy po chybě hlavního serveru, přebere funkčnost sekundární server. Vyvážení zatížení je způsob škálovatelnosti databázového systému, kdy více serverů poskytuje stejná data a tím snižuje celkové zatížení. K dosažení těchto dvou vlastností existuje více řešení. Ty budou nyní popsány podle [10].

První technika se nazývá „Shared Disk Failover“ a využívá jediného sdíleného diskového pole k uložení databáze. Jedná se o architekturu sdíleného disku popsanou v kapitole 6.3. Díky této

architektuře není potřeba synchronizovat databázi s jinými servery a odpadá zatížení způsobené synchronizací. Pokud databázový server selže, nahradí ho záložní server. Záložní server ovšem nesmí přistupovat na sdílené diskové pole, pokud je hlavní server aktivní. Hlavní problém nastává při chybě diskového pole, kdy oba (hlavní i záložní) servery se stávají nefunkčními.

Dalším typy jsou techniky distribuované architektury. Řešení „File System (Block-Device) Replication“ používá k synchronizaci dat replikaci celého souborového systému. Všechny změny v souborovém systému jsou šířeny na záložní server. „Warm and Hot Standby Using Point-In-Time Recovery (PITR)“ zajišťuje vysokou dostupnost pomocí šíření WAL logu (záznam všech operací). Při chybě primárního serveru má záložní server téměř všechna jeho data. Jednou z často používaných technik je „Trigger-Based Master-Standby Replication“ a její implementace „Slony-I“. Toto řešení zajišťuje replikaci dat na záložní server pomocí asynchronních zpráv. Navíc záložní server slouží jako databázový server pro operace čtení („read-only“). Díky těmto vlastnostem dosahuje řešení jak vysoké dostupnosti, tak vyváženého zatížení. „Statement-Based Replication Middleware“ také dosahuje obou vlastností pomocí programu, který rozděluje všechny SQL dotazy mezi oddělené servery. Program zasílá dotazy obsahující čtení a zápis na všechny servery a tím udržuje konzistentní stav. Pokud se jedná jen o operaci čtení, pošle se jen jednomu serveru. Díky tomu zatížení čtecími operacemi je rozděleno na více serverů. „Asynchronous Multimaster Replication“ je první technika, která používá více primárních databází. Řešení slouží pro synchronizaci serverů, které nejsou pravidelně propojeny (server na notebooku, vzdálený server). Při propojení databází se zjišťují konflikty, které se řeší pomocí pravidel nebo uživatelské interakce. „Synchronous Multimaster Replication“ používá také více primárních databází, jak již slovo „multimaster“ napovídá. Jedná se o techniku, kdy na každém serveru je možné vykonat dotaz se zápisem. Server vykonávající tento dotaz zasílá před operací „commit“ dotaz ostatním serverům. Tento systém předávání dotazů může způsobovat rozsáhlé zamykání a vést k nízkému výkonu. Metoda „Data partitioning“ slouží k rozdělování tabulek na více skupin dat. Každá skupina může být modifikována pouze jedním databázovým serverem (například rozdělení poboček na Londýn a Paříž [10]). Poslední technikou je „Multiple-Server Parallel Query Execution“, která jako jediná slouží k zpracování jednoho dotazu více servery najednou. Toho bývá často dosaženo pomocí rozdělení dat mezi servery, kdy každý server zpracovává dotaz nad svou částí dat. Nakonec jsou výsledky vráceny centrálnímu serveru, který odešle odpověď klientovi. Implementací této metody je například „pgpool-II“ [10].

7 Používané databázové systémy

Než se dostaneme k měření výkonnosti a ladění databázových systémů, musíme si představit dnešní nejpoužívanější databázové systémy. V následující kapitole bude představena podle [1] open-source databáze PostgreSQL a komerční databázové systémy (IBM DB2, Oracle, Microsoft SQL Server). Také bude uvedena nejpoužívanější open-source databáze MysSQL podle [13]. Každý systém má unikátní vlastnosti a rysy. Mezi ně patří například pomocné nástroje, SQL rozšíření, architektura systému a uložení dat, zpracování dotazů a další.

7.1 PostgreSQL

PostgreSQL je open-source projekt vytvářející objektově-relační databázový systém. Je to následník jednoho z dřívějších databázových systémů. PostgreSQL byl vyvinut profesorem Michaelem Stonebrakerem na Kalifornské univerzitě v Berkeley a jeho vývoj stále pokračuje. V dnešní době podporuje PostgreSQL standardy SQL92 a SQL:1999 a nabízí vlastnosti jako složené dotazy, cizí klíče, triggery, pohledy, transakční zpracování a multigenerační architekturu (Multiversion concurrency control - MVCC). Podle [11] architektura MVCC implementuje přístup k datům bez zámků. Tedy čtenáři (operace select) nikdy nečekají na písaře (operace update, delete a insert) a naopak. Dále architektura řeší zpracování modifikovaných n-tic, ať už se jedná o operaci mazání nebo úpravy. MVCC neupravuje stávající řádky, ale místo toho provede kopii řádku a tím vytvoří novou verzi. Podle verzí n-tic se řeší jejich viditelnost, která poté slouží při vykonávání dotazů k získání validních dat. Mazání starých a nepotřebných verzí je řešeno operací VACUUM, která se musí opakovaně volat po dobu běhu databáze. Dále systém nabízí nové datové typy, funkce, indexační metody a možnost propojení s velkým množstvím programovacích jazyků (C, C++, Java, Perl, Tcl, Python). Asi největším přínosem tohoto databázového systému je vývoj pod BSD licencí a tedy možnost využití, úpravy a distribuce zdrojových kódů a dokumentace bez žádného poplatku.

7.2 Oracle

Firma Oracle byla založena roku 1977 Larry Ellisnem, Bobem Minerem a Edem Oatsem. Nejdříve se firma nezbývala tvorbou komerčního databázového systému. Poté se ale přeorientovala na tvorbu relační databáze jako komerčního produktu a byla první firmou, která ho uvedla na trh. Od této doby si Oracle udržuje vedoucí pozici s relačními databázemi na trhu. V dnešní době již nenabízí jen databázový systém, ale spolu s ním vytváří nástroje pro vývoj a správu databází. Oracle nabízí velké množství nástrojů pro návrh databáze, dotazování, generování reportů a analýzu dat s OLAP. Oracle objektově-relační databáze podporuje všechny hlavní vlastnosti standard SQL:1999. Kromě toho Oracle nabízí další jazykové konstrukce, některé ze standardu SQL:1999, další jsou přímo vázané na syntaxi a funkcionalitu Oraclu. Systém využívá dva hlavní procedurální jazyky Java a PL/SQL. Dále implementuje indexy, materializované pohledy, optimalizaci výkonu, paralelní zpracování, replikaci a všechny důležité operace a vlastnosti dnešních databázových systémů.

7.3 DB2

DB2 je rodina komerčních produktů vyvíjená společností IBM obsahující databázový systém, business inteligenci, systém pro správu obsahu. Všechny tyto produkty jsou známy pro jejich komplexnost a robustnost. První komerční verze databázového systému DB2 byla uvedena na trh roku 1984. DB2 je možné nasadit na velký počet různých hardwarových zařízení s různými operačními systémy. Systém obsahuje mnohé nástroje pro administraci, replikaci, distribuovaný datový přístup a OLAP. DB2 nabízí velkou sbírku SQL funkcí pro různorodé aspekty databázového zpracování. Velká část funkcí a syntaxe byla použita při návrhu standardů SQL-92 a SQL-99.

7.4 Microsoft SQL Server

Microsoft SQL Server byl původně vytvořen v roce 1980 firmou Sybase pro systém UNIX a byl později byl upraven pro operační systém Windows NT. Roku 1994 firma Microsoft uvedla vlastní databázový systém nezávislý na firmě Sybase. Databázový systém je nabízený ve více variantách (express, standard, enterprise edice) a v mnoha lokalizacích po celém světě. Server obsahuje také analytické služby zahrnující OLAP a dolování dat. Dále je produkt typický velkým množstvím grafických nástrojů a pomocníků pomáhajících při vytvoření, administraci, zálohování a replikaci databázových serverů. Mnoho dnešních vývojových prostředí obsahuje podporu Microsoft SQL serveru.

7.5 MySQL

MySQL je světově nejpopulárnější open-source databázový software. První verze MySQL byla vytvořena Michalem Wideniusem v roce 1995. V roce 2000 byl databázový systém vydán pod dvěma licencemi. Licence GPL umožnila použití MySQL zdarma. Druhá část licence zpoplatňuje distribuci databáze spolu s vytvořeným komerčním softwarem. MySQL je multiplatformní databáze. Jejím hlavním cílem je poskytnout rychlý databázový server, který je jednoduchý na nastavení a používání. Pro komerční účely existuje několik placených verzí, které nabízí rozšířenou funkcionalitu. Existují mnohé webové aplikace (Joomla, WordPress, phpBB, Drupal, Wikipedia), které používají tento databázový server.

8 Měření výkonnosti databáze

V této kapitole jsou popsány techniky měření výkonu databáze. Pro testování a ladění byla vybrána open-source databáze PostgreSQL. První část kapitoly popisuje společné techniky k měření výkonnosti všech typů OLAP databází. Podkapitoly 8.4 a 8.5 představují nástroje k měření výkonu právě databáze PostgreSQL.

8.1 TPS

Dříve než můžeme začít ladit výkon databází, musíme mít nějaký prostředek na jeho měření. Velmi častým měřícím prostředkem je číslo TPS (transaction per second). Tato hodnota udává počet vykonaných transakcí za jednu sekundu.

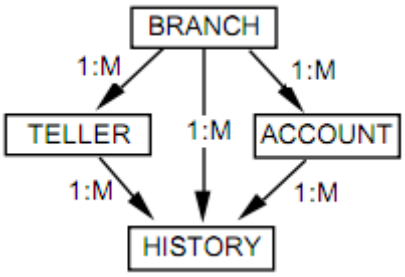
TPS značí počet commitů (úspěšných) a rollbacků (neúspěšných) transakcí za sekundu [12]. Měření probíhá pomocí dříve vytvořených testů, které se opakují po předem zvolenou dobu nebo dokud není dosažen určitý počet transakcí. Doba transakce závisí na počtu a typu operací v jedné transakci. Aby byly testy vypovídající, je nutné vytvořit testy se shodnou transakcí, která se bude vždy vykonávat nad jinou množinou dat. K výpočtu TPS byla vytvořena řada testů, jedním z nich je test TPC-B, který bude později sloužit k testování a porovnávání výkonnosti.

8.2 TPC-B

Zkratka TPC označuje neziskovou organizaci Transaction Processing Performance Council. Tato organizace se specializuje na transakční zpracování a tvorbu databázových výkonnostních testů. Kromě zaměření na počítačové funkce obsahující sadu operací ke čtení a zápisu, systémových volání nebo přenosu dat mezi subsystémy, se TPC také zajímá transakcemi na podnikové úrovni[14]. Jejich testy jsou označovány pomocí zkratky TPC-x, kde x je nahrazeno písmenem definujícím určitý test.

Podle [15] TPC-B je výkonnostní test vytvořený organizací TPS roku 1990. Jedná se již o druhý schválený test (prvním byl TPC-A). Oproti TPC-A není tento test zaměřen na OLTP, ale místo toho vytváří databázový zátěžový test charakteristický početnými diskovými operacemi, mírnou systémovou a aplikační výpočetní dobou a testováním transakční integrity. TPC vytvořila ještě třetí nejnovější test TPC-C, který také slouží spolu s TPC-A a TPC-B k porovnávání databázového systému zahrnující hardware i software počítače. Jelikož každý test má odlišnou specifikaci, není možné, aby byly mezi sebou porovnávány výsledky různých typů testů. Od 6. 6. 1995 je TPC-B test označen za zastaralý.

TPC-B se zaměřuje na dávkové testování systému řízení báze dat a jeho databáze. Test simuluje banku s více pobočkami. Každá pobočka má více bankovních úředníků. Banka má mnoho klientů, kdy každý z nich vlastní v bance účet. Databáze reprezentuje pozici peněz každé entity (pobočka, úředník a klientský účet) a historii proběhlých transakcí v bance. Transakce reprezentuje operaci vložení nebo výběru peněz klientem z jeho účtu. Transakce je vykonávána úředníkem určité pobočky [15]. Obrázek 8.2.1 zobrazuje ER diagram reprezentující banku s pobočkami (branch), úředníky (teller), účty (account) a transakční historií (history).



Obrázek 8.2.1 - TPC-B ER diagram [15]

TPC-B může vypočítat maximální možný počet simultánních transakcí, které může systém vykonat. Test neobsahuje a nezatěžuje testování žádnými uživateli, komunikačními kanály a terminály (kromě toho, na kterém je test spuštěn). TPC-B je analogií k elektronickému zpracování data (electronic data processing - EDP), kdy zpracovávající dávka je spuštěna přes noc a žádný zákazník není do systému připojen. Transakce jsou předávány programem k paralelnímu výpočtu. Program vždy čeká na dokončení jedné transakce, než spustí další. Více těchto programů může být spuštěno najednou. Testování slouží jak k měření výkonu a zjištění hraničních hodnot systému, tak i k nalezení jeho nejoptimálnější konfigurace.

8.3 JDBC TPC-B benchmark

Implementací TPC-B standardu je program JDBC TPC-B, který slouží k testování a porovnávání různých distribucí databází. Test podporuje Oracle, DB2, Derby, MySQL, PostgreSQL a generic JDBC. Protože je program implementován v jazyce Java, autor zvolil k testování použití uložených procedur (stored procedures). Díky tomu by testování mělo být méně ovlivněno režií spojenou s Java klientem připojeným k databázi a jeho implementací [16].

8.4 pgbench

Pgbench je aplikace na porovnávání výkonnosti dodávaná přímo s instalačním balíčkem databáze PostgreSQL. Jedná se o jednu z nejznámějších implementací TPC-B testu. Mimo tohoto typu program nabízí vytvoření vlastních testů spustitelných nad libovolnou databází. Pgbench tvoří škálovatelný více klientský databázový testovací program [17].

Při použití základní struktury vytváří pgbench databázové schéma shodné s diagramem na obrázku Obrázek 8.2.1. Toto schéma je následně naplněno daty podle zvoleného měřítka (scale). Při výchozím nastavení, kdy je měřítko rovno 1, se vytváří 1 pobočka s 10 úředníky a 100 000 účty. Měřítko určuje počet poboček banky a s tím i celkový počet úředníků a účtů. Pokud bychom prováděli testování nad vlastní databází, není možné určit měřítko. Proto předpokládané měřítko všech testů nad vlastní databází je 1.

Pgbench obsahuje tři předdefinované testy spustitelné nad základní strukturou databáze TPC-B. Výchozí transakce se jmenuje „TPC-B (sort of)“ neboli TPC-B (podobná). Tato transakce je skoro stejná jako standard TPC-B, liší se jen v intervalu delta a velikosti datového typu pro stav účtu [16]. Na následujícím obrázku 8.4.1 je zobrazen skript této transakce. První tři řádky nastavují podle měřítka, kolik existuje záznamů poboček, úředníku a účtů. Další čtyři řádky vytváří náhodné proměnné simulující bankovní transakci, kdy klient (aid) šel za určitým úředníkem (tid) v zvolené

pobočce (bid) vložit nebo vybrat peníze (delta). Hlavní část skriptu je tvořena uvnitř transakčního bloku (mezi begin a end). Díky transakčnímu zpracování se buď provedou operace všechny, nebo žádná z nich. Výsledkem musí být konzistentní stav peněz od pobočky po účet klienta.

Každá operace má jiný dopad na výkonnostní testování. První update je spuštěn nad největší tabulkou databáze, a tedy s největší pravděpodobností způsobí čtecí a zapisovací operace na disku. Následný select by měl využít informací z předchozí operace zanechané ve vyrovnávací paměti a vytvořit odpověď velice rychle. Protože tabulka úředníků je velice malá v porovnání s účty, měla by být nahrána do paměti RAM. I tak její velikost někdy způsobuje problémy se zámky. Poslední update se provádí nad extrémně malou tabulkou poboček. Ta je ve většině případů také nahrána celá do paměti RAM a tedy přístup k ní by měl být velice rychlý. Poslední operace transakce vkládá do tabulky historie. Do této tabulky se vždy jen vkládají data a nikdy v ní není nic upravováno nebo čteno. Tabulka také neobsahuje žádné indexy a proto doba zápisu je relativně malá oproti úpravám předchozích tabulek a jejich indexů.

```
\set nbranches :scale
\set ntellers 10 * :scale
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom bid 1 :nbranches
\setrandom tid 1 :ntellers
\setrandom delta -5000 5000
BEGIN;
UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid =
    :aid;
SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid =
    :tid;
UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid =
    :bid;
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES
    (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
END;
```

Obrázek 8.4.1 - Pgbench TPC-B (sort of) [17]

Kromě výchozího skriptu pgbench obsahuje ještě skript zaměřený na operaci update. Transakce provádí úpravy nad tabulkami úředníků a poboček. Druhou transakcí provádí pouze operaci select nad tabulkou účtů. Testováním posledního typu transakce může být výhodné k prozkoumání vyrovnávací paměti a zjištění maximální rychlosti.

8.5 pgFouine

Aplikace pgFouine patří mezi nejpopulárnější analyzátory logovacího souboru databáze PostgreSQL. Skript je vytvořen v jazyce PHP. Tedy k jeho chodu je potřeba PHP kompilátor a některé další knihovny, které nejsou přímo přibaleny v instalaci PHP. Právě proto může být zprovoznění programu trochu složitější zejména u systému Windows a starších unixových systémů. Jelikož aplikace analyzuje logovací soubor, nemusí být nainstalována na stejném počítači, jako je databázový server [17].

Po spuštění ve výchozím nastavení je vytvořena html stránka obsahující agregovaná data získaná ze všech databázových operací zaznamenaných v logu. Mezi tato data patří celková statistika, typy provedených dotazů a jejich počet, nejpomalejší dotazy, dotazy s nejdelší vykonávací dobou

(doba záleží na době vykonávání jednoho dotazu a celkovém počtu vykonaných dotazů) a nejčastější dotazy. Kromě agregovaných dat obsahuje report i nenormalizované nejpomalejší dotazy. Každá tabulka agregovaných dat obsahuje deset normalizovaných dotazů. Každý normalizovaný dotaz obsahuje tři nenormalizované dotazy totožné s logem. Normalizovaný dotaz má nahrazeny všechny znakové řetězce (nahrazeno za ‘’) a číselné hodnoty (nahrazeno za 0). Na řádku tabulky je dále viditelný počet vykonání, průměrný čas dotazu, celková doba vykonávání.

Vstup i výstup programu je možné nadefinovat pomocí různých přepínačů. Je možné zvolit například výstupní typ souboru (text, html nebo html s grafy), počet normalizovaných a nenormalizovaných dotazů v rámci tabulky, typy agregovaných dat. Jedny z nejdůležitějších přepínačů definují typ (csv, stderr, syslog), název vstupního logovacího souboru a dotazy, které se mají zpracovávat (datové rozmezí dotazů, typ dotazů).

