

Vyhodnocení výkonu a migrace nerelačních distribuovaných datových skladů

Performance Evaluation and the Migration of Non-relational
Distributed Data Stores

Bc. Miroslav Oujeský

Diplomová práce
2012



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2011/2012

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Miroslav OUJESKÝ**
Osobní číslo: **A10389**
Studijní program: **N 3902 Inženýrská informatika**
Studijní obor: **Informační technologie**

Téma práce: **Vyhodnocení výkonu a migrace nerelačních distribuovaných datových skladů**

Zásady pro vypracování:

1. Prostudujte problematiku relačních (MySQL apod.) a nerelačních databázových systémů (Hadoop, MongoDB, CouchDB, Cassandra, apod.)
2. Najděte a na příkladech vhodně popište datové-objemové limity relačního databázového systému MySQL.
3. Vyhodnoťte výkon operací čtení/zápis v obou typech databází při velmi vysokých objemech v nich existujících dat a najděte extrémy, od kterých relační databáze přestávají být prakticky použitelné.
4. Analyzujte datové struktury systému Webnode a navrhněte, které součásti bude vhodné migrovat na nerelační databázové systémy.
5. Proces migrace popište, částečně realizujte a v praxi porovnejte výkon migrovaných dat s daty relačními.
6. Vyberte si modelový příklad čtení a zápisu dat a ukažte rozdíl mezi relační a nerelační databází.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. TIWARI, Shashank. Professional NoSQL. [s.l.] : Wiley, 2011. 384 s. ISBN 978-0470942246.
2. HEWITT, Eben. Cassandra: The Definitive Guide. [s.l.] : OReilly, 2010. 330 s. ISBN 978-1-4493-9041-9.
3. ANDERSON, J. Chris; LEHNARDT, Jan; SLATER, Noah. CouchDB: The Definitive Guide : Time to Relax. [s.l.] : OReilly, 2010. 272 s. ISBN 978-0-596-15589-6.
4. HOLT, Bradley. Scaling CouchDB : Replication, Clustering, and Administration. [s.l.] : OReilly, 2011. 50 s. ISBN 978-1-4493-0343-3.
5. WHITE, Tom. Hadoop: The Definitive Guide. Second Edition. [s.l.] : OReilly, 2010. 626 s. ISBN 978-1449389734.
6. CHODOROW, Kristina; DIROLF, Michael. MongoDB: The Definitive Guide. [s.l.] : OReilly, 2010. 216 s. ISBN 978-1449381561.
7. CHODOROW, Kristina. Scaling MongoDB. [s.l.] : OReilly, 2011. 62 s. ISBN 978-1449303211.
8. BROWN, Amy; WILSON, Greg. The Architecture of Open Source Applications. [s.l.] : [s.n.], 2011. 432 s. ISBN 978-1-257-63801-7.

Vedoucí diplomové práce:

Ing. Kateřina Ježková

Ústav automatizace a řídicí techniky

Datum zadání diplomové práce:

24. února 2012

Termín odevzdání diplomové práce:

21. května 2012

Ve Zlíně dne 24. února 2012



prof. Ing. Vladimír Vašek, CSc.
děkan



doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby,
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce,
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3,
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona,
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo - diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše),
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům,
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor,
- odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....

podpis diplomanta

OBSAH

ÚVOD	10
I TEORETICKÁ ČÁST	11
1 RELAČNÍ DATABÁZOVÉ SYSTÉMY	12
1.1 ACID	13
1.1.1 Atomičnost	13
1.1.2 Konzistence	13
1.1.3 Izolace	14
1.1.4 Trvanlivost	14
1.1.5 Implementace ACID požadavků	14
1.2 DATOVĚ-OBJEMOVÉ LIMITY MYSQL	15
1.2.1 MySQL	15
1.2.2 Limity	15
1.2.3 Zámky	16
1.2.4 Škálování	17
2 NOSQL DATABÁZE	18
2.1 ROZDÍLY OPROTI RELAČNÍM DATABÁZÍM	18
2.2 CAP TEORÉM	19
2.3 ARCHITEKTURA NOSQL DATABÁZÍ	20
2.4 MAPREDUCE	21
3 ROZDĚLENÍ NOSQL DATABÁZÍ	23
3.1 ÚLOŽIŠTĚ KLÍČ-HODNOTA	23
3.2 DOKUMENTOVÁ DATABÁZE	24
3.3 GRAFOVÁ DATABÁZE	26
3.4 OBJEKTOVÁ DATABÁZE	28
3.5 SLOUPCOVÉ ÚLOŽIŠTĚ	29
4 DŮLEŽITÉ VLASTNOSTI A PARAMETRY NOSQL DATABÁZÍ	31
4.1 VYSOKÁ DOSTUPNOST	31
4.1.1 Single Point of Failure	32
4.1.2 Multi-Version Concurrency Control	33
4.2 VYSOKÝ VÝKON	34
4.3 ŠKÁLOVATELNOST	34
4.3.1 Replikace master-slave	35
4.3.2 Replikace master-master	36
4.3.3 Sharding	36

4.4	FLEXIBILNÍ DATOVÁ SCHÉMATA.....	37
5	NEJROZŠÍŘENĚJŠÍ NOSQL DATABÁZE.....	39
5.1	CASSANDRA	39
5.2	MONGODB	41
5.3	COUCHDB	42
5.4	REDIS	44
5.5	HBASE.....	46
5.6	RIAK	48
5.7	NEO4J	49
II	PRAKTICKÁ ČÁST.....	52
6	TESTY DATOVĚ-OBJEMOVÝCH LIMITŮ MYSQL	53
6.1	OPERACE ČTENÍ.....	53
6.2	KOMBINACE OPERACÍ ČTENÍ A ZÁPISU	54
7	VÝKONNOSTNÍ SROVNÁNÍ NOSQL DATABÁZÍ.....	56
7.1	HARDWAROVÁ KONFIGURACE.....	56
7.2	YCSB BENCHMARK	56
7.3	VÝKON PŘI VELKÉM MNOŽSTVÍ ČTENÍ	57
7.4	VÝKON PŘI VELKÉM MNOŽSTVÍ ZÁPISŮ	58
7.5	VÝKON PŘI POUZE ČTENÍ.....	59
7.6	CELKOVÉ SROVNÁNÍ VÝKONU	60
8	ANALÝZA VHODNOSTI PRO MIGRACI DO NOSQL DATABÁZE..	62
8.1	WEBNODE.....	62
8.2	SUBSYSTÉMY VHODNÉ PRO MIGRACI	62
8.2.1	Uživatelské sessions	63
8.2.2	Logování	63
8.2.3	Centrum plateb.....	64
9	PROCES MIGRACE DO NOSQL DATABÁZE	65
9.1	ČÁSTEČNÁ REALIZACE MIGRACE.....	66
10	MODELOVÝ PŘÍKLAD ČTENÍ A ZÁPISU	68
10.1	PŘÍKLAD ČTENÍ A ZÁPISU V SYSTÉMU MYSQL	68
10.2	PŘÍKLAD ČTENÍ A ZÁPISU V SYSTÉMU CASSANDRA.....	70
	ZÁVĚR.....	72
	CONCLUSION	73
	SEZNAM POUŽITÉ LITERATURY.....	73

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	76
SEZNAM OBRÁZKŮ	77
SEZNAM TABULEK	78
SEZNAM PŘÍLOH	79

ABSTRAKT

Cílem této práce je rozbor tematiky vysoce výkonných a vysoce dostupných distribuovaných datových skladů - databází souhrnně označovaných jako NoSQL - a jejich porovnání s tradičními relačními databázovými systémy v prostředí vysoké zátěže. Je zde provedeno výkonnostní srovnání obou metod na vybraných částech systému Webnode a na základě výsledků je navrženo, které části migrovat na nerelační databázový systém.

Klíčová slova: NoSQL, Webnode, nerelační databáze, distribuovaná databáze, vysoký výkon, vysoká dostupnost

ABSTRACT

The aim of this thesis is the research of high-performance and high-availability distributed data stores - databases commonly known as NoSQL - and the comparison against more traditional relational database systems in high load environment. The performance evaluation of both methods was made on selected parts of Webnode system and based on the evaluation results migration to non-relational database is prepared.

Keywords: NoSQL, Webnode, non-relational database, distributed database, high performance, high availability

Děkuji vedoucí diplomové práce slečně Ing. Kateřině Ježkové za její spolupráci a příkladné vedení. Dále také děkuji kolegům z celého kolektivu a především z vývojového týmu firmy Webnode AG za podnětné informace a veškerou podporu. Zvláštní poděkování pak patří panu Mgr. Petrovi Krátkému za přípravu hardwaru pro výkonnostní testy a slečně Ing. Kateřině Štarhové za jazykové korektury této práce.

ÚVOD

Relační databázové systémy byly vždy velmi výkonné pro práci s pevně strukturovanými daty. Zároveň umožňují na těchto datech provádět širokou řadu dynamických dotazů, které splňují nejrůznější požadavky.

I přesto, že v tomto odvětví není pro relační databázové systémy konkurence, vysoce dostupné internetové aplikace pracující s velkými datovými objemy vyžadují odlišný přístup, který relační databázové systémy nejsou zcela schopny zajistit. Vhodným příkladem jsou například dnes velmi oblíbené sociální sítě, které pracují s komplexně propojenými vyvíjejícími se daty milionů uživatelů.

Vývoj rozsáhlých internetových aplikací přináší spíše objektově orientovaný pohled, se kterým je spojena nutnost definovat velmi komplexní datová schémata. Především pak požadavky na vysokou dostupnost a rychlost odezvy nejsou složité dotazy v jazyce SQL schopny udržet.

V této práci se budeme věnovat široké rodině databázových systémů, souhrnně označovaných jako NoSQL. Tyto dokáží plnit specifické požadavky moderních internetových aplikací, jako je snadná škálovatelnost, flexibilní datové schéma nebo práce v reálném čase.

V teoretické části se seznámíme s nejběžnějšími druhy NoSQL databází a účelů, ke kterým jsou vhodné. Dále zde nalezneme představení vybraných nejrozšířenějších implementací, které jsou dostupné a především spolehlivé k nasazení do produkčního prostředí.

V praktické části této práce se potom budeme věnovat především výkonnostním srovnáním vybraných NoSQL databázových systémů v porovnání s relačním databázovým systémem MySQL. Na základě těchto poznatků, pak bude provedena analýza subsystémů projektu Webnode a následně návrh vhodného řešení pro migraci z MySQL na zvolený nerelační databázový systém.

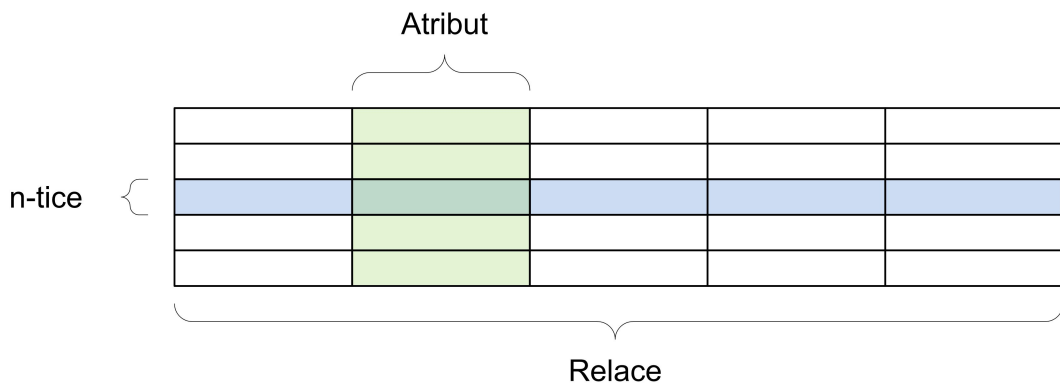
Je však nutné poznamenat, že NoSQL databázové systémy nejsou „magickým“ řešením na všechny problémy a proto je potřeba velmi dobře zvážit, zda případná migrace do nerelačního databázového systému je přínosná.

I. TEORETICKÁ ČÁST

1 RELAČNÍ DATABÁZOVÉ SYSTÉMY

Relační databázové systémy (relační systémy řízení báze dat) jsou v dnešní době nejrozšířenější databázové systémy používané v prostředí internetových aplikací.

Jejich základem je pevně definované datové schéma v podobě tabulek (relací) obsahující sloupce (atributy) různých datových typů (domén) a vzájemných vztahů mezi těmito tabulkami. Tabulka (relace) je definována jako množina řádků (n-tic), které mají stejné sloupce (atributy). Řádek typicky obsahuje informace o popisovaném objektu reálného světa. Na obrázku 1 je naznačena podoba relace.



Obr. 1. Relace v relačním databázovém systému

Data jsou z databázového systému získávána pomocí dotazovacího jazyka SQL (Structured Query Language) nebo jeho odvozenin. Jednotlivé tabulky je možné na základě definovaných vztahů mezi nimi v dotazech spojovat (JOIN) a data je možné agregovat pomocí vestavěných agregačních funkcí.

Podstatnou částí návrhu datového schématu je normalizace. Jedná se o sadu procedur, které zajišťují eliminaci neatomických hodnot a redundance (duplikace) dat v tabulkách, což přeneseně zajišťuje odolnost vůči ztrátě datové integrity. Normalizace je obvykle dosaženo splněním některé z normálních forem. Na druhou stranu normalizace určitým způsobem zvyšuje komplexnost databázového schématu a zvyšuje výpočetní náklady na provádění operací z důvodu spojování více tabulek[14].

Pro zvýšení výkonu provádění výběrů, je možné k vybraným sloupcům nebo množinám sloupců vytvořit indexy. Při dotazu, který využije takovýto index se potom neprovádí prohledávání celé tabulky, ale optimalizovanou cestou je nalezen konkrétní řádek nebo množina řádků. Indexy jsou typicky implementovány jako B+ strom, R strom nebo

bitová mapa. Efektivní využití indexů může dramaticky zvýšit výkon databázového systému.

Další velmi důležitou vlastností je použití transakcí. Transakce zajišťují atomičnost posloupnosti operací - ke změně v datech dojde jen a pouze tehdy, pokud všechny operace ze kterých se transakce skládá jsou úspěšně dokončeny[14].

Relační databázové systémy jsou dostupné již velmi dlouhou dobu, proto jsou velmi spolehlivé a časem prověřené, vhodné i pro použití v kritických oblastech, kde je nutné mít velmi silné záruky spolehlivosti. Dále to znamená velmi snadné osvojení ze strany programátorů a jednoduché použití v internetových projektech.

Mezi nejznámější relační databázové systémy používané v prostředí internetových aplikací patří MySQL, PostgreSQL, SQLite, Microsoft SQL Server a další.

1.1 ACID

ACID (Atomicity, consistency, isolation, durability) je sada požadavků, kterou musí splňovat databázový systém, aby bylo zaručeno, že databázová transakce je spolehlivě zpracována.

Tento koncept byl definován koncem 70. let 20. století J. Grayem, který také vyvinul technologie, jež umožnily tyto požadavky automatizovaně splnit. V roce 1983 byla tomuto konceptu A. Reuterem a T. Härderem dáno označení ACID, které je zkratkou prvních písmen anglických názvů požadavků[13].

Jednotlivé požadavky jsou specifikovány následovně:

1.1.1 Atomičnost

Atomičnost (anglicky *Atomicity*) vyžaduje, aby každá transakce buď byla dokončena kompletně nebo vůbec. Pokud jedna část transakce selže, pak selže celá transakce a nedojde ke změně stavu databáze. Atomický systém musí garantovat atomičnost v jakékoliv situaci, včetně výpadku energie, chyb nebo pádu systému.

1.1.2 Konzistence

Konzistence (anglicky *Consistency*) zaručuje, že jakákoliv transakce přesune databázi z jednoho platného stavu do dalšího platného stavu. Jakákoliv data zapsaná do da-

tabáze musí být platná na základě všech definovaných pravidel, včetně integritních omezení, kaskádování a triggerů.

1.1.3 Izolace

Izolace (anglicky *Isolation*) znamená požadavek, aby žádné transakci nebylo dovoleno interferovat s jinou transakcí. Jeden ze způsobů, jak tohoto požadavku dosáhnout, je nemožnost dvou transakcí, běžících současně v jednu chvíli, měnit stejný řádek v tabulce, jelikož pořadí jejich provádění, a tím pádem i výsledný stav dat, je nepředvídatelný. Tento požadavek bývá často splněn částečně zjednodušeně, neboť má za následek velký výkonnostní postih.

1.1.4 Trvanlivost

Trvanlivost (anglicky *Durability*) vyžaduje, aby jakmile byla transakce úspěšně dokončena, tak její výsledek zůstane do budoucna zachován, a to i v případě výpadku elektrické energie, chyb nebo pádu systému. Znamená to, že po dokončení transakce jsou všechna data okamžitě uložena do perzistentního úložiště.

1.1.5 Implementace ACID požadavků

Obvykle se u relačních databázových systémů pro zajištění splnění ACID požadavků pracuje s konceptem zámků na úrovni sloupců, řádků nebo případně i celých tabulek. To má bohužel za následek výkonnostní postihy v případě většího množství souběžných transakcí. Výkonnostní postih může nastat i v případě, že z databáze data pouze čteme[14].

Další komplikace nastávají v případě distribuovaného databázového systému. V tomto případě je nutné vyloučit chyby vznikající v rámci síťového spojení, případně je některý uzel systému nucen vrátit změny provedené transakcí, v případě selhání transakce v jiném uzlu systému. Obvykle je nutné transakci provádět ve dvou fázích, kdy v první fázi je nutné všemi uzly systému potvrdit, že transakce může korektně proběhnout a ve druhé fázi je potom transakce finalizována a dokončena.

1.2 Datově-objemové limity MySQL

Pro účely této práce se budeme zabývat především vlastnostmi a omezeními relačního databázového systému MySQL v prostředí vysokých datových objemů.

1.2.1 MySQL

MySQL je jedním z nejrozšířenějších relačních databázových systémů současnosti. Jeho vývoj probíhá již od roku 1995 a nyní je systém vlastněn firmou Oracle. Je dostupný pod open source GNU GPL 2 licencí.



Obr. 2. Logo projektu MySQL

MySQL podporuje velmi širokou podmnožinu dotazovacího jazyka SQL ve standardu ANSI SQL 99 spolu s vlastními rozšířeními. Pro ukládání dat je možné zvolit z několika dostupných úložišť, z nichž každé má určité vlastnosti určující jej pro použití na konkrétní účely[8]. V této práci se budeme věnovat pouze úložišti MyISAM.

1.2.2 Limity

Limity relačního databázového systému MySQL ovlivňuje několik faktorů, které není snadné dopředu předvídat. Jedná se především o:

- Poměr operací čtení a zápisů
- Konfiguraci počtu vláken
- Počet fyzických a virtuálních procesorů serveru
- Konfiguraci cache dotazů
- Kapacitu a rychlost operační paměti RAM
- Přístupovou dobu a propustnost diskového úložiště
- Rozložení zátěže při replikaci uzlů

Pro stanovení konkrétních limitů je nutné provést analýzu těchto bodů a na jejím základě stanovit předpokládané zatížení databázového systému.

Výsledky testů pro zjištění limitů na běžném serverovém hardware se nacházejí v kapitole 6.

1.2.3 Zámky

Relační databázový systém MySQL používá pro řízení souběžného přístupu k datům koncept zámků. V případě datového úložiště MyISAM se jedná o zamykání na úrovni celých tabulek[9].

