

Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informatika

NoSQL databáze

BAKALÁŘSKÁ PRÁCE

Student : Jakub Mrozek

Vedoucí : RNDr. Helena Palovská, Ph.D.

Oponent : Ing. Tomáš Bruckner, Ph.D.

2014

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 9. 12. 2013

Jakub Mrozek

Poděkování

Děkuji RNDr. Palovské, Ph.D. za pomoc při zpracování bakalářské práce.

Poděkování rovněž patří Lukáši Linhartovi za cenné připomínky k textu o MongoDB.

Abstrakt

Tématem bakalářské práce jsou NoSQL databáze. Práce se zaměřuje především na představení nejpoužívanějších druhů NoSQL databází a možností jejich využití. Celý text předpokládá, že je již čtenář dostatečně seznámen se základními principy NoSQL databází a potřebuje se zorientovat v množství NoSQL databází, které jsou aktuálně na trhu k dispozici. Cílem bakalářské práce je tedy poskytnout tomuto čtenáři vodítko k výběru správné databáze pro řešení jeho konkrétního problému.

Práce je rozdělena do dvou základních částí. V teoretické části je čtenář seznámen s danou kategorií NoSQL databází. Důraz je kladen na podrobné představení nejpoužívanějšího zástupce, a to včetně ukázky práce s databází. V praktické části pak čtenář najde porovnání několika zástupců dané kategorie NoSQL databází.

Klíčová slova

Databáze, NoSQL, CouchDB, RavenDB, MongoDB, Neo4j, OrientDB, Titan, Redis, Memcached, Riak, Cassandra.

Abstract

The topic of this thesis is NoSQL databases and is mainly focused on presenting the most common types of NoSQL databases and their use. This text assumes that the reader is sufficiently familiar with the basic principles of NoSQL databases and is aimed at helping the reader to find their way through a number of NoSQL databases that are currently available on the market. The main objective is to provide readers with a guide for choosing the right database solution for the particular problem they are trying to solve.

The work is divided into two main parts. In the theoretical part, the reader will find basic information about the types of NoSQL databases. The most widely used representatives, including examples of work with databases, will be presented in detail. In the practical part, the reader will find a comparison of several representatives in the category of NoSQL databases.

Keywords

Database, NoSQL, CouchDB, RavenDB, MongoDB, Neo4j, OrientDB, Titan, Redis, Memcached, Riak, Cassandra.

Obsah

1	ÚVOD	1
1.1	OČEKÁVANÉ CÍLE A PŘÍNOSY	2
1.2	METODY DOSAŽENÍ CÍLŮ	3
1.3	PŘEDPOKLADY A OMEZENÍ	3
1.4	STRUKTURA PRÁCE	4
2	REŠERŠE VYBRANÝCH PRACÍ	5
3	DĚLENÍ NOSQL DATABÁZÍ	7
4	DOKUMENTOVĚ ORIENTOVANÉ DATABÁZE	8
4.1	MONGODB	9
4.1.1	Struktura dat	9
4.1.2	Atomicita	10
4.1.3	Práce s velkými soubory	10
4.1.4	Indexy	10
4.1.5	Agragace a MapReduce	11
4.1.6	Replikace	11
4.1.7	Dotazování a aktualizace dat	12
4.1.8	Optimalizace	12
4.1.9	Škálovatelnost	13
4.1.10	Licence	14
4.1.11	Rozhraní k programovacím jazykům	14
4.1.12	Hostování a cloud	15
4.1.13	Komunita, dokumentace	15
4.1.14	Nástroje	15
4.1.15	Případové studie	16
4.1.16	Ukázka	16
5	GRAFOVÉ DATABÁZE	23
5.1	NEO4J	24
5.1.1	Struktura dat	24
5.1.2	Dotazovací jazyk Cypher	26
5.1.3	Transakce	28
5.1.4	REST API	29
5.1.5	Webové rozhraní	29
5.1.6	Zálohování	29
5.1.7	Replikace a škálovatelnost	30
5.1.8	Licence	30
5.1.9	Rozhraní k programovacím jazykům	31
5.1.10	Komunita, dokumentace	31
5.1.11	Nástroje	32
5.1.12	Případové studie	32
5.1.13	Ukázka	32
6	DATABÁZE TYPU KLÍČ/HODNOTA	37
6.1	REDIS	38

6.1.1	Struktura dat.....	38
6.1.2	Transakce.....	41
6.1.3	Expirace klíčů.....	42
6.1.4	Publish/Subscribe.....	43
6.1.5	Skriptování v Lua.....	43
6.1.6	Dělení ukládaných dat.....	44
6.1.7	Replikace.....	44
6.1.8	Perzistence, zálohování dat.....	45
6.1.9	Licence.....	45
6.1.10	Rozhraní k programovacím jazykům.....	46
6.1.11	Komunita, dokumentace.....	46
6.1.12	Nástroje.....	46
6.1.13	Případové studie.....	46
6.1.14	Ukázka.....	47
7	SROVNÁNÍ NOSQL DATABÁZÍ.....	50
7.1	ZKOUMANÉ PARAMETRY.....	51
7.1.1	Komunita a dokumentace.....	51
7.1.2	Transakce.....	51
7.1.3	REST API.....	51
7.1.4	Nativní rozhraní, podpora programovacích jazyků.....	51
7.1.5	Serverové skriptování.....	52
7.1.6	Cena a licence.....	52
7.1.7	Nezahrnuté parametry.....	52
7.2	SROVNÁNÍ DOKUMENTOVĚ ORIENTOVANÝCH DATABÁZÍ.....	53
7.2.1	Zhodnocení.....	54
7.3	SROVNÁNÍ GRAFOVÝCH DATABÁZÍ.....	55
7.3.1	Zhodnocení.....	56
7.4	SROVNÁNÍ DATABÁZÍ TYPU KLÍČ/HODNOTA.....	57
7.4.1	Zhodnocení.....	58
8	ZÁVĚR.....	59
8.1	MOŽNOSTI DALŠÍHO VÝZKUMU.....	60
8.1.1	Databáze určené primárně pro vyhledávání.....	60
8.1.2	Cloudové databáze.....	60
	TERMINOLOGICKÝ SLOVNÍK.....	61
	SEZNAM LITERATURY.....	64
	SEZNAM OBRÁZKŮ A TABULEK.....	71

1 Úvod

Využívání NoSQL databází představuje jednu z nejdůležitějších technologických změn posledních let, především v oblasti webových aplikací. S příchodem sociálních sítí jako je Facebook či Twitter a konceptu nazvaného Web 2.0 se stává web také prostředkem pro komunikaci a sdílení obsahu mezi mnoha lidmi po celém světě. Tato revoluce má významný dopad na technologie a nástroje, které autoři webových aplikací ke své práci potřebují a dramatický nárůst zájmu o NoSQL je její průvodní jev.

Mezi odbornou veřejností je poměrně dobře znám příběh největší sociální sítě na světě, Facebooku, jehož databázové požadavky výrazně převyšovaly soudobé možnosti klasických SQL databází, a tak byli jeho programátoři nuceni vytvořit zcela nový databázový systém, Cassandra [1]. Facebook nebyla jediná společnost, která se vydala cestou tvorby vlastního NoSQL systému, další známý příklad je třeba společnost Google a její databáze BigTable [2].

Velmi dobrý zdroj informací o aktuálních trendech v informatice jsou statistiky vyhledávačů práce. Podle populárního vyhledávače Indeed.com dochází každý rok k zdvojnásobení počtu nabídek pracovních pozic, které mají mezi požadavky na uchazeče uvedenu i jednu z NoSQL databází [3]. Není se čemu divit. Podle výzkumné organizace SINTEF bylo 90% všech dat vytvořeno za poslední dva roky [4]. Je zřejmé, že takový nárůst množství dat musí zákonitě vyvolat nové požadavky na databázová uložení.

Relační databáze nemusí být vždy to správné řešení pro každý problém. Kupříkladu Michal Buchman ve své prezentaci na mezinárodní konferenci Webexpo 2012 ukázal porovnání výkonu relační databáze a grafové databáze Neo4j [5]. Pro experiment bylo vytvořeno 1000 generovaných záznamů, které představují osoby. Dále každá z těchto osob měla vytvořen vztah k 50 jiným, náhodně vybraným osobám z daného vzorku. Na databázi byl zadán dotaz, zda existuje cesta mezi dvěma osobami do hloubky 4 (tedy osoba A má vztah k osobě, který má vztah k osobě, který má vztah k osobě, který má vztah k osobě B). Tento dotaz dokážou relační databáze zpracovat v průměru za 2000 ms. Grafová databáze tuto úlohu

zpracovala za 2 ms, tedy 1000x rychleji než relační databáze. Následně se počet osob v databázi ztisícínásobil. Relační databáze v tomto případě již nebyla schopna v akceptovatelném čase dotaz dokončit, grafová databáze však dotaz zpracovala opět za 2 ms, protože je pro tyto typy dotazů přímo optimalizovaná.

Tento experiment není čistě laboratorní příklad, nýbrž jde o úlohu, kterou řeší mnohé aplikace různě na světě. Např. největší sociální síť Facebook obsahuje ještě tisíckrát více uživatelů, než obsahoval experiment, a mnohem více než 50 vazeb mezi nimi [6]. Je zřejmé, že tyto úlohy již pomocí klasické relační databáze řešit nelze a je potřeba použít NoSQL databázi pro tyto účely optimalizovanou.

Lze předpokládat, že v budoucnosti vliv NoSQL databází ještě výrazně poroste. Vše směřuje k další generaci webu, který již za nás „bude myslet“, tedy analyzovat obrovské množství dat ze vstupu mnoha uživatelů a vytvářet z nich výstup pro jednoho konkrétního uživatele přes různé statistické metody a hledání vazeb mezi daty, čemuž bude potřeba přizpůsobit i databázové prostředky. Vidina velké perspektivy NoSQL databází je také hlavní důvod, proč jsem si pro svojí bakalářskou práci vybral právě téma NoSQL databází.

1.1 Očekávané cíle a přínosy

Hlavním cílem práce je představení základních kategorií NoSQL databází vč. nejdůležitějších zástupců v každé z nich. Dalším cílem je pak srovnání nejpoužívanějších NoSQL databází a poskytnutí vodítka ve výběru konkrétní databáze pro daný projekt.

Cíle lze popsat také takto. Představuji si, že práci čte IT odborník, který zná základní pojmy a principy, na kterých NoSQL databáze fungují, avšak nezná dostatečně základní kategorie NoSQL databází. Kupříkladu si není jist, zda pro svůj projekt raději použije dokumentově orientovanou databázi anebo si vystačí jen s databází typu klíč/hodnota. Po prostudování práce by si měl snadno tuto otázku zodpovědět. Navíc by měl mít představu o tom, kterou konkrétní databázi použije a jaká pozitiva a negativa mu tento výběr přinese.

1.2 Metody dosažení cílů

Aby bylo možné stanovené cíle naplnit, jsou použity různé postupy. V první řadě jde o podrobné zkoumání literatury, která je k databázím dostupná. Protože jsou NoSQL databáze ve většině případů velmi mladé, není k dispozici tolik odborné literatury jako v případě relačních databází. Proto jsou obvykle hlavním zdrojem informací oficiální manuály. Avšak byla-li k dané databázi dobře recenzovaná kniha, byla zahrnuta mezi informační zdroje. Další informace jsou pak čerpány z mnoha různých oblastí, vítaným zdrojem informací byly třeba případové studie.

Každá z popisovaných databází byla odzkoušena na jednoduché aplikaci, která je pro danou kategorii NoSQL databází vhodná. Díky tomu bylo možné zahrnout do textu i její subjektivní hodnocení.

1.3 Předpoklady a omezení

Celý text přímo navazuje na bakalářskou práci Richarda Günzla, obhájenou v roce 2012 na Vysoké škole ekonomické v Praze [7]. V této práci jsou čtenáři podrobně vysvětleny základní pojmy a principy fungování NoSQL databází v dostatečném rozsahu, a proto tyto informace ve své práci již nezmiňuji. Od čtenáře se tedy očekává základní úroveň znalostí o NoSQL databázích v rozsahu uvedené práce.

Práce se rovněž nezabývá všemi kategoriemi NoSQL databází, ale pouze těmi, s nimiž se programátoři běžně setkávají ve své praxi. Konkrétně se zabývá těmito kategoriemi NoSQL databází:

- dokumentově orientovanými databázemi,
- grafovými databázemi a
- databázemi typu klíč/hodnota.

1.4 Struktura práce

Práce je rozdělena do dvou základních částí.

V teoretické části jsou čtenáři představeny základní kategorie NoSQL databází. U každého druhu databáze najde čtenář základní informace o jeho odlišnostech a jeho nejpoužívanějšího zástupce. V průběhu práce bude čtenář podrobněji seznámen se třemi databázovými systémy:

- MongoDB,
- Neo4j,
- Redis.

Základní představu o práci s databázovým systémem může získat z praktické ukázky jednoduché aplikace vhodné pro danou kategorii NoSQL databáze. Popis každého databázového systému je navíc doplněn o případové studie.

V praktické části jde o srovnání nejpoužívanějších zástupců NoSQL databází v každé kategorii. Pokud je tedy čtenář rozhodnut pro konkrétní typ databáze, pak v této části si jednotlivé systémy může srovnat a rozhodnout se pro jeden z nich.

2 Rešerše vybraných prací

Téma NoSQL je častým námětem různých vědeckých prací. Na VŠE se tématem zabýval třeba Richard Günzl věnující se obecné problematice NoSQL databází [7], Ondřej Pultera, který zpracoval populární databázi CouchDB [8], či Martin Petera, který dopodrobna rozebral databázi MongoDB [9].

