TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Analýza výkonu a implementace databázových systémů

Bakalářská práce

Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie

Autor práce: **Kristýna Frydrychová**
Vedoucí práce: Ing. Igor Kopetschke
Ústav nových technologií a aplikované informatiky

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Performance analysis and implementation of database systems

Bachelor thesis

Study programme: B2646 Information technology
Study branch: Information technology

Autor: **Kristýna Frydrychová**
Supervisor: Ing. Igor Kopetschke
Institute of New Technologies and Applied Informatics

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Analýza výkonu a implementace databázových systém

Jméno a příjmení: **Kristýna Frydrychová**
Osobní číslo: M17000075
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav nových technologií a aplikované informatiky
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Dle vlastního uvážení a na základě analýzy požadavků zadavatele zvolte minimálně 5 open source databází vhodných pro provoz na Raspberry Pi.
2. Navrhněte sadu konkrétních zátěžových testů zaměřených na větší objem dat.
3. Pro provedení zátěžových testů nepoužijete standardní dostupné nástroje, ale implementujete vlastní validátor, jehož výstupem bude export do validity log souboru.
4. Implementaci validátoru realizujte ve Vámi zvoleném jazyce z následujících variant – C#, .NET, Python nebo Java.
5. Jednotlivé testy budou zaměřeny zejména na rychlost v real time, na zjištění maximálního komprimačního indexu a na další parametry dodatečně požadované zadavatelem.
6. Vypracujte metodiku vyhodnocení validity log souboru, navrhněte a zdůvodněte finálně vybraný databázový SW.
7. Jestli to bude možné, získejte zpětnou vazbu z ostrého provozu.

Rozsah grafických prací: dle potřeby
Rozsah pracovní zprávy: 30-40 stran
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

[1] KROENKE, David a David J. AUER. *Databáze*. Brno: Computer Press, 2015. ISBN 978-80-251-4352-0.
[2] HEROUT, Pavel. *Testování pro programátory*. České Budějovice: Kopp, 2016. ISBN 978-80-7232-481-1.
[3] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.

Vedoucí práce: Ing. Igor Kopetschke
Ústav nových technologií a aplikované informatiky

Datum zadání práce: 9. října 2019
Předpokládaný termín odevzdání: 18. května 2020

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci dne 17. října 2019

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracovala samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědoma toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědoma povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědoma následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

28. května 2020 Kristýna Frydrychová

Poděkování

Ráda bych poděkovala panu Ing. Igoru Kopetschkemu za vstřícnost a odborné vedení mé bakalářské práce. Velké dík také patří mému konzultantovi Michalu Čermákovi ohledně cenných rad, které mi pomohly tuto práci vytvořit. Poděkování také patří společnosti Cermitech, která mi umožnila práci uskutečnit poskytnutím potřebných materiálů a zařízení.

Abstrakt

Cílem bakalářské práce je na základě analýzy volně dostupných databázových systémů zvolit minimálně 5 open source databází, které jsou vhodné pro provoz na zařízení Raspberry Pi. Pro jejich testování je třeba navrhnout sadu zátěžových testů zaměřených na velký objem dat. Pro realizaci zátěžových testů bude implementován vlastní validátor ve zvoleném programovacím jazyce, jehož výstupem bude export do validity log souboru. Jednotlivé testy budou zaměřené především na rychlost práce s daty a jejich komprimaci. Výsledkem bude vyhodnocení validity log souboru, návrh a zdůvodnění finálně vybraného databázového softwaru.

Klíčová slova

Raspberry Pi, databázové systémy, zátěžové testy, validátor

Abstract

Focus of this thesis, is to conduct an analysis of open source database systems and choose 5 of those, which are suitable for operation on Raspberry Pi. A series of load tests on large volume of data is proposed. For the load tests a custom validator will be implemented in chosen programming language. Validator's output will be an export into validity log file. Tests will be focused on data manipulation and compression. Result will be log file validity evaluation and explanation of the final choice of database system.