9 Testovací a vyhodnocovací program

V minulé kapitole byl popsán program pgbench na měření a porovnávání výkonu databáze. Problémem toho programu je nemožnost jednoduchého spuštění více testů pro různé počty připojených klientů a různě nastavenou škálovatelnost. Tento nedostatek řeší z určité míry rozšíření pgbench-tools. Jedná se o bash aplikaci, která dokáže spouštět více testů pgbench s předem zvoleným nastavením. Výsledky poté ukládá a zobrazuje v grafové podobě. Další nevýhodou pgbench skriptu je testování nad předem definovanou databází (schéma TPC-B). Pokud používaná databáze se neshoduje se schématem TPC-B, je velice pravděpodobné, že testování pomocí TPC-B nebude vůbec odpovídat reálné rychlosti používané databáze. Pgbench sice umožňuje testování vlastním skriptem, bohužel rozšířené testování je zbytečně složité. Kvůli těmto problémům a možnému usnadnění při testování výkonnosti databáze byl vytvořen v rámci této diplomové práce nový program. Skript je postaven nad aplikacemi pgbench_tools (verze 0.6) a pgfouine (verze 1.2). Díky kombinaci těchto dvou programů a přidané logice je možné získat výsledky vypovídající o reálné rychlosti jakékoli databáze s logem provedených operací.

Popis implementace je rozepsán do tří podkapitol. Nejdříve je popsána instalace a příprava testů před spuštěním, poté následuje podkapitola testování. V poslední kapitole je uvedeno zpracování výsledků včetně vytvoření grafů a reportů.

9.1 Instalace a příprava testů

Dříve než je možné začít s testy, je nutné skript připravit. Aby mohl program bezchybně fungovat, je zapotřebí mít nainstalovaný program pgbench, gnuplot, python a PHP interpret. Pomocí pgbench probíhají testy s PostgreSQL databází. Gnuplot slouží k tvorbě grafů z výsledných hodnot. Python vykonává skript na zpracování hodnot latence. PHP interpret se stará o vykonávání skriptu na analýzu logovacího souboru databáze. Instalace testovacího a vyhodnocovacího programu se provede zkopírováním složky „pgbench-with-fouine“ na distribuci linuxu a nainstalováním požadovaných programů (funkčnost byla testována na distribuci CentOS 5.5).

Po úspěšné instalaci všech programů musíme nastavit konfigurační soubor „config“. Ten definuje připojení k testované databázi a databázi uchovávající výsledky testů, logovací soubor testované databáze a typy testů ke spuštění. Databáze s výsledky může být uložena na odlišném serveru, než kde probíhají testy. Schéma databáze je nutné vytvořit pomocí předem vytvořeného SQL skriptu „resultdb.sql“ (schéma je podrobněji popsáno v následující kapitole 9.2).

Aby bylo možné využít generování transakcí z logovacího souboru, je nutné zapnout logování do souboru. V konfiguračním souboru PostgreSQL databáze můžeme nastavit logování pomocí výpisu přes standardní chybový výstup (stderr), který je přesměrovaný do souboru nebo pomocí systémového logu. Při využití chybového výstupu je ještě nutné upravit formát výpisu pomocí atributu log_line_prefix na formát:'%t [%p]: [%l-1]. Doporučovaným nastavením je použití systémového logování. Při ukládání přes chybový výstup není garantována konzistence víceřádkových operací [19].

Testy jsou rozděleny na dva hlavní typy. Prvním typem jsou testy typu TPC-B. Tyto testy nepoužívají analýzu logu. Do toho typu lze zařadit i testy, kdy si vytvoříme předem transakční soubor, který bude použit při testech. V tomto případě jde testovat předem zvolenou databázi s jakýmkoli schématem. Druhým typem jsou automaticky vytvořené transakce na základě analýzy

databázového logu. Pro tento typ je třeba v konfiguračním souboru zvolit, z kterých operací chceme transakci vytvořit. Na výběr máme z nejčastějších, nejpomalejších nebo průměrně nejdelších dotazů. Dále volíme jejich počet v rámci jedné transakce. Všechny dotazy jsou vloženy v normalizované podobě a je potřeba k nim zvolit rozmezí hodnot, které se budou v testech dosazovat. Tyto hodnoty se volí z předem zvoleného počtu konkrétních operací z logovacího souboru.

Další nastavení definuje, kolik testů se má provést. K určení různého počtu klientu se používá pole hodnot s počty klientů. Například hodnota "1 2 4" provede každý test s počtem klientů 1,2 a 4. Dále lze nastavit počet opakování každého testu. Pro testování je výhodné spustit stejný test vícekrát, aby se ověřilo, zda jsou výsledky stejné. Škálovatelnost se nastavuje polem hodnot stejně jako počet klientů. Spolu s ní je možné nastavit inicializaci databáze před každým testem. Oba parametry (inicializace a škálovatelnost) fungují pouze při použití TPC-B testů. Další parametr nastavuje počet obsluhujících vláken. Toto nastavení je výhodné při více jádrovém procesoru a více klientském testu, kdy každé vlákno zpracovává jednoho klienta. Posledním důležitým nastavením je délka testu. Ta může být buď určena počtem vykonaných transakcí, nebo dobou testování. Zbylé parametry určují názvy pomocných souborů, které jsou vytvářeny při testech. Všechny dočasné soubory jsou na konci testů uklizeny.

9.2 Testování

V předchozí podkapitole byl uveden popis k přípravě testovacího skriptu a jeho konfiguračního souboru. Nyní je možné spustit testování pomocí souboru „runTest". Skript „runTest" byl upraven v rámci diplomové práce a vychází ze skriptu „runset" programu pgbench-tools. Oproti výchozí verzi skript umožňuje testování pomocí analýzy logovacího souboru. Po spuštění skript načte hodnoty konfigurace a otestuje připojení k testovací databázi a databázi uchovávající výsledky. Dále se kontroluje přítomnost potřebných programů (pgbench a PHP interpret) a souboru (databázový log). PHP interpret a log je vyžadován a kontrolován pouze pokud je spuštěn test s analýzou databázového logovacího souboru. Tento test následně spustí skript „transactionMaker" na vygenerování transakčního souboru, který je později předán programu pgbench. TransakctionMaker generuje ve výchozím nastavení transakční soubor obsahující v názvu časové razítko a tím zabraňuje jeho ztrátě nebo přepsání při spuštění stejného testu vícekrát. Nové transakční soubory se ukládají do adresáře se skripty (přesněji podadresář pgfouine) a můžeme je možné kdykoli znovu využít.

Skript na tvorbu nových transakčních souborů spolu s úpravou aplikace pgfouine byl vytvořen v rámci této diplomové práce. Implementace vychází z programu pgfouine verze 1.2. Doimplementovaná část nabízí nový formát („text-with-variables") uložení výsledků. Formát obsahuje agregované operace stejně jako formát „text". Dále oproti formátu „text" přidává hodnoty intervalů jednotlivých číselných proměnných nacházejících se v operacích. Na obrázku 9.2.1 je vidět pomocný soubor se šesti dotazy vygenerovanými pomocí upraveného pgfouine skriptu. Hodnoty intervalů byly vytvořeny z minimálních a maximálních hodnot deseti provedených operací. Každá číslice 0 označuje jednu proměnnou. První řádek „ExampleVariable" označuje první proměnou. Každá další proměnná je uložená na novém řádku.

```
#####  Most frequent queries (N)  #####

1) 40 - 0.3s - UPDATE pgbench_accounts SET abalance = abalance + 0
     WHERE aid = 0;
ExampleVariable Min: -2373 Max: 4143
ExampleVariable Min: 1719 Max: 90928
--
2) 40 - 0.0s - END;
--
3) 40 - 0.0s - INSERT INTO pgbench_history (tid, bid, aid, delta,
    mtime) VALUES (0, 0, 0, 0, CURRENT_TIMESTAMP);
ExampleVariable Min: 2 Max: 10
ExampleVariable Min: 1 Max: 1
ExampleVariable Min: 1719 Max: 96503
ExampleVariable Min: -2373 Max: 4143
--
4) 40 - 0.0s - UPDATE pgbench_tellers SET tbalance = tbalance + 0
     WHERE tid = 0;
ExampleVariable Min: -2462 Max: 4143
ExampleVariable Min: 1 Max: 10
--
5) 40 - 0.0s - SELECT abalance FROM pgbench_accounts WHERE aid = 0;
ExampleVariable Min: 1719 Max: 96503
--
6) 40 - 0.0s - UPDATE pgbench_branches SET bbalance = bbalance + 0
     WHERE bid = 0;
ExampleVariable Min: -4203 Max: 4143
ExampleVariable Min: 1 Max: 1
--
```

Obrázek 9.2.1 - Pomocný soubor s nejčastějšími dotazy

Po vytvoření pomocného souboru je tento soubor zpracován skriptem transactionMaker, který z něj vytvoří transakční soubor splňující specifikaci programu pgbench. Transakční soubor je zobrazen na obrázku 9.2.2. V konfiguračním souboru bylo nastaveno vytvoření transakce ze tří operací. Protože jsou při testování důležité jen operace select, insert, update a delete, byl vytvořen pomocný soubor s dvojnásobkem potřebných operací. Takto se předpokládá, že v dvojnásobku bude vždy přítomný potřebný počet CRUD operací. Jak je vidět z obrázku 9.2.1 druhá operace byla při tvorbě transakce vynechána. Všechny proměnné jsou definovány na začátku souboru písmenem V a číslem. Zbytek souboru tvoří transakce s operacemi.

```
\setrandom V0 -2373 4143
\setrandom V1 1719 90928
\setrandom V2 2 10
\setrandom V3 1 1
\setrandom V4 1719 96503
\setrandom V5 -2373 4143
\setrandom V6 -2462 4143
\setrandom V7 1 10
begin;
update pgbench_accounts set abalance = abalance + :V0 where aid =
:V1;
insert into pgbench_history (tid, bid, aid, delta, mtime) values
    (:V5, :V2, :V3, :V4, current_timestamp);
update pgbench_tellers set tbalance = tbalance + :V6 where tid =
:V7;
end;
```

Obrázek 9.2.2 - Transakční soubor

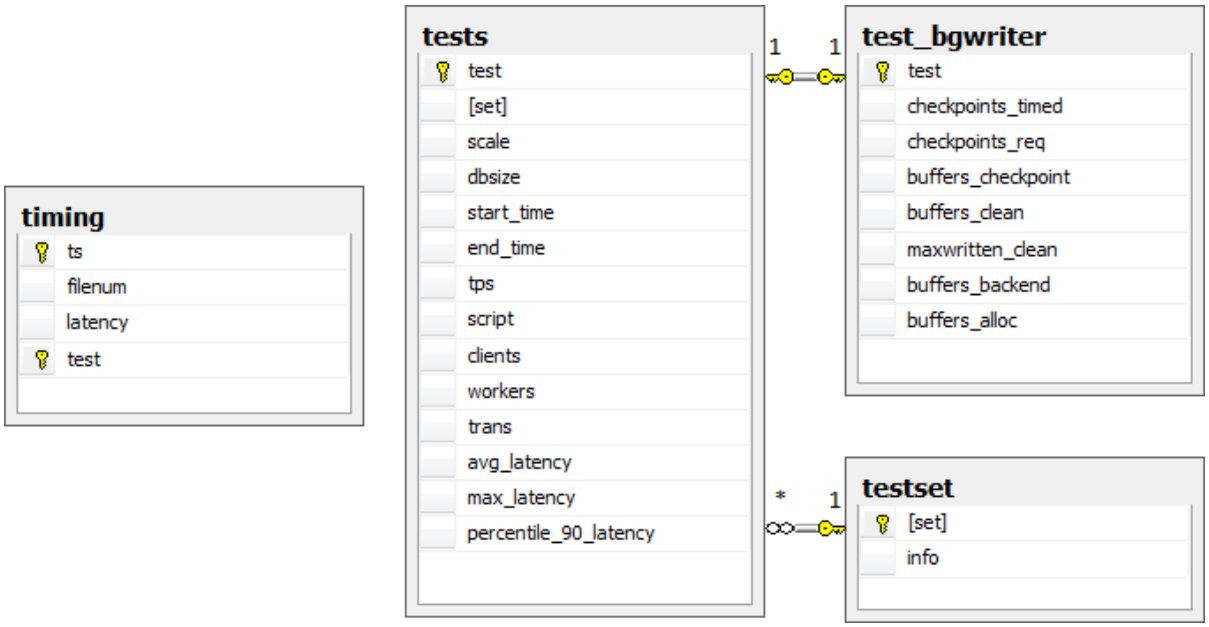
Jakmile skončí skript „transationMaker“ a transakční soubor je úspěšně vytvořen přechází se k testovací fázi. Pokud nastane jakákoli chyba během chodu skriptu, je vypsáno chybové hlášení a skript je přerušen. Testovací část je společná pro testování s i bez analýzy logovacího souboru. Testování probíhá sekvenčně pro každou kombinaci počtu klientů, testovacích pokusů a hodnot škálovatelnosti (škálovatelnost je použita pouze při testech bez analýzy logu). Každý test je vykonáván skriptem „benchmark“. Tento skript byl upraven v rámci diplomové práce a vychází ze skriptu „benchwarmer“ programu pgbench-tools. Úprava umožňuje použití vygenerovaného transakčního souboru a uchovávání logovacích souborů. Získané výsledky jsou ukládány do databáze a do předem zvolené složky.

Všechny hlavní výsledky jsou uloženy ve zvolené databázi. Schéma databáze je vidět na následujícím obrázku 9.2.3. Schéma obsahuje čtyři tabulky, z toho jedna tabulka je použita pro pomocné výpočty a zbylé tři slouží k permanentnímu uchování výsledků. Dočasná tabulka „timing“ je využívána v testech k nahrání všech záznamů z logu jednoho testu pgbench. Tyto data poté slouží k vytvoření grafu hodnot TPS v čase a grafu latence v čase. Z dat se také počítá hodnota průměrné a maximální latence, která je následně zapsána v tabulce „tests“. Dlouhodobé uchovávání těchto dat není důležité, a proto je tabulka na konci každého testu vymazána.

Při přechodu do testovací části je vždy vytvořen záznam do tabulky „testset“, která uchovává sady testů. Záznam obsahuje číslo sady a popis. Do popisu jsou ve výchozím nastavení vloženy základní informace sady testů (cesta k transakčnímu souboru, hodnoty počtu klientů a škálovatelnosti, počet opakování a počet zpracovávajících vláken). Pro každý test je vytvořen údaj v tabulce „tests“ a „test_bgwriter“. Tabulka „tests“ uchovává všechny informace ohledně vykonaného testu. Obsahuje číslo testu (test), referenci na testovou sadu (set), hodnotu škálovatelnosti (scale - pro testy s analýzou logu je tato hodnota vždy rovna 1), velikost databáze (dbsize), počáteční a koncový čas testu (start_time, end_time), průměrnou hodnotu transakcí za sekundu (TPS), cestu k transakčnímu souboru (script), počet klientů (clients), počet obsluhujících vláken (workers), počet vykonaných transakcí (trans), průměrnou latenci, maximální latenci a průměrnou hodnotu latence z nejmenších 90% hodnot.

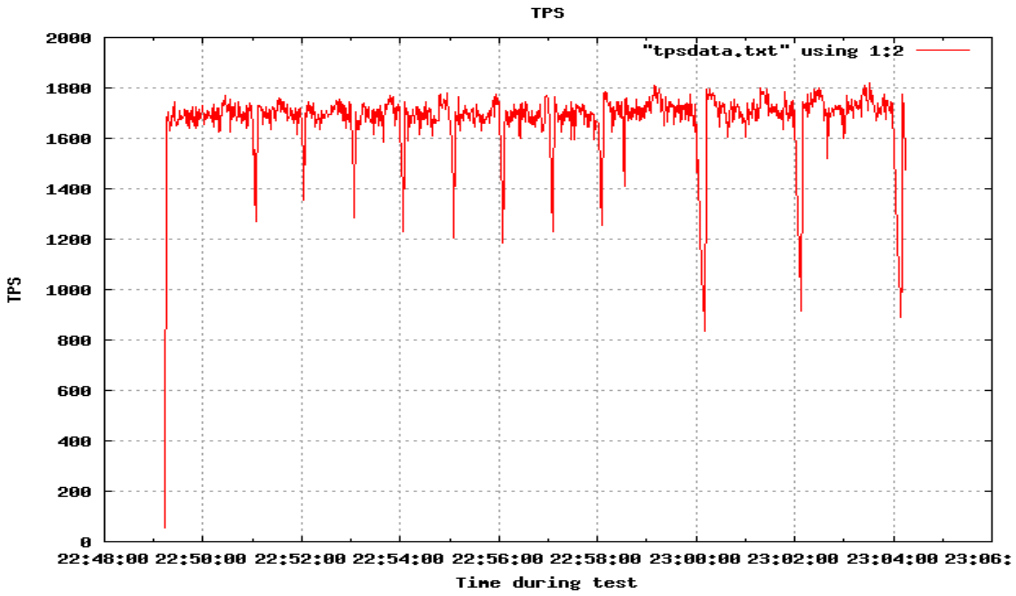
Tabulka „test_bgwriter“ je ve vztahu 1:1 s tabulkou „tests“ a uchovává hodnoty PostgreSQL databáze vypovídající o využívání kontrolních bodů (checkpoint) a vyrovnávací paměti (buffer). Tedy statistika ukazuje, jak moc bylo během testu zapisováno na disk. Popis atributů vychází z [17]. Sloupec „test“ jednoznačně identifikuje dokončený test. Kontrolní body slouží k redukování času potřebného k zotavení při pádu databáze. Kvůli tomu požadujeme, aby byly vykonávány co nejčastěji. Na druhou stranu velké množství kontrolních bodů má za následek častý zápis na disk a tím způsobuje zpomalení systému. Četnost tvorby kontrolních bodů se dá nastavit pomocí dvou parametrů. První definuje potřebný počet zapsaných WAL segmentů k vyžádání vytvoření dalšího kontrolního bodu. Vytvoření checkpointu se poté připisuje do statistiky „checkpoints_req“ (nastaven proměnou checkpoint_segments). Do sloupce „checkpoints_timed“ se zapisují vytvořené kontrolní body vynucené uplynutím určitého časového intervalu (nastaven proměnou checkpoint_timeout). Hodnoty „buffers_checkpoint“, „buffers_backend“ a „buffers_clean“ zaznamenávají tři možné způsoby zápisu upraveného bloku dat z paměti na disk. První typ se týká vyčištění všech upravených bloků z paměti při vytvoření kontrolního bodu (počet bloků je zaznamenán do sloupce „buffers_checkpoint“). Pokud jakýkoli proces kromě procesu starajícího se o kontrolní body nahrává data do vyrovnávací paměti a musí stávající upravená data zapsat na disk, připisuje se tato statistika k „buffers_backend“. Posledním typem je uklizení bloků dat pomocí procesu „background writer“. Ten se snaží uvolnit nepoužívaná data z paměti pro další možné alokování v budoucnu. Počet bloků je poté připsán do „buffers_backend“. Hodnota „maxwritten_clean“ udává počet přerušení čistícího

skenování kvůli zápisu více bufferů, než specifikuje „bgwriter_lru_maxpages“ [21] (hodnota udává maximální počet zápisů bufferů na pozadí [22]). Poslední atribut udává počet požadovaných bloků dat. Pokaždé když je vyžadován blok dat v paměti, ať už tam již je nebo se musí načíst, je čítač „buffer_alloc“ inkrementován.