Práce Richarda Günzla rozebírá základní dělení databází. Vysvětluje zde třeba přesný význam zkratky NoSQL, která může v některých lidech vyvolávat pocit, že jsou NoSQL databáze fanaticky vymezeny vůči SQL. Opak je však pravdou. Mnohé NoSQL databáze jsou jazykem SQL výrazně inspirovány a přímo počítají s tím, že pro různé aplikace mohou běžet vedle sebe klasické relační databáze i NoSQL databáze a data mezi sebou synchronizují. Jako konkrétní příklad lze uvést Neo4j, která má v dokumentaci přímo sekci, která se integrací s SQL databázemi zabývá [10]. Pojem NoSQL tedy neznamena striktní odmítání SQL jako jazyka, ale skutečný význam zkratky je „Not only SQL“, tedy „nejen SQL“.

Richard Günzl také ve své práci okrajově narazil na téma cloud computingu a databází v cloudu. Toto téma by samo o sobě vydalo na rozsáhlou práci. Existuje velké množství poskytovatelů cloud hostingů, kteří nabízejí levné hostování dat v NoSQL databázích. Jako ukázkový příklad uvedu hosting Heroku, který umožňuje napojení aplikací na databáze Redis, MongoDB, Neo4j nebo Elasticsearch.

Dále Richard Günzl popisuje různé modely dat u NoSQL a způsoby získávání dat. V sekci Dotazování je uvedeno: *„nejméně propracovanou komponentou v NoSQL databázích je dotazování“*. S tím však (podobně jako s celou danou částí práce) nemohu rozhodně souhlasit. Kupříkladu Neo4j využívá k dotazování vlastní deklarativní jazyk Cypher, který je z SQL velmi inspirován. K získávání dat využívá pattern matching, se kterým se subjektivně pracuje lépe než s SQL a nabízí podobné možnosti jako SQL. Dojem z nepropracovanosti dotazování v NoSQL nejspíš vychází z autorovy zkušenosti s Cassandrou, kterou v práci podrobněji rozebírá. Nelze ale jednoznačně tvrdit, že je dotazování nejméně propracovanou komponentou u všech NoSQL databází.

V práci je také rozebráno velké téma NoSQL databází, a to transakce. Jak Richard píše, některé NoSQL se dobrovolně vzdávají ACID transakcí ve prospěch jiných vlastností, jako je třeba lepší škálovatelnost.

Podobnou tematikou se zabývá i práce Ondřej Pultera, která byla obhájena na VŠE v roce 2011 [8]. Práce se zaměřuje podrobně na CouchDB. Na CouchDB je zajímavé třeba to, že obsahuje REST API, což znamená, že s databází může komunikovat kdokoliv a není potřeba vytvářet speciální nativní rozhraní pro daný programovací jazyk.

REST API u NoSQL databází má velký význam pro moderní HTML5 aplikace. Ty totiž nepotřebují zvláštní serverovou část, mohou komunikovat se vzdálenou databází přímo pomocí REST API. CouchDB rozhodně není jediná databáze, která REST API podporuje, stejně je na tom třeba Elasticsearch nebo grafová databáze Neo4j.

Zajímavou práci na téma NoSQL databází vytvořil Martin Petera, který si zvolil jako téma MongoDB [9]. Tato databáze je podrobněji rozebrána také v tomto textu, protože jde dlouhodobě o vůbec nejpoužívanější NoSQL databázi [11].

Cenným zdrojem informací při zpracování práce byla také kniha Erika Redmonda a Jima R. Wilsona *Seven Databases in Seven Weeks* [42]. V této knize jsou probrány databáze Redis, Neo4j, CouchDB, MongoDB, HBase, Riak a Postgres a kniha byla důležitým zdrojem informací při zpracování textů o Neo4j, MongoDB a Redisu.

3 Dělení NoSQL databází

Protože mezi NoSQL databázemi existuje výrazné rozdíly a jejich jediným společným znakem je fakt, že k dotazování nepoužívají (jen) jazyk SQL, existuje několik druhů dělení NoSQL databází.

Jedním z používaných dělení je to, které použil ve své prezentaci Ben Scofield [12]. Databáze rozdělil do kategorií podle kvalitativních parametrů a podle jejich datového modelu. Dnes se pravděpodobně toto dělení používá vůbec nejčastěji.

Při dělení NoSQL databází určil 4 kategorie:

- dokumentově orientované databáze,
- databáze typu klíč/hodnota,
- grafové databáze,
- sloupcové databáze.

Zde jde o užší pojetí významu NoSQL databází. Do širšího pojetí lze dále řadit objektové databáze, XML databáze, multidimenzionální databáze, více-hodnotové databáze, grid a cloud řešení [7].

4 Dokumentově orientované databáze

Dokumentově orientované databáze ukládají a operují s daty ve formě dokumentu. Dokument se skládá ze semi-strukturovaných dat, což znamená, že dokument obsahuje jak data samotná, tak popis jejich struktury [13]. Zde je patrný rozdíl oproti klasickým relačním databázím, které definují předem formu tabulek a každý záznam v tabulce pak musí mít přesně danou strukturu.

Každý dokument může mít svojí vlastní strukturu, která se může lišit napříč různými dokumenty ve stejné kolekci. Dokumenty mohou být v sobě navíc různě vložené, což je opět rozdíl oproti relačním databázím, kde je struktura tabulek lineární a hierarchická data je nutné rozložit mezi více záznamů, které jsou pak spojeny pomocí relací.

Výhodou dokumentově orientovaných databází je snadná manipulace s daty. U relačních databází je potřeba složitou strukturu dat nejprve dekomponovat a vložit do databáze, v případě dokumentových databází lze často data vložit tak, jak je s nimi operováno uvnitř programu. Proto se třeba databáze MongoDB stala velmi populární u programátorů Node.js, protože je možné jedním příkazem překonvertovat objekt v JavaScriptu na JSON, které se pak do databáze přímo vloží.

Mezi nejpoužívanější dokumentově orientované databáze patří MongoDB, CouchDB a RavenDB [11].

CouchDB je ze zmíněných databází nejstarší, její první verze byla vydána v roce 2005. Pro dotazování nabízí REST API a data ukládá v JSON. Umožňuje nad databází spouštět vlastní skripty napsané v JavaScriptu. [14]

RavenDB je napsaná v .NET a pro tuto platformu je i určena. Pro komunikaci s databází je možné využít jak nativního klienta v .NET, tak REST API. Oproti některým jiným NoSQL databázím podporuje ACID transakce. [15]

4.1 MongoDB

MongoDB patří mezi nejpoužívanější dokumentově-orientované NoSQL databáze [11]. Dle statistik populární sítě LinkedIn z přelomu roku 2011 a 2012 největší počet uživatelů zařazuje z NoSQL databází právě MongoDB mezi své dovednosti [16].

MongoDB je škálovatelná, vysoce výkonná open-source databáze, která ukládá data ve formátu BSON, což je serializovaný formát JSON do binární podoby [17]. Protože uživatel zadává a získává data přes formát JSON a komunikuje s databází prostřednictvím JavaScriptu, stala se databáze velice populární u vývojářů webových aplikací, kteří mohou začít používat MongoDB velmi rychle, aniž by se museli učit zcela nový dotazovací jazyk.

4.1.1 Struktura dat

MongoDB data ukládá v kolekcích, které jsou obdobou tabulek v relačních databázích. Každá kolekce dále obsahuje BSON dokumenty, které lze chápat jako záznamy v relačních tabulkách. Na rozdíl od analogie s relačními databázemi však není potřeba explicitně uvádět strukturu kolekce při jejím vytváření, která je tedy v MongoDB mnohem flexibilnější. Kolekce se automaticky vytvoří při prvním vložení dokumentu. Maximální velikost dokumentu je 16 MB, pro větší dokumenty se používá GridFS. [18]

Vedle klasických kolekcí podporuje MongoDB kolekce s limitní velikostí (capped collections), kterým lze nastavit maximální velikost. V případě překročení velikosti kolekce je nejstarší záznam přemazán novým záznamem. Tento typ kolekcí se hodí např. pro logy, kde správce nechce, aby velikost kolekce s logy překročila stanovenou velikost souboru. [18]

MongoDB ve verzi 2.2 přidává možnost nastavit u kolekce datum expirace dokumentů. Po uplynutí stanoveného času dojde ke smazání dokumentu z kolekce. Tato možnost je velmi užitečná např. u správy relací či ukládání kešovaných dat, které jsou validní pouze po určitou dobu. [18]

Každý dokument se skládá z několika sloupců (polí), které obsahují hodnoty. Hodnota může být jakéhokoliv JSON typu (řetězce, čísla, hodnoty ano/ne, null, pole a objekty). Kromě toho MongoDB podporuje ještě další datové typy: datum, ObjectID, binární data, regulární výraz a kód v JavaScriptu. [18]

MongoDB (většinou) pro každý záznam v kolekci vytváří unikátní řetězec (ObjectID), který slouží jako jednoznačný identifikátor daného záznamu. Tento řetězec je uložen pod identifikátorem `_id`. Jedná se o 12 bytový řetězec složený z data vložení, kódu stroje, kódu procesu a čísla. [18]

4.1.2 Atomicita

Atomické operace jsou v MongoDB podporovány pouze na úrovni jednoho dokumentu v kolekci. MongoDB nepodporuje ACID transakce a nehodí se tedy pro aplikace, pro které je podpora transakcí na více objektech důležitá. [19]

4.1.3 Práce s velkými soubory

MongoDB umožňuje ukládat BSON objekty v maximální velikosti 16 MB. Pokud je potřeba uložit větší soubor, používá se GridFS, což je mechanismus pro rozdělení velkého souboru mezi několik menších dokumentů, které pak vystupují jako jeden celek. [20]

4.1.4 Indexy

S Indexy se v MongoDB pracuje velice podobně jako u relačních databází. Je možné používat indexy na jednom i více sloupcích. Přidání indexu do kolekce zablokuje všechny ostatní operace v databázi, proto dochází-li k doplnění indexů v produkci za plného provozu, je nutné při jeho vytváření specifikovat volbou *background*, která říká, že má vytváření proběhnout na pozadí. Databáze tak zůstane schopna odpovídat příchozím požadavkům, ačkoliv bude po dobu vytváření indexu pracovat s menší odezvou. [19]

4.1.5 Agragace a MapReduce

MongoDb podporuje několik způsobů provádění agregačních operací. Jednodušší operace nad danou kolekcí lze provádět pomocí příkazů `count()`, `distinct()` a `group()`. Druhý způsob je k dispozici od nové verze 2.2 a jde o agregační framework Aggregation Pipeline a příkaz `aggregate()`. Třetí způsob představuje využití konceptu MapReduce. [21]

MapReduce je často používán i u jiných NoSQL databází. Nejprve jsou získány výsledky klasickým dotazem. Ty jsou poté předány v JavaScriptu napsané funkci `map()`, která vytvoří výsledek pro jeden dokument. Všechny výsledky pro agregovaný sloupec jsou předány JavaScriptové funkci `reduce()`, který z nich vytvoří jeden výsledek. [21]

MongoDB umožňuje výstup z MapReduce vložit do RAM, vytvořit s výsledky novou kolekci, popř. sloučit výsledky s existující kolekcí. Poslední zmíněná možnost je vhodná pro kešování obsahu, kdy stačí zavolat MapReduce pouze jednou na nových datech a nový výsledek sloučit s existující kolekcí. [19]

4.1.6 Replikace

MongoDB má velmi dobrou podporu pro replikaci dat, tedy vytváření více kopií databáze. Obvykle je zde jeden uzel, který je primární a umožňuje zápis a čtení a pak je zde několik dalších uzlů, které jsou k dispozici pouze pro čtení. Uživatel pracuje s primárním uzlem a sekundární uzly se automaticky aktualizují, pokud dojde ke změně v primárním uzlu. [17]

V případě, že dojde k pádu u primárního uzlu, některý ze sekundárních uzlů se stane primárním. Je možné také sekundární uzly využít pro čtení dat a rozložit tak zátěž na více uzlů. Dále se replikace hodí při administrátorské práci, jako je například vytváření záloh či aktualizace software. [17]

Databázový systém ve výchozím nastavení podporuje tzv. journaling, který slouží pro obnovu dat po pádu databáze. [22]

4.1.7 Dotazování a aktualizace dat

Systém umožňuje vytvářet vlastní JavaScriptové funkce, které lze pak opakovaně nad databází spouštět. Používá se k tomu příkaz `db.eval()`. Ten však zamkne celou databázi pro čtení i zápis, dokud běh funkce neskončí. [23]

Systém podporuje pro vkládání, aktualizaci a mazání dat podobné možnosti jako klasické SQL databáze. Navíc obsahuje možnosti aktualizace jen některé zanořené části v dokumentu tak, aby nebylo potřeba nejprve záznam vrátet, aktualizovat a zase ho zpět do kolekce vložit. Za zmínku stojí třeba možnost přejmenovat název sloupce nebo přidat prvek do pole. [23]

Při aktualizaci jednoho dokumentu je dostupná volba *upsert*, která vykoná aktualizaci dat v případě, že dokument v databázi již je. V opačném případě je dokument do databáze vložen. MongoDB také umožňuje vložit více dokumentů jedním příkazem. [17]