Zamykání pro operace zápisu i operace čtení má velmi podobný princip:

- Pokud není tabulka uzamčená, operace nastaví zámek pro čtení nebo zápis.
- Pokud tabulka je uzamčená, pak se požadavek na zámek zařadí do fronty zámků pro čtení nebo zápis.

V okamžiku, kdy je zámek uvolněn, je zamčení tabulky nabídnuto prvnímu procesu, který je ve frontě zámků pro zápis, následně potom prvnímu procesu ve frontě zámků pro čtení. Z toho vyplývá, že v případě velké zátěže na stráně operací zápisu budou operace čtení muset čekat, dokud nebudou všechny zápisy dokončeny. To může mít za následek nezanedbatelný výkonnostní postih pro celou aplikaci.

Důležité je, že na základě těchto pravidel může dojít k situaci, kdy zamčení tabulky pro čtení způsobí jiná operace čtení a ne zápisu. Tato situace může nastat, pokud probíhá operace čtení s dlouhým trváním a ještě než je dokončena, uzamče tabulku jiná operace čtení, která ale nemůže být dokončena, dokud neskončí původní operace čtení. Všechny ostatní příchozí operace čtení pak čekají na uvolnění zámku a nemohou být provedeny. To může způsobit velmi citelné prodlevy v práci aplikace závislé na daném databázovém systému.

Řešením tohoto problému je nastavení vyšší priority pro operace čtení, což však s sebou nese nebezpečí toho, že nebudou provedeny některé operace zápisu. Jinou možností je replikace, kdy uzel typu master je používán pouze k zápisu a uzly typu slave pouze ke čtení.

1.2.4 Škálování

Databázový systém MySQL pro potřeby škálování podporuje databázovou replikaci typu master-slave, kdy data z jednoho databázového serveru (master) jsou asynchronně replikována na jeden nebo více dalších databázových serverů (slave). Asynchronnost replikace zajišťuje odolnost vůči výpadkům spojení mezi servery[9].

Podle potřeby je možné replikovat všechny databáze, vybrané databáze nebo pouze vybrané tabulky z určité databáze.

Replikace umožňuje horizontální škálování (viz 4.3) databázového systému pro operace čtení, které mohou být prováděny na kterémkoliv z databázových serverů ve skupině (ať už master nebo slave). Operace čtení mohou být prováděny pouze na databázovém serveru master. Tato konfigurace může dramaticky zvýšit rychlost operací čtení a částečně i rychlost operací zápisů, pokud je databázový server master vyhrazen pouze pro operace zápisů. Tím pádem ale v aplikaci vzniká kritické místo "single point of failure" (viz 4.1.1), a v případě selhání databázového serveru master dojde k selhání celé aplikace.

2 NOSQL DATABÁZE

Jako NoSQL databáze se v dnešní době označuje široká skupina databázových systémů, které se mohou zásadně odlišovat od klasických relačních databázových systémů (relačních systémů řízení báze dat). Nejzásadnější rozdíl je v tom, že NoSQL databáze nevyužívají k dotazům jazyk SQL. Databázový systém také nemusí vyžadovat striktní, pevná schémata tabulek, obvykle nepodporují operace spojování výsledků (join), nemusí garantovat dodržení všech prvků ACID (atomicity, consistency, isolation, durability - atomičnost, konzistence, izolace, trvanlivost) a obvykle u nich dobře funguje horizontální škálování (distribuovaná úložiště)[1].

Mezi příklady nasazení NoSQL databází v reálných podmínkách můžeme zmínit například projekt ENSEMBLE Evropské komise, který pro modelování kvality vzduchu využívá cca 6 TB dat. Za zmínku také stojí cca 50 TB databáze Facebooku pro vyhledávání ve zprávách zaslaných mezi uživateli.

Původ NoSQL databází je úzce spojen s produkty významných internetových společností jako je Google, Amazon, Facebook nebo Twitter. Tyto společnosti u svých produktů pracují s tak velkými datovými objemy, u kterých již tradiční relační databázové systémy nedokáží držet krok, při současném požadavku práce v reálném čase. Výkon a povaha práce v reálném čase je v tomto případě důležitější než požadavky na konzistenci dat, na kterou tradiční relační databázové systémy vynakládají podstatně velké množství výpočetního výkonu. Důsledkem jsou pak NoSQL databáze vysoce optimalizované pro operace čtení a zápisu dat, ale kromě ukládání dat neposkytují téměř žádnou další funkčnost (typickým příkladem tohoto přístupu jsou úložiště typu klíč-hodnota - viz 3.1).

2.1 Rozdíly oproti relačním databázím

Typičtí zástupci dnešních relačních databázových systémů (například MySQL, PostgreSQL, Oracle, MSSQL) vykazují při specifických aplikacích s vysokým datovým objemem relativně slabý výkon. Jedná se například o indexování velkého množství dokumentů, obsluhu požadavků na vysoce navštěvovaných webových aplikacích nebo streamování multimedií.

Klasické relační databázové systémy jsou obvykle optimalizované buď pro malé, ale

velmi časté operace čtení/zápis nebo pro velké dávkové operace z nízkým počtem zápisů. Oproti tomu NoSQL databáze jsou schopny velmi dobře obsluhovat čtení i zápisy velké intenzity.

Relační databázové systémy poskytují oproti NoSQL databázím silnější záruky konzistence dat. Škálování relačních databázových systémů je možné, nicméně u NoSQL databázových systémů je to jedna z jejich základních vlastností, a proto je snazší tyto systémy dále škálovat.

Pro určité úlohy, v relačních databázových systémech snadno proveditelné díky jazyku SQL, je nutné u NoSQL databázových systémů využít jiného přístupu, například konceptu Map-Reduce. Typicky se jedná o dotazy s využitím agregace a agregačních funkcí, případně druhé restrikce.

2.2 CAP teorém

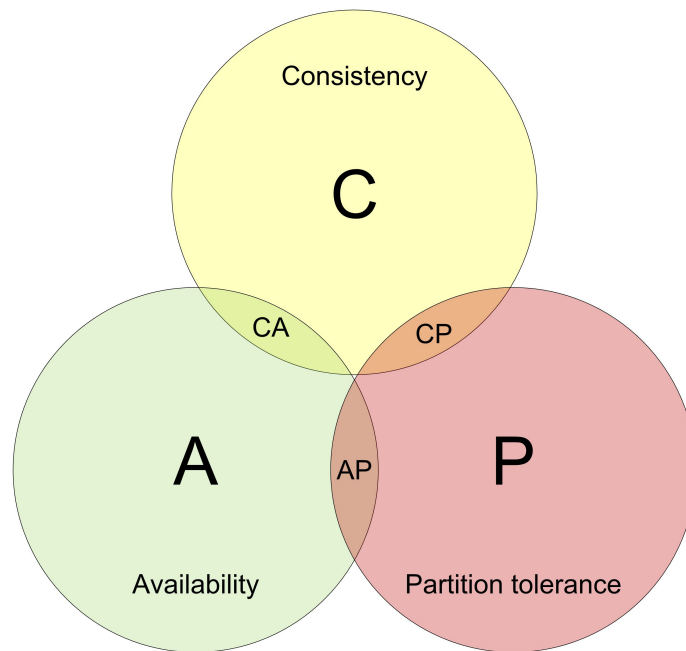
CAP teorém (známý taktéž jako Brewerův teorém) říká, že není u distribuovaném počítačovém systému možné dosáhnout situace, aby systém splňoval všechny tři z následujících vlastností:

- **Konzistence** (Consistency) - všechny uzly vidí ve stejný okamžik stejná data.
- **Dostupnost** (Availability) - záruka, že na každý dotaz k systému bude přijata odpověď, zda byl úspěšný nebo nikoliv.
- **Odolnost proti výpadku** (Partition tolerance) - systém zůstává provozuschopný i přes to, že došlo k nedoručení nějaké zprávy nebo došlo k výpadku části systému.

Podle CAP teorému je možné, aby distribuovaný systém splňoval dvě z těchto vlastností, ale nikdy ne všechny tři zároveň[10].

Tento koncept byl poprvé vysloven jako domněnka americkým informatikem Erikem Brewerem z UCLA v roce 2000. V roce 2002 byl pak S. Gilbertem a N. Lynchovou vytvořen formální důkaz této domněnky a tím vznikl zmiňovaný teorém. Jeho název je odvozen zkratkou prvních písmen trojice požadavků (Consistency, Availability, Partition tolerance).

Grafické znázornění vztahu požadavků CAP teorému je možné vidět na obrázku 3.



Obr. 3. CAP teorém

Tradiční relační databázové systémy splňují vlastnosti CP, tedy konzistenci a odolnost proti výpadku. Oproti tomu většina NoSQL databází, nebo obecně všechny distribuované systémy využívající model "eventual consistency" splňují vlastnosti AP, tedy dostupnost a odolnost proti výpadku.

2.3 Architektura NoSQL databází

Obvyklými prvky architektury NoSQL databází jsou relativně slabé záruky datové konzistence, typicky model označovaný jako "eventual consistency" (po uplynutí dostatečně dlouhé doby je možno optimisticky předpokládat, že veškeré změny byly propagovány do všech uzlů systému a tyto uzly jsou tak navzájem konzistentní) nebo transakce omezené na jednotlivé datové položky[1]. Některé NoSQL databáze ovšem za určitých okolností, obvykle za použití doplňkové mezivrstvy, dokáží zajistit kompletní konzistenci splňující ACID.

Dalším typickým znakem NoSQL databází je distribuovaná architektura, kdy jsou data redundantně ukládána napříč více servery, často za použití distribuované tabulky hashů (distributed hash table - DHT). To umožňuje databázový systém podle potřeby dále škálovat přidáním dalších serverů a také tolerovat selhání některého serveru.

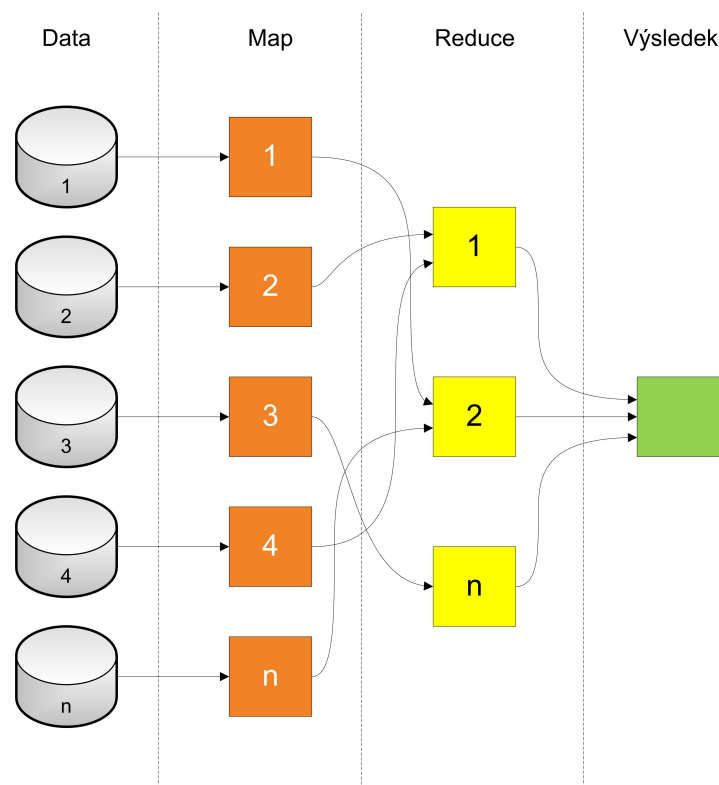
Časté je použití velmi jednoduchého rozhraní pro přístup k NoSQL databázi - například

pomocí asociativních polí nebo dvojic klíč-hodnota. Některé další systémy, jako jsou XML databáze, používají pro přístup standard XQuery.

2.4 MapReduce

MapReduce je softwarový framework vyvinutý firmou Google, který je využíván ke zjednodušení zpracování velkých objemů dat efektivní cestou. V hojně míře je užíván právě v prostředí NoSQL databází.

Základem MapReduce jsou funkce map a reduce, které se běžně využívají ve funkcionálním paradigmatu programování. Pro zpracování dat je specifikována mapovací funkce, která z dvojic klíč-hodnota vygeneruje přechodnou sadu dvojic klíč-hodnota, která je následně pomocí zadané redukční funkce zpracována tak, že všechny hodnoty se stejným klíčem jsou pomocí této funkce sjednoceny do jedné hodnoty[11].



Obr. 4. proces MapReduce

Jinými slovy je možné říci, že data jsou rozčleněna do několika podmnožin, které jsou pak jednotlivě zpracovány distribuovaně na více strojích a výsledná data jsou potom sloučena do jedné sady.

Graficky je proces MapReduce operace znázorněn na obrázku 4.

Aby mělo využití MapReduce smysl, je nutné, aby objem dat byl dostatečně velký pro to, aby se projevila výhoda vyššího výkonu v distribuovaném výpočtu. Výpočty nejsou obecně nijak závislé na externím vstupu, ale pouze na zadané datové sadě, která se má zpracovat. Výpočty prováděné na jednotlivých podmnožinách dat je nutné spojit do jedné sady. Výsledná data jsou obvykle menšího objemu než data vstupní.

MapReduce umožňuje v prostředí NoSQL databází suplovat neexistující klauzule JOIN a GROUP BY tak, jako jsou užívané v relačních databázových systémech.

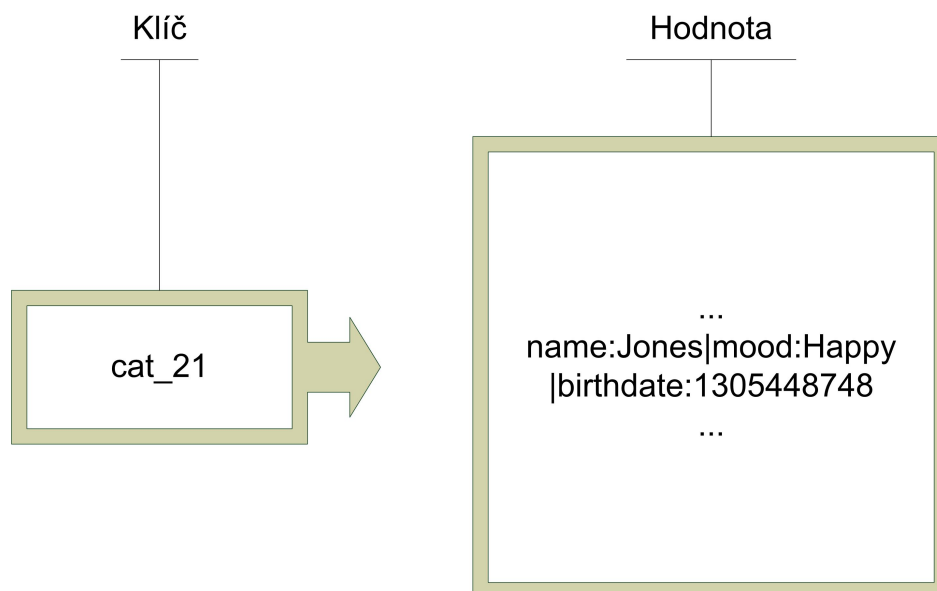
3 ROZDĚLENÍ NOSQL DATABÁZÍ

Obvykle jsou NoSQL databáze rozdělovány na základě způsobu, jakým jsou v nich data ukládána. Některé NoSQL databáze je možné zařadit do více kategorií, protože využívají více zde uvedených konceptů.

Tento výčet si neklade za cíl být kompletní. Protože množina NoSQL databázových systémů se dynamicky rozrůstá o nové inovativní, ne však příliš prověřené a rozšířené systémy, budeme se zabývat jen podmnožinou nejvýznamnějších druhů NoSQL databází.

3.1 Úložiště klíč-hodnota

Anglicky *key-value store*. Úložiště typu klíč-hodnota umožňují uchovávat data bez jakéhokoliv datového schématu. Tato data jsou obvykle reprezentována řetězcem, který představuje klíč a samotnými daty, která tvoří hodnotu ve dvojici klíč-hodnota. Data hodnoty jsou obvykle "primitivním" typem programovacího jazyka (řetězec, číselná hodnota, pole) nebo představují objekt serializovaný rozhraním mezi programovacím jazykem a úložištěm. Na obrázku 5 je možné vidět naznačené spojení klíče s hodnotou[1].



Obr. 5. Úložiště klíč-hodnota

Oproti relačním databázovým systémům je u databází typu klíč-hodnota podstatně vyšší výkon pro operace čtení a zápisu. Také výsledný kód aplikace je mnohem čistší a jednodušší oproti kódu protkanému dotazy v jazyce SQL. Ukládání dat v databá-

zích klíč-hodnota je také mnohem bližší objektově-orientovanému paradigmatu programování. V prostředí aplikací využívajících relačních databázových systémů je tento přístup emulován využitím knihoven pro objektově-relační mapování (například Hibernate, Active Record, Propel atd.), které ale mají za následek další zhoršení výkonu. To je důsledkem použití relativně složité mezivrstvy mezi objektově-orientovanou aplikací a samotným SQL dotazem do relačního databázového systému.

Rozhraní databáze typu klíč-hodnota může být ve své podstatě velmi jednoduché. Minimální sada operací pro základní funkčnost sestává pouze ze tří operací: čtení dat pod daným klíčem, zápisem dat pod daný klíč a vymazání daného klíče. Obvykle jsou však v jednotlivých implementacích zastoupeny další optimalizované operace - například inkrement číselné hodnoty jako jedna operace nebo práce s poli nebo kolekcemi v rámci hodnot.

Obvyklá je snadná možnost vertikálního škálování a replikace. Časté pak také velmi intenzivní využití operační paměti RAM k velice rychlému přístupu do indexu klíčů. Hodnoty jsou pak typicky, na základě určitého cachovacího heuristického algoritmu, také ponechány do určité kapacity v operační paměti RAM. Ostatní jsou pak perzistentně uchovány na pevném disku.

Databázové systémy typu klíč-hodnota je možné dále dělit do několika podskupin:

- Úložiště klíč-hodnota s modelem "eventual consistency" (například Riak)
- Hierarchická úložiště klíč-hodnota (například Caché)
- Úložiště klíč-hodnota v operační paměti RAM (například memcached, Redis)
- Úložiště klíč-hodnota s řazením (například MemcacheDB, Berkeley DB)

3.2 Dokumentová databáze

Anglicky *document-oriented database*. Základním konceptem dokumentové databáze je tzv. dokument, což jsou strukturovaná data zapouzdřená v některém ze standardizovaných formátů. Mezi používané formáty patří XML, YAML, JSON, BSON, ale také binární formáty jako PDF nebo dokumenty Microsoft Office[1].

Dokumenty v dokumentovém databázovém systému jsou do určité míry podobné záznamům (řádkům) v relačních databázích, nicméně se na ně neuplatňují tak striktní

pravidla. Dokumenty nemusí dodržovat pevné datové schéma ani nemusí zachovávat totožné části.