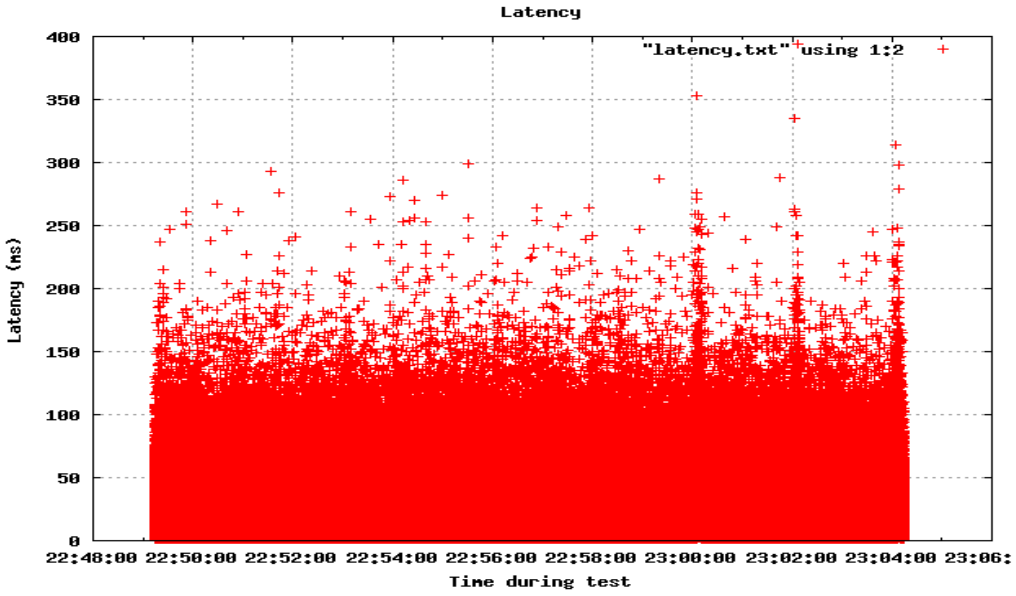


Obrázek 9.2.3 - Schéma databáze s výsledky

Další část výsledků se ukládá přímo do souborů na disk. Jedná se o informace z programu pgbench, jeho logy (pokud je zapnuta volba ukládání v konfiguračním souboru) a vygenerované grafy s html stránkou. V konfiguračním souboru se volí složka, do které se budou ukládat všechny výsledky. Skript vytváří hierarchickou strukturu, kde každý test je uložen do vlastní podsložky. Jméno podsložky je tvořeno číslem definující test (shodné číslo s atributem „test“ v databázovém schématu). Podsložka testu obsahuje vygenerované grafy TPS a latence v čase pomocí aplikace gnuplot. Graf TPS je zobrazen na obrázku 9.2.4. V grafu jsou vidět výkyvy způsobené například zátěží více prioritních procesů. Opakované výkyvy mohou být způsobeny přestavováním indexů databáze a výměnou dat ve vyrovnávací paměti. Graf latence je zobrazen na obrázku 9.2.5. Až na menší výjimky byla latence při testu velmi nízká. Vysoká latence se poté odrazuje v grafu TPS. Dále je ve složce uloženo nastavení PostgreSQL databáze a výsledek vygenerovaný aplikací pgbench. Poslední soubor je html stránka zobrazující grafy a odkaz na soubory výsledku a konfigurace. Pokud je zapnuto ukládání logu, je vytvořena složka log s pomocnými soubory (soubory obsahují data sloužící k výpočtu TPS a latence). Systém výpočtu TPS a ukládání dat do databáze a na disk byl převzat z aplikace pgbench-tools. V rámci diplomové práce byly přidány do konfiguračního souboru proměnné na definování vlastního úložiště výsledků a zaznamenávání logovacích souborů. Po ukončení testů je spuštěn skript na vytvoření agregovaných výsledků. Tento skript je popsán v následující podkapitole 9.3.



Obrázek 9.2.4 - Graf TPS v čase



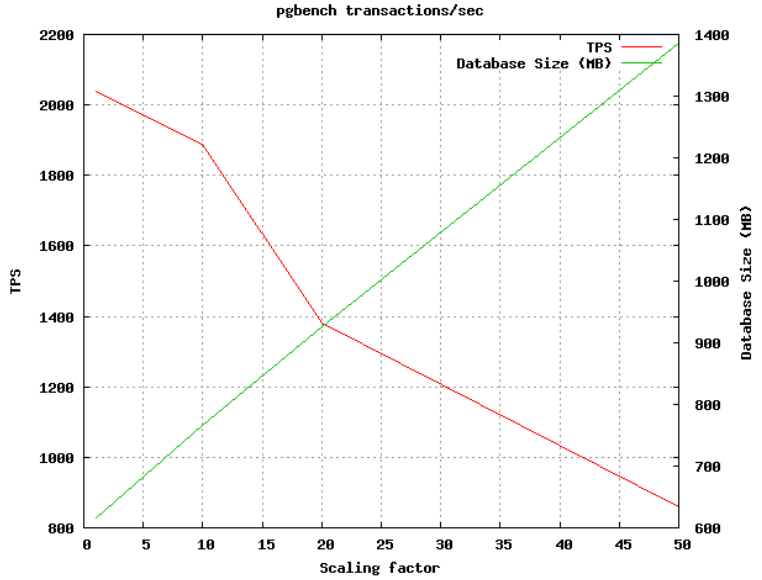
Obrázek 9.2.5 - Graf latence v čase

9.3 Zpracování výsledků

Po ukončení celé sady testů je spuštěn skript „webreport“ zajišťující vytvoření hlavního HTML souboru obsahujícího agregované statistiky všech testů. Skript byl upraven v rámci diplomové práce a vychází ze souboru „webreport“ aplikace pgbench-tools. Oproti původní verzi skriptu bylo přidáno generování grafů pro každou sadu testů a tvorba tabulky s výsledky ukončenými chybou. Vytvořený hypertextový soubor je uložen do kořenového adresáře výsledků testů. Spolu s ním jsou vygenerovány tři typy agregačních grafů. První typ grafu zobrazuje závislost TPS a velikosti databáze na hodnotách škálovatelnosti (graf je zobrazen je při testování bez analýzy logu). Další graf ukazuje vývoj hodnoty TPS k narůstajícímu počtu klientů. Posledním typem je spojení předchozích dvou dvoudimenzionálních grafů do jednoho třídimenzionálního. Graf vytváří 3D síť, kde na ose y je

hodnota TPS, na ose x je počet klientů a na ose z jsou hodnoty škálovatelnosti. Všechny grafy jsou generovány pomocí aplikace gnuplot. Skripty k jejich generování jsou uloženy ve složce skriptů a podsložce „plots“.

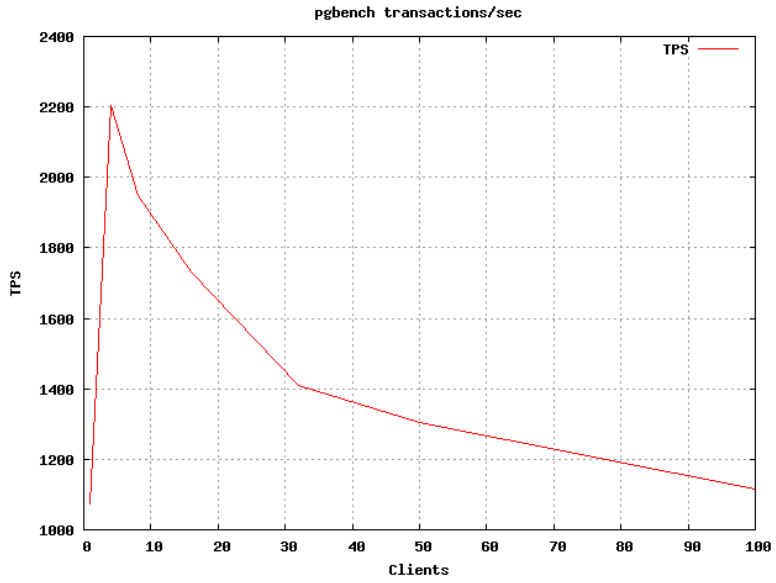
Graf závislosti TPS a velikosti databáze (database size) na hodnotách škálovatelnosti (scaling factor) je zobrazen na obrázku 9.3.1. Křivka TPS je vždy charakteristická podobným tvarem, liší se vždy jen její posunutí a sklon křivky v jednotlivých fázích. Graf obsahuje tři fáze, první a zároveň nejrychlejší z nich je fáze výpočtu všech transakcí v paměti. Jelikož se jedná o malou databázi, všechna potřebná data jsou nahrána do paměti a není potřeba při dotazech nahrávat další data z disku. Tato část je zakreslena v grafu od škálovatelnosti 1 do hodnoty 10. Jak se zvětšuje databáze, není možné mít přístupné všechny data v paměti a občas je tedy nutné nahraní dat z disku. Tato fáze je zobrazena intervalem škálovatelnosti od hodnoty 10 po hodnotu 20. Rychlost TPS se zmenšuje díky operacím s diskem. Poslední fáze je část křivky od hodnoty škálovatelnosti 20. Zde se již většina dat musí nahrát z disku a proto je tato část nejpomalejší. Zelená přímka ukazuje lineární nárůst velikosti databáze se zvětšující se škálovatelností.



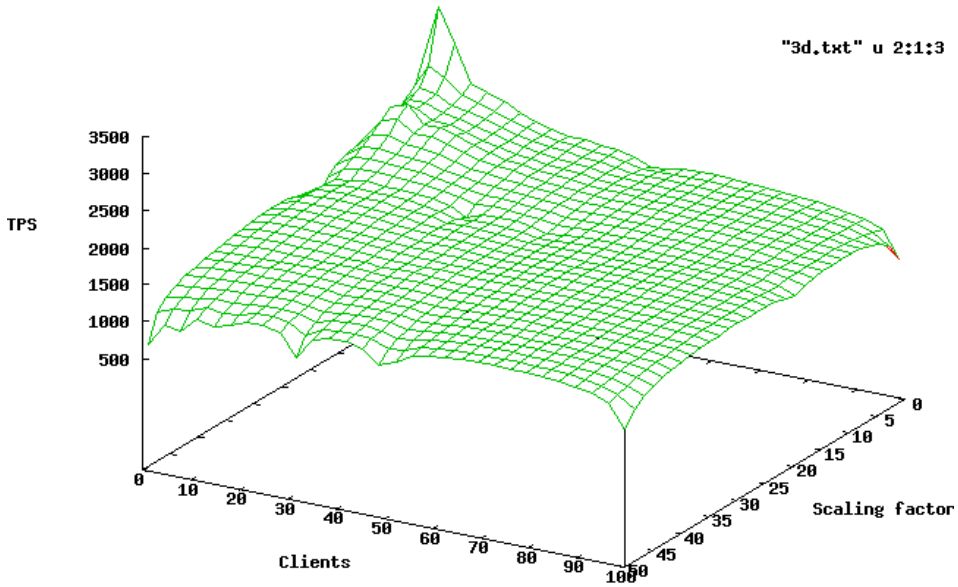
Obrázek 9.3.1 - Graf závislosti TPS a velikosti databáze na hodnotách škálovatelnosti

Druhý typ grafu je zakreslen na obrázku 9.3.2. Graf zobrazuje závislost TPS na počtu klientů (clients) dotazujících se databáze. Na grafu je patrné, že větší počet klientů zvyšuje rychlost databáze. Nejvyšší rychlost je dosažena při pěti klientech, kdy velká část dotazů je zpracována velice rychle díky využití vyrovnávací paměti. Díky provádění podobných dotazů se jejich zpracování optimalizuje a využívá se co nejvíce urychlení přes vyrovnávací paměť. S rostoucím počtem klientů roste i zátěž na databázový systém a velikost sdílené paměti. Proto pro větší počet klientů se TPS zmenšuje.

Posledním typ grafu je zachycen na obrázku 9.3.3. Graf zobrazuje závislost TPS na počtu klientů (clients) a hodnotách škálovatelnosti (scaling factor). Vygenerování grafu vzniká 3D síť pokrývající všechny provedené testy. Osy jsou směřovány tak, aby nejvyšší zátěž, a tedy i nejnižší hodnota TPS, byla zobrazena ve spodní pravé části grafu. Naproti tomu levý horní roh znázorňuje nejrychlejší část. Díky tomu graf představuje plochu klesající z levého horního rohu do pravého dolního rohu. Barva sítě určuje stranu plochy, horní strana je zbarvena zeleně a spodní červeně. Ve většině případů by měl být graf zelený, červená barva by znamenala nějaké neočekávané chování (například část testů byla ovlivněna vysokou zátěží jiných procesů).



Obrázek 9.3.2 - Graf závislosti TPS na počtu klientů



Obrázek 9.3.3 - 3D graf závislosti TPS na počtu klientů a hodnotách škálovatelnosti

Hlavní hypertextový dokument obsahuje kromě grafů tabulky výsledků všech testů zapsaných do databáze s výsledky. Skript prochází všechny přítomné sady testů v databázi a pro každý z nich vytváří předem zmíněné grafy a tabulky s výsledky. Začátek dokumentu obsahuje grafy z úplně všech sad testů. Pro každou sadu testů vzniká tabulka porovnávající průměrnou hodnotu TPS vzhledem k hodnotám škálovatelnosti, k počtům klientů a k hodnotám škálovatelnosti i počtům klientů zároveň. Tyto agregační tabulky vždy počítají průměrnou hodnotu TPS pro všechny testy s určitou hodnotou škálovatelnosti nebo s určitým počtem klientů nebo s obojím. Tabulky obsahují číslo sady testů, hodnotu, na kterou se tabulka zaměřuje (škálovatelnost, počet klientů, obojí), hodnotu TPS, průměrnou hodnotu latence, průměrnou hodnotu nejnižších 90% latence a maximální latenci. Další tabulkou v sadě testů je detailní tabulka jednotlivých testů obsahující kromě škálovatelnosti, počtu klientů, TPS a maximální latence ještě databázové informace ohledně kontrolních bodů a vyrovnávající paměti. V sadě testů se může objevit ještě jedna tabulka obsahující testy ukončené chybou. Pokud nenastala při testování žádná chyba, tabulka je vynechána.

Kromě automaticky generovaného záznamu je možné získat výsledky použitím předem vytvořených skriptů v podadresáři „reports“. Tyto soubory byly převzaty z programu pgbench-tools. Skripty slouží k dotazování databáze a získání výsledků i při spuštěném testování. Skripty získávají informace ohledně využívání vyrovnávací paměti, nejrychlejších testů a agregovaných výsledků.

10 Testy a ladění výkonu

Kapitola testů a ladění výkonu popisuje provedené testy pro různé typy konfigurací. Testy byly provedeny pomocí programu popsaného v 9. kapitole. Každý test byl spuštěn třikrát, aby byly hodnoty co nejvěrohodnější. Při porovnáváním výsledků mezi různými konfiguracemi se vždy vybral medián všech stejných testů. Oproti průměru má medián výhodu v nezapočítání krajních hodnot, které můžou být ovlivněny různými vlivy (velká latence, jiné procesy využívají systémové zdroje a podobně). Kromě opakování testů je také důležitá doba jednotlivých testů. Pro všechny testy byla proto zvolena dostatečně dlouhá doba patnácti minut.

Nejdříve je popsáno ladění výkonnosti databáze PostgreSQL pomocí jeho konfiguračního souboru. Kapitola nejdříve popíše nejdůležitější nastavení a poté je předvede pomocí tří srovnávacích testů. Další kapitola se zaměřuje na porovnání optimalizovaných PostgreSQL databází na čtyřech rozdílných hardwarových konfiguracích. Třetí kapitola srovnává výkonnost databázového systému PostgreSQL a Oracle. Poslední kapitola testuje zrychlení databázového systému pomocí použití paralelní techniky zvané partitioning.

10.1 Testy a ladění konfigurace PostgreSQL

Tato kapitola je rozdělena na dvě části. První z nich popisuje hlavní konfigurační soubor „postgresql.conf“ obsahující důležitá nastavení ovlivňující výkonnost. K jednotlivým nastavením uvádí doporučené hodnoty podle [17] a [23]. Druhá část testuje upravené parametry konfiguračního souboru. Testování obsahuje tři databázové konfigurace, na kterých byla spuštěna sada testů s různými počty klientů a nad různě velikou databází.

10.1.1 Konfigurační soubor PostgreSQL

Dříve než je možné spustit řadu testů na otestování PostgreSQL databáze, je třeba popsat důležité proměnné konfiguračního souboru databáze „postgresql.conf“ ovlivňující výkonnost. Konfigurační soubor obsahuje mnohá nastavení, kdy některá se mohou měnit podle verze databáze. Pro všechny testy byla proto zvolena nejnovější stabilní verze PostgreSQL 9.03 (platné pro 1. 3. 2011). Soubor se nachází v kořenovém adresáři databáze (adresář je přístupný přes proměnnou $PGDATA).

Nastavení se dělí na serverovou a klientskou část. Serverová část definuje připojení k databázi, sdílenou paměť, logování, úklid, kontrolní body a WAL („Write-Ahead Log“). Nejdůležitější klientské proměnné nastavují vyrovnávací a pracovní paměť přidělenou klientům a způsob synchronizace dat paměti s diskem. Dále lze dělit nastavení podle doby, kdy začnou působit jejich upravené hodnoty. Ke každé proměnné je přiřazena úroveň flexibility („level of flexibility“). Ve většině případů je vyžadována právě nejvyšší úroveň „Postmaster“, která potřebuje restart celého databázového serveru. Příkladem jiné úrovně je „Superuser“, kdy superuživatel může měnit proměnné za chodu serveru [23].

Nyní budou popsána serverová nastavení konfiguračního souboru. Prvním důležitým parametrem je připojení k databázi, to je definováno proměnou „listen_addresses“, která ve výchozím nastavení povoluje pouze lokální připojení (pro neomezené připojení je třeba nastavit hodnotu „*“). Proměnná „max_connections“ nastavuje maximální počet připojení k databázi. Sdílená paměť se nastavuje pomocí proměnné „shared_buffers“. Ta definuje kolik paměti je dedikováno pro

vyrovnávací paměť PostgreSQL serveru. Ve výchozím nastavení je tato hodnota velice nízká, doporučuje se hodnota ¼ RAM paměti [23]. Vytváření kontrolních bodů je určeno proměnnou „checkpoint_segments“, která stanovuje počet zapsaných WAL segmentů (každý má velikost 16MB [17]) k vytvoření kontrolního bodu. Výchozí hodnota je 3, nicméně vytváření kontrolních bodu po každých 48MB (3*16) může představovat úzké hrdlo systému. Proto se využívají hodnoty od 32 po 256 podle velikosti a rychlosti databázového systému [23]. Kontrolní body ovlivňuje i proměnná „checkpoint_completion_target“ definující rozprostření zápisu kontrolního bodu v čase. Hodnota definuje, do kdy má být kontrolní bod vytvořen vůči procentuálnímu stavu následujícího kontrolního bodu. Doporučené nastavení je 0.9 (tedy 90% následujícího kontrolního bodu) [17]. PostgreSQL databáze potřebuje pro svůj chod průběžný úklid pomocí „autovacuum“ procesu. Tomu lze definovat maximální počet úklidových procesů („autovacuum_max_workers“) a maximální velikost paměti („maintainance_work_mem“). Typickým vyšším nastavením velikosti paměti „maintainance_work_mem“ při předpokladu, že počet procesů zůstal stejný, je 5% z celkové velikost RAM paměti [17]. Důležitým faktorem pro rychlé vykonávání operací jsou statistiky vytvářené analýzou dotazů. Analýza je také zajišťována procesem „autovacuum“ a lze definovat pomocí hodnoty „default_statistics_target“. Proměnná určuje množství informací ukládaných analyzačním procesem. Zvýšením hodnoty se zvětšuje doba analýzy a zatížení databáze, ale zlepšuje se vykonávací plán dotazů nad tabulkami. Integritu dat zajišťuje WAL. Jeho nastavení můžou mít dopad na výkonnost databáze. Parametrem WAL je hodnota „wal_buffers“ definující velikost paměti pro WAL data v rámci celkové sdílené paměti. Proměnná by měla být dostatečně velká, aby dokázala uchovat všechny operace v rámci jedné transakce. Výchozí hodnota je 64kB. Při velkém zatížení způsobeným zápisem, je pro optimální výkon lepší zvětšit hodnotu na 1MB a více [17]. Další důležitou částí nastavení je logování. Pomocí konfiguračního souboru je možné nastavit mnohé proměnné. Mezi nejpoužívanější patří typ logování (příklad: systémový log, přesměrování standardního chybového výstupu), cílové místo logování a struktura logu. Další využívané nastavení je zápis do logu všech provedených operací překračujících minimální doby zpracování „log_min_duration_statement“.