4.1.8 Optimalizace

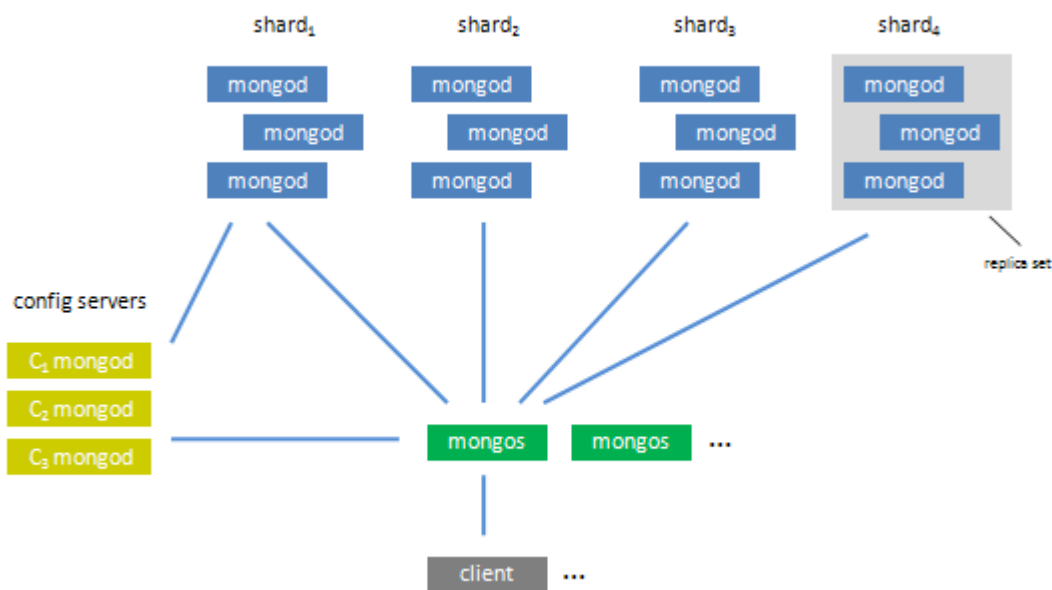
MongoDB obsahuje základní sadu nástrojů pro optimalizaci databáze. Vedle klasických doporučení jako využívat indexy, používat klauzuli `limit` pro získání skutečně potřebných sloupců a specifikování jen později využívaných sloupců, databázový systém také obsahuje profiler a nástroj `explain` známy i v SQL. [19]

Příkaz `explain()` se přidává k existujícímu dotazu a místo výsledků dotazů vrací informace o jeho vykonání, tedy např. počet dokumentů, které bylo nutné zpracovat, dobu zpracování či podrobné informace o použití indexů při běhu dotazu. [19]

Profiler je možné aktivovat pro všechny dotazy nebo pro dotazy trvající déle než je stanovaný limit. Do interní kolekce system.profile jsou následně zaznamenány všechny dotazy, které databáze zpracovala opět s podrobným průběhem zpracování. [19]

4.1.9 Škálovatelnost

MongoDB umožňuje horizontální škálování přes koncept sharding, který umožňuje dělit kolekce dat na menší celky (chunks) uložené na jiném serveru. Jako příklad se často uvádí aplikace podobná sociální síti Twitter, která by měl v jedné kolekci obrovské množství dokumentů. Pomocí shardingu je kolekce automaticky rozdělena na menší části, které se pak mohou spravovat samostatně. [24]



Obrázek 1: sharding (zdroj: [23])

Celá architektura aplikace se skládá z několika částí:

- **Shards** jsou samostatné stroje (servery), které obsahují část dat (chunks). Každý shard obsahuje několik procesů mongod, které obsahují vždy stejná data v rámci shardu. Tvoří tedy replikační set. Pokud dojde k pádu jednoho mongod procesu uvnitř jednoho shardu, stane se ihned jiný shard primárním strojem.
- **Config servers** jsou konfigurační servery, které obsahují informace o tom, která část dat je uložena v jakém shardu. Konfigurační servery obsahují vlastní logiku replikací.
- **Mongos** je proces, který funguje jako router. Poslouchá příchozí požadavky a přeposílá je na konkrétní shardy podle konfiguračních serverů. [24]

Celá práce s rozsáhlou databází je pak jednoduchá. Koncová aplikace má k dispozici stejné rozhraní, ať už pracuje s jednou databází, nebo s rozsáhlým databázovým clusterem. Vše je zautomatizováno. Z pohledu tvůrce aplikace je důležité vybrat správný klíč, podle kterého se bude kolekce dělit na menší části. [19]

4.1.10 Licence

Databáze je distribuovaná pod licencí GNU AGPL v3.0. Databázi je možné zdarma používat na komerčních i nekomerčních projektech. [25]

4.1.11 Rozhraní k programovacím jazykům

Pro MongoDB je k dispozici rozhraní pro všechny populární programovací jazyky. Není problém používat MongoDB v C, C++, Javě, .NET, JavaScriptu (přes Node.js), PHP, Pythonu, Ruby i Scale.

4.1.12 Hostování a cloud

MongoDB je dnes součástí velkého množství hostingů, ať už jde o PaaS (např. Heroku), IaaS (Amazon, Joyent, Rackspace či Windows Azure) nebo DaaS jako MongoHQ či MongoLab.

4.1.13 Komunita, dokumentace

K dispozici je rozsáhlá dokumentace na oficiálních stránkách. Za zmínku také stojí online shell, který obsahuje tutoriál se základními pokyny pro práci s databází. K dispozici je velké množství knih, téměř každý den se konají na různých místech setkání vývojářů pracujících s databází. Na populárním serveru StackOverflow patří komunikace kolem MongoDB mezi nejaktivnější.

4.1.14 Nástroje

Pro MongoDB je k dispozici velké množství nástrojů. Součástí distribuce jsou i nástroje pro export a import dat či vytváření zálohy.

Pro běžnou programátorskou práci s databází existuje několik nástrojů, z nichž se subjektivně nejlépe pracuje s komerčním MongoVUE.

4.1.15 Případové studie

Protože je MongoDB nejpoužívanější NoSQL databází [11], existuje i velké množství případových studií.

MongoDB používá třeba společnost eBay. Je zde využíváno pro nápovědu u vyhledávání (search suggestion), uložení pro metadata či cloud management. [27]

Dalším uživatelem MongoDB je CERN, kde je MongoDB používáno pro interní CMS. MongoDB slouží mimo jiné jako uložení pro dokumenty pro cca 3 tis. vědců. [28]

Zajímavá případová studie se týká společnosti Cisco, která používá MongoDB jako uložení pro real-time data pro WebEx Social, což je Enterprise 2.0 platforma. MongoDB se používá k získávání dat pro různá doporučení či pro generování statistik. Převod platformy z relační databáze na NoSQL databázi MongoDB způsobil rapidní zvýšení výkonu aplikace. Některé dotazy dříve mohly trvat až 30 vteřin, dnes zaberou pouze několik milisekund. [29]

MongoDB dále využívá populární síť Forsquare. Dříve byla postavena na relační databázi, kvůli nutnosti automatického škálování však síť přešla na MongoDB, které využívá doteď. [30]

4.1.16 Ukázka

Práci s databází ukážu na jednoduchém systému pro správu obsahu.

Instalace pod Windows je velmi jednoduchá. Z hlavních stránek databáze uživatel stáhne klasický Windows instalátor, přes který vše potřebné nainstaluje. Po instalaci je potřeba vytvořit složku, kam bude MongoDB ukládat veškerá data. Pak je potřeba spustit program mongod.exe, což je MongoDB daemon.

Celá aplikace si vystačí pouze se dvěma kolekcemi pro články a autory.

- **Articles** obsahuje všechny vydané články. Každý článek obsahuje název, datum, štítky (slouží ke kategorizování), autora, perex a samotný text článků. Kromě toho může obsahovat i komentáře uživatelů (jméno a text).
- **Authors** obsahuje autory příspěvků. Každý má povinně uvedeno jméno (přezdívkou), e-mail a heslo.

Nejprve je potřeba spustit MongoDB daemon (program mongod.exe). Dále je potřeba spustit mongo shell a přihlásit se do databáze. V případě testování databáze na localhostu není potřeba zadávat žádné další parametry, vytvoří se spojení pro localhost na výchozím portu 27017. Všechny další příkazy zadávám již pouze do mongo shellu.

Databázi není potřeba vytvářet, vytvoří se sama při vložení prvního dokumentu do kolekce. Pro přepnutí na danou databázi se používá stejně jako v SQL příkaz use.

```
MongoDB shell version: 2.4.8
connecting to: test
> use vse
switched to db vse
```

Dále vytvořím nového uživatele, čímž se zároveň vytvoří i nová kolekce. Heslo je uloženo jako hash, který vygeneruje aplikace.


```
> var user = {
...   login: "Jakub Mrozek",
...   email: "jakub.mrozek@gmail.com",
...   pass: "6e017b5464f820a6c1bb5e9f6d711a667a80d8ea"
... };
> db.users.insert(user);
```

V tuto chvíli je uživatel úspěšně vložen do databáze. Lze se o tom přesvědčit vypsáním obsahu kolekce uživatelů příkazem `find()`.

```
> db.users.find()
{
  "_id" : ObjectId("52abdad2762a6f99dd13485b"),
  "login" : "Jakub Mrozek",
  "email" : "jakub.mrozek@gmail.com",
  "pass" : "6e017b5464f820a6c1bb5e9f6d711a667a80d8ea"
}
```

Databáze správně vrátila jeden záznam. Je zde také vidět, že byl automaticky vygenerován sloupec `_id` s unikátní hodnotou pro dokument v kolekci.

Bylo by ještě vhodné přidat k uživateli do databáze identifikaci, zda jde o hlavního správce či o obyčejného autora článků, který bude mít nižší úroveň oprávnění. Aktuálně je v kolekci pouze jeden správce, takže je možné spustit dotaz na všechny záznamy bez podmínky `where`.

```
> db.users.update({}, {$set: {"superadmin": true}})
```

Následně vytvořím ještě autory a články.

```
> var user = {  
...   login: 'Tomáš Peterka',  
...   email: 'tom@gmail.com',  
...   pass: '2cf146e1da1240d6fbffec0d91a0ac3994d8c9ff',  
...   superadmin: false  
... };  
> db.users.insert(user);  
  
> var article = {  
...   title: "Programování v PHP",  
...   url: "programovani-v-php",  
...   date: new Date(),  
...   author: "Tomas Peterka",  
...   tags: ["programovani", "php"],  
...   perex: "Lorem ipsum set dolorem",  
...   text: "Lorem ipsum set dolorem"  
... };  
> db.articles.insert(article);
```

Na úvodní stránce chci vybrat posledních 10 článků a chci vybrat pouze titulek, perex a URL.

```
> db.articles.find({}, {title: 1, perex: 1, url:
1}).sort({date: -1}).limit(10);
{
  "_id" : ObjectId("52abdd6b762a6f99dd13485d"),
  "title" : "Programování v PHP",
  "url" : "programovani-v-php",
  "perex" : "Lorem ipsum set dolorem"
}
```

V detailu článku zobrazím vše. Článek vyhledávám podle URL.

```
> db.articles.findOne({url: "programovani-v-php"});
{
  "_id" : ObjectId("52abdd6b762a6f99dd13485d"),
  "title" : "Programování v PHP",
  "url" : "programovani-v-php",
  "date" : ISODate("2013-12-14T04:24:09.517Z"),
  "author" : "Tomas Peterka",
  "tags" : [ "programovani", "php" ],
  "perex" : "Lorem ipsum set dolorem",
  "text" : "Lorem ipsum set dolorem"
}
```

V aplikaci chci také umožnit uživatelům vyhledat mezi vloženými texty. Frázi budu hledat v titulku, perexu a textu. Uživatel zadal např. frázi „programování“.

```
> db.articles.find({"$or": [{title: /programování/i}, {perex: /programování/i}, {text: /programování/i}]})
{
  "_id" : ObjectId("52abdd6b762a6f99dd13485d"),
  ...
  "text" : "Lorem ipsum set dolorem"
}
```

Uživatel může také chtít procházet články s určitým štítkem, např. se štítkem PHP.

```
> db.articles.find({tags: "php"})
{
  "_id" : ObjectId("52abdd6b762a6f99dd13485d"),
  ...
  "text" : "Lorem ipsum set dolorem"
}
```

Nebo je možné procházet záznamy od daného autora.

```
> db.articles.find({author: "Tomas Peterka"});
{
  "_id" : ObjectId("52abdd6b762a6f99dd13485d"),
  ...
  "text" : "Lorem ipsum set dolorem"
}
```

K článku mohou uživatelé přidávat komentáře.

```
> var comment = {  
...   login: "Jakub",  
...   text: "Pěkný článek, jen tak dále"  
... };  
> db.articles.update({url: "programovani-v-php"}, {$push : {  
comments: comment}}  
);
```

Může se stát, že některý článek již nechci v systému mít, takže ho smažu.

```
> db.articles.remove({url: "programovani-v-php"})
```

5 Grafové databáze

Grafové databáze jsou určeny k řešení specifických úloh, které je velmi těžké, ne-li nemožné, provádět v klasických relačních databázích (příklad byl uveden v úvodu). Jsou optimalizovány třeba pro hledání nejkratší cesty v grafu. Používají se často k práci s daty, které mezi sebou obsahují velké množství vazeb.