Keywords

Raspberry Pi, database systems, load tests, validator

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

Seznam použitých symbolů a zkratek

SQL – Structured Query Language

FSF – Free Software Foundation

RDBMS – Relational Database Management System

IoT – Internet of Things

OLAP – Online Analytical Processing

OLTP – Online Transaction Processing

FTP – File Transfer Protocol

BLOB – Binary Large OBject

JVM – Java Virtual Machine

JDK – Java Development Kit

Úvod

Při vývoji nového softwaru je členy vývojového a testovacího týmu obvykle kladen velký důraz na testování GUI, a to z důvodu, že grafické uživatelské rozhraní se stává nejvíce viditelnou částí aplikace. Ovšem důležité je ověřit především funkci jádra aplikace, čímž je databázový systém. Testování databází se dělí do tří částí. Strukturální testování databáze zahrnuje kontrolu všech prvků v datovém uložišti a především ověření databázového serveru. Funkční testy ověřují správnost veškerých transakcí a operací prováděných uživatelem tak, aby byla v souladu se specifikacemi požadavků. Poslední částí je testování výkonu databázových systémů.

Cílem této bakalářské práce je realizovat tento poslední bod, který se týká testování výkonu vybraných databází a porovnání jejich výsledků. Následný návrh databázového systému podle analýzy výsledků by měl sloužit jako základ pro reálnou aplikaci fungující jako multiinstanční software pro průmysl 4.0.

První část je zaměřená především na prozkoumání dostupných databázových systémů, porovnání jejich vlastností a funkcí pro získání představy jejich použití v jednotlivých aplikačních systémech.

V druhé části jsou na základě předchozí analýzy představeny vybrané open source databáze, vyzdvihnuty jejich specifické vlastnosti a využití. Následně je popsána jejich implementace na zařízení Raspberry Pi s operačním systémem Raspbian.

Třetí část je především věnována různým způsobům testování výkonu databáze a realizaci vybraných zátěžových testů ve zvoleném programovacím jazyce Python.

Na závěr jsou z validity log souboru porovnány a vyhodnoceny výsledky jednotlivých testů provedených na zvolených databázích a představeno finální řešení databázového systému pro reálnou aplikaci.

1 Teorie databází a jejich testování

Pokud je součástí softwaru databáze, na kterou je daná aplikace závislá, mělo by samozřejmostí být i testování této databáze. Kontrolou by měla projít struktura databáze, její funkčnost a výkon. Struktura databáze zahrnuje kontrolu kompatibility definice tabulky s definicí použitou pro uživatelské rozhraní, kontrolu nepřiřazených tabulek a sloupců a kontrolu mapování databáze s aplikací. Při testu funkčnosti se ověřuje správnost zpracování transakcí. Zda při jejich zpracování nebyla žádná data ztracena či nedošlo k uložení informací, které byly provedené jen částečně či byly zrušené. Také zda k databázi přistupují pouze pověřené osoby. Poslední variantou je zátěžové testování, které prověří, zda má databáze dostatečný výkon na zpracování dat. Především testování výkonu je tématem této práce. (1)

1.1 Porovnávací tabulka open source databází

Každá databáze má jiné funkce a jiné zaměření, proto se každá hodí na různé typy projektů. Některé databáze jsou určené pro rozsáhlou analýzu dat, jiné se zaměřují na co nejrychlejší ukládání dat do databáze nebo vytváření grafů ze získaných dat. Záleží tak na požadavcích, které jsou od databáze očekávány a na základě nich zvolit nejvhodnější databáze pro vývoj aplikace.

Většina databázových systému je multiplatformní, což znamená, že je k dispozici na všech hlavních platformách jako je Linux, Windows, MacOS, Solaris nebo BSD. Na relační databáze se lze také dotazovat i jinými jazyky jako např. Python, PHP, Perl, Ruby, C++, Java nebo Node.js než pouze dotazovacím jazykem SQL.

Databázové systémy, které poskytují své služby už řadu let, jsou spolehlivější, výkonnější a snadno se používají, proto patří k velmi oblíbeným u komunity. Díky jejich rozsáhlé síti vývojářů, velké podpoře u obrovské a aktivní komunity, které poskytuje bezplatnou podporu, a rozsáhlého testování jde jejich vývoj stále dopředu. S přibývajícími možnostmi rostou i nároky na databáze. Již osvědčené open source databáze často přináší nové vylepšení v podobě lepší rychlosti a přidaných funkcí pro pohodlnější práci s databázovými systémy. Což je také důvod, proč je důležité se zaměřit na databáze, které spolehlivě ukládají data již několik let a jejich vývojáři často přináší nové verze pro udržení kroku s dobrou.