Následuje příklad dvou dokumentů ve formátu JSON:

```
{
  "firstname" : "Alice",
  "address" : "U vodárny 2, Brno",
  "phone" : 123456789
}
```

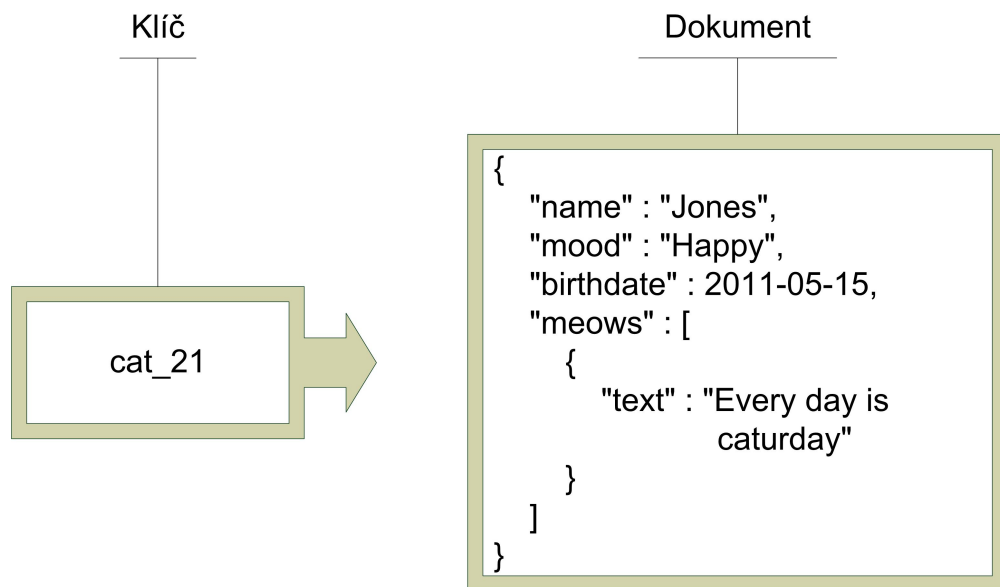
Druhý dokument může vypadat například následovně:

```
{
  "firstname" : "Bob",
  "phone" : 987654321,
  "children" : ["Carol", "Dave", "Eve"]
}
```

Na těchto příkladech je ilustrována relativní volnost ve struktuře dokumentu - oba dokumenty mají některé položky společné, jiné některé se vyskytují jen v jednom z dokumentů. Není zde zapotřebí "prázdných" položek tak jako by tomu bylo nutné v případě klasického relačního databázového systému.

Každý dokument v databázi má přiřazený unikátní klíč, který jej jednoznačně identifikuje. Obvykle se jedná o textový řetězec bez speciálního formátu, někdy pak URI nebo cesta. Na základě tohoto klíče je možné k dokumentu přistupovat. Klíče jsou pak obvykle databázovým systémem indexovány, aby byla zajištěna vysoká rychlost čtení. Tento princip je obdobný jako u úložišť typu klíč-hodnota. Na obrázku 6 je vidět naznačené spojení klíče a dokumentu.

Jinou možností přístupu k dokumentům, mimo přímý přístup pomocí klíče, je vyhledávání. Dokumentová databáze poskytuje rozhraní nebo dotazovací jazyk, který umožňuje číst dokumenty v závislosti na jejich obsahu. Je možné například vybrat všechny dokumenty, jejichž určitá položka obsahuje určitou hodnotu nebo množinu hodnot. Rozsah dotazovacího jazyka nebo rozhraní se mezi jednotlivými dokumentovými databázovými systémy podstatně liší.



Obr. 6. Dokumentová databáze

Dokumenty jsou obvykle dále různými způsoby organizovány, například pomocí kolekcí, tagů (značek), na základě metadat nebo adresářovou strukturou.

Mezi nejvýznamnější zástupce dokumentových databází patří například CouchDB, MongoDB, Redis.

3.3 Grafová databáze

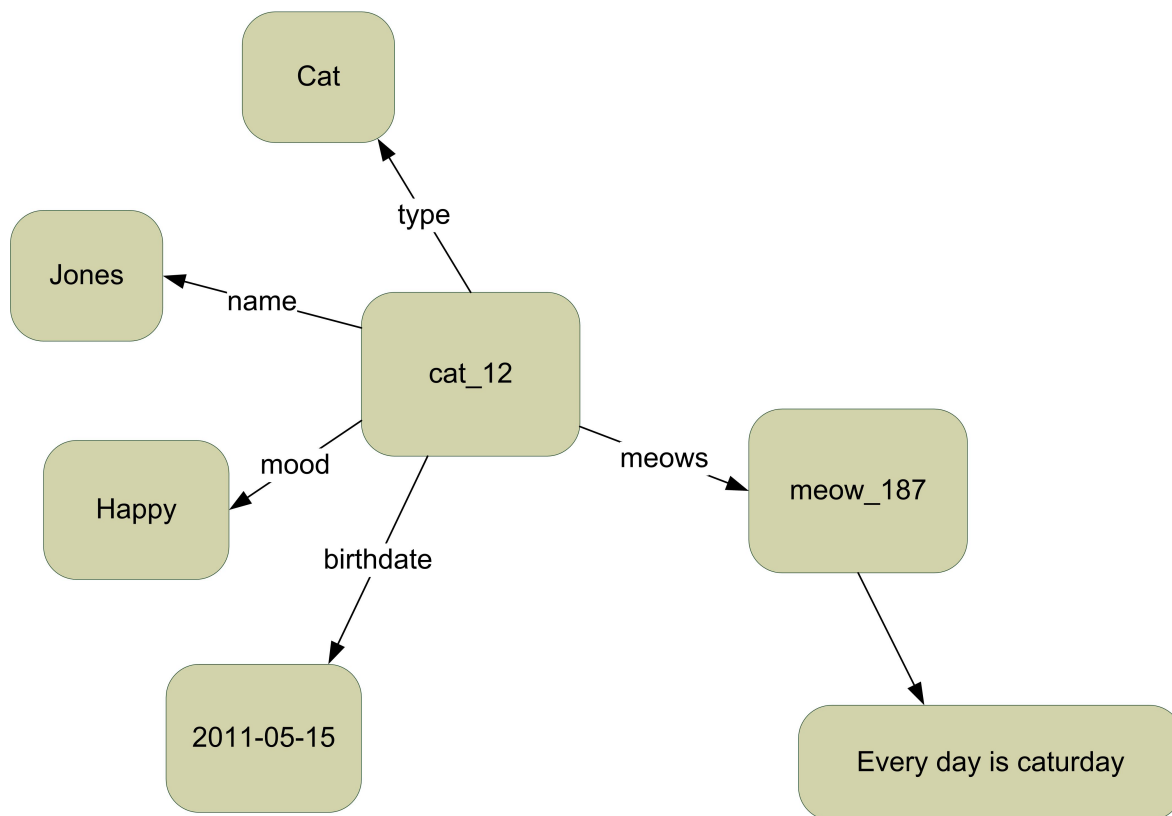
Anglicky *graph database*. Grafové databáze k reprezentaci a ukládání dat využívají grafové struktury obsahující uzly, hrany a vlastnosti. V podstatě je možné tvrdit, že jakýkoliv databázový systém, který umožňuje přístup k sousedním datům bez použití indexu je grafová databáze. To znamená, že každý prvek obsahuje přímý odkaz na vedlejší prvek a není zde žádná nutnost vyhledávání v indexu[1].

Obrázek 7 znázorňuje příklad struktury dat v grafové databázi.

Uzly (anglicky *nodes*) reprezentují entity, jako jsou například osoby, firmy, účty nebo jakékoliv další údaje, o kterých je třeba uchovávat informace. Jsou ve své podstatě velmi podobné objektům v objektově-orientovaném paradigmatu programování.

Vlastnosti (anglicky *properties*) jsou informace, které jsou určitým způsobem relevantní k uzlům.

Hrany jsou orientované spojnice, které propojují jednotlivé uzly navzájem, nebo uzly s vlastnostmi a reprezentují jejich vzájemné vztahy. Nejdůležitější informace jsou



Obr. 7. Grafová databáze

obvykle uloženy právě v hranách.

Ve srovnání s relačními databázovými systémy jsou grafové databáze často rychlejší pro práce s asociativními datovými sadami (poli) a snadněji je možné je mapovat na strukturu objektově-orientovaných aplikací. U grafových databází je přirozenější škálování při velkých objemech dat a typicky není potřeba náročných spojovacích (join) operací.

Vzhledem k tomu, že grafové databáze nejsou tolik závislé na striktním datovém schématu, jsou na rozdíl od relačních databází vhodnější pro správu ad-hoc dat nebo dat, která v průběhu času vyvíjí a mění své schéma. Naopak relační databáze typicky vykazují větší výkon při provádění stejné operace na velkém množství datových prvků.

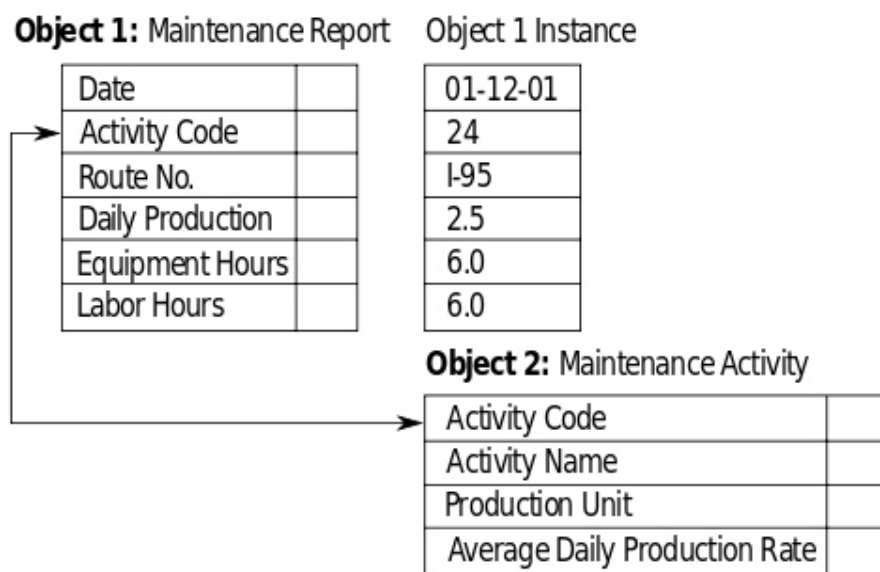
Grafové databáze jsou velmi vhodným nástrojem pro operace a dotazy vycházející z teorie grafů, jako je například nalezení nejkratší cesty mezi dvěma uzly grafu.

Mezi rozšířené grafové databáze patří mimo jiné projekty Neo4j, OrientDB nebo Horton.

3.4 Objektová databáze

Anglicky *object database* nebo *object-oriented database*. Objektová databáze je databázový systém, ve kterém jsou data reprezentována ve formě objektů, tak jako jsou používány v objektově-orientovaném paradigmatu programování[1].

Vzhledem k velmi těsné integraci objektového databázového systému s programovacím jazykem, je možné pro programátora udržet konzistenci v rámci jednoho prostředí - jak databázový systém tak i programovací jazyk může používat stejnou reprezentaci datového modelu. Naopak relační databázové systémy vytyčují velmi jasný předěl mezi databázovým modelem a aplikací, který je za cenu zvýšených výpočetních nároků možné alespoň částečně překlenout použitím knihovny pro objektově-relační mapování. Na obrázku 8 je možné vidět příklad datového modelu v objektové databázi.



Obr. 8. Objektová databáze

Většina objektových databází poskytuje určitý druh dotazovacího jazyka, který umožňuje nacházení objektů pomocí principů deklarativního programování. V této oblasti jsou nejpatrnější rozdíly mezi jednotlivými implementaci objektových databázových systémů.

Přístup k datům může být rychlejší než u relačních databázových systémů, jelikož se nevyužívají náročné spojovací (join) operace. Objekty mohou být získávány přímou cestou bez vyhledávání pomocí ukazatelů. Objekty taktéž nevyžadují serializaci a deserializaci při ukládání respektive načítání z databáze, což ušetří jak výpočetní čas, tak

i čas k naprogramování. Datový model objektové databáze je také snáze uchopitelný, neboť se zakládá na fungování skutečného světa.

Rozdíly mezi jednotlivými implementacemi objektových databází jsou obvykle ve způsobu definice databázového schématu, nicméně obecně lze říci, že datové typy v databázovém schématu odpovídají datovým typům v programovacím jazyce.

Objektové databáze jsou vhodné pro multimediální aplikace, protože metody třídy asociované s daty jsou zodpovědné za jejich správnou interpretaci. Častá je také přítomnost verzování objektů - na objekt je možné nahlížet jako na sadu všech jeho verzí, ale také je možné každou verzi objektu brát jako samostatný objekt. Objektové databáze jsou velmi efektivní v aplikacích, ve kterých je vyžadováno velmi rozsáhlé množství dat o jednotlivé položce.

Tak jako objektově-orientované paradigma programování, i objektové databáze splňují určité předpoklady - zapouzdření, dědičnost, polymorfismus.

3.5 Sloupcové úložiště

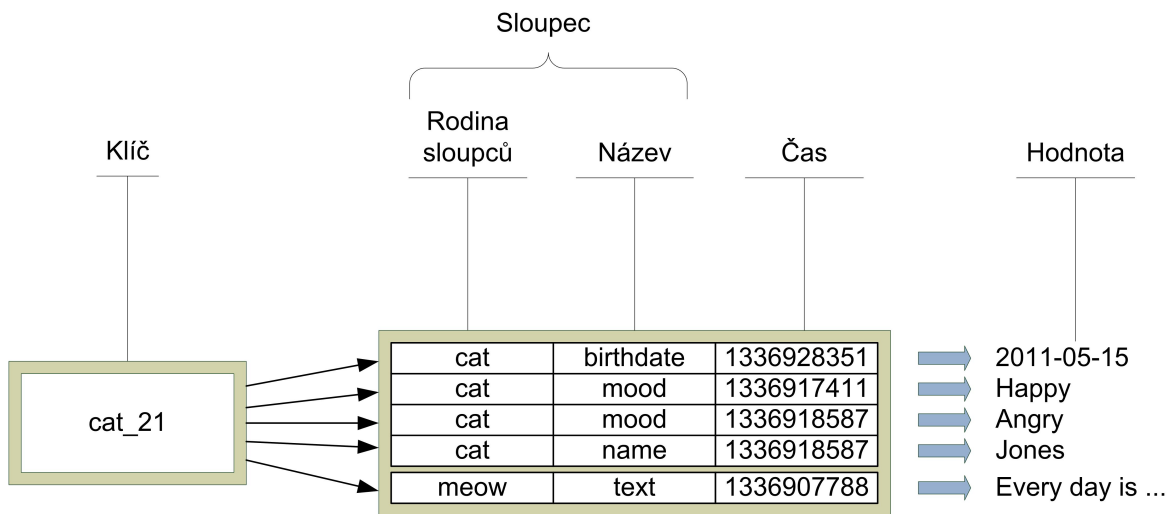
Anglicky *wide column store* nebo také *tabular*. Sloupcové úložiště je řídké distribuované perzistentní multidimenzionální seřazené asociativní pole (mapa). Oproti relačním databázovým systémům jsou sloupcová úložiště vhodnější na velmi vysoké objemy dat (řádově miliony řádků a miliony sloupců)[1].

Sloupcová úložiště jsou obvykle postavena tak, aby využívala distribuované souborové systémy, což umožňuje efektivní rozložení úložiště mezi množství nezávislých serverů. Například úložiště HBase využívá distribuovaný souborový systém HDFS (Hadoop Distributed File System), úložiště BigTable pak využívá distribuovaný souborový systém GFS (Google File System).

Data ve sloupcových úložištích připomínají zdánlivě tabulky, tak jako jsou využívány v relačních databázových systémech. Jedná se ale o multidimenzionální řídké asociativní pole, tedy není nutné, aby všechny sloupce byly obsaženy u všech záznamů v sloupcovém úložišti. Samotné hodnoty sloupců jsou tvořeny polem bytů, tedy nemají pevně definován datový typ.

Oproti jiným databázovým systémům, jsou data ve sloupcových úložištích, ukládána seřazená podle klíčů. To je značná výhoda vzhledem k distribuované podstatě těchto

databázových systémů, jelikož obvykle jsou příbuzné klíče uloženy na stejném fyzickém serveru.



Obr. 9. Sloupcové úložiště

Sloupce je možné sdružovat do rodin sloupců (column family), jejichž struktura je definovaná předem a je později neměnná. Rodiny sloupců ovšem opět nemusí obsahovat u všech záznamů stejné sloupce. Výhodou rodin sloupců je to, že mají více či méně pevnou strukturu, u které je možné bez kompletního načtení všech záznamů zjistit všechny vyskytující se sloupce. To není možné u samotných sloupců.

Všechny sloupce mají ještě jeden rozměr, a tím je verze, reprezentovaná časovým razítkem (celočíslné hodnota počtu vteřin od počátku epochy). Pro jednotlivé rodiny sloupců je možné stanovit pravidla, kolik verzí daného záznamu je třeba uchovávat. Bez specifikovaného údaje o čase sloupcové úložiště vrací nejaktuálnější verzi záznamu. Pokud je časový údaj při dotazu zadán, pak je vrácena verze s časem rovným nebo nižším než je čas zadaný.

Na obrázku 9 je naznačena možná struktura dat ve sloupcovém úložišti.

Mezi nejrozšířenější implementace sloupcových úložišť patří systémy BigTable a HBase.

4 DŮLEŽITÉ VLASTNOSTI A PARAMETRY NOSQL DATABÁZÍ

Aby bylo možné pro existující relační databázové systémy, které narážejí v určitých situacích na své datově-objemové limity, efektivně nahradit NoSQL databázovým systémem, musíme si nejprve představit důležité požadavky, které takový databázový systém musí splňovat, aby byl nejen plnohodnotnou náhradou, ale především aby přinesl zvýšení výkonu oproti původnímu řešení.

4.1 Vysoká dostupnost

Anglicky *High Availability*. Aby webová aplikace využívající databázový systém byla pro uživatele kvalitně použitelná, musí mít při všech základních operacích velmi nízké časy odezvy a také je nutné eliminovat jakkoliv dlouhé výpadky. Proto musí využívaný databázový systém splňovat kritérium vysoké dostupnosti.

Dostupnost se obvykle měří v procentech celkové dostupnosti aplikace bez výpadku v průběhu jednoho roku. V tabulce 1 jsou znázorněny vybrané hodnoty dostupnosti s přepočtem celkového času výpadku v průběhu jednoho roku, jednoho měsíce a jednoho týdne[17].

Tab. 1. Přepočet poměru dostupnosti na čas výpadku

Dostupnost	Výpadek ročně	Výpadek měsíčně	Výpadek týdně
90%	36,5 dní	72 hodin	16,8 hodin
95%	18,25 dní	36 hodin	8,4 hodin
97%	10,96 dní	21,6 hodin	5,04 hodin
98%	7,30 dní	14,4 hodin	3,36 hodin
99%	3,65 dní	7,20 hodin	1,68 hodin
99,5%	1,83 dní	3,60 hodin	50,4 hodin
99,8%	17,52 hodin	86,23 minut	20,16 minut
99,9%	8,76 hodin	43,2 minut	10,1 minut
99,95%	4,38 hodin	21,56 minut	5,04 minut
99,99%	52,56 minut	4,32 minut	1,01 minut
99,999%	5,26 minut	25,9 vteřin	6,05 vteřin
99,9999%	31,5 vteřin	2,59 vteřin	0,605 vteřin

Nejčastější problémy a úzká hrdla, která ovlivňují dostupnost aplikace využívající databázový systém jsou obvykle zámky v databázi, je především řízení souběžného přístupu

k datům, problémy vznikající replikací dat a vstupně-výstupní limity operační paměti RAM a diskových úložišť.