Struktura grafové databáze se skládá z vrcholů (nodes, vertex) a hran (edges, relationships). Každý vrchol má obvykle několik hran do něj vstupujících a několik hran z něj vystupujících, v databázi však mohou existovat i vrcholy, které žádné vazby mezi sebou nemají. Oproti tomu má hrana vždy jeden vrchol, do kterého vstupuje, a jeden vrchol, ze kterého vystupuje. [31]

Data jsou obvykle ukládána ve vrcholech, není to však pravidlem a např. Neo4j umožňuje stejným způsobem ukládat data i v hranách. [32]

Mezi nepoužívanější grafové databáze patří Neo4j, OrientDB a Titan. [11]

OrientDb je databáze, která kombinuje vlastnosti dokumentově orinetované a grafové databáze. Je napsaná v Javě. Pro přístup k datům podporuje jak Java API, tak také REST API. Podobně jako Neo4j i OrientDB podporuje ACID transakce. [33]

Titan je velmi mladá grafová databáze (první verze byla vydána v roce 2012). Také ona podporuje ACID transakce. Titan podporuje rozšiřitelnou architekturu uložení, je možné databázi doplnit ještě jinými databázemi (třeba Cassandra, HBase, ElasticSearch a další). [34]

5.1 Neo4j

Neo4j je nejpoužívanější grafová databáze na světě [11]. Je optimalizovaná pro vysoký výkon a dostupnost pro aplikace, které potřebují zpracovávat miliony uzlů ve velmi krátkém čase. Na rozdíl od mnoha jiných NoSQL databází se Neo4j může pyšnit podporou ACID transakcí. [35]

Pro dotazování má Neo4j vlastní deklarativní jazyk Cypher, který je inspirován jazykem SPARQL pro manipulaci s daty v RDF a také jazykem SQL používaným u relačních databází. Manuál proto obsahuje sekci určenou speciálně pro vývojáře v SQL. [35]

Neo4j obsahuje vlastní REST API. To může mít velký význam v době moderních webových aplikací, které přesouvají logiku na stranu klienta (webového prohlížeče), který tak může získávat data bez nutnosti zvláštní serverové logiky aplikace. Díky REST API mohou s Neo4j vývojáři pracovat z jakéhokoliv jazyka. [35]

Databáze je vyvíjena v Javě. Proto je i mnoho ukázek v Javě, část manuálu je o propojení Javy a Neo4j, v Javě je hlavní programátorské rozhraní pro komunikaci s Neo4j a v Javě lze i Neo4j dále rozšiřovat. Protože patří Java k nejpoužívanějším programovacím jazykům současnosti, má propojení Javy a Neo4j významný vliv na to, že je Neo4j nejpoužívanější grafovou databází na světě, podobně jako je tomu v případě MongoDB a JavaScriptu. [35]

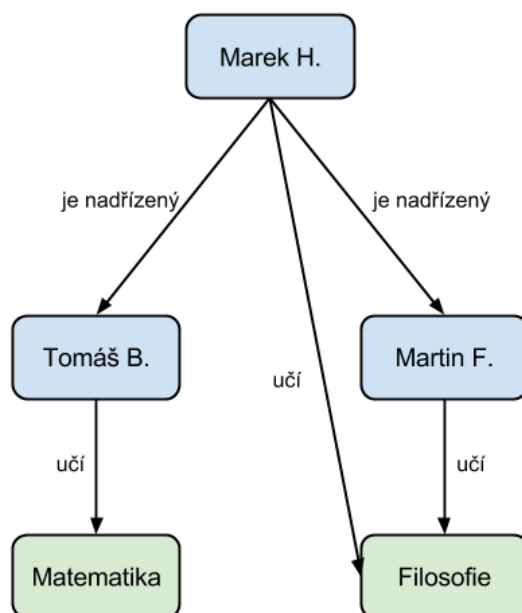
5.1.1 Struktura dat

Všechna data jsou ukládána do grafu, který se v případě Neo4j dělí na 4 základní části: uzly (nodes), vztahy/hrany (relationships), štítky (labels) a vlastnosti (properties). [35]

Uzly se používají nejčastěji k reprezentaci entit a jsou navzájem spojeny pomocí hran. Hran jsou vždy orientované, mají určený směr, ze kterého uzlu vystupují a do kterého vstupují. [35]

Jak uzly, tak hrany mohou mít přiřazeny vlastnosti typu klíč/hodnota. Klíč je vždy řetězec, hodnotou může být jakýkoliv primitivní datový typ v Javě (boolean, int, float atd.) nebo pole složené z primitivních hodnot. Kromě vlastností je možné od verze Neo4j 2.0 sdružovat uzly do skupin pomocí štítků. [35]

Jako příklad lze uvést graf, ve kterém budou jako uzly učitelé a předměty:



Obrázek 2: vzorová struktura databáze (zdroj: autor)

Učitelé budou spojeni s předměty pomocí hran (vyjádření vztahu „učí“), učitelé navzájem však mohou být spojeni také (vyjádření vztahu „je nadřizený“). Učitelé mohou mít přiřazeny různé vlastnosti (jméno, věk, pohlaví), předměty mohou mít vlastnosti také (název, počet kreditů). Pro odlišení uzlů, které jsou učitelé a které jsou předměty, se použijí štítky Učitel a Předmět.

5.1.2 Dotazovací jazyk Cypher

Cypher je speciální dotazovací jazyk pro Neo4j, přes který je možné data získávat nebo aktualizovat. Vzdáleně připomíná jazyk SQL, základem je zde však vzor (pattern) a data se získávají porovnáváním vzorů se strukturou grafu (pattern matching). [35]

Vzory

Vzor se skládá z jednoho či více uzlů a vztahů mezi nimi. Vzor může obsahovat štítky, vlastnosti, typy vztahů mezi uzly, je ale třeba také možné určit minimální a maximální vzdálenost mezi hledanými uzly. [36]

Vytváření vzorů je jednoduché a intuitivní, jak ukazují následující příklady:

Výběr všech učitelů s předměty, ke kterým mají nějaký vztah (třeba ho garantují, vyučují ap.):

```
(u:Ucitel) → (p:Predmet)
```

Výběr vše dvojic nadřízený/podřízený:

```
(nadrizeny:Ucitel) -[:nadrizeny]-> (podrizeny:Ucitel)
```

Výběr všech učitelů, kteří učí matematiku:

```
(u:Ucitel) -[:uci]-> (p:Predmet {nazev: "matematika" })
```

Klauzule

Cypher je podobně jako jazyk SQL tvořen několika klauzulemi. Základ jazyka tvoří klauzule **MATCH**, **START** a **WHERE** pro čtení a **CREATE**, **MATCH**, **SET**, **REMOVE** a **DELETE** pro aktualizaci dat. [36]

Za klauzulí **MATCH** následuje vzor, který bude v grafu hledán. Používá se společně s klauzulí **RETURN**, která definuje, jaké hodnoty má dotaz vrátet. [36]

Zápis pro získání všech předmětů v Cypheru vypadá takto:

```
MATCH (p:Predmet)
RETURN p
```

Podporovány jsou i různé speciální grafové funkce, třeba získání nejkratší cesty mezi dvěma městy lze napsat snadno:

```
MATCH cesta = shortestPath((a:Mesto) → (b:Mesto))
RETURN cesta
```

Pomocí klauzule **WHERE** lze podmínky různě zpřesňovat. [36] Což takhle vybrat všechny učitele, kteří mají nadřizeného staršího 50 let?

```
MATCH (n:Ucitel) -[:nadrizeny]-> (p:Ucitel)
WHERE n.vek >= 50
RETURN p
```

Kromě výše uvedeného lze dále používat:

- **START** pro určení počátku vyhledávání,
- **DISTINCT** pro odstranění duplicit,
- **ORDER BY** pro seřazení výsledků,
- **LIMIT** a **SKIP** pro získání určené podmnožiny z vrácených výsledků,
- **UNION** a **WITH** pro sloučení výsledků několika dotazů. [36]

Pro manipulaci s daty slouží tyto klauzule:

- **CREATE** a **MERGE** pro vytvoření nového uzlu či vztahu,
- **SET** pro úpravy vlastností a štítků,
- **DELETE** pro mazání uzlů a vztahů,
- **REMOVE** pro odstranění vlastností a štítků a
- **FOREACH** pro aplikaci úprav na každý nalezený záznam. [36]

V nové verzi Neo4j přibyla podpora pro indexy (příkaz **CREATE INDEX ON**). V kombinaci s příkazem **USING** je možné Neo4j napovědět, která data je potřeba zpracovávat rychleji. [36]

5.1.3 Transakce

Neo4j podporuje ACID transakce. Všechny transakce drží Neo4j v paměti, což znamená i vyšší nároky na paměť v případě práce s velkými grafy. Proto je doporučeno počítat s dělením velké transakce na několik menších, je-li to v daném kontextu možné. [31]

Neo4j podporuje tzv. vložené (nested) transakce. Vložené transakce vznikají tehdy, jestliže je zde požadavek na běh transakce v kontextu, na kterém již jiná transakce běží. V takovém případě se spustí uvnitř jiné běžící transakce a její výsledky nejsou uloženy na disk, dokud nebude dokončena hlavní transakce. Jestliže vložená transakce selže a vynutí si rollback, pak dojde k rollbacku hlavní i všech vložených transakcí v daném kontextu. [37]

5.1.4 REST API

Server Neo4j podporuje také REST API pro dotazování i manipulaci s daty. HTTP požadavky i odpovědi jsou zasílány ve formátu JSON. Při zasílání dotazů je možné používat Cypher jako při jiných formách dotazování. [35]

Komunikace přes REST API má několik odlišností oproti standardnímu přístupu. Třeba je možné spouštět jedním HTTP požadavkem transakci a druhým ho potvrdit (commit). Je zde stanoven časový limit, po kterém musí dojít k uzavření transakce, jinak Neo4j zavolá rollback automaticky (standardně nastaveno na 60 vteřin). Pokud však z nějakého důvodu limit v určitém případě nestačí, je možné zaslat pod daným ID transakce HTTP požadavek s prázdným polem příkazů, což způsobí vynulování časovače. [38]

Zvláštní možností je vrácení výsledků dotazu v grafovém formátu, který se hodí pro vizualizaci. Pokud je třeba zaslán dotaz na vytvoření několika uzlů a bude požadován grafový formát, budou vráceny všechny informace (id, štítky, vlastnosti) o ovlivněných uzlech i vztazích. [38]

5.1.5 Webové rozhraní

Spolu s Neo4j je dodáváno i webové rozhraní. Přes webové rozhraní je možné procházet data, manipulovat s nimi, monitorovat stav Neo4j serveru. Po instalaci a spuštění je rozhraní k dispozici na adrese <http://127.0.0.1:7474>. [35]

5.1.6 Zálohování

Neo4j podporuje dva druhy zálohování. První způsob (full backup) probíhá formou vytvoření kompletní kopie aktuálně běžící databáze bez nutnosti zamykání dat. Druhý způsob (incremental backup) pracuje s logem transakcí, přičemž jsou zpětně postupně vyvolány všechny operace a dále aplikovány na poslední existující zálohu. [39]

5.1.7 Replikace a škálovatelnost

Neo4j má dobrou podporu pro replikace a horizontální škálovatelnost (v Enterprise verzi). Podobně jako v případě MongoDB je zde jedna hlavní instance (master) a několik vedlejších (slaves). [35]

Na rozdíl od ostatních řešení však umožňuje zapisovat na jakoukoliv vedlejší instanci a není nutné přeměrovat operace měnící data na hlavní instanci, jako je tomu právě v případě MongoDB. Při provádění transakcí z vedlejší instance jsou všechny operace synchronizovány na ostatní instance. Commit je proveden nejprve na hlavní instanci a pokud proběhne vše v pořádku, spustí se commity i na ostatních instancích. [35]

Neo4j je *fault tolerant* a pokud dojde k pádu hlavní instance, je automaticky vybrána jedna z vedlejších instancí za hlavní. Zároveň sleduje nefunkční instanci a ve chvíli, kdy je zase k dispozici, tak ji vrátí zpět do clusteru. [35]

5.1.8 Licence

Neo4j je k dispozici ve 4 základních verzích, které se liší podle toho, kdo ji chce využívat. [34]

Neo4j Community je verze určena jen pro open source projekty a pro tyto účely je k dispozici zdarma. Jde o oseknanou verzi, která neobsahuje možnosti důležité pro provozování vysoce výkonných aplikací jako např. monitorovací nástroje, clustering, online zálohování či vysoce výkonnou keš. [40]

Neo4j Personal je plná verze databáze Neo4j, která je rovněž dostupná zdarma. Platí zde však omezení, že může být využívána pouze firmami, které mají 3 nebo méně zaměstnanců (vč. těch, kteří s firmou spolupracují jinak než na klasický pracovní poměr), firma je založena bez investorů a její roční příjem nepřesahuje 100 tis. dolarů. [40]

Neo4j Startups je rovněž plná databáze Neo4j, která je k dispozici pro firmy s ročním příjmem menším než 5 mil. dolarů a s počáteční investicí menší než 10 mil. dolarů. Pro tyto firmy již však databáze zdarma není a licence k použití Neo4j vyjde na 12 tis. dolarů (cca 250 tis. Kč) za rok. [40]

Neo4j Business & Enterprise je k dispozici pro firmy, které předchozí podmínky nesplňují a cena je pro ně určena individuálně. [40]

5.1.9 Rozhraní k programovacím jazykům

Pro Neo4j je k dispozici rozhraní pro všechny používanější jazyky. Lze najít knihovny pro komunikaci s Neo4j (drivery) pro Javu, Ruby, Node.js, PHP, Python, Perl, Scalu a další.

5.1.10 Komunita, dokumentace

Dokumentace je přehledná, velmi dobře se čte. Za velmi užitečné považuji uživatelské komentáře na každé stránce, což je již dnes u dokumentací naštěstí standard. Za klíčové považuji přidání sekce s množstvím složitějších příkladů přímo do manuálu. Je to věc, která u jiných databází často chybí.

Za špičkové považují videa na hlavní stránce neo4j.org. Je zde množství krátkých videí k různým aspektům databáze, především videa obecně o grafových databázích jsou zdařilá.

Za velmi překvapivou (navzdory licenční politice) považuji velikost komunity, která kolem Neo4j existuje. Pořádají se konference (GraphConnect) a setkání po celém světě (vč ČR), lze říci, že každý den je někde na světě setkání vývojářů Neo4j.

5.1.11 Nástroje

K dispozici jsou také nástroje pro práci s Neo4j, nicméně je jich mnohem méně než třeba v případě MongoDB či Redisu. Za zmínku stojí třeba Neoclipse, studio podobné Eclipse pro Javu. Pak jsou zde nástroje pro vizualizaci uložených grafů, především Linkurious.

5.1.12 Případové studie

Neo4j používá celá řada aplikací z oboru, jako jsou telekomunikace, média, sociální výzkumy či biometrika. Používá se také často pro různé doporučovací systémy. [5]

Francouzská telekomunikační společnost Telecom používá Neo4j pro monitoring jejich sítě. Veškerá infrastruktura je rozkreslená do grafu. Kdykoliv je potřeba třeba kvůli údržbě nějaké místo vypnout, pohledem do grafu je okamžitě jasné, které oblasti a služby budou výpadkem postihnuty. [5]

Neo4j dále používají společnosti jako Adobe, Cisco či H&P. [41]

5.1.13 Ukázka

Instalace je velice jednoduchá. Pro Windows je k dispozici klasický instalátor. Po stažení a instalaci se ihned spustí Neo4j a nabídne přechod na webové rozhraní (<http://localhost:7474>). Zde je možné rovnou do databáze vložit rozsáhlý předpřipravený demo soubor a zkusit jednotlivé operace.