Tabulka 1: Dostupnost a použití open source databází

Databáze	Licence	Typ databáze	Zaměření
MySQL	GNU GPLv2 i komerční	RDBMS	–
PostgreSQL	PostgreSQL Licence	RDBMS, ORDBMS	Objektově relační databáze
MariaDB	GNU GPLv2, GNU LGPLv2.1	RDBMS	Podpora ColumnStore
Neo4j	GNU GPLv3 a GNU AGPLv3 i komerční	NoSQL	Grafová databáze
MongoDB	GNU AGPLv3	NoSQL	Dokumentově orientovaná databáze
RethinkDB	Apache Licence 2.0	NoSQL	Dokumentově orientovaná databáze
Redis	BSD 3-bodová	NoSQL	Key-value databáze
SQLite	Volné dílo	RDBMS	–
Cassandra	Apache Licence 2.0	NoSQL	Sloupcově orientovaná databáze
TimescaleDB	Apache Licence 2.0	SQL i NoSQL	Time series DBMS
CouchDB	Apache Licence 2.0	NoSQL	Dokumentově orientovaná databáze
MonetDB	Mozila Public Licence 2.0	RDBMS i NoSQL	Sloupcově orientované databázové systémy
HBase	Apache Licence 2.0	NoSQL	Sloupcově orientovaná databáze
Druid	Apache Licence 2.0	NoSQL	Sloupcově orientované, distribuované a real-time databázové systémy
OrientDB	Apache Licence 2.0	NoSQL	Multimodální databáze (graf, key-value, objekty, dokumenty)
Couchbase	Apache Licence 2.0	NoSQL	Multimodální databáze (key-value, dokumenty)
Apache Hive	Apache Licence 2.0	RDBMS	Datový sklad
Berkeley DB	Různé licence	NoSQL	Embedded databáze
Apache Derby	Apache Licence 2.0	RDBMS	–
Titan	Apache Licence 2.0	NoSQL	Grafová real-time databáze

Tabulka 2: Informace o vývoji open source databází

Databáze	Vývojáři	Rok vydání	Poslední verze	Vyvíjeno v jazyce
MySQL	Oracle Corporation	1995	2020	C, C++
PostgreSQL	PostgreSQL Global Development Group	1996	2019	C
MariaDB	MariaDB Corporation Ab, MariaDB Foundation	2009	2020	C, C++, Perl, Bash
Neo4j	Neo4j	2007	2020	Java
MongoDB	MongoDB Inc.	2009	2020	Go, C++, C, Python, JavaScript
RethinkDB	RethinkDB	2009	2019	C++, Java, Python, JavaScript, Bash
Redis	Redis Labs	2009	2020	ANSI C
SQLite	D. Richard Hipp	2000	2020	C
Cassandra	Apache Software Foundation	2008	2020	Java
TimescaleDB	Timescale	2017	2020	C
CouchDB	Apache Software Foundation	2005	2019	Erlang, JavaScript, C, C++
MonetDB	MonetDB B.V.	2002	2019	C
HBase	Apache Software Foundation	2008	2019	Java
Druid	Apache Druid	2011	2020	Java
OrientDB	OrientDB Ltd	2010	2020	Java
Couchbase	Couchbase, Inc.	2010	2019	C++, Go, Erlang, C
Apache Hive	Contributors	2010	2019	Java
Berkeley DB	Oracle Corporation	1994	2018	C
Apache Derby	Apache Software Foundation	1996	2019	Java
Titan	Aurelius	2012	2020	Java

2 Kritéria pro výběr požadovaných databází

Cílem je najít databázi s tolerantnějšími licenčními podmínkami, která umožňuje snadnější použití, modifikaci a následnou distribuci softwaru ať pro komunitní, tak i pro komerční vývoj.

Databáze by měla umět pojmout co největší množství dat v co nejkratším čase a rychle zpracovat různé dotazy. Zároveň by měla umožňovat vysokou komprimaci dat, aby zabírala co nejméně místa na disku. Výhodou by byla možnost ukládání obrázků přímo do databáze.