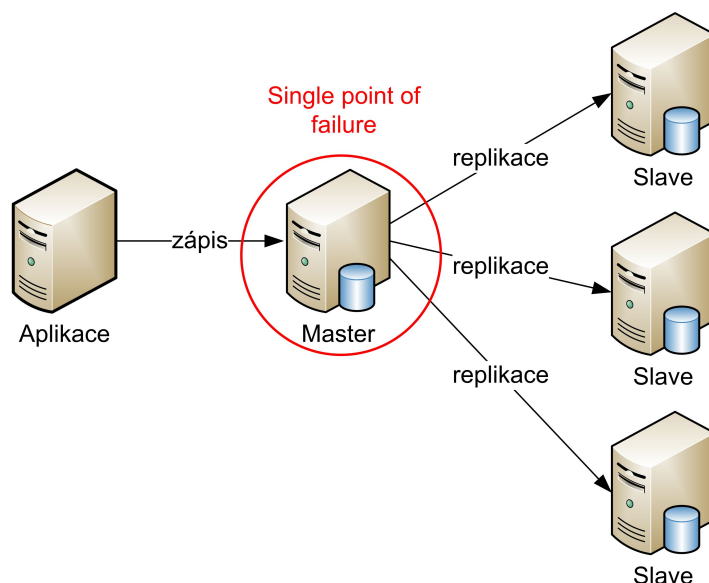
Vstupně-výstupní limity operační paměti RAM a diskových úložišť je možné řešit několika způsoby. Buď vhodným nahrazením nebo doplněním nedostačujících hardwarových komponent serverů (přidání dalšího disku do RAID pole, zvýšení kapacity operační paměti RAM, výměna celého serveru za výkonnější a podobně), případně pak škálováním replikací nebo rozdělením dat na více fyzických serverů.

Horizontální škálování databázového systému však s sebou nese některé další potenciální problémy, obzvláště, pokud databázový systém obsahuje nějaké kritické místo "single point of failure" (viz 4.1.1).

Problémy se řízením souběžného přístupu k datům vznikají především, pokud je souběžný přístup zajišťován pomocí zamykání tabulek nebo řádků. Zde mohou nastat potenciální prodlevy při čekání na uvolnění zámku. Částečným řešením je škálování databázového systému replikací. Jiným řešením je potom použití odlišného přístupu k řízení souběžného přístupu jako je například pomocí verzování (viz 4.1.2).

4.1.1 Single Point of Failure

Single Point of Failure (SPOF) - kritický bod selhání - je nějaká součást systému, jejíž výpadek má za následek přerušení fungování celého systému. Kritické body selhání jsou nežádoucí v jakémkoliv systému, jehož cílem je vysoká dostupnost a spolehlivost[20].



Obr. 10. Příklad Single Point of Failure

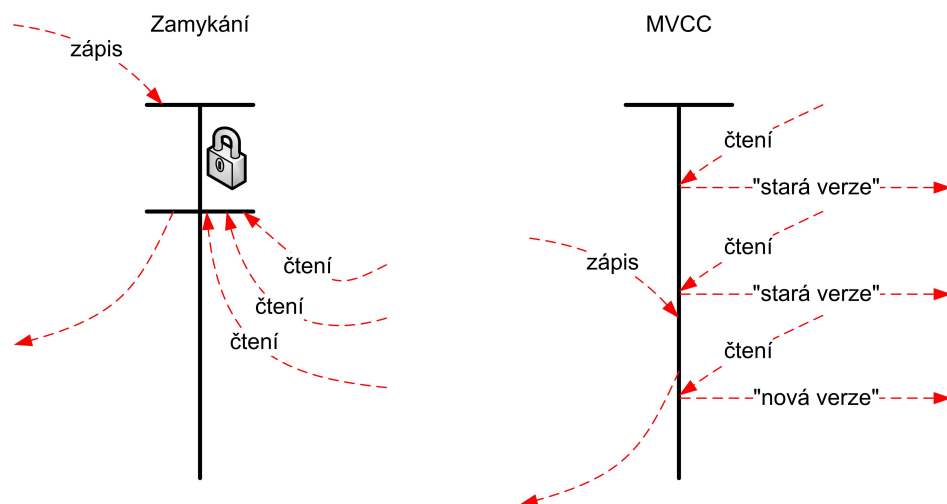
System může dosáhnout robustnosti zvýšením redundance všech potenciálních kritických míst selhání. Redundance může být dosaženo na úrovni vnitřních komponent systému, na úrovni systému (více serverů) nebo replikací celého systému.

Na obrázku 10 je možné vidět příklad potenciálního místa "single point of failure" v prostředí replikace dat relačního databázového systému. Jeden databázový server je v tomto případě master a jako jediný umožňuje zápis dat. Na všechny ostatní slave servery jsou potom tato data replikována. V případě kritického selhání master serveru nemůže aplikace zapisovat data a tím pádem nesplňuje svůj účel.

4.1.2 Multi-Version Concurrency Control

Multi-Version Concurrency Control (MVCC) - řízení souběžného přístupu pomocí verzí - data v databázovém systému se při operaci zápisu nemění in situ, ale minimálně po dobu než je operace dokončena zůstává předchozí hodnota zachována a nová hodnota je vytvořena jako další verze. To umožňuje provádět změny dat bez přítomnosti zamykání databáze a/nebo jejich částí pro čtení a tedy i tím větší dostupnost.

Rozdíl mezi řízením souběžného přístupu pomocí zámků a pomocí verzí je možné vidět na obrázku 11.



Obr. 11. Rozíl mezi řízením souběžného přístupu pomocí zámků a verzí

Verze bývají typicky označovány časovým údajem, časový údaj je zároveň poslán s požadavky na provedení operace, proto je možné určit, který stav databázového systému se při odpovědi na požadavek má brát v potaz.

To, že jsou jednotlivé verze dat po nějakou zachovávány, umožňuje efektivní využití

paměťového nebo diskového prostoru, protože není nutné řešit fragmentaci dat. Obvykle databázový systém, využívající řízení pomocí verzí, obsahuje mechanismus, který pravidelně odstraňuje již nepotřebné staré verze, aby uvolnil jinak zbytečně obsazený prostor.

Zřejmou nevýhodou řízení souběžného přístupu pomocí verzí je nutnost uchovávat více verzí objektů databázového systému na diskovém nebo paměťovém úložišti. Naopak největší výhodou je nepřítomnost zámků pro čtení, což může být velmi důležité pro aplikace s velkým objemem operací čtení.

4.2 Vysoký výkon

Anglicky *High Performance*. Logickým požadavkem na databázový systém s vysokým výkonem jsou vysoce optimalizované operace čtení a zápisu. Důležité je již při výběru vhodného databázového systému uvažovat, v jakém poměru budou operace čtení a operace zápisu zastoupeny a na základě toho volit databázový systém optimalizovaný právě pro tuto situaci.

Nejvyššího výkonu dosahují typicky databázové systémy, které využívají co nejefektivněji práci s operační pamětí RAM. Zde je ovšem nutné počítat s potenciálními omezeními vyplývajícími s požadavkem na perzistenci dat. V případě, že předem víme nebo můžeme dostatečně spolehlivě odhadnout, jaké datové objemy bude databázový systém uchovávat a zpracovávat, pak pokud se tento datový objem vejde do dostupné operační paměti RAM, je velmi vhodné o takovémto řešení uvažovat.

4.3 Škálovatelnost

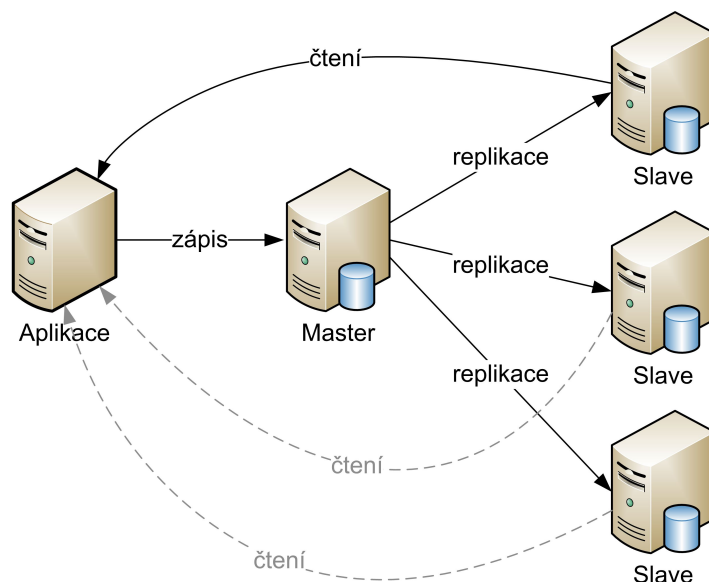
Anglicky *Scalability*. Aby bylo možné vyhovět vzrůstajícím nárokům na vysokou dostupnost a vysoký výkon databázového systému, je obvykle nutné postupovat cestou horizontálního škálování. To narušuje od vertikálního škálování, což znamená zvyšování výkonu jednoho uzlu systému, jde cestou přidávání dalších uzlů do systému, mezi které je poté zátěž určitým způsobem rozložena.

Pro zachování vysoké dostupnosti databázového systému je podstatné, aby přidání nového uzlu do systému žádným způsobem nenarušilo chod systému. To samé platí i pro případ výpadku některého z uzlů.

Existuje několik metod horizontálního škálování databázového systému, které je navíc možné v určitých případech kombinovat především za účelem eliminace potenciálních kritických míst a zvýšení spolehlivosti celého systému. Ty nejrozšířenější metody jsou popsány níže:

4.3.1 Replikace master-slave

V distribuovaném databázovém systému se nachází jeden uzel master, který má na starosti operace zápisu, které potom následně replikuje na jeden nebo více dalších uzlů slave, které jsou dostupné pouze pro operace čtení. Z principu zde vzniká kritické místo "single point of failure" v podobě master uzlu. V případě jeho výpadku není možné provádět žádné zápisy.



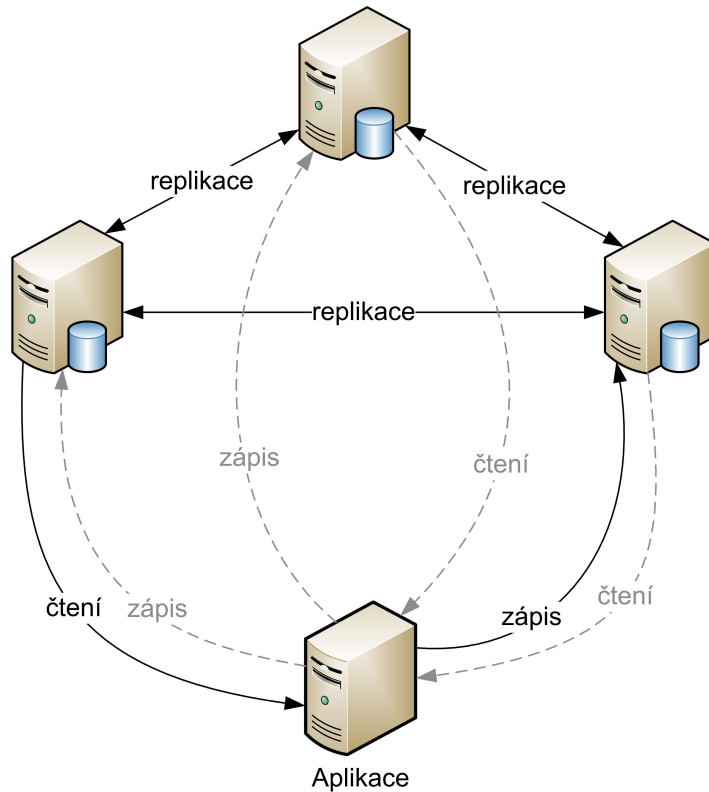
Obr. 12. Replikace master-slave

Tento nedostatek je možné eliminovat několika způsoby. Jednou možností je v případě výpadku automatické "povýšení" jednoho ze slave uzlů na nový master uzel. Další možností je potom víceúrovňová replikace, kdy některé slave uzly jsou master uzly pro další slave uzly. V případě výpadku jednoho z master uzlů dojde tedy k omezení činnosti pouze části celého systému.

Na obrázku 12 je naznačena struktura databázového systému využívající replikaci typu master-slave.

4.3.2 Replikace master-master

V distribuovaném databázovém systému jsou všechny uzly rovnocenné, na každém z nich je možné vykonávat jak operace čtení tak i operace zápisu a změny provedené v jednom uzlu jsou replikovány na všechny ostatní uzly.



Obr. 13. Replikace master-master

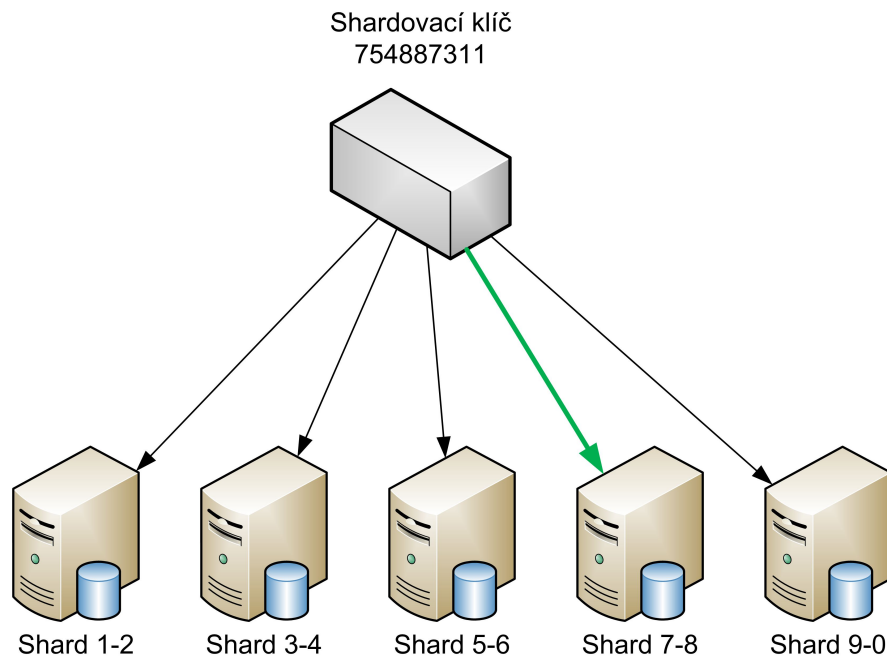
Databázový systém musí mít možnost řešení konfliktů v datech, které mohou nastat, pokud na dvou uzlech systému během jednoho okamžiku dojde ke změně stejných dat. Obvykle není možné dosáhnout plné shody s ACID (viz 1.1), především pak v požadavku na konzistenci dat.

Velkou výhodou tohoto přístupu je, že uzly mohou být rozmístěny v geograficky odlišných lokacích a obsluhovat operace zápisu v blízkosti přistupujícího klienta.

4.3.3 Sharding

Sharding je metoda horizontálního rozložení dat v databázovém systému mezi více logických nebo fyzických serverů. Každé části dat se potom říká *shard*.

Narozdíl od horizontálního rozdělení databáze, je každý shard samostatnou jednotkou, která může fungovat nezávisle na ostatních uzlech systému.



Obr. 14. Příklad shardingu

Data jsou rozdělována pomocí takzvaného shardovacího klíče, což může být u relačních databázových systémů například primární klíč, ale je možné použít téměř jakýkoliv jiný klíč. Na základě shardovacího klíče je možné rozhodnout, ve kterém shardu se hledaná data budou nacházet a není tedy nutné provádět dotaz na všech uzlech systému. Data mohou být do shardů rozdělena buď podle rozsahů hodnot shardovacího klíče, nebo pro zajištění větší rovnoměrnosti pak pomocí hashovací funkce shardovacího klíče. Další možností je potom rozdělení dat do shardů s přihlédnutím ke geografické lokaci datacentra, kde jsou uloženy[18].

Na obrázku 14 je zobrazen příklad možného shardování databáze na základě celočíselného klíče - shardování zde probíhá na základě první číslice klíče.

4.4 Flexibilní datová schémata

Možnost použití datového modelu, který nemá pevné datové schéma (ať už je schéma snadno modifikovatelné nebo není přítomno vůbec), je nespornou výhodou v případě, že aplikace využívající databázový systém počítá v budoucnu se změnami ve schématu, nebo pokud uchovávaná data nemají pevně definovanou strukturu[21].

V prostředí vysokých datových objemů je v takovém případě rigidní datové schéma tak, jak jej známe z relačních databázových systémů, spíše překážkou, a to z důvodu

časové náročnosti prováděných změn (změny, přidávání nebo odebrání sloupců, vytváření nebo rušení indexů), která ve většině případů znamená dočasné přerušení činnosti systému, popírající požadavek vysoké dostupnosti.

Pokud neznáme dopředu strukturu dat, která budeme v databázovém systému uchovávat, pak v případě pevně definovaného schématu musíme pro každý záznam uchovávat prázdné hodnoty sloupců a pokud v budoucnosti bude struktura dat rozšířena o další atribut, musíme pro něj vytvořit příslušný sloupec.

5 NEJROZŠÍŘENĚJŠÍ NOSQL DATABÁZE

Pro volbu nejvhodnějšího řešení je nutné představit si nejznámější zástupce NoSQL databází aktuálně dostupné. Vzhledem k udržení nízkých nákladů na celý systém nás zajímají především databázové systémy uvolněné zdarma nebo ideálně pod open source licencemi.

Všechny zde představené a popsané databázové systémy jsou již delší dobu vyvíjeny a jsou prověřeny aplikací v reálných projektech, které plně využívají jejich výhody práce s velkými datovými objemy.

Ke každému databázovému systému zde uvádíme popis jeho vlastností, stručnou historii a výčet výhod oproti relačním databázovým systémům v konkrétních situacích.

Z technického hlediska budeme předpokládat provoz na serverech s operačním systémem GNU/Linux.

5.1 Cassandra

Apache Cassandra (<http://cassandra.apache.org>) je distribuovaný databázový systém typu sloupcové úložiště. Byl vyvinut pro ukládání a zpracování velmi rozsáhlých datových objemů rozložených na mnoha běžných serverech. Poskytuje vysokou dostupnost bez přítomnosti kritických míst single point of failure[2].



Obr. 15. Logo projektu Apache Cassandra

Cassandra byla vyvinuta v roce 2008 ve firmě Facebook jako databázový systém pro Inbox Search ve Facebooku. Později byl celý projekt publikován jako open source pod licencí Apache Licence 2 a správu nad ním převzala Apache Foundation.

Cassandra je sloupcové úložiště s nastavitelnou mírou konzistence dat. Klíče jsou mapovány na více hodnot, které je možné sdružovat do rodin sloupců. Tyto rodiny sloupců jsou neměnné a jsou nastaveny při založení databáze, nicméně je možné v budoucnu další sloupce do rodin sloupců přidávat. Sloupce se navíc mohou přidávat pouze vybraným klíčem, proto je možné mít rozdílné počty sloupců v rámci jedné rodiny sloupců. Hodnoty jedné rodiny sloupců jsou ukládány do úložiště pohromadě, což Cassandra

dělá hybridním úložištěm na pomezí sloupcového databázového systému a řádkového databázového systému.