Pro ukázkou databáze poslouží aplikace podobná Facebooku. Databáze bude tvořena uzly se štítkem Clovek reprezentující jednu osobu. Mezi některými osobami může být vazba vyjadřující přátelství.

Nejprve se vytvoří 5 uživatelů:

```
CREATE (n:Clovek {jmeno: "Jakub", pohlavi: "muz", vek: 26});
CREATE (n:Clovek {jmeno: "Monika", pohlavi: "zena", vek: 24});
CREATE (n:Clovek {jmeno: "Petr", pohlavi: "muz", vek: 31});
CREATE (n:Clovek {jmeno: "Jana", pohlavi: "zena", vek: 21});
CREATE (n:Clovek {jmeno: "Honza", pohlavi: "muz", vek: 49});
```

Dále několik vazeb:

```
MATCH (a:Clovek), (b:Clovek)
WHERE a.jmeno = 'Jakub' AND b.jmeno = 'Monika'
CREATE (a)-[r:`jsou_pratele`]->(b)

MATCH (a:Clovek), (b:Clovek)
WHERE a.jmeno = 'Jakub' AND b.jmeno = 'Petr'
CREATE (a)-[r:`jsou_pratele`]->(b)

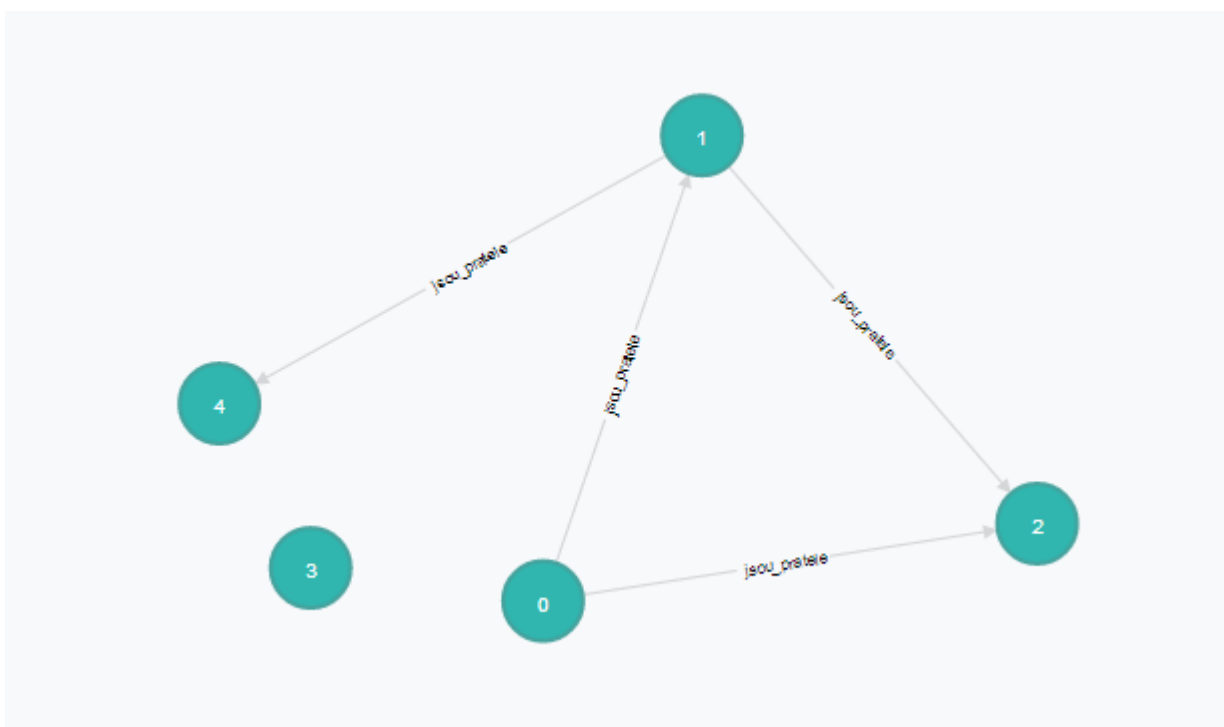
MATCH (a:Clovek), (b:Clovek)
WHERE a.jmeno = 'Monika' AND b.jmeno = 'Petr'
CREATE (a)-[r:`jsou_pratele`]->(b)

MATCH (a:Clovek), (b:Clovek)
WHERE a.jmeno = 'Monika' AND b.jmeno = 'Honza'
CREATE (a)-[r:`jsou_pratele`]->(b)
```


Nejprve budu chtít vybrat všechny osoby v databázi:

```
MATCH (a:Clovek)
RETURN a
```

Webové rozhraní v tomto případě ukazuje rovnou vytvořený graf, proto výstup zadaného dotazu vypadá takto:




Obrázek 3: výstup zadaného dotazu ve formě grafu (zdroj: autor)



Dále bych chtěl znát všechny dvojice přátel, použiji tedy tento dotaz:

```
MATCH (a)-[p:`jsou_pratele`]->(b)
RETURN a.jmeno, b.jmeno
```

Webové rozhraní k Neo4j umí zobrazit místo grafu i tabulku, což se výborně hodí právě pro tento dotaz, takže výsledek může vypadat takto:

```
CYPHER MATCH (a)-[p:jsou_pratele]->(b) RETURN a.jmeno, b.jmeno
```

a.jmeno	b.jmeno	
Jakub	Monika	
Jakub	Petr	
Monika	Petr	
Monika	Honza	

✓ Returned 4 rows in 141 ms  

Obrázek 4: výstup zadaného dotazu ve formě tabulky (zdroj: autor)

Dále mě bude zajímat osoba jménem Jakub. A zajímají mě jména všech přátel jeho přátel.

```
MATCH (a)-->( )-->(b)
WHERE a.jmeno = "Jakub"
RETURN b.jmeno
```

V tomto případě budou vráceny dva záznamy: Petr a Honza. Jakub má totiž mezi přáteli Moniku, která má mezi přáteli právě Petra a Honzu.

Dále bych chtěl zjistit, zda má Jakub a Monika nějaké přátele společné.

```
MATCH (a)-->(b)
WHERE a.jmeno = "Jakub"
WITH b
MATCH (c)-->(b)
WHERE c.jmeno = "Monika"
RETURN b.jmeno
```

Tento dotaz nejprve vybere přátele Jakuba a pak se ptá, zda má Monika tyto přátele také. Dotaz lze však položit ještě jednodušším způsobem:

```
MATCH (a)-->(b)<--(c)
WHERE a.jmeno = "Jakub"
AND c.jmeno = "Monika"
RETURN b.jmeno
```

Oba dotazy vrátí jen jeden výsledek, a to Petra, který je skutečně společným přítelem obou osob.

6 Databáze typu klíč/hodnota

Databáze typu klíč/hodnota patří k databázím, které mají nejjednodušší strukturu dat. Obvykle je zde jeden unikátní řetězec jako klíč a hodnota, jejíž formát se výrazně liší napříč různými systémy. Tyto databáze se obvykle nepoužívají jako hlavní databáze uvnitř nějaké aplikace, ale spíše zajišťují specifickou funkcionalitu, pro kterou jsou výborně optimalizovány. Velmi často se třeba používají pro ukládání a načítání kešovaných dat. [42]

Pod databáze typu klíč/hodnota se často zařazují i jako zvláštní podtyp sloupcové databáze. Oproti klasickým klíč/hodnota databázím pod daný klíč ukládají množství dvojic sloupec/hodnota.

Mezi nejznámější představitele databází typu klíč/hodnota patří Redis, Memcached a Riak. Nejpoužívanější sloupcovou databází je Cassandra. [11]

Memcached je databáze držící data v paměti (in-memory database), která byla speciálně navržena pro ukládání kešovaných dat. [43]

Riak je distribuovaná databáze napsána v Erlangu. Pro dotazování podporuje REST API, podporuje však i nativní rozhraní pro Erlang. Umožňuje spouštění server-side skriptů (úložných procedur) v jazyce Erlang nebo JavaScript. [44]

Cassandra patří mezi 10 nejpoužívanějších databází a po MongoDB se jedná o druhou nejpoužívanější NoSQL databázi. Byla vytvořena ve společnosti Facebook a je určena pro aplikace, které potřebují zpracovávat velké množství dat. [45]

6.1 Redis

Redis patří mezi nejznámější databáze typu klíč/hodnota [11]. Velmi často se používá jako uložení pro identifikátory relací nebo pro ukládání kešovaných dat. Použije se třeba pro uložení výsledku SQL dotazu, který je velmi náročný na výpočetní čas.

Redis oproti některým jiným klíč/hodnota databázím nenabízí jen primitivní ukládání řetězce k zadanému klíči. Naopak, umožňuje ukládat i složitější struktury jako jsou seznamy či mapy hodnot.

Hlavní odlišnost oproti jiným zástupcům v kategorii databází typu klíč/hodnota je v tom, že Redis drží data v paměti. To má zásadní vliv na výkon, neboť přístup k datům v paměti bývá řádově rychlejší než přístup k datům uloženým na disku. Zároveň však Redis umožňuje ukládání dat na disk, pokud tedy dojde k pádu databáze, neznamená to, že dojde ke ztrátě všech dat. [46]

Kromě výše zmíněného podporuje Redis také transakční zpracování. Dále umožňuje nastavit klíčům omezenou dobu životnosti, po které se samy smažou (resp. nejsou již dostupné), což se právě výborně hodí při ukládání kešovaných dat, která musí mít nastavenou dobu validity. Redis také podporuje příkaz eval, pomocí kterého je možné spouštět nad databází skripty napsané v jazyce Lua. Tyto i další vlastnosti budou podrobněji rozebrány v následujícím textu. [46]

6.1.1 Struktura dat

Redis podporuje 5 datových typů: řetězce (strings), seznamy (lists), množiny (sets), tříděné množiny (sorted sets) a mapy (hashes). [46]

Redis nepodporuje koncept tabulek jako relační databáze, vše se ukládá do jednoho společného prostoru. Tabulky se simulují prefixováním klíče. [46]

Řetězce

Řetězcem se rozumí posloupnost znaků do maximální velikosti 512 megabytů. S řetězci je možné provádět základní operace jako přidávání řetězce k některému již uloženému (příkaz APPEND), získávání zadané části uloženého řetězce (příkaz GETRANGE) ap. [46]

Redis navíc umožňuje provádět některé matematické operace i s řetězci. Konkrétně jde o atomické zvýšení či snížení uložené hodnoty o zadanou hodnotu (příkazy INCR, INCRBY, DECR, DECRBY, INCRBYFLOAT). Pokud je takový příkaz nad databází zavolán, pak se Redis pokusí o konverzi řetězce na číslo, nad ním provede příslušnou operaci a číslo opět uloží jako řetězec. [46]

Velmi užitečné jsou bitové operace s řetězci, které Redis také podporuje. Chandra Patni popisuje, jakým způsobem pomocí bitových operací v Redis mohou získávat v reálném čase za méně než 50 ms informace o návštěvnících při simulaci 128 milionů aktivních návštěvníků. [47]

Seznamy

Seznam je kolekce řetězců, které jsou seřazeny podle pořadí, v jakém byly vloženy. Je možné přidat nový řetězec na začátek (příkaz LPUSH) nebo na konec seznamu (příkaz RPUSH). [46]

Seznamy je velmi vhodné použít tehdy, pokud je potřeba číst data převážně ze začátku či konce seznamu (pro tyto operace jsou seznamy optimalizované) a naopak nevhodné tehdy, pokud je potřeba přistupovat k prvkům, které jsou uprostřed. Seznamy je možné třeba využít pro implementaci chatu nebo zobrazování zpráv v aplikacích typu Twitter, kde uživatelé zobrazují téměř výhradně jen naposledy vložené zprávy. [46]

Množiny

Pod pojmem množiny se rozumí kolekce řetězců, které jsou neseřazené. Jsou vysoce optimalizované pro přidávání, odstraňování, testování existence a další operace s prvky v dané kolekci. Přestože může mít množina miliony prvků, přístup k libovolnému probíhá v konstantním čase. Specifikum množin je v tom, že mohou obsahovat pouze unikátní prvky. [46]

Množiny obsahují několik užitečných příkazů. Je možné třeba hledat rozdíl mezi množinami (příkaz SDIFF), průnik (příkaz SINTER) nebo množiny sloučit dohromady (SUNION), přesouvat prvky mezi množinami (příkaz SMOVE) a získávání náhodného prvku z množiny (SRANDMEMBER). [46]

Množiny je možné využít třeba u štítkování v redakčním systému. Pokud redaktor vloží dvakrát stejný štítek k nějakému článku, v dané množině bude pouze jednou. [46]

Zajímavá možnost je zmíněna v [2], kde se množiny používají při implementaci sociální sítě. Do množiny se uloží přátelé dané osoby. Pak lze porovnávat přátele dvou osob a získat rychle jejich společné přátele. [46]

Tříděné množiny

Tříděné množiny se odlišují od klasických množin v tom, že se u prvků zadává skóre (číselná hodnota), podle kterého jsou pak záznamy v množině seřazeny. Tříděné množiny rozšiřují klasické množiny o další možnosti, třeba je možné vybírat jen množiny v určeném intervalu skóre. [46]

Tříděné množiny mají rozsáhlé možnosti využití. Například je-li potřeba získat nějakou množinu s osobami velmi často a v pořadí podle data narození, pak lze jako skóre použít unix timestamp jejich data narození. [46]

Mapy

Nejkomplexnějším datovým typem jsou mapy. Ty se nejčastěji používají k ukládání objektů. Jedná se o kolekci, kde jsou pod daným klíčem uloženy vždy dvojice pole a hodnota. [46]

Používání map místo ukládání hodnot objektů do několika jiných datových typů má zásadní vliv na spotřebu paměti. Proto je doporučeno používat mapy kdekoliv je to jen možné. [46]

6.1.2 Transakce

Redis sice podporuje transakce, avšak nikoliv na také úrovni jako klasické SQL databáze. [46]

U SQL databází jsou transakce zajištěny příkazy BEGIN, COMMIT a ROLLBACK. Příkazem BEGIN se transakce zahajuje, následují příkazy, které mají být provedeny buď všechny, nebo žádný. Celý tato sekvence je zakončena příkazem COMMIT, který transakci uzavře. V případě, že v průběhu transakce nějaký příkaz selže, zavolá se příkaz ROLLBACK, který vrátí databázi do stavu před transakcí. [48]

Redis zahajuje transakci příkazem MULTI. Za ním následuje zadání dalších příkazů, které se však okamžitě neaplikují, ale pouze se ukládají do fronty. Na konci následuje zavolání příkazu EXEC a až po něm jsou všechny příkazy postupně zpracovány. [48]

Příklad zvýšení dvou hodnot:

```
MULTI
INCR klic1
INCR klic2
EXEC
```


Redis deklaruje, že příkazy uvnitř jedné transakce budou provedeny najednou a že nemůže být jiný požadavek zpracován v průběhu probíhající transakce. Také deklaruje, že všechny příkazy uvnitř transakce se provedou najednou nebo vůbec. Zde je však jedna zásadní odlišnost od klasických SQL databází. Redis kontroluje pouze, zda jsou všechny příkazy po volání MULTI zadány správně a pokud některý zadán správně není (třeba špatný počet parametrů příkazu), transakce selže. Neselže však nikdy, nastane-li chyba uvnitř bloku EXEC, tedy pokud některý z příkazů v bloku EXEC selže, ostatní příkazy budou dále zpracovány. [48]

Redis nepodporuje nic jako ROLLBACK v případě SQL databází. Vzhledem k tomu, jak rychlý Redis je a pro jaké řešení problémů se používá, nemusí být absence plnohodnotných transakcí zásadní problém pro dané řešení.