Všechny následující proměnné mohou být nastaveny klienty. Výchozí nastavení se načítá opět z konfiguračního souboru „postgresql.conf“. Hodnota „effective_cache_size“ určuje, jak moc velký prostor má použít operační systém pro vyrovnávací paměť na disku. Standardní nastavení se pohybuje mezi ½ až ¾ RAM paměti [23]. Parametr „synchronous_commit“ určuje, jestli se budou transakce hned zapisovat na disk. Ve výchozím stavu je synchronní commit zapnutý. Toto nastavení představuje velkou zátěž fyzických disků a tedy i úzké hrdlo při operacích commit. Standardním nastavením je vypnutí tohoto přepínače, což má za následek zrychlení systému a použití asynchronního commitu. Asynchronní commit garantuje zápis transakce na disk do maximálně trojnásobku hodnoty „wal_writer_delay“ (ve výchozím nastavení se jedná o hodnotu 600 milisekund) [17]. Pokud by nastal výpadek serveru během této doby, nebude tato transakce obnovena, ale zároveň není porušena integrita databázového systému. Další důležitou hodnotou je velikost pracovní paměti „work_mem“ rezervované pro každého klienta. Tato paměť se využívá, když operace potřebuje třídit dotazovaná data. Pokud je paměť nedostatečná, použije se diskový paměťový prostor. Jestliže databázový server má dostatek paměti, může zvýšení parametru „work_mem“ zásadně zrychlit výkon celého serveru. Proměnná „random_page_cost“ napovídá optimalizátoru, jestli má upřednostňovat sekvenční hledání před použitím přístupu pomocí náhodných indexů. Pro rychlé disky (RAID pole) je doporučováno snížení výchozí hodnoty (4.0) a tím podpoření přístupu přes náhodné indexy [23].

10.1.2 Testování

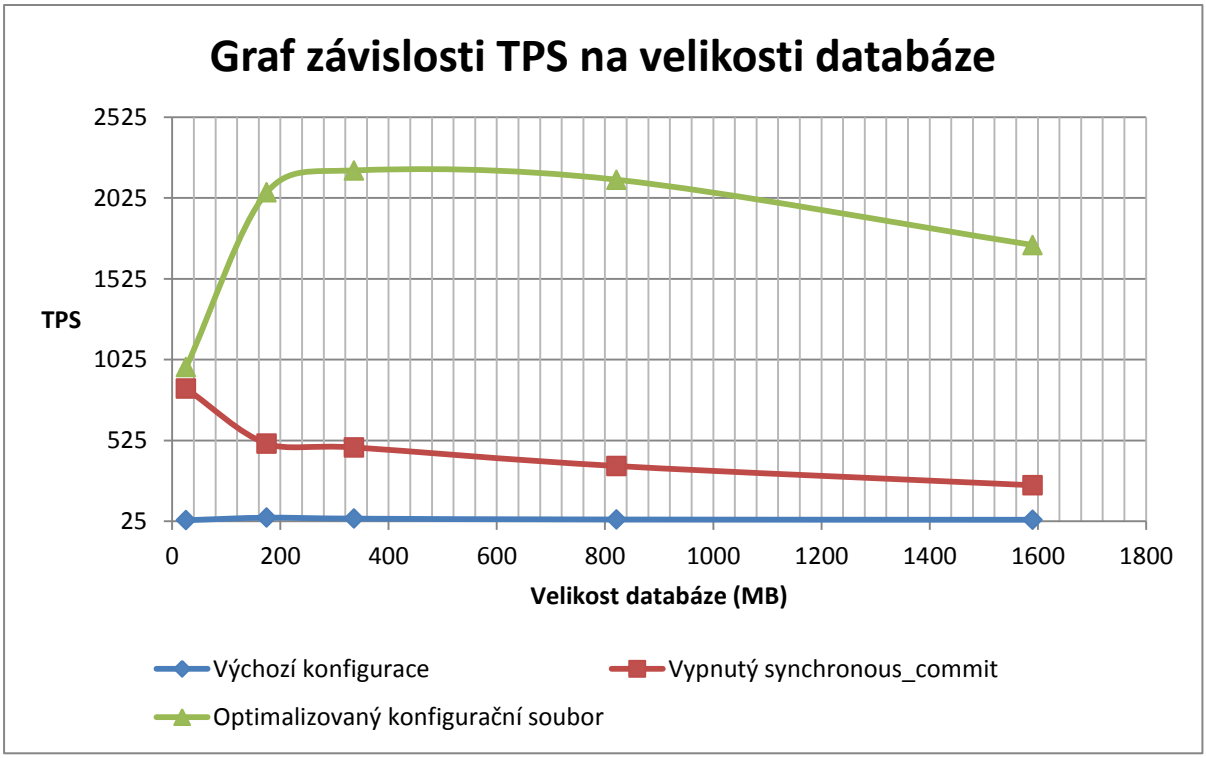
Po uvedení hlavních parametrů konfiguračního souboru PostgreSQL databáze je možné začít s testováním a laděním konfiguračního souboru. Každé nastavení je velice specifické systémem, na kterém databázový serveru funguje a využívanou databází. Právě proto neexistuje jedna univerzální konfigurace, která by byla optimální na všechny případy. Testování bude provedeno na serveru Minerva3 a bude zaměřeno na krátké a početné transakce simulované testem TPC-B. Testy proběhnou pomocí vytvořeného programu probraného v kapitole 9. Pro každou sadu testů byly provedeny testy všech kombinací následujících počtů klientů: 1, 4, 8, 16, 32, 50, 100 a hodnot škálovatelnosti: 1, 10, 20, 50, 100. Každý jednotlivý test trval 15 minut a byl zopakován třikrát. Pro porovnávání byl vybrán jejich medián. Výsledky testů jsou uvedeny v příloze A ve formě tabulky a 3D grafů závislosti TPS na počtu klientů a hodnotách škálovatelnosti (graf je popsán v kapitole 9.3).

První sada testů byla spuštěna nad databází s výchozím nastavením konfiguračního souboru. Na výsledcích z tabulky A.1 a grafu z obrázku A.2 „výchozí konfigurace“ je vidět, že se všechny hodnoty pohybují v rozmezí 25 až 65 TPS. Při porovnání s 3D grafem z kapitoly 9.3 je patrné, že nově naměřené hodnoty jsou daleko nižší. Díky podobnosti grafů lze usoudit, že nižší hodnoty nejsou způsobeny chybným testováním, ale nastavením databázového serveru. Hlavním důvodem velkého rozdílu ve výkonnosti je úzké hrdlo zpomalující všechny testy. Po testování bylo zjištěno, že úzkým hrdlem je okamžitá synchronizace s diskem („synchronous_commit“). Právě proto další sadou testů bude výchozí nastavení s vypnutým přepínačem „synchronous_commit“. Dalším faktorem zpomalení je celkové nastavení konfiguračního souboru optimalizované na malé databáze a bezchybný chod na většině počítačů specifických různým hardwarem a operačními systémy).

Výsledky sady testů jsou uvedeny v 3D grafu obrázku A.3 a tabulce A.1 pod sloupcem „vypnutý synchronous_commit a zobrazují výchozí konfiguraci s vypnutým nastavením synchronizovaného zápisu na disk. Podle porovnání hodnot z tabulky vyplývá, že se výpočet zrychlil průměrně 13krát. Zrychlení je dáno vlastností synchronního zápisu, kdy server čeká na úspěšné potvrzení o zapsání transakce na disk, než oznámí úspěšné vykonání transakce klientovi. Pokud se tento přepínač vypne, neztrácí se čas čekáním a další transakce mohou být v rámci této doby vykonány [28].

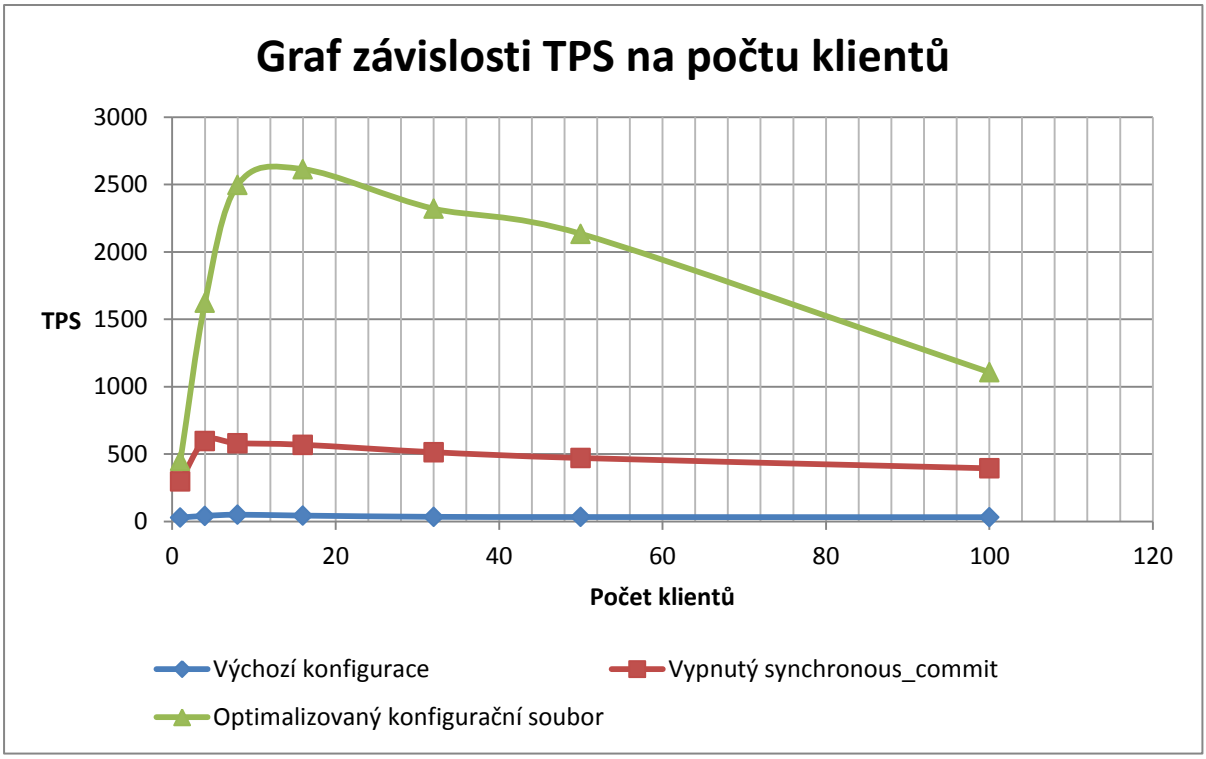
Poslední sada testů ukazuje nejlepší dosažené nastavení. Hodnoty jsou opět zobrazeny v tabulce A.1 a 3D grafu obrázku A.4. Výsledné hodnoty jsou opět mnohonásobně vyšší (téměř 4krát rychlejší než minulá sada testů a 47krát rychlejší než výchozí konfigurace) než v minulých testech. Zrychlení je dosaženo pomocí využití více hardwarových prostředků serveru.

Na obrázku 10.1.1 je zobrazen graf závislosti transakcí za sekundu na velikosti databáze. Graf porovnává průměrné hodnoty počtu klientů pro každou hodnotu škálovatelnosti. V grafu jsou zaneseny všechny tři sady testů. Výsledky výchozího nastavení jsou velice nízké a ani jejich hodnoty se příliš nemění s rostoucí databází. U zbylých dvou křivek je vidět podobná počáteční hodnota pro nejmenší databázi. Nízká hodnota u serveru Minerva3 je dána maximální rychlostí pole disků RAID 6, které je optimalizováno pro velké databáze a pro operace na malých databázích může mít vyšší režii. Další omezení rychlosti u malé databáze je způsobeno zamykáním v tabulce poboček, která obsahuje velice málo záznamů pro nízkou hodnotu škálovatelnosti [17]. Vývoj TPS u druhé sady testů s asynchronním potvrzováním je klesající kvůli zvětšující se databázi a nízké paměti RAM. Křivka optimalizované konfigurace ukazuje maximální TPS kolem 200MB databáze. V tento moment dochází k nejefektivnějšímu využití sdílené paměti a rychlosti diskového pole. Poté se rychlost snižuje díky zvětšování databázi, a tedy i vyšší režii při vykonávání operací.



Obrázek 10.1.1 - Graf závislosti TPS na velikosti databáze na serveru Minerva3

Obrázek 10.1.2 ukazuje graf závislosti transakcí za sekundu na počtu klientů dotazovaných se databáze. Graf porovnává průměrné hodnoty škálovatelnosti pro každou hodnotu počtu klientů. Opět jsou v grafu zobrazeny všechny tři sady testů. Každá sada testů má podobný průběh. Křivka začíná u testu s jedním klientem, poté se výkonnost zvyšuje do maxima definovaného určitým počtem klientů, od kterého výkonnost začíná klesat. Díky efektivnímu využití vyrovnávací paměti je dosaženo výrazného zrychlení při dotazování více klienty najednou. Počet klientů, pro který je výkonnost databáze nejvyšší, je z velké části definován velikostí sdílené paměti. Z obrázku 10.1.2 je viditelné, že pro optimalizovanou konfiguraci se výkonnost zvýšila nad 2500 TPS a optimální počet klientů posunul na hodnotu 12 klientů. Z obou grafů je zřejmé, jak moc ovlivňuje výkonnost databáze správné nastavení jejího konfiguračního souboru.



Obrázek 10.1.2 - Graf závislosti TPS na počtu klientů na serveru Minerva3

10.2 Porovnání hardwarových konfigurací

Pomocí minulé kapitoly optimalizující server Minerva3 byly otestovány a optimalizovány další tři hardwarové konfigurace. Tato kapitola se zaměřuje na porovnání výkonu podle použitého hardwaru. Na testy byly použity dva servery a dva notebooky. Nejdříve budou popsány hardwarové konfigurace a poté budou zobrazeny výsledky porovnávacích testů. Testy byly opět provedeny pro různé počty klientů a na různých velikostech databází.

10.2.1 Hardwarové konfigurace

K testování byly vybrány čtyři odlišné hardwarové konfigurace (dva servery a dva notebooky). Hlavními rozdíly těchto sestav jsou různé typy úložných zařízení (SAS RAID, SSD, běžný magnetický disk), velikost paměti RAM a rychlost a počet procesorů. Následuje seznam hardwarových konfigurací seřazený podle názvu:

- Minerva2 – Jedná se o server Supermicro obsahující dva dvoujádrové procesory AMD Opteron taktovány na 2.8GHz, 8 GB RAM a 5,5TB diskové pole. Na serveru je spuštěn operační systém CentOS 5.5 a databázový server Oracle 11g.
- Minerva3 – Jedná se o výpočetní server Supermicro obsahující dva šestijádrové procesory Opteron 2435, 64GB RAM, rychlé diskové pole SAS RAID6 o kapacitě 2,4TB a rychlosti 15 000 otáček/minutu a úložné diskové pole SAS RAID6 o kapacitě 20TB. Databáze je uložena na diskovém poli o kapacitě 20TB.
- Notebook Lenovo ThinkPad X301 – Notebook s dvoujádrovým procesorem U9400 taktovaným na 1,4GHz, 4GB RAM a 120GB SSD disk. Na notebooku je spuštěn operační systém Ubuntu 11.04.

- Notebook Dell E6400 – Notebook s dvoujádrovým procesorem Intel P9600 taktovaným na 2,66GHz, 4GB RAM a 250GB pevný disk s rychlostí 7200 otáček/minutu. Na notebooku je spuštěn operační systém CentOS 5.5.

Příloha B obsahuje grafy rychlosti čtení použitých disků (diskové pole – obrázek B.6, magnetický disk – obrázek B.7, SSD – obrázek B.8). Grafy ukazují 10krát vyšší rychlost čtení u diskového pole a 2krát vyšší rychlost u SSD oproti normálnímu magnetickému disku.

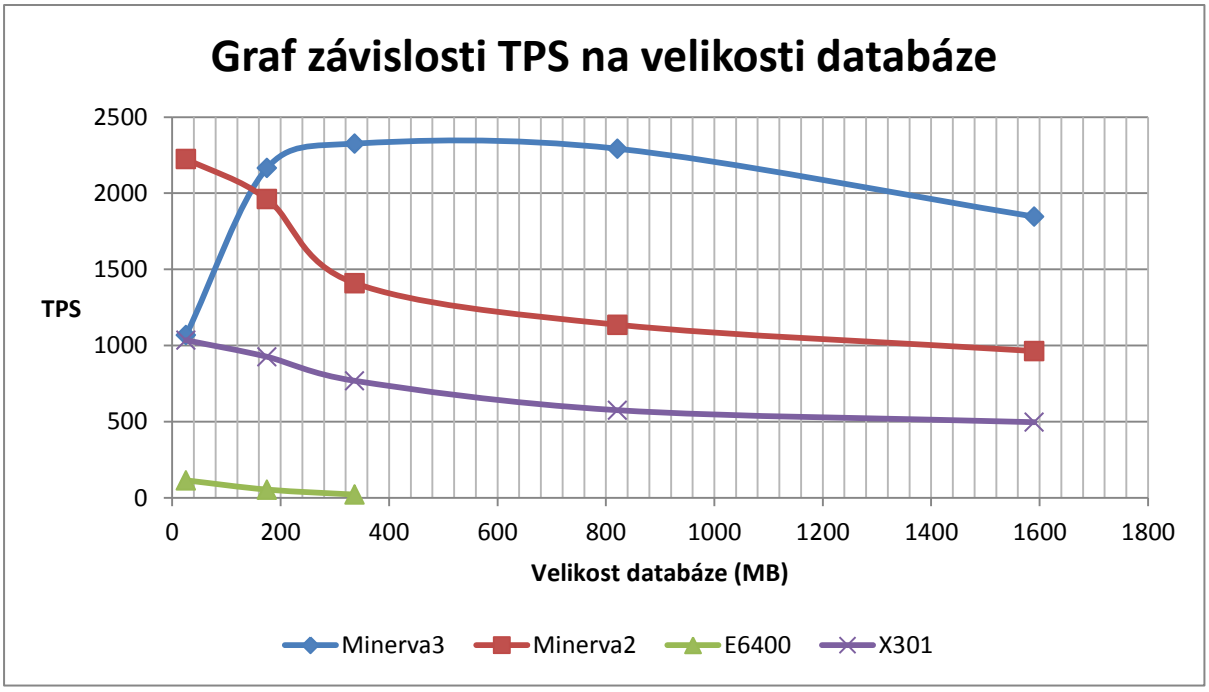
10.2.2 Porovnání hardwarových konfigurací

Pro testování hardwarových konfigurací byly zvoleny tyto počty klientů: 1, 10, 20, 50, 100 a tyto hodnoty škálovatelnosti: 1, 4, 8, 16, 32, 50. Hlavní změny v konfiguračních souborech jsou zobrazeny v tabulce 10.2.1. Výsledky porovnávající výkonnost optimalizovaných hardwarových konfigurací jsou zobrazeny v příloze B ve formě tabulky a 3D grafů. Dále jsou výsledky znázorněny v grafu obrázku 10.2.2 a obrázku 10.2.3. Grafy opět zobrazují závislost TPS na velikosti databáze a počtu klientů a byly vytvořeny stejně jako grafy z minulé podkapitoly 10.1.2.