2.1 Licence free a open source softwaru

Software s open source licencí je možné na základě licenčních podmínek zdarma využívat, upravovat jeho zdrojové kódy nebo šířit dále ať už pro svoje osobní účely či pro komerční využití. Licenční podmínky obvykle zahrnují požadavek na přiložení kopie licenčních podmínek, zachování jmen autorů a jejich zřeknutí se odpovědnosti za dílo či zachování stejné licence v případě distribuce softwaru.

Open source software zde klade důraz na dostupnost zdrojového kódu i legální dostupnost čili způsob, jakým může uživatel se softwarem nakládat. Dostupný zdrojový kód vždy ale nemusí být základem a může se pouze jednat o software vyvíjený komunitou, který je dostupný zdarma.

Výhodou open source softwaru je možnost jeho větvení, tzv. fork, kdy mohou zcela jiní vývojáři kompletní zdrojový kód tohoto softwaru použít jako základ, který budou dále nezávisle na původním softwaru vyvíjet v jiný program. Vzniká tak zcela nový samostatný software, který v případě, že má svoji vlastní komunitu vývojářů a vlastní jméno, vytváří nové obchodní příležitosti. Při kopii zdrojového kódu je také důležité zahrnout licenční soubor softwaru.

V případě, že se tento software vydá cestou komerčního vývoje, je možné na open source databázi využít fork a přizpůsobit si licenční podmínky. V případě použití databáze s placenou licencí v softwaru, který by firma dále distribuovala, by se software prodražil kvůli licenčním poplatkům.

Ve srovnávací tabulce open source databází (Tabulka 1) se objevují licence jako GNU GPL, AGPL a LGPL, BSD licence, Apache Licence či Mozilla Public Licence.

2.1.1 GNU General Public License (GPL)

K jedné z nejpoužívanějších a velmi známých free software licencích se řadí GPL napsané pro GNU Richardem Stallmanem z Nadace pro svobodný software (FSF). Je příkladem silně copyleftové licence, jejímž účelem je šíření díla pouze pod stejnými licenčními podmínkami. To zajišťuje stálou svobodu softwaru dalším uživatelům počítačového programu, i když je dílo modifikováno či přidáno k jinému produktu. Zásadně se tím liší od permisivních licencí free softwaru, kdy je tyto díla možné dále šířit bez původní licence.

Distributoři softwaru začali využívat mezeru v původní verzi GPL, kde nebyl přesně specifikovaný způsob, jak dílo dále šířit, proto svůj software publikovali formou binárních souborů, které jsou sice spustitelné, ale nejdou modifikovat ani přečíst, čímž omezovali svobodu softwaru. Vznikla GPL verze 1, která specifikovala podmínky distribuce díla tak, aby byl v případě dalšího šíření díla zpřístupněn člověkem čitelný zdrojový kód.

Během šíření díla bylo také možné přidávat další restrikce k licenci a tím dílo omezovat či ho kombinovat s jiným softwarem, který má přísnější omezení ohledně dalších distribucí, což vedlo k dalším omezením softwaru licencovaným pod GPL. GPLv1 si tak svoje dílo začalo chránit zavedením podmínek, které nařizují distribuci díla jako celek pouze za licenčních podmínek GPL, proto lze tento software kombinovat pouze se softwarem s tolerantnějšími licencemi, jako jsou BSD či Apache, čímž se nijak neomezí licenční podmínky GPL.

V GPLv2 došlo ke změně klauzule, která vypovídá o tom, že pokud je přidáno omezení, které narušuje licenční podmínky GPL a brání tak svobodnému šíření softwaru pro další uživatele, nesmí se dílo šířit vůbec. Nabyvatelé licence tak mohou dílo dále distribuovat pouze v případě, že mohou splnit všechny licenční podmínky GPL.

Nyní nejčastěji používaná verze 3 licence GPL byla obohacena o přesné definice zdrojového kódu, vztahy k softwarovým patentům či lepší kompatibilitu s ostatními licencemi open source softwaru, jako například Apache Licence 2.0. Dále omezovala tivoizaci, čili systém, který zamezuje uživatelům spouštění modifikovaných verzí softwaru na hardwaru tak, že šířený software pod licencí GPL využívá určitá hardwarová omezení. To bránilo uživatelům využívat svobody softwaru GNU GPL.