6.1.3 Expirace klíčů

Jedna ze zásadních vlastností databáze Redis je podpora expirace klíčů (u všech datových struktur). U každého klíče je možné zadat délku jeho životnosti a po její uplynutí bude automaticky z databáze odstraněn. To se výborně hodí právě pro implementaci relací, které mají dobu trvání omezenou, nebo pro ukládání kešovaných výsledků, které mají také omezenou dobu platnosti. [46]

Nastavení expirace se provádí příkazem EXPIRE se zadáním kolik vteřin má klíč existovat než bude smazán. Alternativně je možné zadat EXPIREAT jako parametr uvést unix timestamp, kdy má být klíč smazán. Životnost je vrácena pomocí příkazu TTL (time-to-live). [46]

```
SET klic "hodnota"  
EXPIRE klic 30  
TTL klic
```

Redis obsahuje možnost označit danou instanci jako uložisko pro kešovaná data. V takovém případě není potřeba zadávat pro klíče expirace, ale je potřeba vymezit paměť pro danou instanci a Redis bude automaticky mazat staré záznamy, pokud se již do keše nevejdou. [46]

6.1.4 Publish/Subscribe

Redis implementuje několik příkazů, které umožňují používání vzoru publish/subscribe. Příkazem SUBSCRIBE se uživatel přihlásí k zasílání zpráv z určeného kanálu (kolekce zpráv). Kdykoliv dojde k přidání nové zprávy do kanálu, přihlášený uživatel okamžitě novou zprávu získá. Novou zprávu zašle jiný uživatel příkazem PUBLISH. [40]

Vzor publish/subscribe lze využít na různých místech, vyloženě se nabízí třeba implementace chatu. [46]

6.1.5 Skriptování v Lua

Podobně jako mnoho jiných databází, i Redis podporuje volání vlastních skriptů nad instancí databáze. Všechny skripty musí být napsány v jazyce Lua. Skript se volá příkazem EVAL a jako první parametr získává kód skriptu. [49]

Jako nejjednodušší příklad lze uvést vložení nového řetězce „retezec“ pod klíč „klic“:

```
redis 127.0.0.1:6379[1]> EVAL "return redis.call('SET', 'klic',  
'retezec')" 0  
OK  
redis 127.0.0.1:6379[1]> GET klic  
"retezec"  
redis 127.0.0.1:6379[1]>
```

6.1.6 Dělení ukládaných dat

Při práci na velkých projektech je obvykle potřeba použít více instancí Redisu, které budou spravovat jen určenou část dat. V případě Redisu se nabízí několik řešení, použití každého z nich však znamená omezení možností, které Redis standardně nabízí (např. není možné spočítat rozdíl dvou množin, pokud je každá v jiné instanci ap.). [50]

První možnost je správa dělení dat na straně klienta. Klient může mít svůj algoritmus, který rozhoduje o tom, která data pošle na kterou instanci. Platí to třeba pro redis-rb, knihovnu pro komunikaci s Redisem pro programovací jazyk Ruby. [50]

Další možnost představuje Redis Cluster. V době psaní práce však nebyl ještě doporučen pro produkční prostředí. V budoucnu ale půjde o preferovanou možnost řešení problému. [50]

Třetí a v současné době preferovanou variantou pro dělení dat je Twemproxy vyvinutý pro Twitter. Jedná se o proxy postavenou mezi klienta a jednotlivé instance Redisu. Klient tedy nekomunikuje přímo s Redisem, nýbrž z Twemproxy, který se stará o přeposílání požadavků na konkrétní instanci Redisu. [50]

6.1.7 Replikace

Redis obsahuje vestavěnou podporu pro master-slave replikaci, tedy vytváření přesných kopií hlavní instance databáze. Replikace jsou asynchronní a neblokující, tzn., že je možné standardně pracovat s databází, zatímco jsou data synchronizována s jiným uzlem. [46]

Replikace na Redisu se hodí jak pro redundanci dat, tak také pro škálovatelnost. Například složitější výpočetní operace mohou proběhnout bokem na kopii hlavní instance, zatímco hlavní instance není výpočtem vůbec zatížena. [46]

Replikace se spravuje pomocí příkazu SYNC a SLAVEOF pro určení dané instance jako hlavní (master) nebo jako vedlejší (slave). [46]

Pro vysokou dostupnost je také vyvíjen Redis Sentinel, což je systém pro správu několika běžících instancí Redisu. Používá se pro monitorování, notifikaci správců v případě chyb a především pro automatické přepnutí z mastera na slave v případě, že master nepracuje správně. V době psaní práce byl Redis Sentinel ve vývoji a nedoporučován pro produkční prostředí, proto se jím práce dále nezabývá. [51]

6.1.8 Perzistence, zálohování dat

Redis podporuje dva druhy ukládání dat na disk: RDB a AOF. [51]

RDB funguje podobně jako u jiných databází. Je možné nastavit určený interval, po jehož uplynutí se vždy aktuální kopie databáze uloží na disk. Hodí se pro obnovu dat po katastrofě a pro klasické zálohování. Naopak se nehodí tehdy, nemůžeme-li si dovolit ani minimální ztrátu dat od poslední zálohování. Pokud je třeba zálohování nastaveno na každou celou hodinu a dojde k pádu databáze 5 minut před dalším plánovaným zálohováním, dojde k ztrátě všech uložených dat za posledních 55 minut. [51]

AOF persistence funguje jinak. Jakákoliv operace, která ovlivňuje data v databázi (tedy mimo operací čtení dat) je zapsána do logu. V případě obnovy dat jsou pak všechny příkazy zpětně vyvolány a je sestavena databáze v poslední známé podobě. Nevýhodou je, že AOF log bývá obvykle větší než samotná databáze (vyvolání 1000 operací znamená zapsání 1000 řádků do logu) a obnova dat pomocí AOF je pomalejší než v případě RDB. [51]

6.1.9 Licence

Databáze je distribuovaná pod licencí BSD. Databázi je možné používat na komerčních i nekomerčních projektech. [53]

6.1.10 Rozhraní k programovacím jazykům

Pro Redis je k dispozici velké množství rozhraní pro všechny populární programovací jazyky. K dispozici je také mnoho různých nástrojů (viz dále).

6.1.11 Komunita, dokumentace

Komunita kolem Redisu je také velmi aktivní, i když ne tolik jako v případě MongoDB. Dotazy je doporučeno pokládat na StackOverflow, pokročilejší pak na mailing listu. V San Franciscu a Londýně se pravidelně pořádají setkání uživatelů a správců Redisu.

Za výbornou lze považovat dokumentaci. Je velmi přehledná, jednoduchá a kompletní, často doplněna o uživatelské komentáře. Podobně jako v případě MongoDB je i zde k dispozici interaktivní tutoriál, přes který se uživatel naučí základy práce s Redisem během několika minut.

6.1.12 Nástroje

Pro Redis je k dispozici velké množství nástrojů a doplňků, mezi ty nejvýznamnější patří například Redmon (webové rozhraní pro správu a monitoring běžících instancí Redisu), Webdis (HTTP rozhraní pro Redis), FnordMetric (real-time event tracking aplikace) či Sidekiq (systém pro správu procesů).

6.1.13 Případové studie

Redis používá velké množství aplikací. Poměrně známým uživatelem Redisu je Twitter, který pro Redis vytváří některá rozšíření. S pomocí Redisu umožňuje Twitter vytvářet platformu, která obsluhuje až 300 tis. tweetů za vteřinu. [54]

Redis používá dále Github. Používá se zde pro informace o routování a jako uložisko pro konfigurační data. [55]

Zásadní význam má Redis pro Instagram, nejpulárnější uložště fotografií, nyní pod správou sociální síte Facebook. Zde se Redis používá pro funkce activity feed a main feed nebo třeba pro práce s relacemi. [56]

Redis dále používá třeba Flickr od Yahoo!, další aplikace pro správu fotografií. Konkrétně jsou zde používány seznamy jako fronta úkolů. [57]

Zcela zásadní význam má Redis pro StackOverflow, pokročilé diskuzní fórum pro mnoho různých oborů. Kešování dat je zde velmi důležité a propracované a je postaveno téměř výhradně na databázi Redis. [58]

6.1.14 Ukázka

Ukázka práce s Redisem bude na implementaci relací, pro což se Redis používá velmi často. Když se uživatel přihlásí, vytvoří se nový identifikátor (náhodně vygenerovaný řetězec), ke kterému se přiřadí ID uživatele. Identifikátor bude uložen v cookie na straně prohlížeče a bude zasílán s každým požadavkem.

Redis se doporučuje používat na unixových systémech a není zde oficiální vývojová větev pro Window. Nicméně existuje neoficiální větev spravovaná Microsoft Open Tech, kterou lze pro vývoj výborně používat. Instalace na Windows je pak triviální, stačí rozbalit stažený archiv a spustit server programem redis-server.exe a klienta programem redis-cli.exe.

Nejprve tedy bude potřeba spustit server (redis-server.exe) a klienta (redis-cli.exe). Při prvním spuštění je předvybraná výchozí databáze, pokud je potřeba změnit databázi, složí k tomu příkaz SELECT za kterým následuje index databáze:

```
redis 127.0.0.1:6379> SELECT 1
OK
redis 127.0.0.1:6379[1]>
```

Pro ukládání relací si lze vystačit s datovým typem řetězec. Jako klíč se bude ukládat náhodně vygenerovaný řetězec (hash, řeší aplikace) a jako hodnotu ID uživatele. Takže pokud se uživatel úspěšně přihlásí, zapíše se vznik nové relace (klíč 123, ID 13, platnost 3600 vteřin):

```
redis 127.0.0.1:6379[1]> SET 123 13 EX 3600
OK
redis 127.0.0.1:6379[1]>
```

Kdykoliv pak uživatel bude přistupovat do oblasti, kde je vyžadováno přihlášení, je potřeba ověřit, že je uživatel stále přihlášen a bude potřeba vrátit jeho ID:

```
redis 127.0.0.1:6379[1]> GET 123
"13"
redis 127.0.0.1:6379[1]>
```

Pokud stačí získat informaci, zda je uživatel přihlášen a není potřeba vracet přímo jeho ID, stačí příkaz EXISTS:

```
redis 127.0.0.1:6379[1]> EXISTS 123
(integer) 1
redis 127.0.0.1:6379[1]>
```

Uživatel se také může z aplikace odhlásit a v takovém případě je potřeba relaci z databáze smazat:

```
redis 127.0.0.1:6379[1]> DEL 123
(integer) 1
redis 127.0.0.1:6379[1]>
```

Uživatele však může odhlásit sám Redis, pokud dojde k určené expirace klíče. Pokud je potřeba zjistit, jak dlouho ještě uživatel bude přihlášen, lze použít příkaz TTL, který vrací počet vteřin, dokdy je klíč ještě validní:

```
redis 127.0.0.1:6379[1]> TTL 123
(integer) 3599
redis 127.0.0.1:6379[1]>
```


7 Srovnání NoSQL databází

Praktická část bakalářské práce obsahuje srovnání nejpoužívanějších databází v dané kategorii. Po prostudování teoretické části by již čtenář měl být rozhodnut, který typ NoSQL databáze pro svůj projekt vyžije a měl by znát jeho zástupce. Z praktické ukázky by měl mít také představu o tom, jak se s danou databází pracuje z pohledu správce či programátora.

Tato část by mu měla pomoci s výběrem konkrétního zástupce na základě srovnání několika základních parametrů. Pro srovnání byly použity pouze databáze, které jsou v produkci úspěšně nasazeny významnými firmami. V době psaní bakalářské práce bylo evidováno více než 150 NoSQL databází [64], řada z nich je však pouze ve stádiu vývoje či jsou k dispozici ve svých prvotních verzích. Byť řada z nich slibuje zajímavé možnosti, jejich nasazení lze doporučit jen tomu, kdo je ochoten obětovat čas na řešení problémů vycházejících z minimálních zkušeností z ostrého provozu s touto databází.

Databáze jsou vždy srovnány po kategoriích. Několik mých předchůdců srovnávalo NoSQL databáze napříč všemi kategoriemi. To však nepovažuji za správné, protože je-li každý typ databáze určen pro jiné řešení problémů, pak lze očekávat, že se i zkoumané parametry budou lišit minimálně ve váze důležitosti. Třeba srovnání databáze Memcached s Neo4j nemá vůbec žádný praktický význam, neboť si lze jen těžko představit, že by se někdo rozhodoval mezi těmito dvěma databázemi.