Nastavení \ HW konfigurace	Minerva3	Minerva2	X301	E6400
shared_buffers	4096MB	128MB	1024MB	128MB
work_mem	12MB	1MB	1MB	1MB
maintenance_work_mem	512MB	16MB	64MB	32MB
synchronous_commit	off	off	off	off
wal_buffers	64MB	512kB	512kB	256kB
checkpoint_segments	50	16	16	16
checkpoint_completion_target	0,9	0,9	0,9	0,9
effective_cache_size	8192MB	256MB	2048MB	256MB
default_statistics_target	7500	1000	1000	500

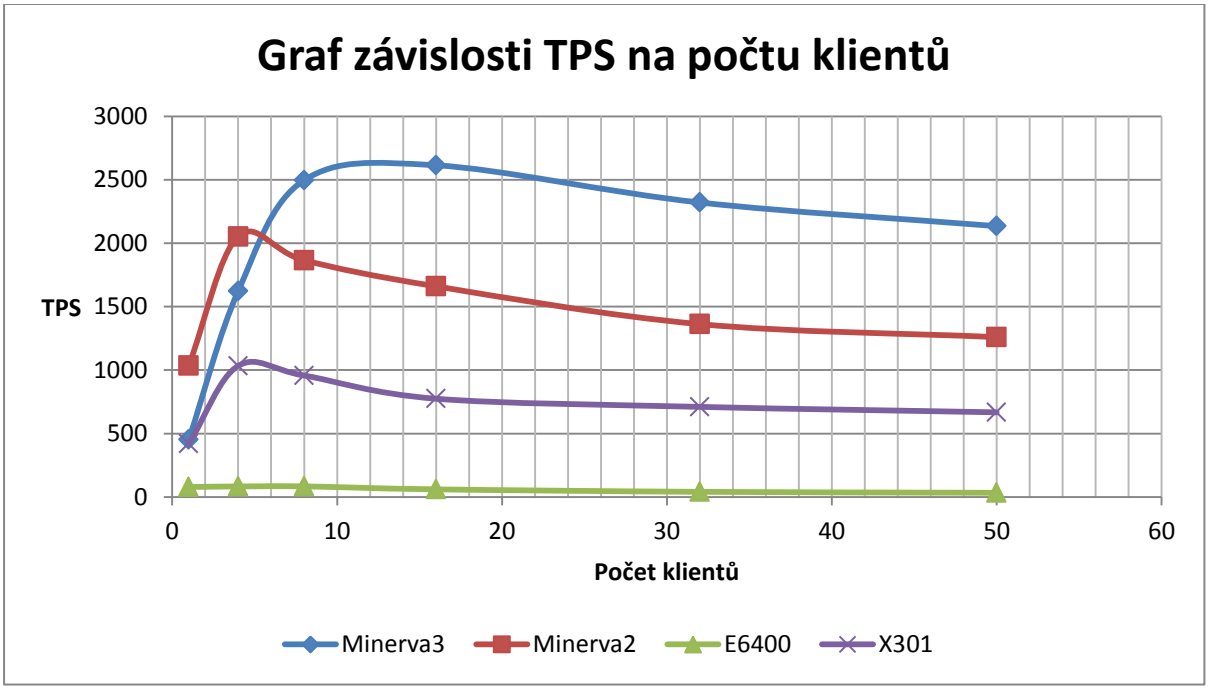
Tabulka 10.2.1 - Hlavní parametry v konfiguračních souborech databází

Obrázek 10.2.2 ukazuje rozdíl výkonnosti podle velikosti testované databáze. Z obrázku je patrné, že notebook E6400 má velice nízkou výkonnost a pro databáze větší než 350MB není použitelný. Jeho výkon nejvíce ovlivňuje rychlost pevného disku. Pro větší databáze se začnou více projevovat diskové operace čtení a zápisu. Ty mají za následek zpomalení celého počítače až do momentu, kdy počítač přestane úplně reagovat. Notebook X301 je průměrně 12krát rychlejší než notebook E6400. Databázový server lze používat na této hardwarové konfiguraci bez problémů. Jeho výkonnost je velmi ovlivněna vysokou rychlostí SSD disku. Ve srovnání je X301 2krát pomalejší než server Minerva2 a 2,5krát pomalejší než server Minerva3. I když server Minerva2 obsahuje méně výkonný hardware, je vidět z grafu, že pro nejmenší databázi je jeho hodnota TPS, a tedy i výkonnost, větší než u serveru Minerva3. To je způsobeno pomalejším diskovým polem serveru Minerva3, které je optimalizováno pro operace s většími objemy dat (například většími databázemi). Další vývoj výkonnosti obou serverů je klesající díky narůstající velikosti databáze. V průměru je server Minerva3 rychlejší o 26% než server Minerva2 díky rychlejšímu hardwaru a větší RAM paměti.



Obrázek 10.2.2 - Graf závislosti TPS na velikosti databáze na různých hardwarových konfiguracích

Obrázek 10.2.3 zobrazuje závislost TPS na počtu klientů dotazujících se databáze při testování. Z grafu jsou opět viditelné velice nízké hodnoty notebooku E6400. Ty jsou stejně jako v minulém grafu zapříčiněny pomalým hardwarem. Vývoj křivek u obou serverů a notebooku X301 je podobný. Posunuté maximum u serveru Minerva3 je dáno výkonnější hardwarovou konfigurací a větší nastavenou sdílenou pamětí. Pokud v paměti zůstávají nahraná data, která jsou vícekrát dotazována jinými klienty, dochází k výraznému urychlení. I když má X301 nastavenou větší sdílenou paměť než Minerva2, díky celkově pomalejšímu hardwaru jsou jeho výsledky dvojnásobně nižší. Průměrné zrychlení serveru Minerva3 vůči serveru Minerva2 a notebooku X301 je stejné jako u minulého grafu 10.2.2.



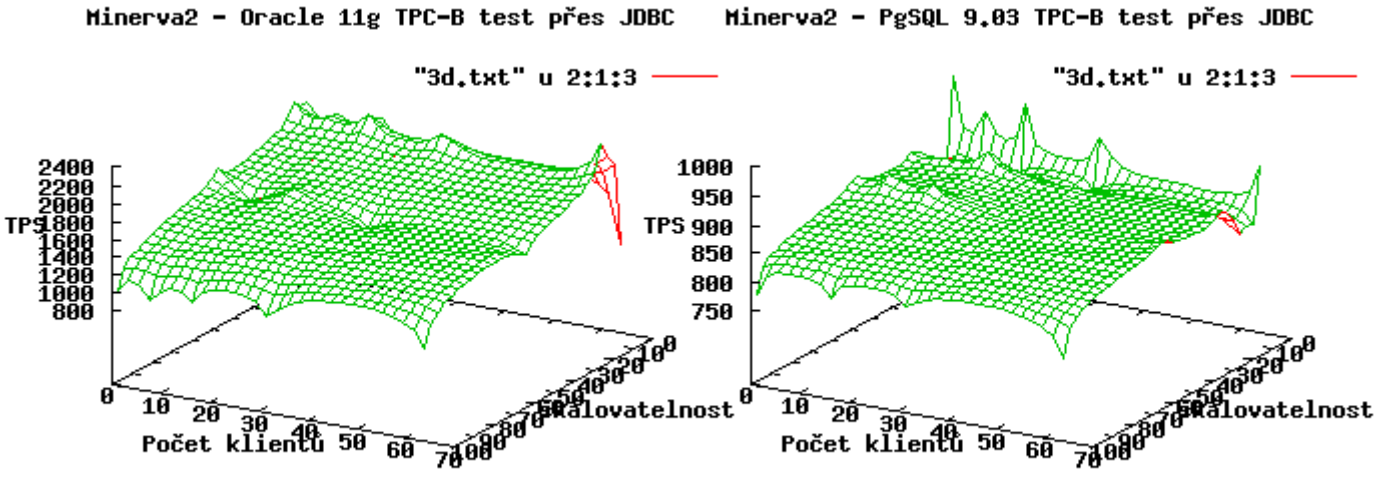
Obrázek 10.2.3 - Graf závislosti TPS na počtu klientů na různých hardwarových konfiguracích

10.3 Porovnání výkonu PostgreSQL a Oracle

Protože všechny databázové systémy jsou si podobné, měly by pro ně platit stejné principy. Právě proto se jedny z provedených testů zabývají porovnáním výkonu mezi databázemi PostgreSQL a Oracle. Testy byly provedeny pomocí programu implementující testy TPC-B pro různé databázové systémy popsaného v kapitole 8.3. K testování byla zvolena optimální konfigurace PostgreSQL databáze získaná pomocí testů z podkapitoly 10.1. Testy byly provedeny na serveru Minerva2, na kterém je spuštěn již optimalizovaný databázový server Oracle 11g Enterprise Edition (11.1.0.6.0 - optimalizováno Jaroslavem Rábem, server využívání 4GB paměti RAM).

Opět proběhlo testování, kdy každý test byl zopakován třikrát a jako výsledek byl vybrán jejich medián. Testy proběhly pro všechny kombinace následujících počtů klientů: 1, 8, 16, 32, 64 a hodnot škálovatelnosti: 1, 5, 10, 50, 100. K zobrazení výsledků byly vybrány 3D grafy závislosti TPS na počtu klientů a hodnotách škálovatelnosti, protože nejlépe zachycují všechny získané hodnoty. Tabulka naměřených výsledků je vypsána v příloze C.

Obrázek 10.3.1 obsahuje oba výše zmíněné 3D grafy. Vlevo lze vidět výsledky testování Oracle databáze. Výkonnost databáze závisí na hodnotě škálovatelnosti (velikosti databáze), pohybuje se od 900 (pro největší databázi, škálovatelnost je 100) po 2650 TPS (pro nejmenší databázi, škálovatelnost je 1). Počet klientů ovlivňuje rychlost zpracování transakce jen lehce, pro interval od 1 do 50 klientů, se hodnota mění nejvíce o 200 TPS. V pravé části obrázku je vidět graf PostgreSQL databáze. Zde jsou všechny hodnoty TPS v intervalu od 730 po 840 TPS. Testování proběhlo pomocí softwaru z kapitoly 8.3, který používá ke komunikaci JDBC ovladač a k testování databáze uložené procedury („stored procedures“). Jelikož tento ovladač není optimalizován pro PostgreSQL databázi, jsou všechny dosažené hodnoty velice podobné. Lze usoudit, že úzkým hrdlem těchto testů se stává právě ovladač JDBC. U Oracle databáze zprostředkovává JDBC ovladač nativní propojení a proto i výsledky ukazují daleko větší výkonnost. Při porovnání hodnot serveru Minerva2 z minulé kapitoly 10.2 s hodnotami Oracle databáze je zřejmé, že výsledky se pohybují ve stejném rozmezí a grafy (obrázek 10.3.1 a obrázek B.3) jsou si velice podobné. Tedy pro obě databáze platí podobné principy a jejich výkonnost je úzce spjata s hardwarovou konfigurací, na které je databáze spuštěna.



Obrázek 10.3.1 – 3D grafy TCP-B testů Oracle a PgSQL databáze

10.4 Partitioning

K testování a ladění výkonu byla vybrána jedna metoda paralelního urychlení. Jedná se o metodu partitioningu na úrovni tabulky, kdy velká tabulka je rozdělena na několik menších podle určitého klíče. Testování proběhlo podle třídílného článku [24,25,26].

První částí testů bylo ověření postupů implementace metody partitioningu a porovnání naměřených výsledků. Podle [24] bylo vytvořeno schéma databáze. Schéma obsahuje jednu otcovskou tabulku „hashvalue_PT“, která se skládá z atributu bitové pole „hash“ a časového razítka „hashtime“. Dále schéma obsahuje deset tabulek, které dědí z „hashvalue_PT“ a jsou rozděleny podle měsíce od ledna po říjen roku 2008. Pro každou dědící tabulku je vytvořen index nad atributem „hashtime“. Aby mohl partitioning fungovat, je třeba zajistit správné vkládání dat do tabulek. Proto [24] definuje trigger na vkládání n-tice do jedné tabulky podle určitého datového intervalu. Po vytvoření schématu je nutné databázi naplnit. Z [25] byla převzata funkce na nahrání dat do tabulky. Následné testování proběhlo na tabulkách o velikosti 40 miliónů záznamů. Vždy se tabulky otestovaly na tři typy dotazů. Každý dotaz byl testován třikrát a vybrán jejich medián. První dotaz vybírá všechna data, která mají stejné datum. Druhý dotaz vybírá data v intervalu určitých dat v rámci jedné dědící tabulky. Poslední dotaz vybírá data v intervalu překrývající dvě tabulky. Tabulka 10.4.1 zobrazuje výsledky testů pro dotazy vykonané nad 40 milióny záznamy. Velikost databáze je 5,85GB. V [26] není popsána hardwarová sestava, na které byly testy vykonány. Podle výsledků je zjevné, že se jedná o pomalejší konfiguraci. Nicméně poměr zrychlení zůstává podobný. Použití metody partitioningu je velice výhodné pro tento typ databáze a dotazů.

Typ dotazu	Počet záznamů (v milionech)	Bez partitioningu (ms)	Partitioning (ms)	Zrychlení (%)
		Server Minerva3		
Určité datum	40	13379,345	2187,845	611,53%
Interval z jedné tabulky	40	45281,007	12176,861	371,86%
Interval přesahující dvě tabulky	40	46094,737	16193,323	284,65%
		Server [26]		
Určité datum	40	259844	34610	750,78%
Interval z jedné tabulky	40	306485	98953	309,73%
Interval přesahující dvě tabulky	40	334468	197687	169,19%

Tabulka 10.4.1 – Porovnání výsledků metody partitioningu podle data

Použití metody partitioningu se jevilo výhodné i pro testy TPC-B, kde pro větší hodnoty škálovatelnosti se výrazně zvětšuje tabulka „pgbench_accounts“. Proto druhá část testů byla zaměřena na zrychlení testů TPC-B pomocí partitioningu. Opět byly testy provedeny nad 40 milióny záznamy. Vygenerovaná tabulka „pgbench_accounts“ má velikost 5,98GB. Tabulka 10.4.2 porovnává operace výběru („select“) záznamů nad oběma typy tabulek. Testy byly spuštěny pomocí programu z kapitoly 9 pro tři různé počty klientů (1, 12, 100). Jako testovací dotaz byl vybrán výchozí dotaz programu pgbench testující pouze operaci „select“ nad tabulkou „pgbench_accounts“. Z výsledků je viditelné, že partition tabulka je daleko pomalejší než základní tabulka. V průměru je zpomalení desetinásobné. Vykonávací plán tohoto typu dotazu je zobrazen na obrázku 10.4.3. Lze vidět, že složitost a výsledná cena je vyšší u dotazu do partition tabulky. Díky tomu v testech výběru jednoho prvku je vždy rychlejší základní tabulka.

V další části tabulky 10.4.2 jsou uvedeny testy výběru více účtů najednou. Opět byly zvoleny dva typy dotazů „select“, kdy jeden vybírá interval v rámci jedné partition tabulky a druhý se dotazuje na účty ze dvou a více tabulek. Testy byly provedeny na různém počtu získaných záznamů (100, 600, 900, 1900, 5000). Výsledky ukazují zrychlení metody partitioningu v průměru o 20% oproti základní tabulce. Tyto výsledky jsou potvrzeny vyhodnocovacím plánem zobrazeným na obrázku 10.4.4. I když je dotazování do partition tabulky složitější, jednotlivé dotazy jsou vykonány daleko rychleji než u základní tabulky.

Typ dotazu	**Počet záznamů (v milionech)**		***Server Minerva3***		
		Počet klientů	**Bez partitioningu (TPS)**	**Partitioning (TPS)**	**Zrychlení (%)**
pgbench – dotaz select	40	1	3358,865248	298,165037	-1126,51%
pgbench – dotaz select	40	12	21004,13134	1967,06243	-1067,79%
pgbench – dotaz select	40	100	16098,43549	1628,28828	-988,67%
		Získaných záznamů (v tisících)	**Bez partitioningu (ms)**	**Partitioning (ms)**	**Zrychlení (%)**
Interval z jedné tabulky	40	100	472,005	402,119	120,67%
Interval z jedné tabulky	41	600	2364,68	2004,762	119,16%
Interval z jedné tabulky	42	900	4263,868	3533,473	117,95%
Interval přesahující dvě tabulky	40	900	4249,568	3566,185	117,38%
Interval přesahující dvě tabulky	41	1900	9139,991	7428,449	123,04%
Interval přesahující dvě tabulky	42	5000	23704,807	19318,146	122,71%

Tabulka 10.4.2 - Porovnání výsledků metody partitioningu podle id účtu

Metoda partitioning je výhodná pro tabulky s velmi vysokým počtem záznamů a v případě, kdy potřebujeme načítat více záznamů najednou. Velmi častým použitím jsou tabulky se záznamy s časovým razítkem. Díky použití této metody mohou být data odděleny podle určitého časového období. Kromě rychlejšího přístupu k datům implementace nabízí rychlejší údržbu tabulek (operace „VACUUM“) a přeindexování [17].

```
QUERY: "explain select * from pgbench_accounts where aid=10;"

          QUERY PLAN – Základní tabulka bez partitioningu
-------------------------------------------------------------------------
 Index Scan using pgbench_accounts_pkey on pgbench_accounts
     (cost=0.00..12.65 rows=1 width=97)
     Index Cond: (aid = 10)

                  QUERY PLAN – Partition tabulka
-------------------------------------------------------------------------
 Result  (cost=0.00..21.00 rows=2 width=352)
   ->  Append  (cost=0.00..21.00 rows=2 width=352)
         ->  Seq Scan on pgbench_accounts  (cost=0.00..12.62 rows=1
              width=352)
      Filter: (aid = 10)
           ->  Index Scan using pgbench_accounts_pt_1_aid on
          pgbench_accounts_pt_1 pgbench_accounts  (cost=0.00..8.38
          rows=1 width=352)
          Index Cond: (aid = 10)
```

Obrázek 10.4.3 – Vyhodnocovací plán dotazu na získání jednoho účtu

```
QUERY: "explain select * from pgbench_accounts where aid>20000000
     and aid <25000000"

          QUERY PLAN – Základní tabulka bez partitioningu
-------------------------------------------------------------------------
 Index Scan using pgbench_accounts_pkey on pgbench_accounts
     (cost=0.00..236531.98 rows=4993967 width=97)
     Index Cond: ((aid > 20000000) AND (aid < 25000000))

                  QUERY PLAN – Partition tabulka
-------------------------------------------------------------------------
 Result  (cost=0.00..102043.15 rows=5000001 width=352)
   ->  Append  (cost=0.00..102043.15 rows=5000001 width=352)
         ->  Seq Scan on pgbench_accounts  (cost=0.00..13.15 rows=1
              width=352)
               Filter: ((aid > 20000000) AND (aid < 25000000))
         ->  Seq Scan on pgbench_accounts_pt_21 pgbench_accounts
               (cost=0.00..20406.00 rows=1000000 width=352)
               Filter: ((aid > 20000000) AND (aid < 25000000))
         ->  Seq Scan on pgbench_accounts_pt_22 pgbench_accounts
               (cost=0.00..20406.00 rows=1000000 width=352)
               Filter: ((aid > 20000000) AND (aid < 25000000))
         ->  Seq Scan on pgbench_accounts_pt_23 pgbench_accounts
               (cost=0.00..20406.00 rows=1000000 width=352)
               Filter: ((aid > 20000000) AND (aid < 25000000))
         ->  Seq Scan on pgbench_accounts_pt_24 pgbench_accounts
               (cost=0.00..20406.00 rows=1000000 width=352)
               Filter: ((aid > 20000000) AND (aid < 25000000))
         ->  Seq Scan on pgbench_accounts_pt_25 pgbench_accounts
               (cost=0.00..20406.00 rows=1000000 width=352)
               Filter: ((aid > 20000000) AND (aid < 25000000))
```

Obrázek 10.4.4 – Vyhodnocovací plán dotazu na více účtů

11 Závěr

V rámci diplomové práce jsem měl možnost prostudovat výkonnostní problémy dnešních databázových systémů. Také jsem si osvojil mnohé možnosti zrychlení vykonávání databázových operací přidáním výkonnějšího hardwaru, optimalizací konfiguračního souboru nebo implementací paralelizačních metod.