Nejzásadnější vlastnosti databázového systému Cassandra jsou:

- **Decentralizace** - každý uzel v systému má stejnou roli, není zde žádné kritické místo single point of failure. Data jsou distribuovaně uložena v rámci clusteru (takže každý uzel obsahuje jiná data), nicméně zde není žádný server v roli master, takže každý uzel může obsluhovat stejné požadavky.
- **Replikace** - úroveň replikace dat je nastavitelná. Cassandra byla navržena jako distribuovaný systém pro nasazení na velkém množství uzlů v rámci více data-center.
- **Elastičnost** - výkon operací čtení i zápisu roste lineárně s tím, jak jsou do systému přidávány další uzly, bez toho aby bylo nutné běh systému přerušovat.
- **Odolnost vůči chybám** - data jsou automaticky replikována na více uzlů, aby v případě selhání uzlu nebyla přerušena činnost systému. Uzly, které selhaly je možné nahradit bez výpadku systému.
- **Nastavitelná konzistence** - jak operace zápisu tak čtení umožňují nastavitelnou úroveň konzistence mezi uzly. Je možné nastavit úroveň konzistence v celém rozsahu od situace kdy žádný zápis nesmí selhat až po situaci kdy čtení má vždy přednost a žádný uzel nesmí být zápisem blokován.
- **MapReduce** - využívá MapReduce implementaci systému Hadoop.

Databázový systém Cassandra je implementován v programovacím jazyce Java a pro přístup k datům využívá vlastní binární protokol. Pro mnoho dalších programovacích jazyků včetně PHP existují rozhraní umožňující přístup k datům v databázovém systému.

Operace zápisu jsou v prostředí databázového systému Cassandra rychlejší než operace čtení. To jej činí vhodným pro aplikace kde jsou zápisy důležitější než čtení - například logování nebo real time analýza dat.

Databázový systém Cassandra je v praxi využíván například v projektech Facebook, Twitter, Digg a dalších.

5.2 MongoDB

MongoDB (<http://www.mongodb.org>) je dokumentově orientovaný databázový systém, který data ukládá ve strukturovaném formátu BSON blízkému standardizovanému formátu JSON, bez nutnosti definovat datové schéma. Umožňuje na datech spouštět na straně databázového systému vlastní kód v JavaScriptu[6].



Obr. 16. Logo projektu MongoDB

Vývoj databázového systému MongoDB začal v roce 2007 firmou 10gen jako součást vlastní platformy pro hosting internetových aplikací. V roce 2009 byl samostatný projekt MongoDB zpřístupněn pod open source licencí GNU AGPL v3.0. Od března 2011 je potom s vydáním verze 1.4 považován za vhodný pro běh v produkčním prostředí.) Nejzásadnější vlastnosti databázového systému jsou:

- **Ad hoc dotazování** - je podporováno vyhledávání v dokumentech v databázi na základě polí, rozsahů hodnot a regulárních výrazů. Dotazy mohou vracet vybraná pole dokumentu a také mohou obsahovat vlastní JavaScriptové funkce.
- **Indexování** - jakékoliv pole v dokumentu v databázovém systému může být indexováno. Indexy jsou koncepčně velmi blízké těm z relačních databázových systémů.
- **Replikace** - je vyřešena způsobem master-slave. Uzel master může provádět operace čtení i zápisu, slave pak může být použit jen pouze pro čtení nebo zálohy. Slave uzly mají v případě selhání master uzlu možnost povýšit jeden z uzlů na nový master[7].
- **Rozložení zátěže** - horizontální škálování databázového systému je prováděno pomocí shardingu.
- **Souborové úložiště** - databázový systém může být nativně využit i jako distribuované úložiště souborů, využívající všechny výhody replikace a rozložení zátěže

mezi více serverů.

- **MapReduce** - je možné využít konceptu MapReduce pro získání agregovaných pohledů na data.
- **JavaScript na straně serveru** - v dotazech do databázového systému a v agregačních funkcích je možné využít JavaScript, který se provede přímo nad daty na straně serveru.
- **Kolekce pevné velikosti** - jsou dostupné kolekce, které mají pevně stanovenou velikost. Je zachováno pořadí vkládání a pokud je dosaženo limitu velikosti, chová se takováto kolekce jako kruhová fronta.

System MongoDB je implementován v programovacím jazyce C++ a pro přístup k datům se využívá vlastní binární protokol. Pro mnoho programovacích jazyků včetně PHP existují připravená rozhraní pro přístup k databázovému systému. Výhodou může být, že si databázový systém MongoDB ponechává některé vlastnosti relačních databázových systémů (indexování, dotazy), což může usnadnit přechod mezi oběma paradigmaty.

Celý projekt zastává filosofii preferování výkonu před množstvím funkcionality, proto je rozhraní relativně strohé, nicméně databázový systém poskytuje vynikající výkon. Data jsou ukládána na pevném úložišti, nicméně pokud je to možné pak se co největší množství dat drží současně v operační paměti RAM.

Databázový systém MongoDB je v praxi využíván například projekty Craigslist nebo Foursquare.

5.3 CouchDB

Apache CouchDB (<http://couchdb.apache.org>) je dokumentově orientovaný databázový systém, jehož hlavním záměrem je velmi jednoduché použití v prostředí webu. Data jsou v databázovém systému uložena ve standardizovaném formátu JSON, přístupná jsou pomocí unikátního klíče a pro dotazování je využíváno jazyka JavaScript ve spojení s MapReduce funkcemi. Jako přístupové rozhraní je využit standardizovaný protokol HTTP[3].



Obr. 17. Logo projektu CouchDB

Vývoj systému CouchDB začal D. Katz v roce 2005 a již od počátku byl zamýšlen jako databázový systém pro webové aplikace. V roce 2007 byl uvolněn pod open source licenci GNU GPL. V roce 2008 byl převeden pod Apache Foundation a licence byla změněna na Apache Licence 2. První stabilní verze byla potom vydána v roce 2010.

Nejzásadnější vlastnosti databázového systému jsou:

- **ACID** - díky využití Multi-Version Concurrency Control (řízení souběžného přístupu pomocí verzí) splňuje CouchDB ACID požadavky.
- **Pohledy a indexy** - uložená data jsou strukturována za pomoci pohledů (view). Každý pohled je vytvořen JavaScriptovou funkcí, která funguje jako funkce map v MapReduce. Funkce bere jako vstup dokument a transformuje jej na jednu výslednou hodnotu. CouchDB může pohledy indexovat a tyto indexy udržovat aktuální vůči změnám v datech.
- **Replikace** - umožňuje obousměrnou replikaci (synchronizaci) s přihlédnutím k možnosti práce offline. Více replik může mít svou vlastní upravenou kopii stejných dat a později tyto změny synchronizovat.
- **REST API** - každý dokument má své unikátní URI, ke kterému je možné přistupovat pomocí protokolu HTTP. REST rozhraní využívá HTTP metody POST, GET, PUT a DELETE pro zpřístupnění základních CRUD (vytvoření, čtení, úprava, smazání) operací na dokumentech.
- **Eventual consistency** - pro zajištění konzistence dat je využito přístupu "eventual consistency", což umožňuje zajistit vysokou dostupnost a zároveň i odolnost vůči výpadkům.
- **Offline režim** - databázový systém je možné replikovat do zařízení (například

mobilního telefonu), které mohou přejít do režimu offline a poté co jsou opět online se data sesynchronizují do aktuálního stavu.

Databázový systém CouchDB je implementován v programovacím jazyce Erlang. Vzhledem k REST rozhraní není nutné používat žádnou nadstavbovou knihovnu pro přístup k tomuto konkrétnímu databázovému systému, nicméně existují rozhraní pro velké množství programovacích jazyků, které práci s CouchDB lépe integrují do prostředí daného programovacího jazyka.

Replikační a synchronizační schopnosti databázového systému CouchDB z něj činí vhodný nástroj pro použití v mobilních zařízeních, kde není zaručena nepřetržitá síťová konektivita, ale aplikace musí bez přerušení pracovat i v offline režimu[4].

Vhodné je použití v aplikacích, jejichž data se především akumulují a ne příliš často se mění, je na nich potřeba spouštět předdefinované dotazy a kde je důležité verzování dat. Velmi přínosná je replikace typu master-master, což umožňuje velmi snadné nasazení na více místech.

Databázový systém CouchDB je v praxi využíván například na projektech Meebo nebo v televizní společnosti BBC pro jejich platformu dynamického obsahu.

5.4 Redis

Redis (<http://www.redis.io>) je úložiště typu klíč-hodnota, kdy celý obsah úložiště je uložen v operační paměti RAM s volitelnou možností trvanlivosti dat zápisem na pevný disk. Data, uložená a přístupná pod daným unikátním klíčem, mohou být na rozdíl od jiných implementací úložišť typu klíč-hodnota nejen řetězce, ale jsou podporovány i některé další abstraktní datové typy[12].



Obr. 18. Logo projektu Redis

Databázový systém Redis začal být vyvíjen italským vývojářem S. Sanfilippo v roce 2009 a byl uvolněn pod open source licenci BSD. V roce 2010 do projektu finanční podporou vstoupila firma VMWare.

Nejzásadnější vlastnosti databázového systému jsou:

- **Abstraktní datové typy** - databázový systém Redis umožňuje ukládat data krom řetězce také v několika dalších abstraktních datových typech. Ty jsou:
 - seznam řetězců
 - množina řetězců (kolekce unikátních neseřazených elementů)
 - uspořádaná množina řetězců (kolekce unikátních elementů řazených pomocí skóre, což je číslo s plovoucí řádovou čárkou)
 - hash - asociativní pole (kde klíče i hodnoty jsou řetězce)

Na těchto datových typech umožňuje databázový systém provádět vysoce optimalizované atomické operace (například sjednocení množin, rozdíl množin, průnik množin nebo seřazení seznamů).

- **Expirace klíčů** - jednotlivým klíčům je možné nastavit čas expirace, po kterém je databázový systém vyřadí.
- **Nastavitelná perzistence** - databázový systém Redis typicky celý datový obsah udržuje v operační paměti RAM a je možné vybrat ze dvou způsobů, jak data ukládat na diskové úložiště. První možností je nechat databázový systém v určitých intervalech vytvářet obrazy aktuálních dat. Je nicméně možné, že dojde ke ztrátě dat, pokud dojde k selhání serveru před tím, než je obraz zapsán. Druhou a bezpečnější alternativou je žurnálování, kdy každá provedená operace je okamžitě po dokončení připojena do souboru s žurnálem a v případě potřeby je možné rekonstruovat aktuální stav.
- **Replikace** - je podporován víceúrovňový model replikace master-slave, kdy jakýkoliv uzel je možné replikovat na neomezené množství slave uzlů. Jednotlivé slave servery umožňují zápisy, může tedy dojít mezi uzly k datové nekonzistenci (ať už úmyslně nebo neúmyslně).
- **Výkon** - v případě, že není vyžadována perzistence dat, poskytuje databázový systém Redis extrémně vysoký výkon, ve srovnání s databázovými systémy, které vyžadují před dokončením jakékoliv transakce její zápis na disk. Neexistují výrazné rozdíly výkonu mezi operacemi zápisu a operacemi čtení.

- **Publish/Subscribe** - databázový systém Redis má vestavěnou podporu pro možnost zaslání zpráv a jejich naslouchání. To umožňuje reagovat na určité události připojenými posluchači.

Databázový systém je vyvinut v programovacím jazyce ANSI C. Pro přístup k datům je používán velmi jednoduchý textový komunikační protokol, který využívá sadu jednoduchých příkazů. Pro celou řadu programovacích jazyků jsou pak dostupná rozhraní pro přístup k datům v databázovém systému.

Atomické příkazy je možné spojovat do větších transakcí, které se následně také chovají atomicky. Z podstaty atomičnosti transakce ale není možné v jejím průběhu číst hodnoty, které by se jinak v průběhu transakce změnily.

Pro větší datové objemy je nutné počítat se zvýšenou potřebou operační paměti RAM, jelikož databázový systém Redis využívá pro dosažení co nejvyššího výkonu více paměti. Další množství operační paměti RAM je potřeba pro operace spojené s perzistencí dat.

Nejvhodnější využití databázového systému je pro situace, kdy se data velmi často mění, a kdy se dá předem odhadnout přibližný objem ukládaných dat.

Databázový systém Redis je v praxi používán například projekty Flickr, StackOverflow nebo Github.

5.5 HBase

Databázový systém Apache HBase (<http://hbase.apache.org>) je distribuované sloupcové úložiště určené k spolehlivému uchovávání a operacím nad velkými objemy řídko strukturovaných dat. Pro svůj provoz využívá distribuovaný souborový systém HDFS (Hadoop Distributed Filesystem) a poskytuje vlastnosti sloupcových úložišť pro projekt Apache Hadoop[5].



Obr. 19. Logo projektu HBase

Vývoj projektu HBase započal v roce 2006 firmou Powerset. V roce 2008 se stal součástí projektu Hadoop a byl uvolněn pod open source licenci Apache Licence 2. V roce 2010

nad projektem převzala záštitu Apache Foundation.

Nejzásadnější vlastnosti databázového systému jsou:

- **Datová konzistence** - databázový systém HBase poskytuje plné záruky datové konzistence (na rozdíl od způsobu "eventual consistency")
- **Lineární škálovatelnost** - škálování je zajištěno automatickým shardováním v clusteru pomocí regionů. Regiony jsou automaticky rozdělovány a redistribuovány tak, jak objem dat roste, a jak jsou do systému přidávány další uzly.
- **MapReduce** - je možné využít vysoce paralelizovaného MapReduce pro získání agregovaných pohledů na data.
- **REST API** - kromě přímého Java API je možné využít i REST API pomocí protokolu HTTP pro aplikace, které nevyužívají Javu jako programovací jazyk.
- **Výkon náhodného přístupu** - databázový systém HBase je velmi dobře optimalizován pro operace čtení nebo operace zápisu s náhodným přístupem na velkých objemech dat v reálném čase.

Databázový systém HBase je implementován v programovacím jazyce Java a obsahuje knihovny pro rozhraní v jazyce Java. Pro ostatní programovací jazyky je možné použít REST API nebo na jeho základě vystavěné knihovny.

Pro jednotlivé rodiny sloupců v databázovém systému může existovat odlišná konfigurace parametrů, jako je komprese dat, počty uchovávaných verzí jednotlivých záznamů nebo nastavení cachování. Rodiny sloupců pro jednotlivé záznamy jsou uloženy v oddělených datových jednotkách, takže si navzájem neblokuje možnost operací zápisu nebo čtení.

Pro výběr položek je možné, kromě samotného přístupu pomocí klíče položky, využít i hledání pomocí zadaného rozsahu klíčů.

Databázový systém HBase je nejvíce optimalizován pro opravdu velké objemy dat, typicky v řádu stovek milionů záznamů. Dále je nutné poznamenat, že architektura systému vyžaduje, aby pro produkční provoz běžel alespoň na šesti fyzických uzlech.

Systém HBase je v praxi používán například projekty Facebook nebo Twitter a také projekty firmy Adobe.

5.6 Riak

Databázový systém Riak (<http://wiki.basho.com/Riak.html>) je distribuované úložiště typu klíč-hodnota s volitelnou možností výběru backendu pro distribuované ukládání dat na bázi shardování. O hodnotách jsou uchovávány další informace ve formě metadat, například jaký typ obsahu je uložen[19].



Obr. 20. Logo projektu Riak

Systém Riak je vyvíjen firmou Basho Technologies, která s jeho vývojem započala v roce 2009. Systém je dostupný pod open source licencí Apache Licence 2.0.

Nejzásadnější vlastnosti databázového systému jsou:

- **Typy obsahu** - každá hodnota může mít indikován typ obsahu (*Content-Type*), který je v ní uložen. Díky tomu programátor nemusí řešit správnost kódování vkládaných binárních dat, ale stará se o to přímo databázový systém.
- **Propojení dat** - jednotlivé klíče a jejich hodnoty je možné mezi sebou propojit a dosáhnout tak určité míry vlastností grafové databáze. Je možné jednou operací získat množinu dat propojených k jedné položce databáze.
- **MapReduce** - databázový systém Riak má vestavěnou možnost využívat metodu MapReduce pro práci s daty. Definice funkcí je systému předávána ve formátu JSON a je možné za sebe skládat více fázi Map i Reduce.
- **Indexace** - databázový systém obsahuje podporu indexů. Ta je zajištěna pomocí přiřazení několika hodnot danému indexovacímu klíči. Indexy je možné přímo využít i pro MapReduce.
- **REST API** - kromě přímého API pro programovací jazyk Erlang je možné využít i REST API pomocí protokolu HTTP pro aplikace, které Erlang nevyužívají.
- **Škálování** - databázový systém Riak využívá replikaci master-master spolu se shardingem. Data jsou automaticky rovnoměrně rozdělena mezi uzly v systému

a v případě přidání nebo odebrání uzlu se automaticky redistribují.

Databázový systém Riak byl vytvořen v programovacím jazyce Erlang a obsahuje rozhraní jak přímé rozhraní pro tento jazyk, tak i REST rozhraní, na kterém jsou vystavěny knihovny pro další programovací jazyky.

Data jsou kromě klíčů organizována v takzvaných kbelících (anglicky *Bucket*). Unikátním identifikátorem jedné konkrétní hodnoty je dvojice bucket, klíč.

Při operaci čtení je možné specifikovat počet uzlů systému, které musí vrátit výsledek čtení, než je tato operace požadována za úspěšnou. Obdobně i pro operace zápisu je možné stanovit počet uzlů, které musí zápis úspěšně potvrdit. Na základě nastavení replikace a shardingu je takto možné zajistit dostatečnou spolehlivost a dopředu detekovat potenciální selhání uzlů.

Z podstaty replikace typu master-master mohou při souběžných zápisech datové konflikty napříč uzly. Je možné volit řešení, že poslední operace zápis je považována za aktuální, případně je možné vrátit klientu obě konfliktní verze a nechat je rozhodnout, která je správná.

Databázový systém Riak je v praxi využit například v projektu GitHub nebo v projektech firem AOL a Bestbuy.

5.7 Neo4j

Databázový systém Neo4j (<http://neo4j.org>) je grafová databáze, využívající flexibilní síťovou strukturu popisu dat založenou na principech teorie grafů. Plně podporuje transakce a obsahuje vestavěné funkce pro práci s grafy.



Obr. 21. Logo projektu Neo4j

Systém Neo4j je vyvíjen firmou Neo Technologies, Inc. od roku 2007. Je dostupný buď pod open source licencí GNU GPL 3 nebo pod kombinací licence AGPL 3 a komerční licence. V roce 2010 byla vydána první stabilní verze vhodná pro nasazení do produkčního prostředí.

Nejzásadnější vlastnosti databázového systému jsou:

- **Procházení grafem** - databázový systém obsahuje velmi silnou a vysoce optimalizovanou sadu funkcí pro procházení grafem a další grafové operace, jako je například hledání nejkratší cesty grafem.
- **ACID** - systém Neo4j splňuje všechny požadavky ACID, je zde tedy obsažena plná podpora transakcí. Pro zajištění řízení souběžného přístupu je využívána metoda MVCC (viz 4.1.2).
- **Škálování** - databázový systém může pracovat v replikačním režimu master-slave, s možností provádět zápisy na slave uzlech. Data zapsaná ve slave uzlu se s možným zpožděním předají uzlu master, který je dále replikuje na ostatní uzly v systému.
- **REST API** - kromě přímého Java API je možné využít i REST API pomocí protokolu HTTP pro aplikace, které nejsou naprogramovány v jazyce Java.
- **Dotazovací jazyk** - systém Neo4j má vestavěnou podporu pro deklarativní dotazovací jazyk Cypher, který umožňuje přehledný a efektivní dotazování grafové databáze. Syntaxe jazyka Cypher je uzpůsobena k zápisu toho, co je potřeba z databáze získat, ne jak to získat.
- **Indexování** - pro velmi rychlý přímý přístup k prvkům databázového systému je možné vytvářet vlastní indexy a jednotlivé uzly nebo hrany do nich přiřazovat.