7.1 Zkoumané parametry

Pro porovnání bylo použito několik parametrů, v dalším textu je jejich výběr zdůvodněn.

7.1.1 Komunita a dokumentace

Tento parametr považuji za jeden z nejdůležitějších. Existuje zde obrovské množství databází, které nabízejí i zajímavější možnosti než databáze nejčastěji používané. Nicméně jakmile nastane problém, je potřeba najít rychle řešení a v případě velké komunity uživatelů se řešení obvykle najde snáze. K dispozici je pak také větší množství tutoriálů, článků na různá témata ap.

7.1.2 Transakce

Pro velké množství aplikací je podpora ACID transakcí životně důležité. U jiných aplikací naopak jejich absence neznamena zásadní překážku k použití.

7.1.3 REST API

Pro moderní HTML5 aplikace je existence REST API vítaným prvkem. Výhoda je také v tom, že s databází nabízející REST API je možné komunikovat ze všech jazyků. Na druhou stranu, pokud databáze obsahuje jen REST API, je obvykle celý proces zpracování na straně aplikace časově náročnější.

7.1.4 Nativní rozhraní, podpora programovacích jazyků

Vývojářská firma pravděpodobně nepřejde kvůli databázi na jinou platformu či programovací jazyk, proto je podpora jimi používaných programovacích jazyků velmi důležitá.

7.1.5 Serverové skriptování

Pro řadu složitějších úloh bude důležité vědět, zda je možné spouštět vlastní procedury.

7.1.6 Cena a licence

Cena je jeden ze zásadních parametrů. Mnohé databáze mají různá omezení, neumožní třeba využití všech možných funkcí zdarma vždy nebo je jejich použití omezeno pro komerční využití. Najde také jen o cenu samotné licence k použití, ale také cenu za nástroje či hostování databáze, které se také výrazně liší.

7.1.7 Nezahrnuté parametry

Ve srovnání nebyla zahrnuta rychlost zpracování dotazů. Byť jde o velmi zásadní parametr, je bohužel nesmírně obtížné dojít k jednoznačnému závěru a prohlásit, že je daná databáze pomalejší či rychlejší. Takový výsledek by záležel na platformě, na pokládaném dotazu a mnoha dalších parametrech. Je-li při výběru databáze rychlost zcela zásadní parametr, pak doporučuji vytvořit jednoduchý prototyp aplikace a vyzkoušet na ní uvažovanou aplikaci.

Další významný parametr nezahrnutý do srovnání je škálovatelnost. Srovnání možností škálovatelnosti různých NoSQL databází by vydalo na samostatnou, velmi obsáhlou práci. Jde také o parametr, který bude důležitý především pro velké aplikace a pro kvalifikované rozhodnutí bude nutné prozkoumat mnoho dalších zdrojů.

7.2 Srovnání dokumentově orientovaných databází

Tabulka 1: srovnání dokumentově orientovaných databází (zdroj: autor)

Parametr / databáze	MongoDB	CouchDB	RavenDB
Komunita a dokumentace	výborná, online shell, podrobná dokumentace, množství tutoriálu, velmi aktivní komunita	dobrá, subjektivně horší dokumentace než u konkurence, výrazně slabší aktivita komunity třeba na StackOverflow (cca 6x méně dotazů než u MongoDB)	podprůměrná, subjektivně horší dokumentace než u konkurence, velmi slabá komunita (aktivita na StackOverflow 2x menší než v případě CouchDB).
ACID transakce	NE	NE	ANO
REST API	NE	ANO	ANO
Podpora jazyků	mnoho	mnoho	oficiálně jen platforma .NET
Serverové skriptování	ANO, v JavaScriptu	ANO, v JavaScriptu	ano, na platformě .NET
Cena a licence	svobodná, provozovat lze zdarma i pro komerční projekty	svobodná, provozovat lze zdarma i pro komerční projekty	placená pro komerční účely, cena začíná na \$399 za rok

7.2.1 Zhodnocení

Všechny tři zkoumané databáze jsou vhodnými kandidáty na produkční nasazení.

Pro běžné aplikace doporučuji využít MongoDB kvůli strmé křivce učení, výborné dokumentaci a aktivní komunitě. Významný může být její přínos u aplikací, kde je potřeba ukládat hierarchická data, jejich rozložení do relační struktury by bylo v případě relačních databází pracnější. Výborný je přístup přes JavaScript. Nehodí se pro aplikace, pro které jsou důležité transakce. Diskutabilní je její použití u HTML5 aplikací kvůli absenci REST API, na druhou stranu zde existuje několik cloud hostingů (např. Heroku), které umožňují s MongoDB přes REST API komunikovat.

CouchDB je vhodným kandidátem pro klasické HTML5 aplikace díky komunikaci přes REST API. Navíc jsou zde projekty typu PouchDB, které nabízejí automatickou synchronizaci mezi databází u klienta a vzdálenou CouchDB databází, což usnadňuje vývoj aplikací, které potřebují pracovat i offline. Nehodí se pro aplikace, pro které jsou klíčové ACID transakce napříč kolekcemi.

RavenDB doporučuji použít jen u vývoje aplikace na platformě .NET a jen tehdy, existuje-li zde přímý požadavek na podporu ACID transakcí. Je potřeba počítat s vyššími náklady u pořízení licence.

7.3 Srovnání grafových databází

Tabulka 2: srovnání grafových databází (zdroj: autor)

Parametr / databáze	Neo4j	OrientDB	Titan
Komunita a dokumentace	velmi dobrá, dokumentace je přehledná, k dispozici je množství tutoriálů, aktivní komunita	špatná, dokumentaci lze označit za dobrou, přehlednou a kompletní, avšak komunita je mnohonásobně menší než v případě Neo4j	špatná, dokumentace je přehledná, komunita je i zde mnohonásobně menší než v případě Neo4j
ACID transakce	ANO	ANO	ANO
REST API	ANO	ANO	ANO, skrze doplněk Rexter
Podpora jazyků	mnoho	mnoho	Python, Java, Clojure
Serverové skriptování	ANO, v Javě	ANO, v Javě a JavaScriptu	ANO, v Javě
Cena a licence	zdarma pro malé společnosti, placená pro společnost s 4 a více zaměstnanci, cena začíná na \$12000/rok	svobodná, provozovat lze zdarma i pro komerční projekty	svobodná, provozovat lze zdarma i pro komerční projekty

7.3.1 Zhodnocení

Všechny tři zkoumané databáze jsou vhodnými kandidáty na produkční nasazení.

Pro profesionální aplikace doporučuji využít Neo4j. Jde o špičkovou databázi s množstvím materiálů a postupů k řešení grafových úloh. Za výborný považuji dotazovací jazyk Cypher, který se lze naučit velmi snadno. Dobře se databáze kombinuje s relačními databázemi. Zásadním omezením Neo4j je však velmi vysoká cena, která se navíc každý rok mění [65]. Pokud však cena není zásadní překážkou, pak jednoznačně doporučuji Neo4j.

OrientDb je kombinace grafové a dokumentově orientované databáze. Podporuje dokonce SQL s vlastním rozšířením a nabízí mnoho velmi zajímavých možností. Jako nevýhodu vidím její mládí, při zkoumání jejího vývoje jsem také narazil na nezvykle vysoké množství reportovaných chyb a v commit logu časté „hotfixy“. Její nasazení do projektu bude nejspíš provázet časté řešení a obcházení chyb. Přesto ji považuji za vhodné řešení pro nízkonákladové projekty a pro startupy, které potřebují řešit grafové úlohy.

Titan je grafovou distribuovanou databází, která využívá jako uložisko jiné databáze, podpora je zde např. Cassandra či HBase, fulltextové vyhledávání je zajištěno třeba pomocí Elasticsearch. Od začátku je stavěna pro nasazení na projekty, které potřebují pracovat s miliardami záznamů. Proto ji spíše doporučuji pro velké projekty, kde není možné z nějakého důvodu využít Neo4j.

7.4 Srovnání databází typu klíč/hodnota

Tabulka3: srovnání databází typu klíč/hodnota (zdroj: autor)

Parametr / databáze	Redis	Memcached	Riak
Komunita a dokumentace	výborná, velmi přehledná, doplněna o uživatelské komentáře, velká komunita	dobrá, doplněná o uživatelské komentáře	velmi dobrá dokumentace, komunita je ovšem menší
ACID transakce	NE	NE	NE
REST API	NE	NE	ANO
Podpora jazyků	mnoho	mnoho	mnoho
Serverové skriptování	Lua	NE	JavaScript, Erlang
Cena a licence	zdarma i pro komerční využití	zdarma i pro komerční využití	zdarma i pro komerční využití

7.4.1 Zhodnocení

Všechny tři zkoumané databáze jsou vhodnými kandidáty na produkční nasazení.

Pokud je potřeba jednoduché uložení pro keš, pak je Memcached pravděpodobně nejlepším řešením. Jde o systém přímo navržený pro správu kešovaných souborů.

Má-li uložení umět trochu více než jen ukládat a získávat řetězce, pak rozhodně doporučuji nasazení databáze Redis. Kromě uložení keše se používá třeba na ukládání konfiguračního nastavení aplikace, velmi často také na správu relací.

Riak se naopak může hodit pro HTML5 aplikace díky podpoře REST API. Oproti Redisu je lépe navržen pro distribuovaná prostředí.

8 Závěr

V práci byly představeny tři druhy NoSQL databází, které se používají nejčastěji. Každý druh byl popsán včetně uvedení několika základní zástupců. U každého typu databáze byl nejpoužívanější zástupce podrobně popsán. Popis každého zástupce byl také doplněn jednoduchou ukázkou použití. V praktické části pak došlo ke srovnání nejpoužívanějších zástupů podle několika důležitých parametrů.

Z práce vyplývá, že každý druh NoSQL databází slouží k určenému specifickému významu.

Jestliže jsou k dispozici jednoduchá data typu klíč/hodnota, např. výsledky složitěho databázového dotazu, které je potřeba někde dočasně uložit, pak je vhodné využít některou z databází typu klíč/hodnota. Pokud jsou ukládané hodnoty jednoduché řetězce, pak lze doporučit databázi Memcached, jejíž provoz je nenáročný a je podporována mnoha hostingy. Pokud jsou hodnoty o něco složitější (třeba objekty klíč/hodnota) a je potřeba nad nimi provádět další operace, pak bude vhodnějším řešením Redis. Pokud jde o produkční nasazení velkého rozsahu napříč mnoha instancemi databáze, pak může být vhodnější Riak.

Jestliže je struktura dat složitější a obsahují mezi sebou mnoho vazeb, pak stojí za úvahu, zda nepoužít grafovou databázi a namodelovat datový model jako graf. Grafové databáze jsou pak jasnou volbou v případě, kdy je potřeba provádět grafové operace jako hledat nejkratší cestu grafem ap. Pokud však padne volba na grafovou databázi, je potřeba počítat s vyššími náklady na provoz či nástroje. Nejpoužívanější grafovou databázi Neo4j navíc nelze používat zdarma zcela bez omezení. Mnoho firem nesplní licenční požadavky pro provoz verze zdarma a bude muset investovat do koupě vyšší licence, která však není nejlevnější. Pokud je cena zásadní položka, pak lze doporučit konkurenční OrientDb, kterou lze provozovat zdarma i pro komerční účely.

Pokud není potřeba hledat NoSQL databázi pro nějaký konkrétní účel a je zde spíše snaha použít třeba jednodušší přístup k datům než ten, který nabízí SQL, pak stojí za úvahu

použití některé z dokumentově orientované databáze. Z vlastní zkušenosti mohu doporučit databázi MongoDB. Je však potřeba počítat s řadou omezení, z nichž nejvýznamnější je nepodpora transakcí napříč různými kolekcemi, což platí pro všechny nejpoužívanější dokumentově orientované databáze mimo RavenDB.

8.1 Možnosti dalšího výzkumu

Tak rozsáhlé téma jako jsou NoSQL databáze rozhodně nelze podrobně popsat v rámci jedné bakalářské práce. Na tuto práci proto mohou navazovat kolegové z fakulty či jiných vysokých škol. V závěru pro ně přináším dva zajímavá témata, na které již v této práci nezbyl prostor a rozhodně si zaslouží podrobnější analýzu.

8.1.1 Databáze určené primárně pro vyhledávání

Některé databáze jsou určeny především jako vyhledávací stroje, lze je však výborně použít místo klasických dokumentově orientovaných databází. Za zmínku stojí třeba Elasticsearch či Apache Solr. Oba systémy jsou založeny na platformě Apache Lucene. Oba systémy také zaznamenávají rapidní vzestup zájmu a patří mezi 30 nejpoužívanějších databází vůbec.

[11]

8.1.2 Cloudové databáze

Jde o databáze, které jsou primárně navrženy pro použití v cloudu. Cloud computing a NoSQL databáze je jedno z diskutovaných témat dnešní doby a do budoucna význam těchto databází nepochybně poroste. Jde třeba o databáze Cloudant, Cloudbase či Amazon DynamoDb.