GPLv3 byla také doplněna o způsob řešení porušení licenčních podmínek a přidávání dalších požadavků k licenci při distribuci díla pomocí Affero klauzule, což umožnilo kompatibilitu GPL verze 3 s licencí AGPL.

Licence GPL povoluje komerční užití díla, jeho modifikaci, šíření i prodávání a to bez omezení práv, které GPL poskytuje. Tyto programy se musí dále šířit se čtivým zdrojovým kódem či písemnou nabídkou k poskytnutí zdrojového kódu na vyžádání třetí strany. V případě držení zdrojového kódu v tajnosti může být distributor díla trestně stíhán původním autorem softwaru. Licence GPL se tak co nejvíce snaží zamezit odebrání práv ostatním uživatelům na využívání svobodného softwaru. (2)

2.1.2 GNU AGPL

Při každé distribuci programu pod licencí GPL má její další uživatel právo tento program kopírovat, upravovat a dále distribuovat, k čemuž potřebuje zdrojové kódy, které musí být při každé distribuci zveřejněny. V případě, že je tento program poskytován pouze po síti např. v podobě webových serverů či aplikacích používaných po síti, nedochází přímo k distribuci softwaru a tak zůstává zdrojový kód obvykle skrytý, což narušuje běžné myšlení používání licence GPL. Tento způsob užívání softwaru po síti ovšem není v licenčních podmínkách GPL definován.

Affero GPL doplňuje tyto licenční podmínky o požadavek uživatelům využívající aplikaci běžící po počítačové síti zpřístupnit veškeré zdrojové kódy aplikace na stejné síti. Tím má původní autor jistotu, že se jeho software objeví i v dalších programech. (3)

2.1.3 GNU LGPL

Licence LGPL je na rozdíl od GPL permisivnější tím, že uplatňuje licenční podmínky pouze na software uvolněný pod LGPL, nikoliv na celé dílo, kterého je tento software součástí. Neuplatní se tak silné copyleftové podmínky, které nařizují uvolnění zdrojového kódu nejen softwaru pod danou licencí, ale i vlastních částí kódu.

LGPL se nejčastěji používá pro softwarové knihovny, které je v případě použití či úpravě potřeba zpřístupnit pod stejnou licencí. Díky využívání sdílených knihoven je zde jasný rozdíl mezi softwarem spadající pod licenci LGPL a vlastními částmi aplikace, pokud je program navržen tak, aby s knihovnou pouze pracoval a neobsahoval žádné odvozené části knihovny. Potom tento program není odvozeným dílem, proto nespadá pod licenční podmínky LGPL a je možné ho dále šířit i proprietárně.

2.1.4 BSD licence

BSD licence se řadí mezi permisivní licence free softwaru, která má jen minimální požadavky pro další distribuci díla. Programy, které spadají pod BSD licenci, tak může kdokoliv používat a dále šířit bez omezení. Stačí pouze přiložit kopii s textem licence a zachovat jména autorů s informací o zřeknutí se odpovědnosti za dílo. BSD licenci je možné použít ke komerčnímu využití bez nutnosti uvedení zdrojových kódů díla.

BSD licence byla vyvinuta za účelem distribuce unixového operačního systému BSD obchodní organizací na kalifornské Univerzitě v Berkeley. Odtud dostala i svůj název Berkeley Software Distribution. Původními licenčními podmínkami se inspirovaly i další licence jako např. MIT. Od té doby byla originální licence upravena do několika verzí, které jsou dnes známy jako modifikované licence BSD.

Dnes se nejčastěji používá tříbodová BSD licence, která je poskytována ke všem způsobům užití při dodržení autorských práv. Lze ji kombinovat či učinit součástí dalšího proprietárního počítačového programu. Při používání softwaru, který nese tuto licenci, je nutné přesně specifikovat její verzi, aby nedošlo k záměně s původní čtyřbodovou BSD licencí. Ta je na rozdíl od tříbodové BSD licence doplněná o klauzuli, tzv. „reklamní doložku“, která zahrnovala uvedení informace o původním zdroji softwaru do všech dalších odvozených produktů.