Databázový systém Neo4j je implementován v programovacím jazyce Java a poskytuje přímé rozhraní pro použití v Java aplikacích. Pro ostatní programovací jazyky je možné využít REST rozhraní pro přístup nebo na jeho základě vystavěná rozhraní pro daný jazyk.

Základními koncepty databázového systému Neo4j jsou uzly, reprezentující entity, a orientované hrany reprezentující vztahy mezi entitami. Orientace hran není překážkou v procházení grafu, je tedy možné graf procházet i v opačném směru, než je orientace hrany, bez nutnosti vytvářet druhou hranu v opačném směru.

Oba druhy objektů mohou mít přiřazeny vlastnosti, které jsou ve formě asociativního pole. Klíče jsou v tomto případě řetězce a hodnoty mohou nabývat z některého primitivního typu jazyku Java.

Pro práci s objemnými grafovými strukturami je databázový systém Neo4j několikanásobně rychlejší ve srovnání s uložením a zpracováním stejných dat v relačním databázovém systému[16].

Databázový systém Neo4j je v praxi používán například projekty firem Adobe, Cisco nebo Deutsche Telekom.

II. PRAKTICKÁ ČÁST

6 TESTY DATOVĚ-OBJEMOVÝCH LIMITŮ MYSQL

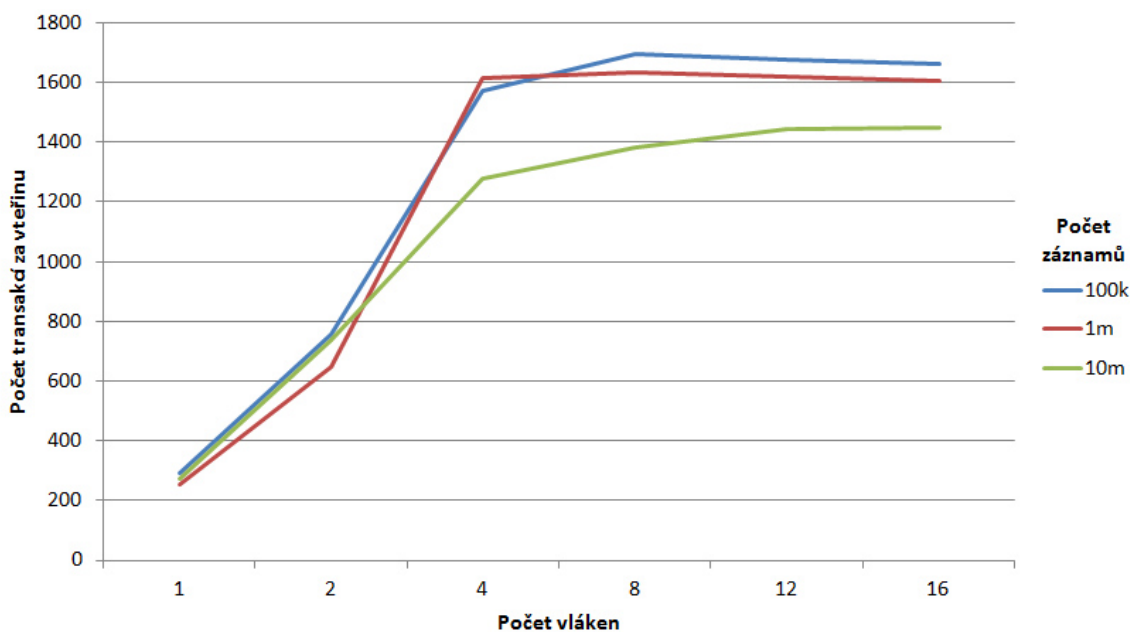
Pro test datově-objemových limitů MySQL serveru verze 5.5.22 byl použit benchmark SysBench (dostupný z <http://sysbench.sourceforge.net>) verze 0.4.12, spouštěný v režimu pouze operací čtení a v režimu kombinace operací čtení a zápisu. Hardwarová konfigurace testovacího serveru je uvedena v sekci 7.1.

Test pracoval s datovým objemem 100 000, 1 000 000 a 10 000 000 řádků v jedné databázové tabulce v úložišti typu MyISAM.

Pro testování bylo zvoleno postupné navyšování počtu vláken provádějící souběžné operace, aby bylo možné určit vliv počtu klientů přistupujících do databáze v jeden okamžik. Použity byly následující počty vláken: 1, 2, 4, 8, 12, 16. Vyšší množství by již bylo negativně ovlivněno dvoujádrovým procesorem serveru, který není schopen u více souběžných vláken zajistit objektivně souběžné provádění.

6.1 Operace čtení

Tento test simuluje práci uzlů typu slave, které provádějí pouze operace čtení.



Obr. 22. MySQL čtení

Výsledky testů jsou uvedeny v grafu na obrázku 22 a v tabulce 2.

Tab. 2. MySQL čtení - počet transakcí za vteřinu

Počet vláken	Počet řádků v tabulce		
	100 000	1 000 000	10 000 000
1	289,90	255,52	272,04
2	758,97	648,31	738,92
4	1572,54	1614,61	1276,54
8	1695,79	1631,97	1381,76
12	1675,91	1617,60	1445,76
16	1662,23	1604,27	1449,35

Z výsledků vyplývá, že s rostoucím počtem položek v tabulce mírně klesá výkon databázového systému, nicméně se nejedná o nijak dramatický pokles. Se vzrůstající mírou souběžného přístupu od počtu 12 vláken výkon dále mírně klesá.

6.2 Kombinace operací čtení a zápisu

Tento test simuluje práci uzlu typu master, na kterém probíhají krom operací čtení také i operace zápisu.

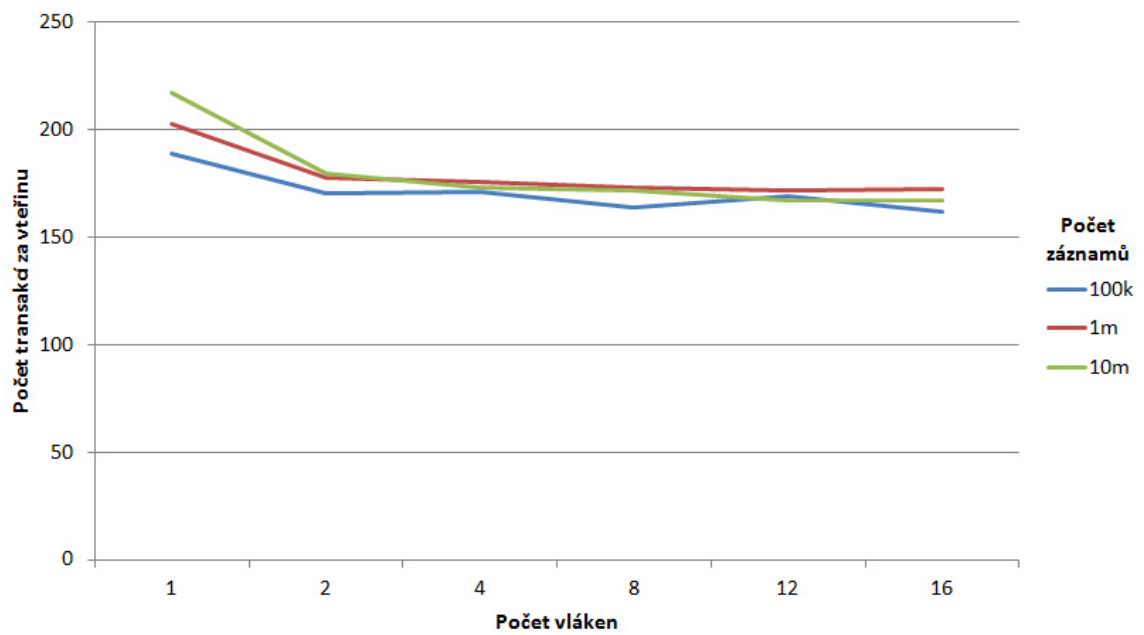
V režimu kombinace operací čtení a zápisů byl stanoven poměr 73% čtení a 26% zápisů, což odpovídá průměrnému dennímu zatížení na master serveru databázového clusteru systému Webnode.

Tab. 3. MySQL čtení/zápis - počet transakcí za vteřinu

Počet vláken	Počet řádků v tabulce		
	100 000	1 000 000	10 000 000
1	188,53	202,50	217,23
2	170,37	177,64	179,25
4	171,10	175,84	173,02
8	163,57	172,81	171,35
12	169,12	171,52	167,01
16	161,83	172,36	166,83

Výsledky testů jsou uvedeny v grafu na obrázku 23 a v tabulce 3.

Z naměřených hodnot je názorně vidět, že pokud se v databázovém systému provádí kromě operací čtení i operace zápisu, pak z důsledku zamykání tabulek dochází k vý-



Obr. 23. MySQL čtení/zápis

raznému poklesu výkonu oproti pouze operacím čtení. Vliv počtu záznamů v tabulce zde již není tak patrný. Zvyšující se množství souběžně přístupujících vláken má však vliv na další snižování počtu transakcí, které databázový systém dokáže provést za daný časový úsek.

7 VÝKONNOSTNÍ SROVNÁNÍ NOSQL DATABÁZÍ

Pro výkonnostní srovnání jsme vybrali čtyři databázové systémy, které jsou na základě jejich vlastností nejvhodnější pro potenciální využití v prostředí systému Webnode. Jedná se o databázové systémy Cassandra (verze 1.0.10), MongoDB (verze 2.0.5) a Redis (verze 2.2.12).

7.1 Hardwarová konfigurace

Výkonnostní testy byly prováděny na běžném komerčně dostupném serveru s následující hardwarovou konfigurací:

- Procesor AMD Dual-Core Opteron 2,4 GHz
- Paměť 8 GB DDR3 ECC
- Pevné disky 2 x 1 TB SAS, 7200 otáček/min, v zapojení RAID0

Operační systém pro testování byl zvolen Ubuntu Linux Server edition verze 12.04 (Linux kernel 3.2.0.24.26).

7.2 YCSB benchmark

K výkonnostnímu testování byl využit nástroj Yahoo! Cloud Serving Benchmark (YCSB) vyvinutý firmou Yahoo!. Tento umožňuje testovat stejnými datovými objemy různá úložiště typu klíč-hodnota, dokumentové databáze a sloupcová úložiště. Zároveň umožňuje testy provádět i na některých relačních databázových systémech včetně MySQL[15].

Použita byla verze benchmarku 0.1.4 dostupná z adresy <https://github.com/brianfrankcooper/YCSB/>.

Pro testování je dostupných několik testovacích scénářů:

- **Update heavy** - poměr operací čtení a zápisu je 50/50.
- **Read mostly** - poměr operací čtení a zápisu je 95/5.
- **Read only** - probíhají pouze operace čtení.
- **Read latest** - jsou přidávány nové záznamy, které jsou také z největší míry čteny.

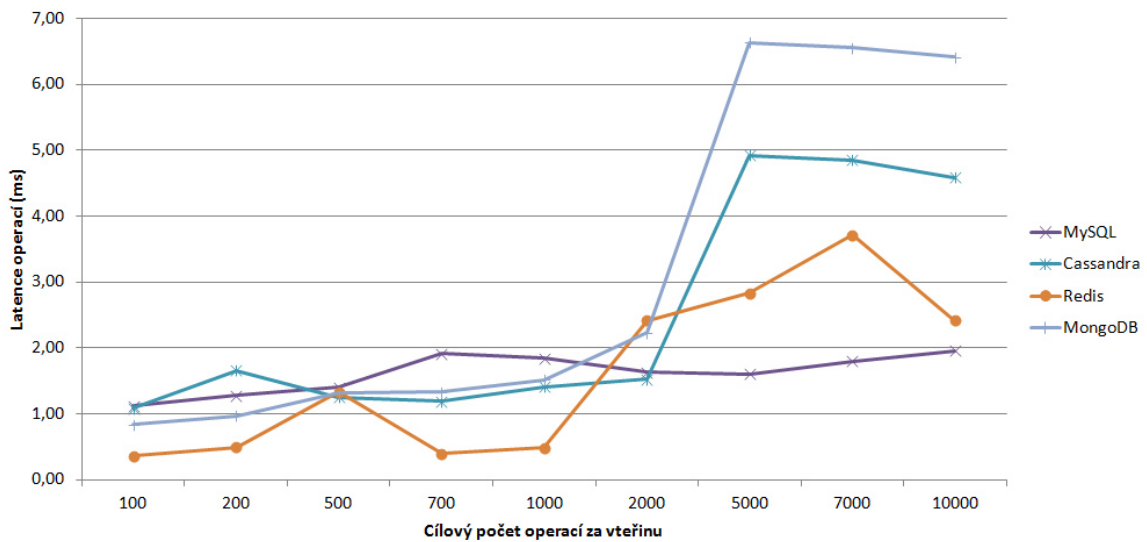
- **Read-modify-write** - záznam je přečten, upraven a změny jsou následně uloženy.

Tyto scénáře velmi dobře simulují běžnou zátěž, se kterou je možné se setkat v širokém poli internetových aplikací. Do každého z databázových systémů byly pro účely testování vloženy 2 000 000 záznamů.

Byly měřeny údaje o počtu provedených operací za vteřinu a také o latenci jednotlivých operací čtení a zápisu v několika testovacích scénářích. Naměřené hodnoty se nacházejí v příloze 2. Měření probíhalo na 10 000 kombinovaných operacích čtení a zápisu podle nastavení jednotlivých testovacích scénářů se zadáním cílového počtu operací za vteřinu od 100 do 10 000. Pro zátěž bylo použito nastavení 30 vláken.

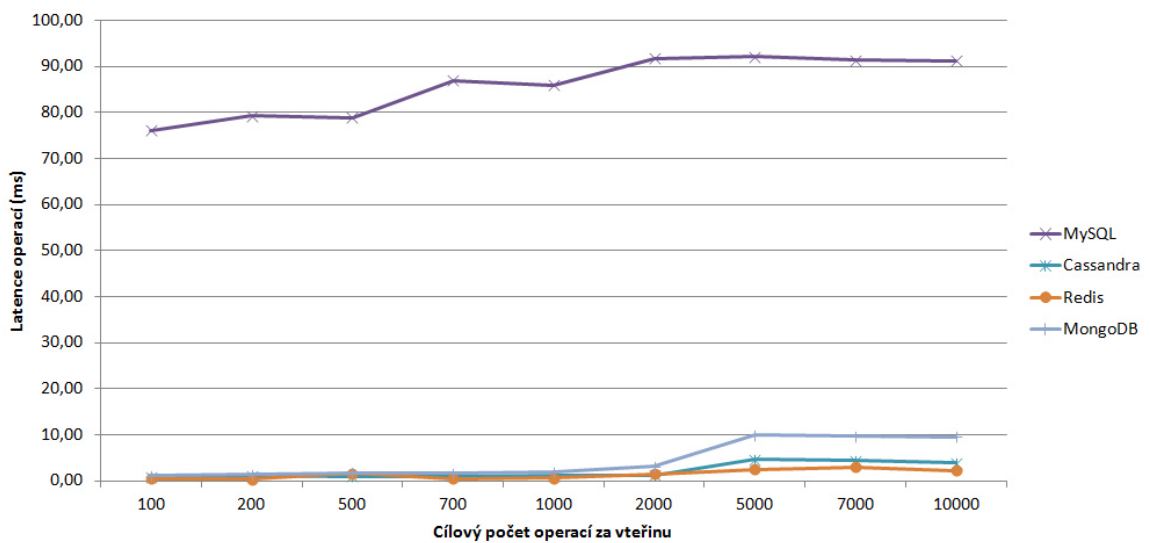
7.3 Výkon při velkém množství čtení

Na grafech v obrázcích 24 a 25 jsou vidět hodnoty latence jednotlivých operací čtení respektive zápisu prováděných v poměru 50/50.



Obr. 24. YCSB - latence čtení v testu 50/50

Nejlepších výsledků v tomto případě dosahuje databázový systém Redis, především z důvodu uložení celé datové sady do operační paměti RAM. Databázový systém Cassandra v tomto případě podává lepší výsledky než při větším poměru čtení a méně zápisů.

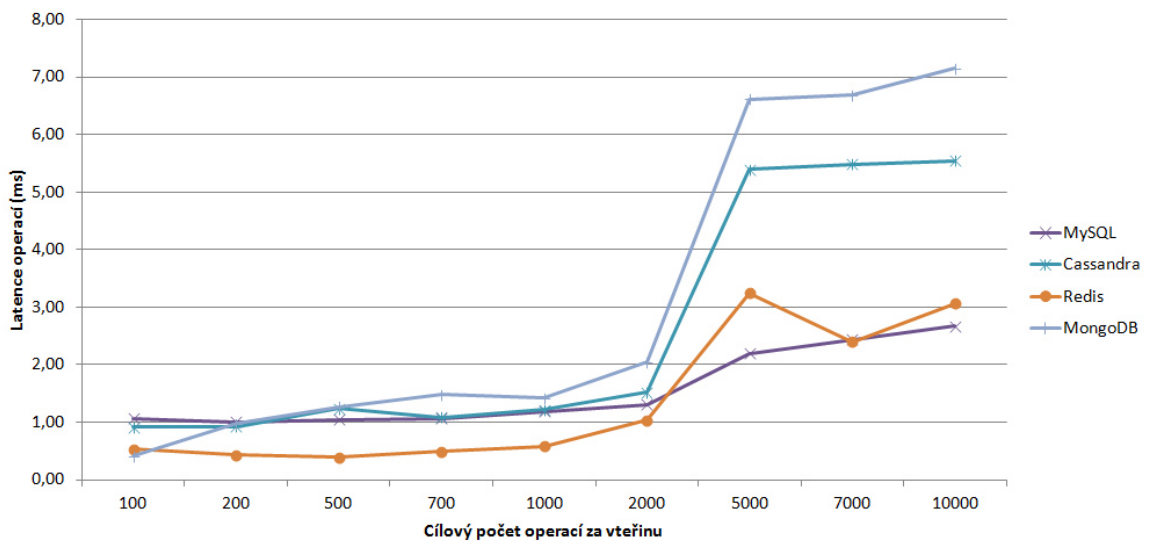


Obr. 25. YCSB - latence zápisu v testu 50/50

Nejhorších výsledků pak v tomto testu dosahuje databázový systém MySQL, který nebyl schopen dosáhnout více než cca 220 operací za vteřinu. To je zapříčiněno především zamykáním tabulek.