V implementační části práce byl vytvořen testovací program schopný dávkově otestovat jakoukoli databázi PostgreSQL (verze 9 a vyšší). Program je specifický vytvářením vlastních transakčních testovacích souborů podle historických operací uložených v logovacím souboru databáze. Tím jsou získány výsledky výkonnosti odpovídající reálnému použití databáze. Kromě toho je program schopen otestovat databázi standardizovaným testem TPC-B. Mimo testování program nabízí možnost vytváření hypertextových záznamů zobrazujících všechny naměřené hodnoty, agregované informace a vygenerované grafy. Celkové řešení bylo založeno na dvou open-source programech pgbench-tools a pgfouine. Vytvořený program v rámci této diplomové práce představuje upravení obou programů a jejich následné spojení. Pgbench-tools je rozšířen o skript na zpracovávání výsledků z programu pgfouine a na vytváření nového transakčního souboru. Dále byla v programu rozšířena možnost konfigurace a zlepšen vyhodnocovací skript vytvářející zprávu o výsledcích. Do program pgfouine byl doimplementován nový formát výstupního souboru obsahující agregované operace spolu s intervaly hodnot všech číselných proměnných získaných z opravdových operací.

Další částí diplomové práce bylo testování výkonnosti databázových systémů. Testování proběhlo kompletně pomocí vytvořeného programu na databázích PostgreSQL. Jedinou výjimkou byly srovnávací testy PostgreSQL a Oracle databáze, které sloužily k potvrzení podobných rysů společných pro všechny databáze. Testování bylo rozděleno na tři hlavní části, aby byl pokryt co největší počet různých testů. První část zkoumá zrychlení pomocí optimalizace konfiguračního souboru. Druhá část porovnává různé hardwarové konfigurace s použitím již optimalizovaných konfiguračních souborů. Třetí část se zabývá implementací a testováním paralelní metody „partitioning“. Další metody paralelního zpracování nemohly být vyzkoušeny kvůli nedostatku práv na školním hardwaru. Porovnáním dvou odlišných databázových systémů bylo dokázáno, že u jiných databázových serverů by stejné testy a optimalizace vypadaly podobně jako výsledky naměřené v provedených testech této diplomové práce. Mimo TPC-B testy bylo vyzkoušeno provádění testů pomocí analýzy databázového logovacího souboru. Správnost implementace byla ověřena na zpracování databázovém logu TPC-B testů, kdy vytvořením transakčního souboru vznikla stejná transakce, jako je transakce TPC-B.

Výsledkem práce je návod, jak zrychlit databázový systém pomocí lepšího nastavení konfiguračního souboru nebo použitím paralelní metody partitioningu. Dále práce slouží k optimálnímu nastavení databáze PostgreSQL 9 na školním serveru Minerva2. Posledním přínosem diplomové práce je testovací program, který bude použit k ladění výkonnosti nového školního serveru.

V pokračování práce by se mohly implementovat a otestovat další standardizované testy testující jiné aspekty databázového systému. Kromě nich by bylo užitečné změřit zlepšení výkonnosti pomocí jiných paralelních metod a technik. Jako rozšíření testovacího programu by bylo výhodné přidat možnost automatického měnění konfiguračního souboru databáze. Testovací program by byl poté schopen sám vyladit výkonnost databáze podle naměřených výsledků testů.

Literatura

[1] SILBERSCHATZ, Abraham; KORTH, Henry; SUDERSHAN, S. *Database System Concepts*. 5th edition. New York : McGraw-Hill, 2006. 1142 s. ISBN 978-0-07-295886-7.

[2] TANIAR, David; LEUNG, Clement H.C.; RAHAYU, Wenny; GOEL, Sushant. *High-Perfomance Parallel Database Processing and Grid Databases*. New Jersey : John Wiley & Sons, Inc., 2008. 551 s. ISBN 978-0-470-10762-1.

[3] MOORE, Gordon E. *Lithography and the Future of Moore's Law*, Proc. SPIE Vol. 2437, 1995.

[4] ZENDULKA, Jaroslav; RUDOLFOVÁ, Ivana: *Databázové systémy IDS studijní opora*. FIT VUT Brno, 2006.

[5] AMDAHL, Gene M. *Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities*. IBM Sunnyvale, California, 1967.

[6] PATTERSON, David A.; GIBSON, Garth; KATZ. Randy H.. *A Case for Redundant Arrays of Inexpensive Disks (RAID)*. University of California Berkley, 1988.

[7] SI, Kiwi. *Techhead* [online]. 2009 [cit. 2011-03-10]. SCSI, Fibre Channel (FC), SATA, PATA, USB & FireWire – Maximum Theoretical Bus Speeds Comparison. Dostupné z WWW: <http://www.techhead.co.uk/scsi-fibre-channel-fc-sata-pata-usb-firewire-maximum-theoretical-bus-speeds-comparison>.

[8] ANDERSON, Dave; DYKES, Jim; RIEDEL, Erik: *More than an interface - SCSI vs. ATA*. 2nd Annual Conference on File and Storage Technology, 2003

[9] SHILOV, Anton. *Xbit laboratories* [online]. 2010 [cit. 2011-03-15]. 4TB Hard Disk Drives May Emerge in 2011 - Analyst. Dostupné z WWW: <http://www.xbitlabs.com/news/storage/display/20101201183538_4TB_Hard_Disk_Drives_May_Emerge_in_2011_Analyst.html>.

[10] *PostgreSQL 9.0.3 Documentation* [online]. 2010 [cit. 2011-01-03]. High Availability, Load Balancing, and Replication. Dostupné z WWW: <http://www.postgresql.org/docs/9.0/static/high-availability.html>.

[11] MITCHELL, Joseph. *Lamp: The open source web platform* [online]. 2001 [cit. 2011-04-13]. PostgreSQL's Multi-Version Concurrency Control. Dostupné z WWW: <http://onlamp.com/pub/a/onlamp/2001/05/25/postgresql_mvcc.html>.

[12] *Burleson Consulting* [online]. 2005 [cit. 2011-04-25]. Oracle transactions per second?. Dostupné z WWW: <http://www.dba-oracle.com/m_transactions_per_second.htm>.

[13] TAHAGHOGHI, Seyed M.M. "Saied"; WILLIAMS, Hugh. *Learning MySQL*. USA : O'Reilly Media, 2006. 618 s. ISBN 978-0-596-00864-2.

[14] *Transaction Processing Performance Council* [online]. 2001 [cit. 2011-04-27]. About the TPC . Dostupné z WWW: <http://www.tpc.org/information/about/abouttpc.asp>.

[15] *TPC BENCHMARK™ B : Standard Specification* [online]. San Jose, USA : Shanley Public Relations, 7.5.1994 [cit. 2011-04-25]. Dostupné z WWW: <http://www.tpc.org/tpcb/spec/tpcb_current.pdf>.

[16] KOLÁŘ, Radim. *Linuxsoft.cz* [online]. 21.10.2010 [cit. 2011-04-27]. Implementace TPC-B testu. Dostupné z WWW: <http://www.linuxsoft.cz/article.php?id_article=1768>.

[17] SMITH, Gregory. *PostgreSQL 9.0 High Performance : Accelerate your PostgreSQL system and avoid the common pitfalls that can slow it down*. Birmingham, UK: Packt Publishing Ltd., 2010. 468 s. ISBN 978-1-849510-30-1.

[18] *PgFouine - a PostgreSQL log analyzer* [online]. 28.10.2006 [cit. 2011-04-28]. Sample reports. Dostupné z WWW: <http://pgfouine.projects.postgresql.org/reports.html>.

[19] *PgFouine - a PostgreSQL log analyzer* [online]. 29.10.2006 [cit. 2011-04-28]. Installation instructions & Tutorial. Dostupné z WWW: <http://pgfouine.projects.postgresql.org/tutorial.html>.

[20] SMITH, Gregory. *PostgreSQL 8.3 Improvements and Migration* [online]. 2007 [cit. 2011-05-01]. Checkpoints and the Background Writer. Dostupné z WWW: <http://www.westnet.com/~gsmith/content/postgresql/chkp-bgw-83.htm>.

[21] *PostgreSQL 9.04 Documentation : Chapter 27. Monitoring Database Activity* [online]. 2011 [cit. 2011-05-01]. 27.2. The Statistics Collector. Dostupné z WWW: <http://www.postgresql.org/docs/9.0/static/monitoring-stats.html>.

[22] *PostgreSQL 9.04 Documentation : Chapter 18. Server Configuration* [online]. 2011 [cit. 2011-05-01]. 18.4. Resource Consumption. Dostupné z WWW: <http://www.postgresql.org/docs/9.0/interactive/runtime-config-resource.html>.

[23] SMITH, Gregory; TREAT, Robert; BROWNE, Christopher. *PostgreSQL Wiki* [online]. 27.1.2011 [cit. 2011-05-04]. Tuning Your PostgreSQL Server. Dostupné z WWW: <http://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server>.

[24] *Mkyong.com* [online]. 2.7.2008, 31.5.2010 [cit. 2011-05-05]. Partition Table In PostgreSQL (Create Partition) – Part 1. Dostupné z WWW: <http://www.mkyong.com/database/partition-table-in-postgresql-create-partition-part-1/>.

[25] *Mkyong.com* [online]. 2.7.2008, 31.5.2010 [cit. 2011-05-05]. Partition Table In PostgreSQL (Simulate Millions Data) – Part 2. Dostupné z WWW: <http://www.mkyong.com/database/partition-table-in-postgresql-simulate-millions-data-part-2/>.

[26] *Mkyong.com* [online]. 8.7.2008, 31.5.2010 [cit. 2011-05-05]. Performance Testing on Partition Table In PostgreSQL – Part 3. Dostupné z WWW: <http://www.mkyong.com/database/performance-testing-on-partition-table-in-postgresql-part-3/>.

[27] Singh, S., K.: Database Systems Concepts, Design and Applications. Dorling Kindersley, India, 2009. ISBN 987-81-7758-567-4

[28] *PostgreSQL 9.04 Documentation : Chapter 18. Server Configuration* [online]. 2011 [cit. 2011-05-10]. 18.5. Write Ahead Log. Dostupné z WWW: <http://www.postgresql.org/docs/current/interactive/runtime-config-wal.html#GUC-SYNCHRONOUS-COMMIT>.

Seznam příloh

Příloha A. Výsledky testů optimalizace serveru Minerva3
Příloha B. Výsledky testů srovnání hardwarových konfigurací
Příloha C. Vygenerovaný záznam výsledků testů serveru Minerva2
Příloha D. Instrukce k instalaci, nastavení a spuštění testů
Příloha E. CD s programem a výsledky testů

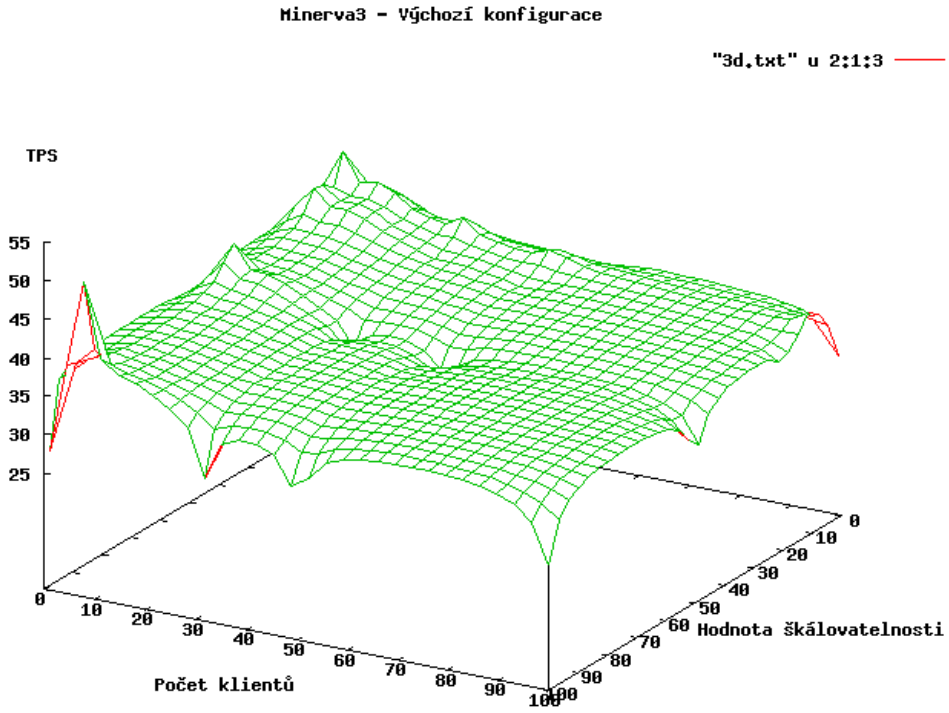
Příloha A. Výsledky testů optimalizace serveru Minerva3

Příloha A obsahuje tabulku výsledků a 3D grafy testů z kapitoly 10.1.2.

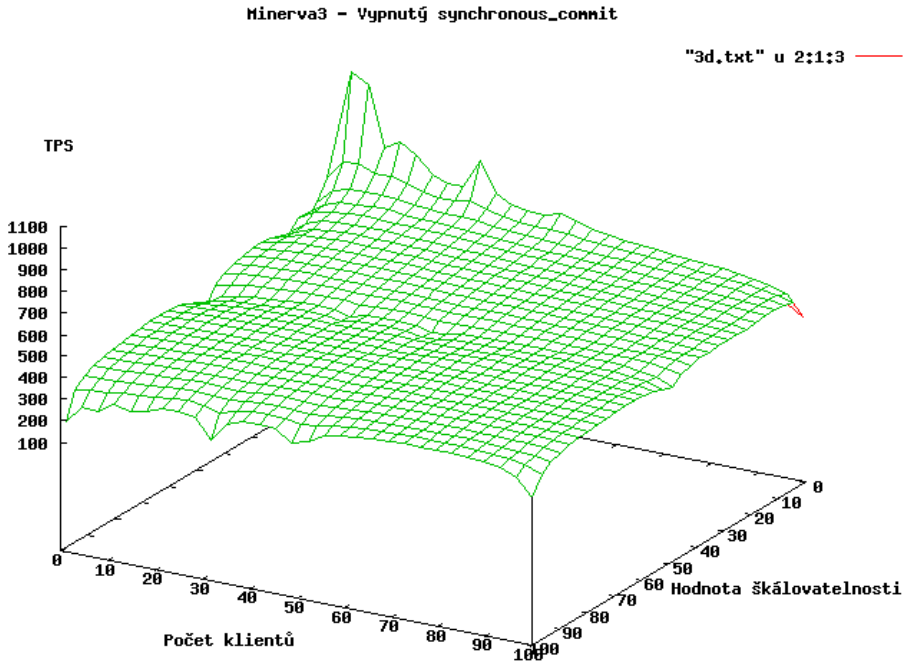
Hodnota škálovatelnosti	Počet klientů	Velikost databáze (MB)	TPS		
			Výchozí konfigurace	*Vypnutý synchronous_commit*	*Optimalizovaný konfigurační soubor*
1	1	22,21745	32,104941	496,293487	591,83255
1	4	22,22837	31,231411	1365,477169	1575,5692
1	8	22,2311	31,225174	1117,281546	1342,4956
1	16	22,22837	31,297117	1035,092831	1211,9471
1	32	22,22837	31,355473	845,175777	962,35229
1	50	22,22837	31,635371	681,420896	732,29947
1	100	22,22837	31,994558	376,169954	411,71152
10	1	163,3438	30,268543	287,115235	447,01697
10	4	163,352	49,462482	509,337442	1809,2163
10	8	163,352	62,49053	576,904669	2918,4514
10	16	163,352	55,251468	591,787059	3114,9241
10	32	163,352	47,376402	561,791032	2551,7109
10	50	163,3547	45,39683	524,059748	2150,4693
10	100	163,3574	41,705119	486,492152	2070,7205
20	1	320,2124	29,294434	272,811816	466,14566
20	4	320,2151	47,391837	472,524834	1801,1295
20	8	320,2124	54,168129	545,608596	3014,7234
20	16	320,2178	49,475924	570,665018	3070,5678
20	32	320,2151	38,99405	539,222993	2953,614
20	50	320,2233	36,519256	514,170262	2648,9387
20	100	320,2206	33,855341	458,734947	2264,9556
50	1	790,5451	28,464706	231,695904	377,07526
50	4	790,5206	41,881158	376,260286	1650,9886
50	8	790,5615	52,954659	401,645326	2971,777
50	16	790,5588	44,72868	403,600157	3156,3376
50	32	790,5697	29,512617	397,648356	2791,1735
50	50	790,567	27,147371	383,391178	2806,8869
50	100	790,5752	27,130273	373,677674	1889,6781
100	1	1574,634	28,282345	198,436164	372,43796
100	4	1574,612	40,089254	266,394615	1275,7477
100	8	1574,631	53,057761	264,066698	2235,3854
100	16	1574,642	40,042905	246,091112	2516,8896
100	32	1574,615	27,518719	227,139743	2343,7823
100	50	1574,604	26,346167	253,515393	2333,5171
100	100	1574,637	26,685457	278,618191	1201,8509

Tabulka A.1 – Výsledky testů serveru Minerva3

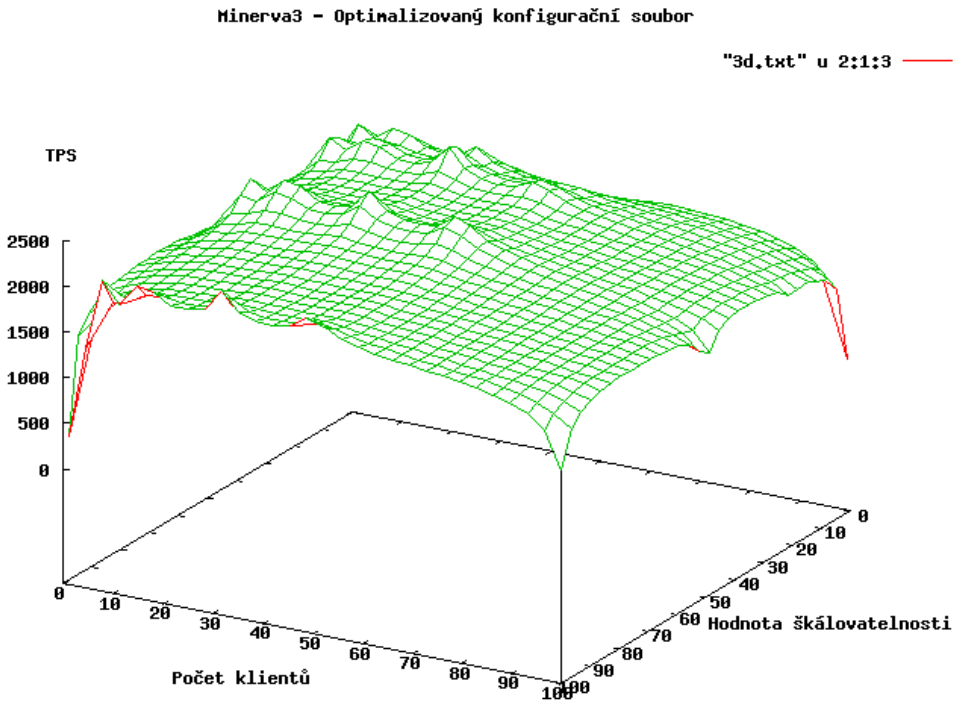
3D grafy závislosti TPS na počtu klientů a hodnotách škálovatelnosti