Tato klauzule vedla k právním problémům týkající se kompatibility s GNU GPL v případě, kdy byly kombinovány programy dvou různých licencí v jednu softwarovou distribuci. Z tohoto důvodu byla původní verze licence revidována odstraněním „reklamní doložky“, proto je nyní možné dílo spadající pod licenci BSD v případě kombinování s dílem s licencí GPL zveřejnit pouze pod licencí GNU GPL. V opačném případě začlenění díla s licencí GLP pod licenci BSD nelze. (4)

2.1.5 Apache Licence

K dalším svobodným softwarovým licencím spadá Apache Licence vytvořená organizací Apache Software Foundation. Programy pod licencí Apache může uživatel bez dalších omezení využívat, upravovat, distribuovat či redistribuovat aniž by došlo k porušení licenčních podmínek. Stačí pouze zachovat informaci o autorství a zřeknutí se odpovědnosti.

Stejně jako u licence BSD, tak i u licence Apache při další distribuci softwaru či jeho modifikaci není nutné šíření pod stejnou licencí. Jedinou podmínkou je zachování nemodifikovaných částí softwaru. Pokud dílo obsahuje soubor NOTICE, je nutné, aby odvozené dílo obsahovalo kopii tohoto souboru. Obsah tohoto souboru slouží pouze pro informativní účely a licenci nijak neupravuje. Jeho obsah může být doplněn o dodatečné oznámení od uživatele. Tento dodatek má opět pouze infomační charakter a znění licence nijak nemění. V případě redistribuce je nutné zachovat informaci o souborech, které byly modifikovány.

Z původní verze licence Apache byl v roce 2000 vypuštěn bod, který pojednával o zachování údajů o autorovi v reklamních materiálech dále distribuovaných produktů. Tento argument přijala společnost Berkeley o rok dřív u licence BSD.

Následovala Apache Licence 2.0, která se odlišovala od původního modelu BSD, což vedlo k usnadnění používání dalších projektů pod touto licencí, zrušení informace o licenci v každém souboru a také podpora lepší kompatibility se softwarem pod licencí GPL.

Apache Licence 2.0 je kompatibilní s licencí GNU GPL verze 3, proto lze software pod těmito licencemi kombinovat a výsledně šířit pouze pod licencí GPLv3. Jakákoliv jiná verze licence Apache není kompatibilní s GPLv3 stejně tak jako Apache Licence 2.0 je nekompatibilní s jakoukoliv jinou verzí GPL. (5)

2.1.6 Mozilla Public License (MPL)

MPL vyvíjená a udržovaná společností Mozilla Foundation je open source licence o něco restriktivnější než permisivní licence typu BSD a zároveň tolerantnější na rozdíl od GNU GPL. Jedná se tak o slabě copyleftovou licenci, jejíž aktuální verze 2.0 zajišťuje kompatibilitu s dalšími licencemi.

Licenční podmínky MPL se oproti BSD liší v tom, že v případě zahrnutí softwaru s licencí MPL do vlastního produktu, je nutné při další distribuci díla tuto část šířit pod licencí MPL. Pokud tak dojde k šíření celého díla, jehož součástí je software s licencí MPL, zajistí se pouze otevřenost kódu tohoto softwaru a zbytek zdrojových kódů tohoto díla zůstane skrytý. Tyto ostatní části softwaru je tak možné šířit pod jinou licencí, čímž se MPL odlišuje od GPL.