Terminologický slovník

Termín	Zkratka	Význam [zdroj]
atomicity, consistency, isolation and durability transaction	ACID	Vlastnosti databázových transakcí: <ul style="list-style-type: none"> • A = transakce se provede buď celá, nebo vůbec, • C = při a po transakci není porušeno integritní omezení, • I = operace uvnitř transakce jsou skryty před ostatními operacemi, • D = změny, které se provedou jako výsledek úspěšných transakcí, jsou skutečně uloženy v databázi a již nemohou být ztraceny. [59]
Big Data		Soubory dat, jejichž velikost je mimo schopnosti zachycovat, spravovat a zpracovávat data běžně používanými softwarovými prostředky v rozumném čase. [60]
Binární JSON	BSON	Binární podoba JSON dokumentu. [61]
cloud computing		Model vývoje a používání počítačových technologií založený na Internetu. [62]
Data as a Service	DaaS	Distribuční model cloud computingu. [autor]
GNU AGPL		Druh licence [autor]

Termín	Zkratka	Význam [zdroj]
Hypertext Markup Language	HTML5	Značkovací jazyk pro tvorbu webu [autor]
Hypertext Transfer Protocol	HTTP	Protokol určený pro přenos dokumentů v HTML [autor].
Infrastructure as a Service	IaaS	Distribuční model cloud computingu. [autor]
Java		Objektově orientovaný programovací jazyk [autor]
JavaScript		Objektově orientovaný skriptovací jazyk [autor]
JavaScript Object Notation	JSON	Formát pro přenos dat [autor]
Node.js		Platforma pro programování aplikací [autor]
Not Only SQL	NoSQL	Druh databází, které používají jinou formu dotazování než jazyk SQL [autor]
Platform as a Service	PaaS	Distribuční model cloud computingu. [autor]
pattern matching		Způsob dotazování, kdy je vzor porovnáván z danou množinou dat. [autor]
Perl		Procedurální programovací jazyk [autor]
Python		Objektově orientovaný programovací jazyk [autor]
Representational State Transfer API	REST API	Rozhraní pro webové služby jako alternativa k SOAP. [autor]

Termín	Zkratka	Význam [zdroj]
Ruby		Objektově orientovaný programovací jazyk [autor]
Scala		Programovací jazyk, založený na principech objektově orientovaného a funkcionálního programování [autor]
semi-strukturovaná data		Data, která kromě hodnoty samotné obsahují i informaci o své struktuře. [13]
Structured Query Language	SQL	Jazyk, který se využívá k dotazování a manipulaci s daty u relačních databází [autor]
Web 2.0		Koncept webu, ve kterém obsah tvoří sami uživatelé. [autor]

Seznam literatury

- [1] MEZRICH, Ben. *The Accidental Billionaires: The Founding of Facebook, A Tale of Sex, Money, Genius, and Betrayal*. USA, 2009. ISBN 978-0-385-52937-2.
- [2] HITCHCOCK, Andrew. Google's BigTable. andrewhitchcock.org [online] [aktualizováno: 31. 8. 2006] Dostupný z: <<http://andrewhitchcock.org/?post=214>>.
- [3] ROBERT, Diana. NoSQL Job Trends. regulargeek.com [online] [publikováno: 29. 8. 2013] Dostupný z: <<http://regulargeek.com/2013/08/29/nosql-job-trends-august-2013>>.
- [4] DRAGLAND, Åse. Big Data – for better or worse. SINTEF [online] [publikováno: 22. 5. 2013] Dostupný z <sintef.no/home/Press-Room/Research-News/Big-Data--for-better-or-worse/>.
- [5] BACHMAN, Michal. Introduction to Neo4j. bachman.cz. [online] [publikováno: 14. 1. 2013] Dostupný z <<http://www.bachman.cz/2013/01/predstaveni-grafove-databaze-neo4j>>.
- [6] ONG, Josh. Facebook now has 1.15 billion monthly active users and 699 million daily active users. The Next Web. [online] [publikováno: 24. 1. 2013] Dostupný z <<http://thenextweb.com/facebook/2013/07/24/facebook-users-q2-2013/#!pLklW>>.
- [7] GÜNZL, Richard. *NoSQL databáze*. Praha, 2012. Bakalářská práce. Vysoká škola ekonomická v Praze.
- [8] PULTERA, Ondřej. *Možnosti použití databázového systému CouchDB*. Praha, 2011. Bakalářská práce. Vysoká škola ekonomická v Praze.
- [9] PETERA, Martin. *MongoDB*. Praha, 2013. Bakalářská práce. Vysoká škola ekonomická v Praze.

- [10] Neo4j manual. Data integration. [online] [citováno: 12. 12. 2013] Dostupný z <http://docs.neo4j.org/chunked/milestone/capabilities-data-integration.html>.
- [11] Db engines. Ranking. [online] [citováno: 12. 12. 2013] Dostupný z <http://db-engines.com/en/ranking>.
- [12] SCOFIELD, Ben. NoSQL talk report. [online] [publikováno: 23. 11. 2009]
- [13] ABITEBOUL, Serge. Querying Semi-Structured Data. [online] [citováno: 12. 12. 2013] Dostupný z <http://www.dtic.mil/dtic/tr/fulltext/u2/a428473.pdf>.
- [14] LEEDS, Randal. Introduction - CouchDB. [online] [aktualizováno: 9. 4. 2012] Dostupný z <http://wiki.apache.org/couchdb/Introduction>.
- [15] RavenDB. Learn RavenDB. [online] [citováno: 12. 12. 2013] Dostupný z <http://ravendb.net/learn>.
- [16] GENESKY, Eric. Most Popular NoSQL Databases According to LinkedIn Skillset. [online] [publikováno: 27. 3. 2012] Dostupný z <http://architects.dzone.com/articles/graph-nosql-database-linkedin>.
- [17] BANKER, Kyle. *MongoDB in Action*. Manning Publications, 2011. ISBN 978-1935182870.
- [18] MongoDB manual. Data models. [online] [citováno: 12. 12. 2013] Dostupný z <http://docs.mongodb.org/manual/data-modeling/>.
- [19] CHODOROW, Christina. *MongoDB: The Definitive Guide*. O'Reilly Media, 2010. ISBN 978-1449381561.

- [20] MongoDB manual. GridFs. [online] [citováno: 12. 12. 2013] Dostupný z <http://docs.mongodb.org/manual/core/gridfs/>.
- [21] MongoDB manual. MapReduce. [online] [citováno: 12. 12. 2013] Dostupné z <http://docs.mongodb.org/manual/core/map-reduce/>.
- [22] MongoDB manual. Journaling Mechanics. [online] [citováno: 12. 12. 2013] Dostupné z <http://docs.mongodb.org/manual/core/journaling/>.
- [23] MongoDB manual. MongoDB CRUD Concepts. [online] [citováno: 12. 12. 2013] Dostupné z <http://docs.mongodb.org/manual/core/crud/>.
- [24] MongoDB manual. Sharding. [online] [citováno: 12. 12. 2013] Dostupné z <http://docs.mongodb.org/manual/sharding/>.
- [25] MongoDB manual. Licensing. [online] [citováno: 12. 12. 2013] Dostupné z <http://www.mongodb.org/about/licensing/>.
- [26] LERNER, Alberto a další. The MongoDB Cookbook. [online] [citováno: 12. 12. 2013] Dostupný z <http://cookbook.mongodb.org/patterns/random-attribute/>.
- [27] FINKELSTEIN, Yuri a FIEBUSCH, John. Storing eBay's Media Metadata on MongoDB [online] [publikováno 10. 5. 2013] Dostupný z <http://www.mongodb.com/presentations/storing-ebays-media-metadata-mongodb-0>.
- [28] MongoDB Case Studies. CERN CMS. [citováno 12. 12. 2013] Dostupný z <http://www.mongodb.com/customers/cern-cms>.
- [29] MongoDB Case Studies. Cisco. [citováno 12. 12. 2013] Dostupný z <http://www.mongodb.com/customers/cisco>.

- [30] MongoDB Case Studies. Foursquare. [citováno 12. 12. 2013] Dostupný z <http://www.mongodb.com/customers/foursquare>.
- [31] ROBINSON, Ian a kol. Graph Databases. O'Reilly, 2013. ISBN 978-1449356262.
- [32] Neo4j manual. Relationships. [online] [citováno 12. 12. 2013] Dostupný z <http://docs.neo4j.org/chunked/milestone/graphdb-neo4j-relationships.html>.
- [33] OrintDB manual. Tutorial: Document and graph model. [online] [citováno: 12. 12. 2013] Dostupný z <https://github.com/orientechnologies/orientdb/wiki/Tutorial:-Document-and-graph-model>.
- [34] Titan manual. Beginner's Guide. [online] [citováno: 12. 12. 2013] Dostupný z <https://github.com/thinkaurelius/titan/wiki/Beginner%27s-Guide>.
- [35] Intro to Neo4j. Youtube. [online] [zveřejněno 7. 1. 2013] Dostupné z <http://www.youtube.com/watch?v=7Fsf5DP9sE>.
- [36] Neo4j manual. Cypher Query Language. [online] [citováno 12. 12. 2013] Dostupný z <http://docs.neo4j.org/chunked/milestone/cypher-query-lang.html>.
- [37] Neo4j manual. Transaction management. [online] [citováno 12. 12. 2013] Dostupný z <http://docs.neo4j.org/chunked/milestone/transactions.html>.
- [38] Neo4j manual. REST API. [online] [citováno 12. 12. 2013] Dostupný z <http://docs.neo4j.org/chunked/milestone/rest-api.html>.
- [39] Neo4j manual. Backup. [online] [citováno 12. 12. 2013] Dostupný z <http://docs.neo4j.org/chunked/milestone/operations-backup.html>.

[40] Neo4j manual. Licenses. [online] [citováno 12. 12. 2013] Dostupný z <<http://www.neo4j.org/learn/licensing>>.

[41] Neotechnology. Who uses Neo4j? Companies like yours.[online] [citováno 12. 12. 2013] Dostupný z <<http://www.neotechnology.com/customers/>>.

[42] REDMOND, Eric and WILSON, Jim. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. Pragmatic Bookshelf, 2012. ISBN 978-1934356920.

[43] Memcached tutorial. About Memcached. [online] [citováno: 12. 12. 2013] Dostupný z <<http://memcached.org/about>>.

[44] Riak. [online] [citováno 12. 12. 2013] Dostupný z <<http://basho.com/riak/>>.

[45] Cassandra manual. Introducing Cassandra. [online] [citováno 12. 12. 2013] Dostupný z <http://www.datastax.com/documentation/gettingstarted/index.html?pagename=docs&version=quick_start&file=quickstart#getting_started/gettingStartedCassandraIntro.html>

[46] SEGUIN, Karl. The Little Redis Book. [online] [citováno 12. 12. 2013] Dostupný z <<http://openmymind.net/redis.pdf>>.

[47] PATNI, Chandra. Fast, easy, realtime metrics using Redis bitmaps. [online] [publikováno 21. 11. 2011]. Dostupný z <<http://blog.getspool.com/2011/11/29/fast-easy-realtime-metrics-using-redis-bitmaps/>>.

[48] Redis manual. Transactions. [online] [citováno 12. 12. 2013] Dostupné z <<http://redis.io/topics/transactions>>.

[49] Redis manual. EVAL. [online] [citováno 12. 12. 2013] Dostupné z <<http://redis.io/commands/eval>>.

[50] Redis manual. Partitioning: how to split data among multiple Redis instances. [online] [citováno 12. 12. 2013] Dostupné z <<http://redis.io/topics/partitioning>>.

[51] Redis manual. Sentinel. [online] [citováno 12. 12. 2013] Dostupné z <<http://redis.io/topics/sentinel>>.

[52] Redis manual. Persistence. [online] [citováno 12. 12. 2013] Dostupné z <<http://redis.io/topics/persistence>>.

[53] Redis manual. License. [online] [citováno 12. 12. 2013] Dostupné z <<http://redis.io/topics/license>>.

[54] KRIKORIAN, Raffi. Real-Time Delivery Architecture at Twitter. [online] [publikováno: 23. 10. 2012] Dostupné z <<http://www.infoq.com/presentations/Real-Time-Delivery-Twitter>>.

[55] WERNER, T. Preston. How We Made GitHub Fast. [online] [publikováno: 20. 10. 2009] Dostupné z <<https://github.com/blog/530-how-we-made-github-fast>>.

[56] HOFF, Tod. The Instagram Architecture Facebook Bought For A Cool Billion Dollars. [online] [publikováno: 9. 4. 2012] Dostupné z <<http://highscalability.com/blog/2012/4/9/the-instagram-architecture-facebook-bought-for-a-cool-billio.html>>.

[57] CAUDILL, Nolan. Talk: Real-time Updates on the Cheap for Fun and Profit. [online] [publikováno: 11. 10. 2012] Dostupné z <<http://code.flickr.net/2011/10/11/talk-real-time-updates-on-the-cheap-for-fun-and-profit/>>.

[58] MONTROSE, Kevin. Does Stack Overflow use caching and if so, how? [online] [publikováno: 3. 10. 2013] Dostupné z

<<http://meta.stackoverflow.com/questions/69164/does-stack-overflow-use-caching-and-if-so-how/69172#69172>>.

[59] Wikipedia. Databázové transakce. [citováno: 13. 12. 2013] [online] *<http://cs.wikipedia.org/wiki/Datab%C3%A1zov%C3%A1_transakce#>*

[60] DOLÁK, Ondřej. Big data, Nové způsoby zpracování a analýzy velkých objemů dat. [citováno: 12. 12. 2013] [online] *<<http://www.systemonline.cz/clanky/big-data.htm>>.*

[61] BSON. [citováno: 12. 12. 2013] [online] *<<http://bsonspec.org/>>.*

[62] CROSMAN, Penny. Cloud Computing Begins to Gain Traction on Wall Street. [publikováno: 6. 1. 2009] [online] *<<http://www.wallstreetandtech.com/it-infrastructure/cloud-computing-begins-to-gain-traction/212700913>>.*

[63] Wikipedia. FIFO. [citováno: 13. 12. 2013] [online] *<<http://en.wikipedia.org/wiki/FIFO>>.*

[64] NoSQL databases. [citováno: 13. 12. 2013] [online] *<<http://nosql-database.org/>>.*

[65] Webarchive.org. [citováno: 14. 12. 2013] [online] *<https://web.archive.org/web/*/http://www.neotechnology.com/price-list/>.*

Seznam obrázků a tabulek

Obrázek 1: Sharding.....	13
Obrázek 2: Vzorová struktura databáze	25
Obrázek 3: Výstup zadaného dotazu ve formě grafu	34
Obrázek 4: Výstup zadaného dotazu ve formě tabulky	35
Tabulka 1: Srovnání dokumentově orientovaných databází	53
Tabulka 2: Srovnání grafových databází	55
Tabulka 3: Srovnání databází typu klíč/hodnota.....	57