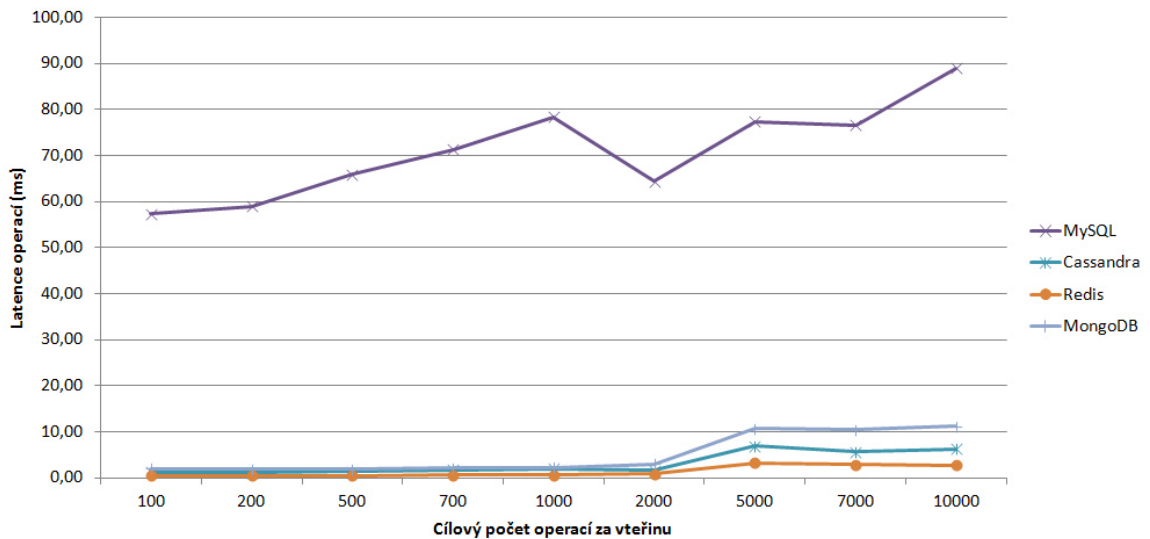
7.4 Výkon při velkém množství zápisů

Na grafech v obrázcích 26 a 27 jsou vidět hodnoty latence jednotlivých operací čtení (poměr čtení vůči zápisu 100/0).



Obr. 26. YCSB - latence čtení v testu 95/5

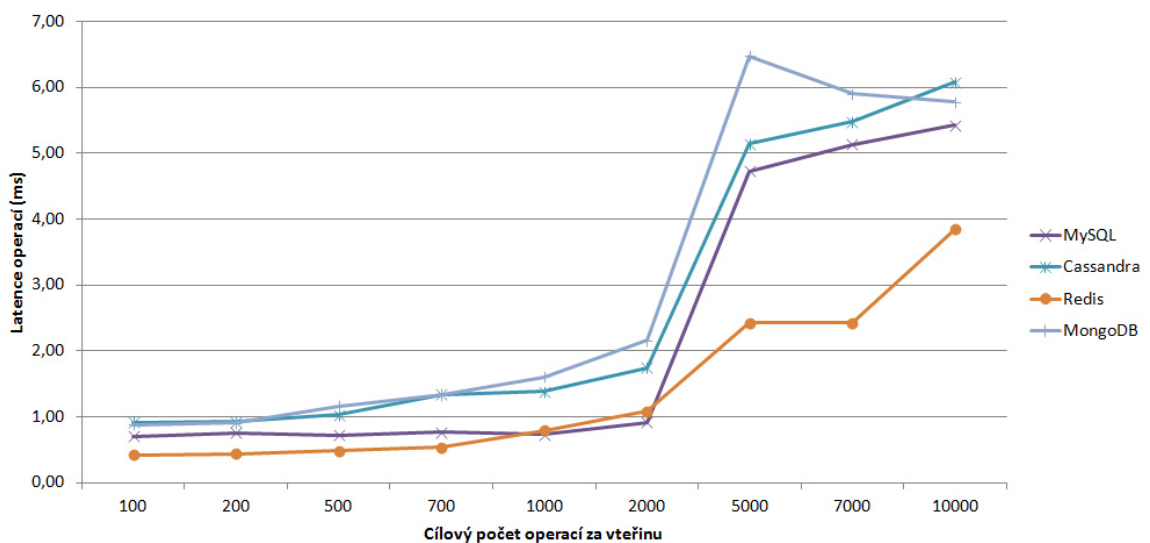
Nejlepší výkon zde opět ze stejných důvodů, jako v předchozím testu, podává databázový systém Redis.



Obr. 27. YCSB - latence zápisu v testu 95/5

Databázový systém MySQL podává velmi špatný výkon především při operacích zápisu, nicméně drží krok s ostatními systémy při čtení do počtu operací 2 000 za vteřinu. Vyššího výkonu již nedosahuje.

7.5 Výkon při pouze čtení



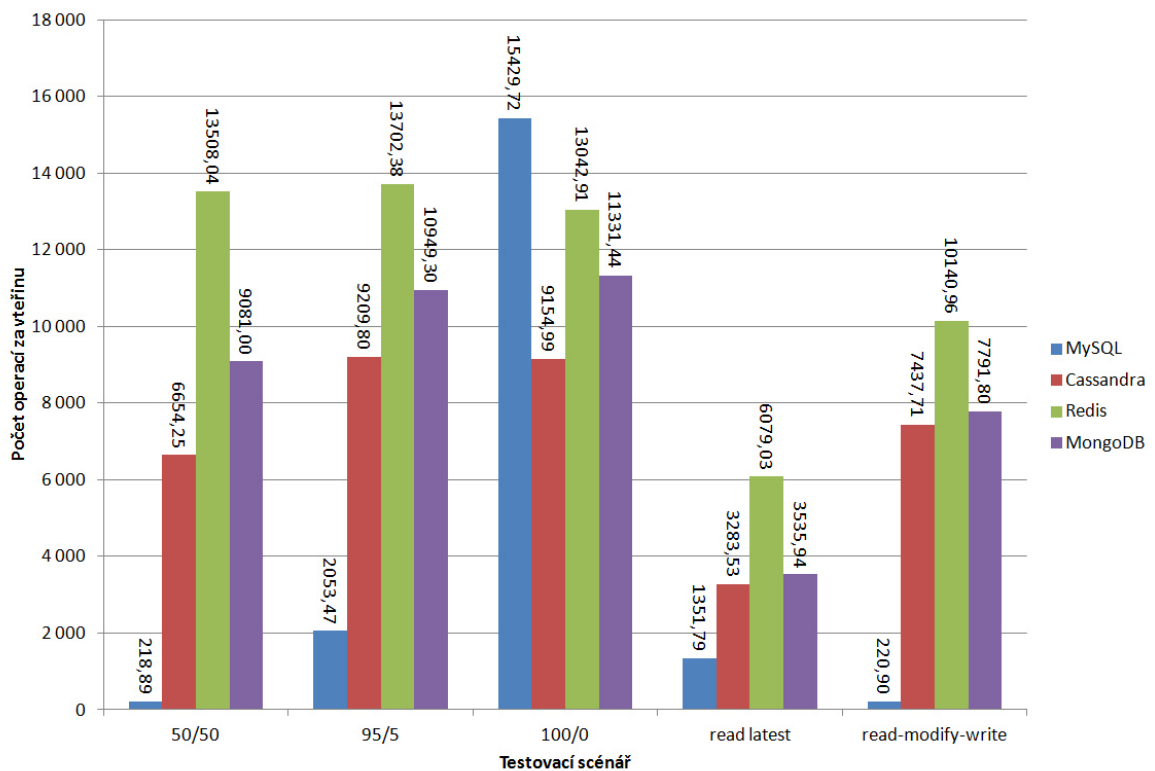
Obr. 28. YCSB - latence čtení v testu 100/0

Na grafu v obrázku 28 jsou vidět hodnoty latence jednotlivých operací čtení respektive zápisu prováděných v poměru 50/50.

Zde vychází jako nejvýkonnější databázový systém MySQL, který není blokován žádnými zámky tabulek pro operace zápisu. Databázový systém Redis jako jediný dokázal zpracovat více než 5 000 operací za vteřinu.

7.6 Celkové srovnání výkonu

V grafu na obrázku 29 a v tabulce 4 je vidět porovnání jednotlivých databázových systémů a jejich výkon v jednotlivých testovacích scénářích.



Obr. 29. Porovnání výkonu v jednotlivých testovacích scénářích YCSB

Nejlépeších výsledků ze všech testovaných databázových systémů dosahoval systém Redis, což je dáno kompletním načtením celé datové sady do operační paměti RAM. Všechny nerelační databázové systémy v kombinovaných testech operací čtení a zápisu velkým poměrem překonávají relační databázový systém MySQL. Jediná situace, kdy MySQL oproti ostatním databázovým systémům podává podstatně lepší výsledky, je test pouze čtení.

Tab. 4. YCSB - počet operací za vteřinu

Testovací scénář	Databázový systém			
	MySQL	Cassandra	Redis	MongoDB
50/50	218,89	6654,25	13508,04	9081,00
95/5	2053,47	9209,80	13702,38	10949,30
100/0	15429,72	9154,99	13042,91	11331,44
read latest	1351,79	3283,53	6079,03	3535,94
read-modify-write	220,90	7437,71	10140,96	7791,80

Databázový systém MySQL dosahuje svých limitů především při velkém množství souběžných operací zápisu, které způsobují zamčení tabulek pro všechny ostatní operace. Na testovací hardwarové sestavě bylo určeno, že od přibližně 500 operací za vteřinu při poměru operací zápisu mezi 5% - 50% je vhodnější využít některého z řešení, které poskytují nerelační databázové systémy. V případě, že je nutné provádět operace čtení, není nutné relační databázový systém MySQL nahrazovat.

8 ANALÝZA VHODNOSTI PRO MIGRACI DO NOSQL DATABÁZE

V této kapitole si představíme některé části systému Webnode, které jsou vhodné pro případnou migraci z relačního databázového systému MySQL do některého z NoSQL databázových systémů.

8.1 Webnode

Webnode (<http://www.webnode.com>) je systém pro snadnou tvorbu webových stránek. Tato služba je v současné době využívána více než 5 000 000 uživateli z celého světa.



Obr. 30. Logo projektu Webnode

Vývoj systému byl zahájen v roce 2008 a během této doby bylo pro chod systému nutné zajistit již více než 100 serverů, na kterých běží relační databázový systém MySQL v replikačním režimu master-slave. Kromě samotné služby tvorby a prezentace webových stránek je součástí systému Webnode několik přidružených interních systémů jako jsou například statistiky, logování přístupů nebo centrum plateb. Většina těchto součástí je implementována v programovacím jazyce PHP.

Pro další růst systému Webnode je důležité zajistit vysoce výkonné datové úložiště, které odstraní problémy, které by jinak databázový systém MySQL v budoucnu mohl způsobovat. Jedná se především o snadnou škálovatelnost a vysokou dostupnost celého systému Webnode. Prioritou je samozřejmě tyto vlastnosti zajistit pro části systému přímo užívané zákazníky, nicméně je důležité tyto vlastnosti poskytovat i pro interní potřeby k zajištění hladkého provozu celého projektu.

8.2 Subsystémy vhodné pro migraci

Pro migraci do NoSQL databázového systému se v prostředí projektu Webnode nabízí hned několik vhodných subsystémů.

8.2.1 Uživatelské sessions

V současném stavu je přihlašování uživatelů k jednotlivým částem systému Webnode řešeno pomocí sessions jazyka PHP, které jsou ukládány do diskového úložiště na serveru, ke kterému je uživatel přihlášen. Pokud se uživatel přihlašuje k více serverům, je nutné data obsažená v session znovu vytvořit a uložit.

Toto omezení by bylo možné eliminovat využitím vhodného vysoce výkonného distribuovaného úložiště typu klíč-hodnota nebo dokumentové databáze. Data uživateleovy session by byla uchovávána v tomto úložišti a byla by dostupná ze všech serverů.

Nejvhodnější se k tomuto účelu jeví migrace do databázového systému Redis.

8.2.2 Logování

Provoz systému Webnode generuje velké množství informací různé podstaty a různého určení. Na jedné straně se jedná o logování přístupů uživatelů k jednotlivým částem systému, dále také vznikají nejrůznější ladící informace nebo informace o výkonu systému.

Tyto informace jsou v současné době ukládány buď do relačního databázového systému MySQL nebo do nejrůznějších souborů napříč systémem. Tyto informace nemají pevně danou strukturu. V případě dat ukládaných do MySQL je po uložení určitého množství záznamu nutné provést rotaci dat a nově vytvořené záznamy ukládat do další nově vytvořené tabulky. To s sebou, v případě že chceme číst historická data, nese potřebu zjišťovat názvy odrotovaných tabulek a velmi to komplikuje navazování na data v dalších tabulkách pomocí spojování (JOIN).

Řešením by bylo sjednotit všechny tyto roztržštěné logy do jediného sloupcového úložiště optimalizovaného pro operace zápisu. Zde se projeví výhoda nepřítomnosti pevného schématu a řídkosti úložiště, tudíž je možné do databázového systému vkládat data z různých zdrojů, které v budoucnu mohou dále přibývat. Další výhodou je možnost provádět pomocí MapReduce funkcí velmi rychlé analýzy na uložených datech.

Nejvhodnější se pro tento účel jeví migrace do databázového systému Apache Cassandra.

8.2.3 Centrum plateb

Centrum plateb je subsystém, který sdružuje veškeré údaje o příchozích platbách za objednané služby systému Webnode. Vzhledem ke globální povaze celého projektu se zpracovávají výpisy plateb z desítek bankovních účtů. Výpisy z rozdílných bankovních účtů mají mírně odlišná pole obsahující data, což klade vysoké nároky na datové schéma.

V současné době je pro každý druh bankovního výpisu v databázi tabulka obsahující pole, která nejsou společná s ostatními bankovními výpisy. V budoucnu je jisté, že budou přibývat další bankovní účty s rozdílným tvarem bankovního výpisu, z čehož vyplývá nutnost dále měnit datové schéma.

V tomto případě by bylo vhodné relační databázi pro tyto data nahradit databází dokumentovou, která nevyžaduje striktně dané datové schéma, ale která nabízí vhodné prostředky pro indexaci obsahu.

Nejvhodnější se pro tento účel jeví migrace do databázového systému MongoDB.

9 PROCES MIGRACE DO NOSQL DATABÁZE

Pro ilustraci procesu migrace z relačního databázového systému MySQL do databázového systému typu NoSQL byl vybrán subsystém logování (viz 8.2.2). Databázovým systémem vybraným pro migraci je Apache Cassandra (viz 5.1)

Prvním krokem procesu migrace je identifikace veškerých míst v systému Webnode, ve kterých probíhá jakákoliv forma logování dat. Tato místa je nutné zdokumentovat a analyzovat průměrné datové objemy a počty operací zápisů zde prováděných.

Na základě této analýzy je potom nutné vytvořit distribuované úložiště Cassandra. Podle zjištěných objemů dat a množství operací zápisů je následně úložiště optimalizováno tak, aby počet uzlů s dostatečnou rezervou pokrýval potřeby logování dat.

Dalším krokem je implementace rozhraní mezi databázovým systémem Cassandra a jazykem PHP za využití existujících knihoven. Za využití tohoto rozhraní je poté implementován zápis logů souběžně se stávajícím řešením (ať už na bazi MySQL nebo souborů). Tím zajistíme, že v průběhu testovacího provozu nemůže dojít k žádné ztrátě dat, v případě že by nový systém nebyl dosud stoprocentně vyladěn. Dále je nutné zápisy do databázového systému Cassandra implementovat tak, aby v případě selhání zápisu mohl skript dále bez přerušení pokračovat.

Zahájením testovacího provozu bude možné sledovat chování databázového systému Cassandra na reálných datech a souběžně porovnávat výkon se stávajícím řešením. Na základě testovacího provozu bude také možné odhadnout, jaké v budoucnu mohou vyvstat potřeby pro rozšíření uzlů databázového systému.

V případech, kdy je to důležité budou potom připraveny skripty zajišťující migraci stávajících historických dat do nového úložiště.

Poté co bude prohlášení ukončeno a nový databázový systém prohlášen za stabilní a lepší náhradu stávajícího způsobu logování, pak je možné původní operace zápisu do relační databáze odstranit.

Poslední fází je potom příprava skriptů pro analýzu nad ukládanými daty, jak z důvodu tvorby statistik tak i například pro automatizované hlídání podezřelých nebo mimořádných situací.

9.1 Částečná realizace migrace

Pro částečnou realizaci migrace do NoSQL řešení byl vybrán subsystém logování, konkrétně logování přihlašování uživatelů do systému Webnode. V současné době probíhá ukládání informací o přihlašování do tabulky v relačním databázovém systému MySQL. Tato tabulka je pak na týdenní bázi rotována a historická data jsou převedena do jiných tabulek.

Na testovacím serveru byla vytvořena keyspace `log` v databázovém systému Apache Cassandra (verze 2.0.12) a byly připraveny následující rodiny sloupců:

`log` Rodina sloupců společná pro všechny logy

`timestamp` Čas logu

`client_ip` IP adresa, odkud požadavek přišel

`server_ip` IP adresa serveru, který požadavek zpracoval

`message` Text logu

`access_log` Rodina sloupců specifická pro logování přihlašování

`login_name` Přihlašovací jméno uživatele

`success` Výsledek, zda bylo přihlášení úspěšné

Na testovacím serveru byl zprovozněn HTTP server Apache spolu se skriptovacím jazykem PHP. Požadavkem pro provoz rozhraní k databázovému systému Cassandra v jazyce PHP je nutnost aktivovat v konfiguraci PHP modul APC (Alternative PHP Cache).

Prvním krokem v přípravě rozhraní je kompilace a instalace modulu Thrift, která umožňuje rozhraní pro konkrétní verzi PHP vytvořit. Tento modul je následně povolen v konfiguračním souboru `php.ini`.

Následně byla nainstalována PHP knihovna `phpcass` verze 1.0.a.2

(<http://thobbs.github.com/phpcassa/>) a byla ověřena její funkčnost v součinnosti s Cassandra serverem. Poté byla implementováno logování přihlašování uživatelů pomocí možností této knihovny tak, aby bylo souběžně zapisováno jak do databázového systému Cassandra tak i do stávajícího databázového systému MySQL.

Současně byla do databázového systému Cassandra přenesena historická data za období předchozích 4 týdnů, která jsou v systému Webnode využívána pro kontrolu nadměrného přihlašování, které může být varovným signálem před pokusy o neoprávněný vstup. Na testovacím serveru byla implementována za využití MapReduce funkcí kontrola počtu pokusů o přihlášení v daném časovém rozpětí, aby bylo možné porovnat rozdíl ve výkonu oproti stávajícímu řešení na bázi databázového systému MySQL.

Objem dat, který se ukládá do logu přihlašování uživatelů je průměrně 62000 záznamů denně, což odpovídá přibližně 43 operací zápisu za minutu. Tyto hodnoty mají vzrůstající tendenci. Vzhledem k tomu, že při každém přihlášení uživatele se provádí kontrola, kolikrát se v daném časovém rozpětí již přihlašoval, pak tento objem dále narůstá o dalších průměrně 43 netriviálních operací čtení, které musí často kombinovat výsledky i z odrotované tabulky s daty za předcházející týden.

Testování těchto operací čtení v prostředí databázového systému Cassandra za využití MapReduce funkcí vykazuje snížení času na provedení jedné operace čtení o 38,7%.

10 MODELOVÝ PŘÍKLAD ČTENÍ A ZÁPISU

Na následujících příkladech si ilustrujeme rozdíly v přístupu k relačnímu databázovému systému MySQL a databázovému systému Cassandra.