2.2 RDBMS nebo NoSQL?

RDBMS neboli SQL databáze byly navrženy za účelem snížení duplicity dat. Svá data mají uložena strukturovaně ve formě řádků a sloupců. Nestrukturované NoSQL databáze neboli distribuované databáze se skládají z různých druhů technologií, které splňují kladené požadavky při vývoji moderních aplikací, čímž šetří čas i peníze. Existuje řada klíčových rozdílů mezi relačními a nerelačními databázemi, které je potřeba zohlednit. Obě varianty jsou správné, záleží pouze na požadavcích a situaci. Každá databáze má svoje využití v jiných projektech. Pro práci s víceřádkovými transakcemi bude efektivnější relační databáze, kdežto ukládání objektových dat, které v dnešní době převažují, zvládnou rychleji distribuované databáze. Před výběrem optimální databáze pro snadné ukládání dat je potřeba se zamyslet, zda budou data ukládána s jasným logickým spojením, zda se bude zapisovat velké množství dat, jak rychle je potřeba načítat data a udržovat aktuální odpovědi na dotazy či s jak velkou podporou lze počítat. (6)

2.2.1 Jazyk

Pro manipulaci s relačními databázemi jako načítání dat z databáze či jejich aktualizace se používá dotazovací jazyk SQL. Je nejpoužívanější a nejbezpečnější možností pro vytváření komplexních dotazů. Ten vyžaduje předem přesně definovanou strukturu dat, se kterými pracuje. Pro dotazování na jeden objekt je potřeba se připojit k datům z více tabulek. Při zvětšujícím se počtu tabulek může být dotazování pomalejší.

Naproti tomu NoSQL ukládá data do nestrukturované podoby, proto nemá žádný dotazovací jazyk. Data jsou ale ukládána způsobem optimalizovaným pro dotazy, proto je dotazování na data mnohem rychlejší. (7)

2.2.2 Datový model a schéma

Relační databáze jsou strukturované, proto je před přidáním dat do systému potřeba předdefinovat přesné schéma, kterého je nutné se držet. To znamená věnovat výraznou přípravu předem k nadefinování struktury. Změna struktury schématu pak bývá velmi obtížná, je časově náročná a velmi nákladná a v některých případech může vyžadovat i přerušení služeb nebo provozu.

NoSQL databáze mají dynamické schéma, proto mohou ukládat i nestrukturovaná data. Ty lze přidávat přímo do databáze, aniž by bylo nutné vynahradit čas na předdefinování schématu. Data jsou ukládána různými způsoby a mají svou vlastní strukturu. Při změně požadavků a dat tak probíhá snadnější aktualizace databází NoSQL bez narušení jejího provozu. (8)

2.2.3 Datová struktura

Datová struktura SQL databází používají tabulky, které jsou mezi sebou propojeny. Efektivní uspořádání dat do řádků a sloupců urychluje prohledávání a lepší filtrování požadovaných vlastností objektu. Tato struktura byla založena v době, kdy byla používána jednoduchá data s přesně definovanými vztahy, což už v dnešní době neplatí.

NoSQL jsou založené na datové struktuře key-value pro jednoduché vyhledávání dotazů, grafových databázích s uzly a pro rychlou analýzu nebo dokumentově či sloupcově orientovaných databázových systémech pro velké množství dat s předvídatelnými vzory dotazů. Díky tomu zpracovávají složitější nestrukturovaná data, jako jsou obrázky, videa, audia, příspěvky, emaily atd. (9)

2.2.4 Škálovatelnost

SQL databáze jsou vertikálně škálovatelné. V případě, kdy překročí požadavky na kapacitu serveru, na kterém běží, je možné jejich výkon zvýšit přidáním hardwarových komponent jako CPU, RAM nebo SSD. Relační databáze je tak hostována na jednom výkonném serveru, což se může značně prodražit. Ovšem distribuce relační databáze na více serverů by velmi zkomplikovala práci s tabulkami. Pro případné škálování na různé servery lze využít virtualizaci, kdy je databáze replikována na více serverů, mezi kterými se může fyzicky pohybovat na diskových polích. Hodí se zejména v případě, kdy dojde k výpadku jednoho serveru, čímž nedojde k přerušení a databáze stále běží dál bez dalších ztrát.