Obrázek A.2 - 3D graf serveru Minerva3 ve výchozí konfiguraci



Obrázek A.3 - 3D graf serveru Minerva3 s vypnutou vlastností synchronous_commit



Obrázek A.4 - 3D graf serveru Minerva3 s optimalizovanou konfigurací

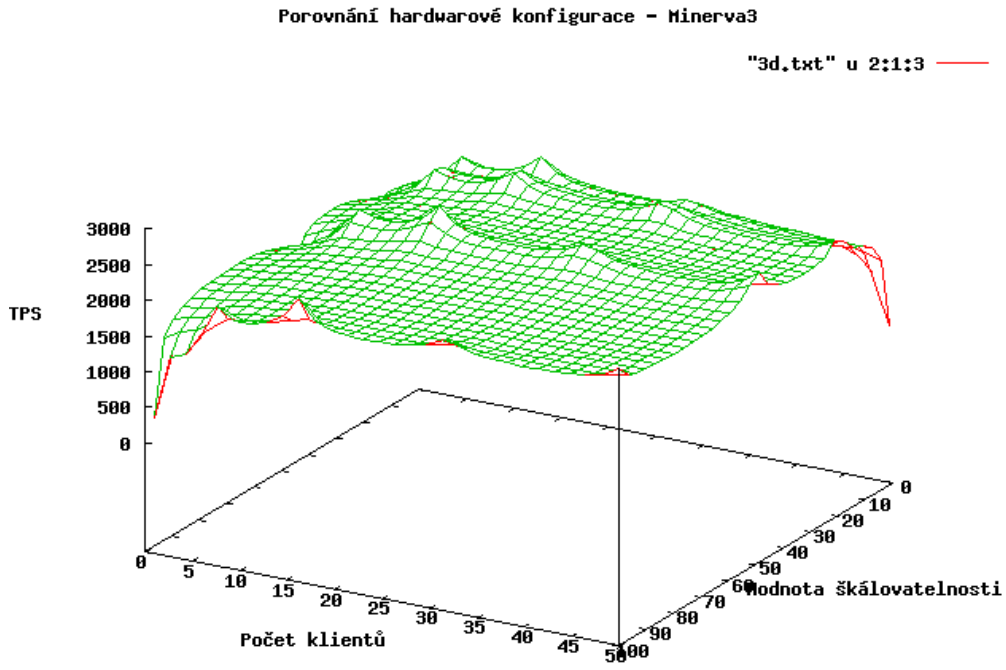
Příloha B. Výsledky testů srovnání hardwarových konfigurací

Příloha B obsahuje tabulku výsledků, 3D grafy závislosti TPS na počtu klientů a hodnotách škálovatelnosti testů z kapitoly 10.2.2 a grafy rychlosti disků vytvořené pomocí aplikace „Palimpsest Disk Utility“ („gnome-disk-utility“).

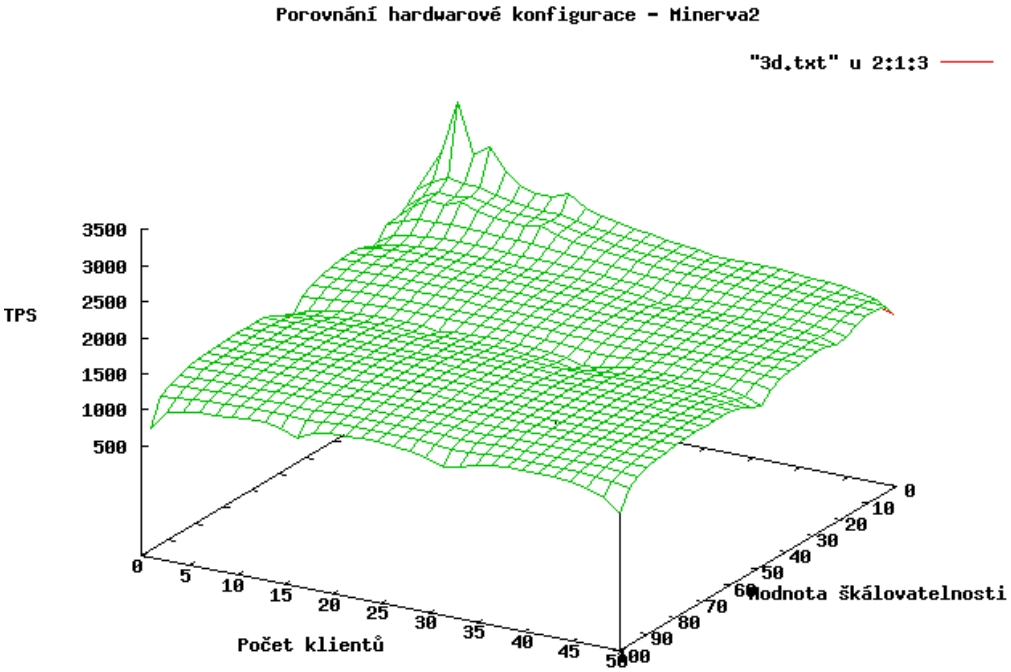
Hodnota škálovatelnosti	Počet klientů	Velikost databáze (MB)	TPS			
			Server Minerva3	*Server Minerva2*	*Notebook X301*	*Notebook E6400*
1	1	645,394	591,8325	1472,189	501,9571	157,6758
1	4	643,261	1575,569	3811,123	574,2789	153,8165
1	8	644,577	1342,496	2754,398	515,267	150,3278
1	16	644,561	1211,947	2222,972	1466,639	100,2217
1	32	646,309	962,3523	1677,625	1521,099	70,18641
1	50	648,930	732,2995	1413,214	1532,445	53,51454
10	1	795,897	447,017	1156,376	1220,706	58,59256
10	4	799,018	1809,216	2486,093	1322,099	67,3144
10	8	803,341	2918,451	2505,966	1327,918	73,90616
10	16	804,065	3114,924	2248,688	1072,075	59,17466
10	32	805,018	2551,711	1746,097	1068,351	34,73498
10	50	804,278	2150,469	1627,939	1088,98	28,47177
20	1	966,037	466,1457	980,3577	961,4637	18,99627
20	4	967,156	1801,13	1589,757	950,6511	27,12491
20	8	974,456	3014,723	1617,457	938,7445	25,69724
20	16	974,488	3070,568	1556,616	823,7388	20,62048
20	32	975,684	2953,614	1375,961	835,0947	16,37674
20	50	979,862	2648,939	1338,848	826,6539	16,72307
50	1	1453,204	377,0753	792,0622	430,7698	
50	4	1458,584	1650,989	1323,69	454,7235	
50	8	1464,045	2971,777	1333,285	467,0974	
50	16	1465,773	3156,338	1273,641	924,3817	
50	32	1474,184	2791,174	1061,281	1283,26	
50	50	1474,042	2806,887	1033,337	1259,559	
100	1	2227,963	372,438	776,6492	927,7788	
100	4	2236,103	1275,748	1053,888	1247,085	
100	8	2238,994	2235,385	1117,856	1240,163	
100	16	2239,005	2516,89	997,5129	853,6048	
100	32	2246,501	2343,782	948,8844	958,6458	
100	50	2209,238	2333,517	885,4127	974,6334	

Tabulka B.1 – Výsledky testů hardwarových konfigurací

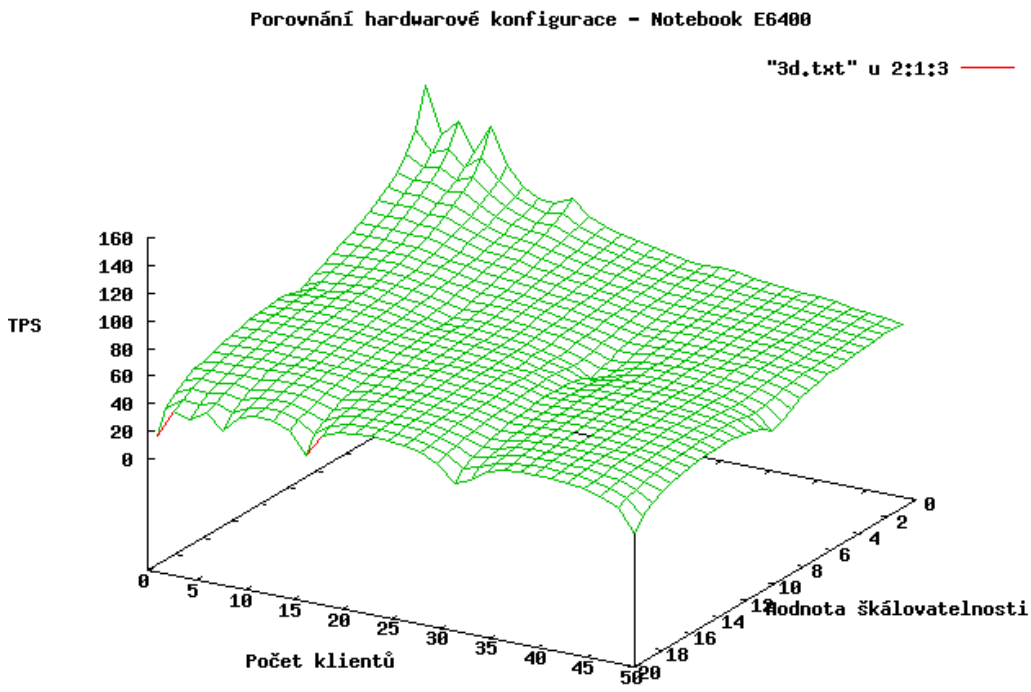
3D grafy závislosti TPS na počtu klientů a hodnotách škálovatelnosti



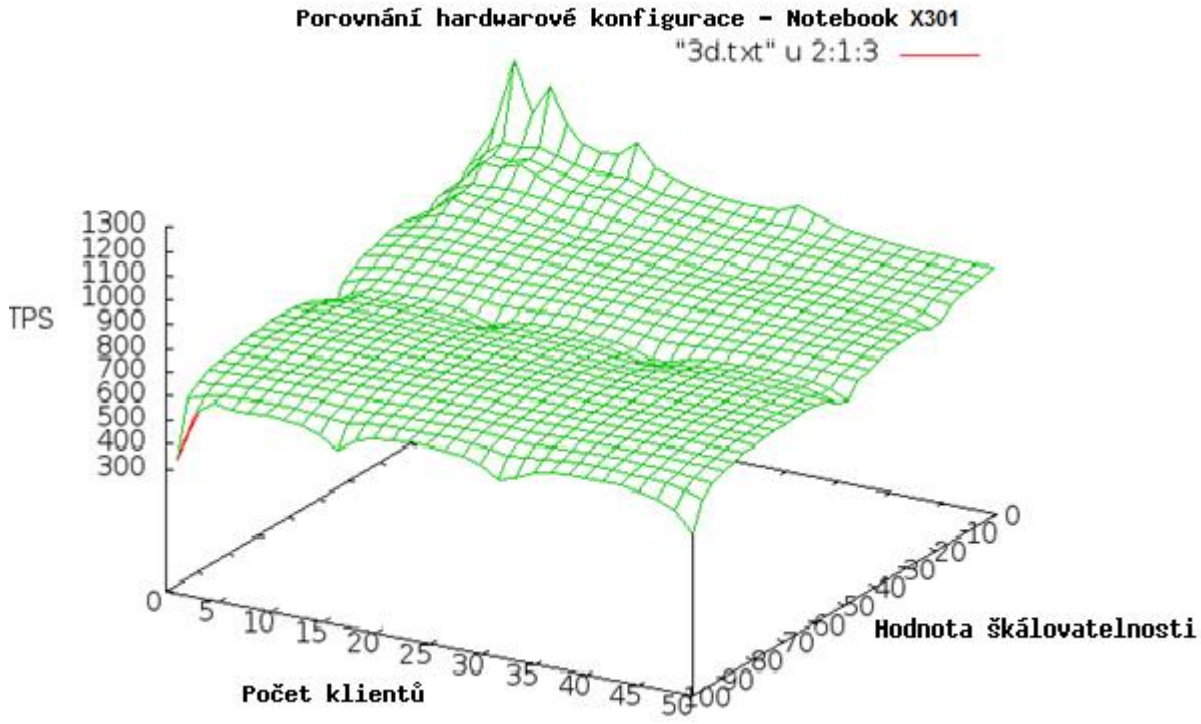
Obrázek B.2 – 3D graf serveru Minerva3 s optimalizovanou konfigurací



Obrázek B.3 – 3D graf serveru Minerva2 s optimalizovanou konfigurací



Obrázek B.4 – 3D graf notebooku E6400 s optimalizovanou konfigurací

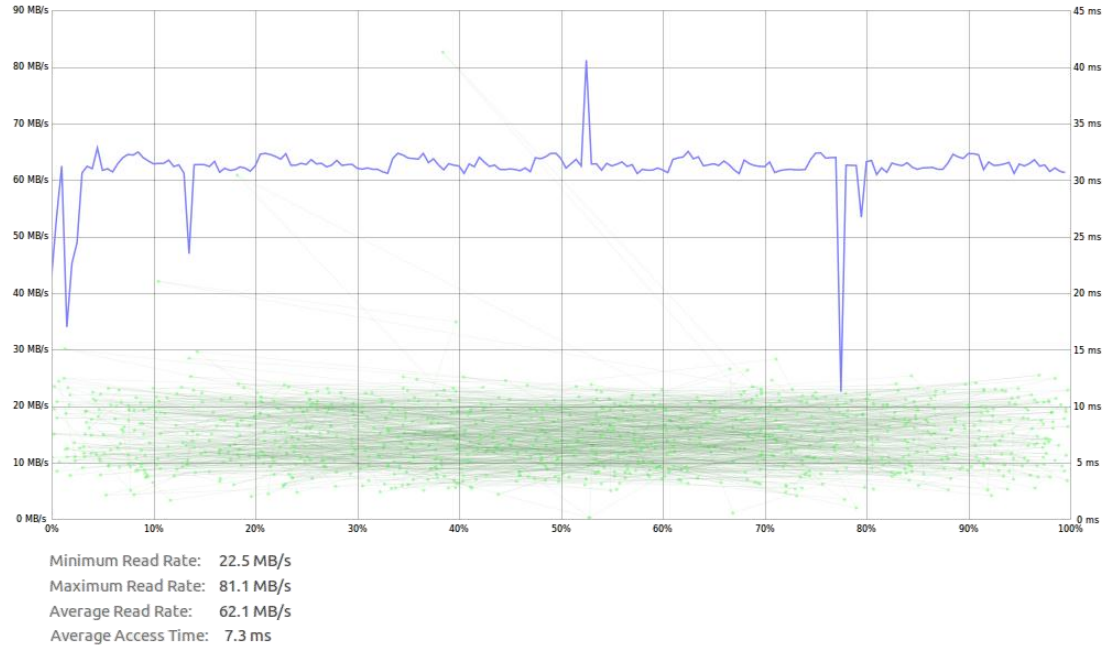


Obrázek B.5 – 3D graf notebooku X301 s optimalizovanou konfigurací

Grafy rychlosti čtení disků s PostgreSQL databází

Notebook E6400

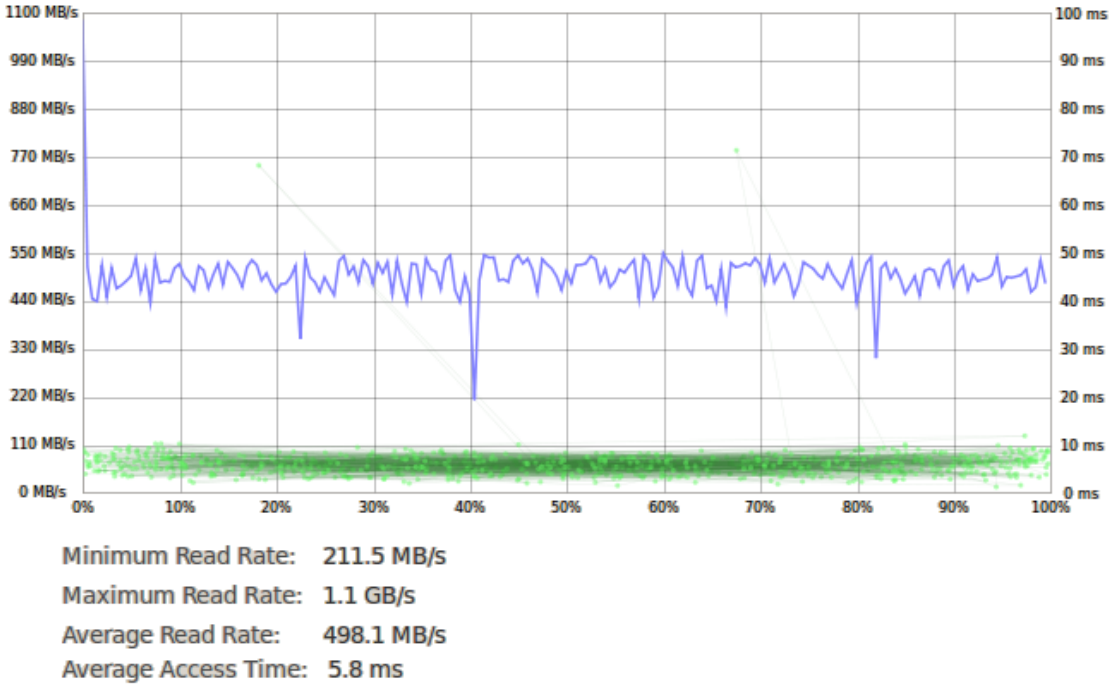
- magnetický disk 7200 otáček za sekundu, 16 MB vyrovnávací paměť



Obrázek B.6 – Graf rychlosti čtení disku notebooku E6400

Server Minerva3

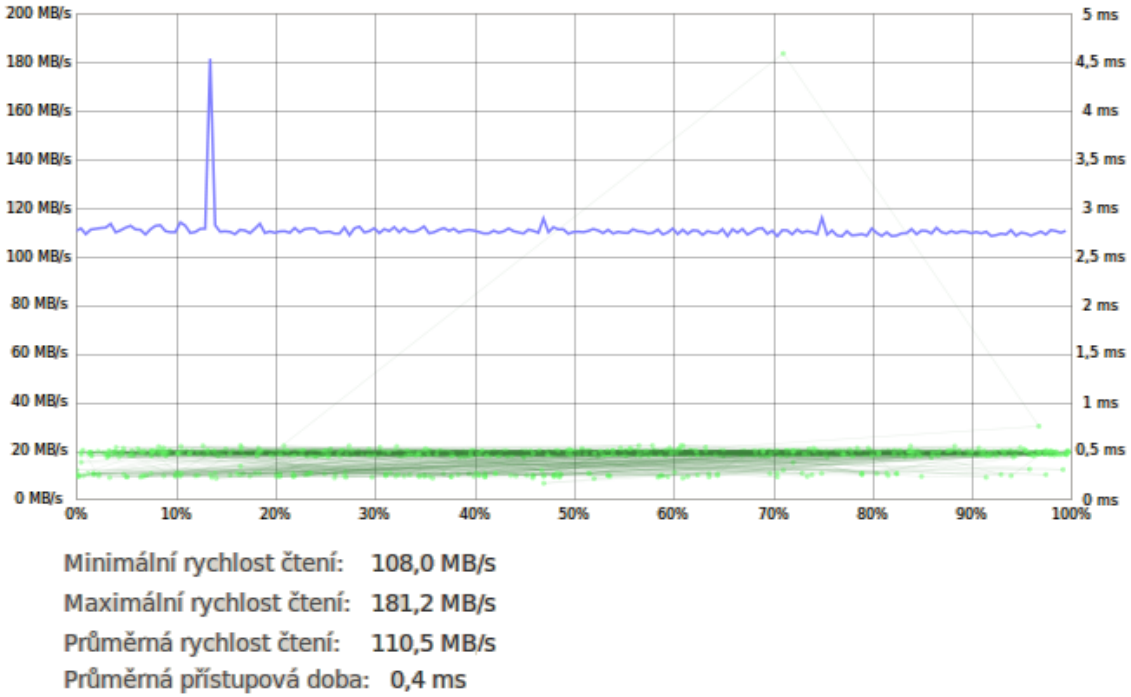
- Diskové pole SAS RAID6



Obrázek B.7 – Graf rychlosti čtení pole disků serveru Minerva3

Notebook

- SSD disk



Minimální rychlost čtení: 108,0 MB/s
Maximální rychlost čtení: 181,2 MB/s
Průměrná rychlost čtení: 110,5 MB/s
Průměrná přístupová doba: 0,4 ms

Obrázek B.8 – Graf rychlosti čtení disku notebooku

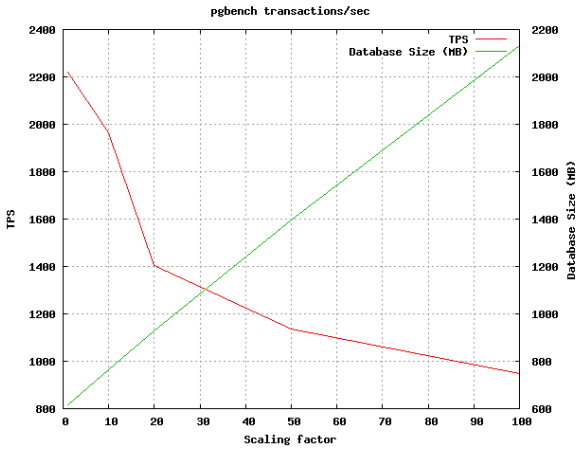
Příloha C. Vygenerovaný záznam výsledků testů serveru Minerva2

Příklad vygenerovaného hypertextového souboru výsledků testů pro server Minerva2.