Příklady předpokládají funkční prostředí programovacího jazyka PHP s rozšířením mysql a Thrift, které jsou nutné pro funkční rozhraní k databázovému systému MySQL respektive Cassandra. Pro přístup k databázovému systému Cassandra využijeme knihovny phpcassa.

V obou příkladech budeme pracovat s datovým modelem seznamu uživatelů systému, kteří mají definováno své uživatelské jméno, heslo a nepovinně jméno a příjmení.

10.1 Příklad čtení a zápisu v systému MySQL

Nejprve si vytvoříme novou databázi `test`, která bude obsahovat tabulku uživatelů `users`. Následuje kód v jazyce SQL v mutaci používané serverem MySQL.

```
CREATE DATABASE 'test';
USE 'test';
CREATE TABLE 'users' (
    'id'          INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    'username'   VARCHAR(255) NOT NULL,
    'password'   CHAR(32) NOT NULL,
    'firstname'  VARCHAR(255) NULL,
    'lastname'   VARCHAR(255) NULL,
    PRIMARY KEY ('id'),
    UNIQUE KEY 'u_username' ('username')
) ENGINE=MYISAM DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
```

Nyní se můžeme spojit s databázovým serverem MySQL a zvolit si příslušnou databázi.

```
$db_server = '127.0.0.1';
$db_user   = 'testuser';
$db_pass   = 'testpass';
$db_name   = 'test';
```

```
$db = mysql_connect($db_server, $db_user, $db_pass);  
mysql_select_db($db_name, $db);
```

Pokud tyto příkazy proběhnou v pořádku, pak můžeme přistoupit k operaci zápisu a naplnit si tak data do tabulky users. Záměrně zde není ilustrována kontrola správnosti provedení spojení. V proměnné \$db je nyní uložen deskriptor spojení k databázi.

Nejprve si připravíme SQL dotaz pro vložení záznamu. Vzhledem k tomu, že vkládaná data mohou pocházet od nedůvěryhodného zdroje, je nutné je správně ošetřit proti útoku typu SQL Injection.

```
$username = 'user123'; $password = 'pass123';  
$firstname = 'Franta'; $lastname = '';
```

```
$sql = 'INSERT INTO 'users'  
      ('username', 'password', 'firstname', 'lastname')  
VALUES (  
      "'mysql_escape_string($username).'",  
      "'md5($password).'",  
      '($firstname ? "'mysql_escape_string($firstname).'" : 'NULL').',  
      '($lastname ? "'mysql_escape_string($lastname).'" : 'NULL').'  
)';
```

Poté můžeme dotaz spustit v databázi a tím zahájit operaci zápisu.

```
$h = mysql_query($sql, $db);
```

Po provedení dotazu se v databázi uloží záznam s automaticky vygenerovaným celočíselným primárním klíčem.

Nyní přistoupíme k operaci čtení. Nejprve si opět připravíme SQL dotaz, který zajistí přečtení uživatele se zadaným uživatelským jménem.

```
$username = 'user123';
```

```
$sql = 'SELECT * FROM `users`  
      WHERE `username` = "'.mysql_escape_string($username)."'  
      LIMIT 1';
```

Připravený dotaz spustíme v databázi. Pokud je nalezen příslušný záznam, pak bude uložen v proměnné `$row` jako asociativní pole s klíči reprezentujícími názvy sloupců.

```
$h = mysql_query($sql, $db);  
$row = mysql_fetch_assoc($h);
```

Nyní můžeme výsledek dotazu uvolnit a uzavřít spojení k databázi.

```
mysql_free_result($h);  
mysql_close($db);
```

10.2 Příklad čtení a zápisu v systému Cassandra

Nejprve v databázovém systému Cassandra pomocí nástroje pro přístup z příkazové řádky `cassandra-cli` připravíme keyspace (ekvivalent databáze v relačních databázových systémech) `Test` a rodinu sloupců `Users`.

```
create keyspace Test;  
use Test;  
create column family Users;
```

Následně se pomocí knihovny `phpcassa` můžeme připojit k databázovému systému. Pokud má systém více uzlů, je možné je definovat jako pole jejich IP adres a případně i portů. Vytvoříme si instanci třídy `ConnectionPool`, která obsluhuje připojení k databázovému systému.

```
$nodes = array('127.0.0.1:9160');  
$pool = new ConnectionPool('Test', $nodes);
```

Následně si vytvoříme instanci třídy `ColumnFamily`, která reprezentuje vybranou rodinu sloupců, která je přibližným ekvivalentem tabulky, tak jak je známá z relačních databázových systémů.

```
$family = new ColumnFamily($pool, 'Users');
```

Nyní můžeme do databázového systému vložit data. Na rozdíl od zápisu pomocí jazyka SQL není nutné vkládaná data nějak zvláště ošetřovat, protože zde nehrozí nebezpečí útoku SQL Injection. Uživatelské jméno v tomto případě slouží jako klíč pro uložení hodnot.

```
$username = 'user123'; $password = 'pass123';  
$firstname = 'Franta'; $lastname = '';
```

```
$family->insert($username, array(  
    'password' => md5($password),  
    'firstname' => $firstname,  
    'lastname' => $lastname,  
));
```

Operace čtení je pak velmi jednoduchá, k datům přistupujeme pomocí unikátního klíče, v tomto případě pomocí uživatelského jména. Data budou vložena do proměnné `$row` jako asociativní pole s klíči obsahujícími názvy sloupců.

```
$username = 'user123';
```

```
$row = $family->get($username);
```

Spojení není nutné implicitně rušit, k uvolnění prostředků dojde automaticky při destrukci objektů, které byly v průběhu skriptu vytvořeny.

Jak je patrné, oproti přístupu k datům relačních databázových systémů pomocí jazyka SQL, je přístup do databázového systému Cassandra mnohem přehlednější a intuitivnější.

ZÁVĚR

Cílem této diplomové práce bylo zpracovat téma nerelačních databázových systémů v prostředí rozsáhlých internetových aplikací s přítomností vysokých datových objemů a požadavku vysoké dostupnosti.

V teoretické části bylo vytvořeno srovnání nejrozšířenějších typů nerelačních databázových systému a zároveň provedena řešerše konkrétních implementací. Byly zde ilustrovány základní koncepty, které se s nerelačními databázovými systémy pojí a v kontrastu k nim byly porovnány koncepty využívané v prostředí relačních databázových systémů.

V praktické části bylo provedeno testování výkonnostních parametrů vybraných nerelačních databázových systémů spolu s relačním databázovým systémem MySQL. Výsledky těchto testů ukázaly vhodnost jednotlivých systémů pro různé scénáře aplikací s rozdílným poměrem operací čtení a zápisu.

Dále byla provedena analýza subsystémů projektu Webnode a za využití získaných údajů z testování bylo navrženo několik subsystémů vhodných pro migraci na nerelační databázový systém. Na subsystému logování byla provedena částečná realizace migrace do databázového systému Cassandra a srovnání výkonu oproti stávajícímu řešení na bázi relačního databázového systému MySQL. Dále bylo popsán modelový případ operace čtení a zápisu jak v relačním, tak nerelačním databázovém systému, a oba tyto přístupy byly porovnány.

Možným dalším rozšířením této práce je potom kompletní migrace dalších rozsáhlejších subsystémů projektu Webnode na nerelační databázový systém a srovnání výkonu pod zátěží řádově v milionech požadavků denně.

CONCLUSION

The aim of this diploma thesis was to cover the wide area of non-relational database management systems suitable for use in large-scale web-based services with high data flows and high availability requirements.

In theoretical part of this work we presented the most common types of non-relational databases and also we showcased selected implementations of such databases. We introduced important concepts associated with non-relational databases and in contrast to them we placed concepts commonly used in relational database systems.

In practical part of this work we made performance testing of selected non-relational databases together with the MySQL relational database. The results of these tests showed us suitability of particular database systems to various scenarios with different read/write ratios.

With this knowledge we made an analysis of Webnode project subsystems and we proposed which subsystems are suitable to migration on non-relational database system. On the logging subsystem we partially implemented the migration on the Cassandra database systems and the solution was evaluated against the existing MySQL database system. We also described a model use-case of read and write operations on both relational and non-relational systems.

The possible topic of further advancement on this work could be the complete migration of a larger subsystem of Webnode project to a non-relational database system and more extensive performance evaluation under load of millions of requests per day.

SEZNAM POUŽITÉ LITERATURY

- [1] TIWARI, Shashank. *Professional NoSQL*. Wiley, 2011. ISBN 978-0470942246.
- [2] HEWITT, Eben. *Cassandra: The Definitive Guide*. O'Reilly, 2010. ISBN 978-1-4493-9041-9.
- [3] ANDERSON, J. Chris; LEHNARDT, Jan; SLATER, Noah. *CouchDB: The Definitive Guide: Time to Relax*. O'Reilly, 2010. ISBN 978-0-596-15589-6.
- [4] HOLT, Bradley. *Scaling CouchDB : Replication, Clustering, and Administration*. O'Reilly, 2011. ISBN 978-1-4493-0343-3.
- [5] WHITE, Tom. *Hadoop: The Definitive Guide. Second Edition*. O'Reilly, 2010. ISBN 978-1449389734.
- [6] CHODOROW, Kristina; DIROLF, Michael. *MongoDB: The Definitive Guide*. O'Reilly, 2010. ISBN 978-1449381561.
- [7] CHODOROW, Kristina. *Scaling MongoDB*. O'Reilly, 2011. ISBN 978-1449303211.
- [8] KOFLER, Michael; KRAMER, David. *The definitive guide to MySQL*. 3rd ed. New York: Apress, 2005. ISBN 15-905-9535-1.
- [9] *MySQL 5.5 Reference Manual*. [online] Oracle, 2012 [cit. 2012-03-27]. Dostupné z: <http://dev.mysql.com/doc/refman/5.5/en/index.html>
- [10] GILBERT, Seth; LYNCH, Nancy. *Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web*. [online] ACM SIGACT News. 2002, roč. 33, č. 2 [cit. 2012-04-02]. Dostupné z: <http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf>
- [11] DEAN, Jeffrey; GHEMAWAT Sanjay. *MapReduce: Simplified Data Processing on Large Clusters* [online]. Google, Inc., 2004 [cit. 2012-04-05]. Dostupné z: <http://research.google.com/archive/mapreduce-osdi04.pdf>
- [12] SEGUIN, Karl. *The Little Redis Book* [online]. 2012 [cit. 2012-05-07]. Dostupné z: <http://openmymind.net/redis.pdf>

- [13] GRAY, Jim; REUTER Andreas. *Transaction processing: concepts and techniques*. San Mateo: Morgan Kaufmann, 1993. ISBN 15-586-0190-2.
- [14] CORONEL, Carlos; MORRIS, Steven; ROB, Peter. *Database Systems: Design, Implementation, and Management*. 9th ed. Australia: Cengage Learning, 2009. ISBN 05-384-6968-4.
- [15] COOPER, Brian F. et al. *Benchmarking Cloud Serving Systems with YCSB* [online]. Santa Clara: Yahoo! Research, 2010 [cit. 2012-05-12]. Dostupné z: <http://nosqlsummer.org/paper/yahoo-ycsb>
- [16] RODRIGUEZ, Marko A. *MySQL vs. Neo4j on a Large-Scale Graph Traversal* [online]. 2011 [cit. 2012-04-23]. Dostupné z: <http://markorodriguez.com/2011/02/18/mysql-vs-neo4j-on-a-large-scale-graph-traversal/>
- [17] PIEDAD, Floyd; HAWKINS, Michael. *High availability: design, techniques, and processes*. Upper Saddle River: Prentice Hall, 2001. ISBN 01-309-6288-0.
- [18] ROY, Rahul. *Shard - A Database Design*. [online]. 2008 [cit. 2012-05-21]. Dostupné z: <http://technoroy.blogspot.com/2008/07/shard-database-design.html>
- [19] BUCKLE, Simon. *Introducing Riak, Part 1: The language-independent HTTP API*. [online]. 2012 [cit. 2012-05-15]. Dostupné z: <http://www.ibm.com/developerworks/opensource/library/os-riak1/index.html?ca=drs->
- [20] Kolektiv autorů. *Single point of failure - Wikipedia, The Free Encyclopedia*. [online]. 2012 [cit. 2012-04-25]. Dostupné z: http://en.wikipedia.org/wiki/Single_point_of_failure
- [21] MONASH, Curt. *Terminology: Dynamic- vs. fixed-schema databases*. [online]. 2011 [cit. 2012-05-05]. Dostupné z: <http://www.dbms2.com/2011/07/31/dynamic-fixed-schema-databases/>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ACID	Atomicity, consistency, isolation, durability
API	Application Programming Interface
BSON	Binary JSON
CRUD	Create, Read, Update, Delete
DHT	Distributed hash table
GFS	Google File System
HDFS	Hadoop Distributed File System
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
REST	Representational State Transfer
URI	Uniform Resource Identifier
SPOF	Single Point of Failure
SQL	Structured Query Language
XML	Extensible Markup Language
YAML	Yet Another Markup Language
YCSB	Yahoo! Cloud Serving Benchmark

SEZNAM OBRÁZKŮ

Obr. 1. Relace v relačním databázovém systému	12
Obr. 2. Logo projektu MySQL	15
Obr. 3. CAP teorém	20
Obr. 4. proces MapReduce	21
Obr. 5. Úložiště klíč-hodnota	23
Obr. 6. Dokumentová databáze	26
Obr. 7. Grafová databáze	27
Obr. 8. Objektová databáze	28
Obr. 9. Sloupcové úložiště	30
Obr. 10. Příklad Single Point of Failure	32
Obr. 11. Rozíl mezi řízením souběžného přístupu pomocí zámků a verzí	33
Obr. 12. Replikace master-slave	35
Obr. 13. Replikace master-master	36
Obr. 14. Příklad shardingu	37
Obr. 15. Logo projektu Apache Cassandra	39
Obr. 16. Logo projektu MongoDB	41
Obr. 17. Logo projektu CouchDB	43
Obr. 18. Logo projektu Redis	44
Obr. 19. Logo projektu HBase	46
Obr. 20. Logo projektu Riak	48
Obr. 21. Logo projektu Neo4j	49
Obr. 22. MySQL čtení	53
Obr. 23. MySQL čtení/zápis	55
Obr. 24. YCSB - latence čtení v testu 50/50	57
Obr. 25. YCSB - latence zápisu v testu 50/50	58
Obr. 26. YCSB - latence čtení v testu 95/5	58
Obr. 27. YCSB - latence zápisu v testu 95/5	59
Obr. 28. YCSB - latence čtení v testu 100/0	59
Obr. 29. Porovnání výkonu v jednotlivých testovacích scénářích YCSB	60
Obr. 30. Logo projektu Webnode	62

SEZNAM TABULEK

Tab. 1. Přepoččet poměru dostupnosti na čas výpadku	31
Tab. 2. MySQL čtení - počet transakcí za vteřinu	54
Tab. 3. MySQL čtení/zápis - počet transakcí za vteřinu.....	54
Tab. 4. YCSB - počet operací za vteřinu	61
Tab. 5. YCSB - latence (ms) čtení v testu 50/50.....	81
Tab. 6. YCSB - latence (ms) zápisu v testu 50/50.....	81
Tab. 7. YCSB - latence (ms) čtení v testu 95/5	82
Tab. 8. YCSB - latence (ms) zápisu v testu 95/5	82
Tab. 9. YCSB - latence (ms) čtení v testu 100/0.....	83

SEZNAM PŘÍLOH

- P I. Struktura přiloženého CD
- P II. Výsledky měření pomocí benchmarku YCSB

PŘÍLOHA P I. STRUKTURA PŘILOŽENÉHO CD

Přiložené CD obsahuje tyto složky:

`latex` Text této ve zdrojové podobě

`pdf` Text této práce ve výsledné podobě

`tests` Výsledky výkonostních testů

`cassandra` Výsledky testů databázového systému Cassandra

`mongodb` Výsledky testů databázového systému MongoDB

`mysql` Výsledky testů databázového systému MySQL

`redis` Výsledky testů databázového systému Redis

`src` Zdrojové kódy modelové operace čtení a zápisu

PŘÍLOHA P II. VÝSLEDKY MĚŘENÍ POMOCÍ BENCHMARKU YCSB

Tab. 5. YCSB - latence (ms) čtení v testu 50/50

Databázový systém	Cílový počet operací za vteřinu				
	100	200	500	700	1 000
MySQL	1,12	1,27	1,40	1,91	1,84
Cassandra	1,08	1,66	1,25	1,19	1,41
Redis	0,36	0,50	1,33	0,40	0,49
MongoDB	0,84	0,97	1,31	1,33	1,52
	2 000	5 000	7 000	10 000	
MySQL	1,63	1,60	1,80	1,95	
Cassandra	1,53	4,92	4,85	4,58	
Redis	2,41	2,83	3,71	2,42	
MongoDB	2,23	6,63	6,55	6,41	

Tab. 6. YCSB - latence (ms) zápisu v testu 50/50

Databázový systém	Cílový počet operací za vteřinu				
	100	200	500	700	1 000
MySQL	76,12	79,24	78,85	86,88	85,97
Cassandra	0,71	1,15	0,98	0,97	1,15
Redis	0,42	0,40	1,56	0,45	0,63
MongoDB	1,16	1,32	1,73	1,55	1,81
	2 000	5 000	7 000	10 000	
MySQL	91,78	92,13	91,30	91,27	
Cassandra	1,21	4,58	4,35	3,93	
Redis	1,49	2,44	2,96	2,28	
MongoDB	3,15	9,92	9,65	9,58	

Tab. 7. YCSB - latence (ms) čtení v testu 95/5

Databázový systém	Cílový počet operací za vteřinu				
	100	200	500	700	1 000
MySQL	1,07	1,00	1,05	1,07	1,19
Cassandra	0,91	0,93	1,24	1,09	1,22
Redis	0,53	0,43	0,40	0,49	0,58
MongoDB	0,41	0,98	1,27	1,48	1,42
	2 000	5 000	7 000	10 000	
MySQL	1,31	2,20	2,44	2,67	
Cassandra	1,52	5,39	5,48	5,54	
Redis	1,04	3,24	2,40	3,06	
MongoDB	2,04	6,61	6,68	7,14	

Tab. 8. YCSB - latence (ms) zápisu v testu 95/5

Databázový systém	Cílový počet operací za vteřinu				
	100	200	500	700	1 000
MySQL	57,26	59,01	65,86	71,35	78,37
Cassandra	1,19	1,07	1,37	1,74	1,92
Redis	0,53	0,53	0,48	0,61	0,63
MongoDB	1,96	1,86	1,89	2,07	2,27
	2 000	5 000	7 000	10 000	
MySQL	64,41	77,37	76,62	89,01	
Cassandra	1,65	6,91	5,60	6,23	
Redis	0,80	3,24	2,81	2,75	
MongoDB	3,03	10,71	10,45	11,13	

Tab. 9. YCSB - latence (ms) čtení v testu 100/0

Databázový systém	Cílový počet operací za vteřinu				
	100	200	500	700	1 000
MySQL	0,70	0,76	0,73	0,77	0,73
Cassandra	0,91	0,92	1,03	1,34	1,39
Redis	0,42	0,43	0,48	0,54	0,79
MongoDB	0,88	0,91	1,16	1,33	1,61
	2 000	5 000	7 000	10 000	
MySQL	0,91	4,72	5,13	5,43	
Cassandra	1,74	5,14	5,48	6,07	
Redis	1,09	2,42	2,42	3,85	
MongoDB	2,16	6,47	5,90	5,77	