Set 1 : Test: transaction file:/homes/eva/xp/xpauli05/DIP/bench/scripts/pgbench-default/tpc-b.sql, scales:1 10 20 50, clients:1 4 8 16 32 50, times:3, workers:1

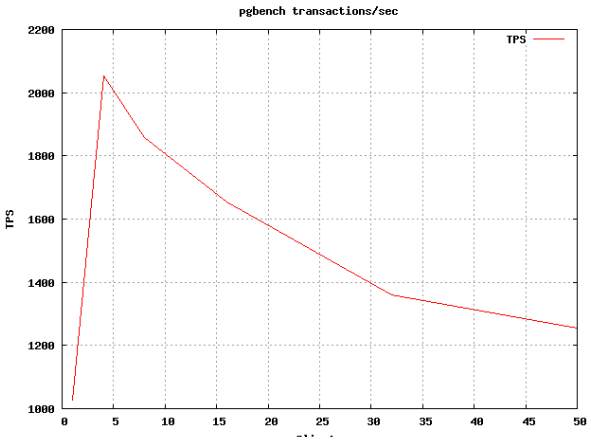
Averages for test set 1 by scale:

set	scale	tps	avg_latency	90% <	max_latency
1	1	2220.12213	11.04	23.16	381.443
1	10	1963.75781	10.29	14.24	2371.64
1	20	1404.98769	13.25	13.50	3040.91
1	50	1135.63193	16.83	12.79	4484.13
1	100	948.426830	20.20	11.20	5657.99



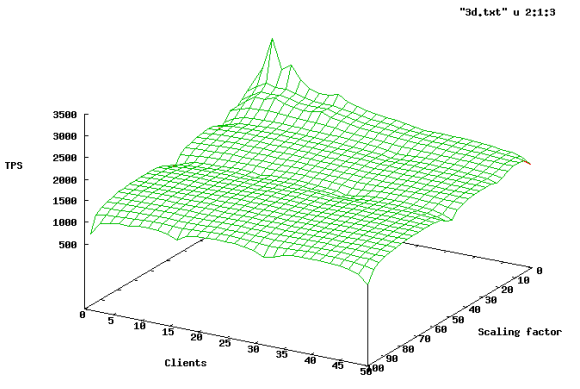
Averages for test set 1 by clients:

set	clients	tps	avg_latency	90% <	max_latency
1	1	1026.41462	1.032	1.042	2159.33
1	4	2053.05073	2.392	1.81	3556.32
1	8	1859.18288	4.86	3.835	3386.21
1	16	1655.25357	10.73	9.756	3283.17
1	32	1358.70467	24.98	26.57	3400.92
1	50	1254.90520	41.96	46.88	3337.38



Averages for test set 1 by scale and client:

set	scale	clients	tps	avg_latency	90% <	max_latency
1	1	1	1473.30788	0.674	0.712	24.851
1	1	4	3805.43492	1.042	1.396	81.179
1	1	8	2752.64437	2.895	4.745	92.242
1	1	1	2193.79708	7.285	14.09	618.202
1	1	3	1683.95298	18.98	40.72	406.365
1	1	5	1411.59555	35.39	77.29	1065.81
1	10	1	1152.96609	0.862	1.002	2089.39
1	10	4	2495.13580	1.593	1.788	2368.76
1	10	8	2499.06577	3.192	3.432	2448.12
1	10	1	2267.25157	7.048	8.264	2594.07
1	10	3	1741.93940	18.35	24.20	2347.05
1	10	5	1626.18822	30.72	46.77	2382.44
1	20	1	970.054965	1.026	1.26	2201.09
1	20	4	1583.27620	2.516	2.251	2780.94
1	20	8	1609.16029	4.962	4.204	3222.62
1	20	1	1566.85942	10.20	9.202	3338.50
1	20	3	1362.16970	23.48	23.89	3711.23
1	20	5	1338.40555	37.33	40.23	2991.06
1	50	1	784.683568	1.269	1.461	3012.56
1	50	4	1321.77384	3.017	2.316	4079.18
1	50	8	1330.44428	6.003	4.051	4092.34
1	50	1	1273.12087	12.55	9.131	4222.53
1	50	3	1066.93334	29.98	22.94	6677.70



1	50	5	1036.83571	48.2	36.86	4820.46
1	100	1	751.060593	1.331	0.774	3468.76
1	100	4	1059.63290	3.792	1.302	8471.52
1	100	8	1104.59967	7.247	2.743	7075.72
1	100	1	975.238908	16.60	8.084	5642.53
1	100	3	938.527911	34.10	21.08	3862.27
1	100	5	861.500989	58.16	33.27	5427.14

Detail for test set 1:

set	test	scale	db_size (MB)	clients	tps	max_latency	chkpts	buf_check	buf_clean	buf_backend	buf_alloc	max_clean
1	1	1	611	1	1465.0278	9.929	10	30637	0	8685	8902	0
1	2	1	618	1	1482.7068	58.074	10	28818	0	8585	9629	0
1	3	1	618	1	1472.1889	6.551	10	29623	0	8546	10963	0
1	4	1	612	4	3771.2583	97.955	29	99413	0	22087	23551	0
1	5	1	614	4	3811.1231	44.181	29	96992	8	22012	24486	0
1	6	1	614	4	3833.9232	101.401	29	95664	4	22269	23963	0
1	7	1	614	8	2754.3976	100.9	21	70936	68	16352	18349	0
1	8	1	615	8	2746.6040	135.177	21	69294	902	15919	18395	0
1	9	1	615	8	2756.9313	40.649	21	69317	0	15932	18394	0
1	10	1	615	16	2239.8510	133.857	18	59624	0	13344	15689	0
1	11	1	614	16	2222.9721	129.788	18	60563	0	13255	15610	0
1	12	1	615	16	2118.5681	1590.96	17	56314	0	12263	14616	0
1	13	1	617	32	1675.9911	393.528	13	42789	0	10231	12919	0
1	14	1	617	32	1677.6251	411.276	14	42747	0	10642	12976	0
1	15	1	615	32	1698.2426	414.291	14	45065	0	10322	12655	0
1	16	1	619	50	1414.3041	1006.39	11	37082	0	8440	10902	0
1	17	1	620	50	1407.2689	1049.32	11	36570	0	8365	10970	0
1	18	1	618	50	1413.2136	1141.72	11	36878	0	8359	10705	0
1	19	10	745	1	1169.6267	1788.92	81	781707	12036	43951	26793	3
1	20	10	767	1	1132.8950	2414.59	70	671815	28463	51929	31581	3
1	21	10	765	1	1156.3764	2064.65	71	679607	28714	47425	32087	13
1	22	10	757	4	2485.4939	2450.94	164	154080	91178	66726	66628	315
1	23	10	765	4	2486.0929	2317.63	160	148718	10324	70557	69210	416
1	24	10	764	4	2513.8205	2337.71	162	150599	10422	71466	69980	424
1	25	10	765	8	2505.9660	2195.39	161	149690	10456	74009	70618	452
1	26	10	767	8	2508.6674	2877.98	163	152010	10460	70517	71133	448
1	27	10	766	8	2482.5638	2270.98	162	150868	10225	70954	70120	410
1	28	10	765	16	2248.6884	2853.30	146	136020	91656	67976	63742	348
1	29	10	768	16	2305.1061	2352.20	150	139955	92314	68090	65337	323
1	30	10	768	16	2247.9601	2576.70	147	137249	89737	67987	63662	301
1	31	10	768	32	1731.7908	2317.39	111	104099	63354	60634	48982	59
1	32	10	768	32	1746.0973	2330.83	113	105704	62155	60135	49490	46
1	33	10	768	32	1747.9299	2392.94	113	106078	61667	60707	49401	52
1	34	10	767	50	1618.9669	2480.01	105	982807	55022	60730	45994	38
1	35	10	767	50	1631.6589	2142.26	106	998303	55465	59499	46398	24
1	36	10	767	50	1627.9388	2525.05	106	995368	54851	60861	46410	30
1	37	20	895	1	982.39526	2187.33	85	701782	12428	87712	66805	980
1	38	20	934	1	947.41197	2140.72	73	562850	14812	113179	65274	1146
1	39	20	935	1	980.35765	2275.22	75	578609	15373	106557	67510	1237
1	40	20	910	4	1566.7838	2669.95	135	110776	15567	216678	11230	1423
1	41	20	929	4	1593.2874	2609.96	131	103444	17442	245013	11447	1633
1	42	20	928	4	1589.7573	3062.92	130	103114	16459	238515	11392	1532
1	43	20	926	8	1618.3390	3085.38	135	108963	16484	241184	11728	1546

1	44	20	931	8	1617.4568	3590.05	133	106143	16876	246987	11696	1586
1	45	20	931	8	1591.6849	2992.42	132	106062	16680	246446	11576	1559
1	46	20	929	16	1588.7143	2568.16	130	103778	17875	232518	11503	1666
1	47	20	929	16	1556.6159	3778.03	129	103717	17596	219897	11286	1645
1	48	20	930	16	1555.2480	3669.32	129	103673	17506	224436	11321	1639
1	49	20	931	32	1375.9607	3884.20	112	886486	19197	162158	98263	1733
1	50	20	932	32	1327.5322	3883.33	107	847873	18454	157306	94082	1683
1	51	20	928	32	1383.0160	3366.15	113	897575	19512	163460	99076	1788
1	52	20	934	50	1349.7728	2789.46	111	877436	19293	155755	96940	1738
1	53	20	934	50	1338.8483	2985.52	110	875065	19255	155755	96297	1762
1	54	20	935	50	1326.5954	3198.20	109	856954	18934	150738	94995	1749
1	55	50	1344	1	769.03680	3063.49	21	91430	26000	470238	96562	2537
1	56	50	1404	1	792.06219	2433.75	19	58788	28451	436434	10291	2769
1	57	50	1410	1	792.95170	3540.43	19	56982	28307	429343	10179	2735
1	58	50	1375	4	1360.5800	3241.78	35	124256	15275	1080204	18049	1525
1	59	50	1397	4	1323.6900	5208.74	34	116149	15803	1052195	17693	1578
1	60	50	1401	4	1281.0514	3787.01	33	113007	17616	1006308	17074	1754
1	61	50	1387	8	1343.8717	4186.85	34	116762	15886	1073231	18075	1587
1	62	50	1399	8	1333.2847	4338.98	34	114222	14906	1061577	17752	1483
1	63	50	1403	8	1314.1762	3751.20	34	116119	16242	1054038	17683	1617
1	64	50	1391	16	1267.4680	4336.75	33	117691	17538	991244	16963	1743
1	65	50	1400	16	1273.6409	4475.68	33	111684	17841	1005499	17214	1773
1	66	50	1403	16	1278.2536	3855.17	33	114098	17669	1017024	17308	1756
1	67	50	1401	32	1061.2812	10507.9	28	100092	23325	766611	14396	2303
1	68	50	1411	32	1095.2375	4061.39	29	99709	23255	811027	14992	2281
1	69	50	1406	32	1044.2812	5463.82	27	92672	21873	760172	13999	2161
1	70	50	1395	50	1033.3372	3727.22	27	96070	22805	750445	14162	2241
1	71	50	1412	50	1048.7120	4345.80	27	93272	23584	757462	14244	2317
1	72	50	1411	50	1028.4578	6388.35	27	92836	21722	758791	13991	2145
1	73	100	2092	1	689.26681	3971.20	22	93197	19120	463339	10464	1880
1	74	100	2138	1	787.26572	2954.01	21	54492	20970	454880	12376	2083
1	75	100	2145	1	776.64923	3481.06	21	48790	20601	451374	13458	2052
1	76	100	2114	4	1170.9093	9377.11	31	72982	10176	871014	19402	1016
1	77	100	2138	4	954.10103	6551.69	29	86164	11075	779340	16115	1099
1	78	100	2145	4	1053.8883	9485.77	28	63257	11256	765048	17757	1119
1	79	100	2119	8	1155.7850	10715.3	30	71973	10774	825035	17743	1070
1	80	100	2141	8	1040.1582	4861.08	30	83440	99716	846401	17526	997
1	81	100	2146	8	1117.8557	5650.71	30	69316	11699	821443	18751	1161
1	82	100	2119	16	834.39529	9376.69	28	99337	12355	799464	14576	1217
1	83	100	2141	16	997.51292	3752.25	28	71764	13811	733209	16699	1372
1	84	100	2146	16	1093.8085	3798.66	29	68000	14880	752085	17248	1473
1	85	100	2138	32	948.88443	4508.38	25	62589	18036	594873	14499	1783
1	86	100	2144	32	903.33152	3571.01	25	61184	17798	602310	15448	1749
1	87	100	2146	32	963.36778	3507.41	25	59103	19187	599773	15415	1891
1	88	100	2092	50	799.37586	7151.40	25	100713	14629	604716	12133	1457
1	89	100	2115	50	899.71439	4909.61	24	56450	15566	592501	15118	1556
1	90	100	2114	50	885.41270	4220.41	23	53674	15265	584775	15066	1525

Příloha D. Instrukce k instalaci, nastavení a spuštění testů

Instrukce k instalaci, nastavení a spuštění testů (soubor README na přiloženém CD)

```
==========================
=== pgbench-with-fouine ===
==========================

Prerequisites
=============

* For all tests::

    gnuplot - Gnuplot is normally included in linux distribution. Otherwise
      It needs to be installed by packaging system or compiled from source code.

    python - Installation is possible by packaging system or by compiling source code.

    PostgreSQL database (version 9 and above) - Installation is possible by packaging system or
      by compiling source code. Source code includes installation notes in "INSTALL" file.
      Setup for logging by syslog:
        - Edit your postgresql.conf file (usually located in /var/lib/pgsql/data/) and set
            the following configuration directives:
            log_destination = 'syslog'
            redirect_stderr = off
            silent_mode = on
        - To log queries slower than n milliseconds (To log every query executed,
            set log_min_duration_statement to 0. Set it to -1 to disable query logging.):
            log_min_duration_statement = n
            log_duration = off
            log_statement = 'none'
        - Then edit your /etc/syslog.conf to set up a PostgreSQL facility:
            local0.*            -/var/log/pgsql
            You should also ignore PostgreSQL facility for the default log file otherwise
              you will log the queries twice:
              *.info;mail.none;authpriv.none;cron.none;local0.none            /var/log/messages
        - Restart syslogd and PostgreSQL.

    pgbench binary - Installation is possible by packaging system or by compiling its source code
      which is included in folder contrib/pgbench of PosgreSQL installation.

* For tests with custom database and log parsing::

    php interpret binary - php-cli package needs to be installed by packaging system
```

Setup
=====

* Create databases for your test and for the results::

    createdb results
    createdb pgbench

    Both databases can be the same, but there may be more shared_buffers
    cache churn in that case. Some amount of cache disruption
    is unavoidable unless the result database is remote, because
    of the OS cache. The recommended and default configuration
    is to have a pgbench database and a results database. This also
    keeps the size of the result dataset from being included in the
    total database size figure recorded by the test.

* Initialize the results database by executing::

    psql -f scripts/init/resultdb.sql -d results

* Setup config file (default values are mentioned)::

    path to requested binaries:
      PHPBIN="/usr/pgsql-9.0/bin/pgbench"
      PGBENCHBIN="/usr/bin/php" (mandatory for tests with custom database and log parsing)

    database connection to test and result database (can be the same):
      TESTHOST="localhost"
      TESTUSER="postgres"
      TESTPORT="5432"
      TESTDB="pgbench"
      RESULTHOST=$TESTHOST
      RESULTUSER=$TESTUSER
      RESULTPORT=$TESTPORT
      RESULTDB="results"

    testing database log file (both settings mandatory for tests with custom
      database and log parsing):
      DABASELOGFILE="/var/log/pgsql"
      DATABASELOGTYPE="syslog"

Running tests
=============

* Edit the config file (section "Test customizations") to run the tests you want
  Default settings will run 3 tests (SETTIMES) for scale values (SCALES): 1, 10, 20, 50, 100 and
    client numbers (SETCLIENTS): 1, 4, 8, 16, 32, 50. Every test will run 300 seconds (RUNTIME).
    SCALES="1 10 20 50 100"
    SETCLIENTS="1 4 8 16 32 50"
    SETTIMES=3
    RUNTIME=300

* Execute::

    ./runTest

In order to execute all the tests

This will create a test set entry with default description from config $TESTSETDESCRIPTION:
'Test: transaction file:$SCRIPT, scales:$SCALES, clients:$SETCLIENTS, times:$SETTIMES, workers:$MAX_WORKERS'
You may want to rename that using something like this::

```
psql -c "UPDATE testset SET info='better name' WHERE set=%set_number%" -d results
```

Results
=======

* You can check results even as the test is running with::

```
psql -d results -f scripts/reports/report.sql
```

This is unlikely to disrupte the test results very much unless you've
run an enormous number of tests already.

* Other useful reports you can run include:
  * fastest.sql
  * summary.sql
  * bufreport.sql
  * bufsummary.sql

* Once the tests are done, the results/ directory will include
a HTML subdirectory for each test giving its results,
in addition to the summary information in the results database.

* The results directory will also include its own index HTML file that
shows summary information and plots for all the tests.

* If you manually adjust the test result database, you can
then manually regenerate the summary graphs by running::

```
./webreport
```

Příloha E. CD s programem a výsledky testů

Přiložené CD obsahuje adresáře:

- pgbench-with-fouine – Adresář obsahuje implementovaný testovací program pgbench-with-fouine
- výsledky testů – Adresář obsahuje výsledky provedených testů
- diplomová práce – Adresář obsahuje soubor diplomové práce ve formátu PDF