Oproti tomu jsou NoSQL databáze škálovatelné horizontálně přidáním dalších serverů do databáze. To je činí silnějšími a lépe rozšiřitelnějšími, proto se stávají preferovanou volbu pro velké a stále se měnící sady dat, které dokážou rychle zpracovávat. Rozšiřování kapacity databáze přes komoditní servery je nákladově mnohem efektivnější ve srovnání se zvyšováním kapacity pro SQL databáze. (10)

2.2.5 Podpora

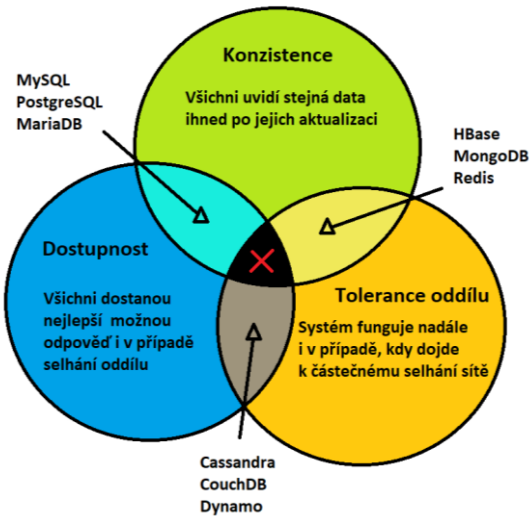
Jelikož existují SQL databáze už řadu let, mají obrovskou podporu od jejich dodavatelů v podobě častých aktualizacích a zákaznické podpory. Je k dispozici spousta návodů nebo kurzů pro práci nebo rozsáhlé nasazení relační databáze.

NoSQL databáze jsou častěji součástí open source spoléhající se na podporu komunity. Pro jejich nasazení ve velkém měřítku je obtížné najít odborníky, jelikož se jedná o novější technologii. (11)

2.2.6 Vlastnosti ACID a CAP

V SQL databází je kladen důraz na integritu jejich dat a zachování její konzistence, což zajišťují operace ACID (Atomicity, Consistency, Isolation a Durability). Operace během transakce jsou nedělitelné, tzn., že se mohou provézt pouze jako celek, v jiném případě se data vrátí do původního stavu. Po dokončení transakce musí být databáze validní a neporušovat žádné integritní omezení. Operace se navzájem neovlivňují, ale vykonávají se postupně a veškeré změny jsou okamžitě uloženy.

Naproti tomu NoSQL ve velkých distribuovaných uložištích dává přednost dostupnosti dat před jejich konzistencí. Teorém CAP (Consistency, Availability a Partition tolerance) udává, že lze současně poskytnout pouze dvě ze tří záruk – konzistence dat, jejich dostupnost nebo odolnost proti výpadkům oddílu. Tudíž při selhání síťového oddílu je potřeba se rozhodnout, zda dokončit operaci pro zajištění dostupnosti za riziko ztráty konzistence nebo zrušit operaci a zajistit tak bezchybnou odpověď ač ne zaručeně aktuální. (12)



Obrázek 1: CAP teorém

2.2.7 Kdy použít SQL a kdy NoSQL?

SQL

- Lepší volba pro aplikace vyžadující víceřádkové transakce s logickými požadavky jako je účetní systém.
- Při vyžadování ACID operací pro zachování 100% integrity dat.
- Lepší podpora vývojářů a osvědčená technologie s dobrými zkušenostmi.
- Rychlejší pro práci s nižším objemem dat, která jsou neměnná a strukturovaná.
- Pro analýzu relací a složité provádění dotazů.
- Použití pro transakční systémy, systém záznamů, finanční transakce, základní bankovní aplikace, správa objednávek, OLTP, ERP, CRM systémy, elektronický obchod atd.

NoSQL

- Lepší volba pro modernější aplikace s nesouvisejícími a vyvíjejícími se požadavky na údaje a často měnícími se a složitějšími sady dat.
- Ukládání velkých objemů dat bez struktury s podporou komprese.
- Pro rychlejší čtení nebo zápis dat.
- K ukládání dočasných dat, jako jsou nákupní vozíky nebo seznam přání.
- Ukládání do cloudových uložišť díky snadné škálovatelnosti.
- Rychlý vývoj a nástup na trh díky flexibilnímu datovému modelu, který nevyžaduje přípravu předdefinovaného schématu.
- Možnost ukládat a zpracovávat data v reálném čase.
- Použití pro správu obsahu, personalizace, vyhledávače, správu a personalizaci uživatelských profilů, datové toky, ukládání dokumentů, digitální komunikace (ukládání zpráv, chaty), velká data, analytiku, strojové učení, datové toky IoT atd.

V případě, kdy je v databázovém systému potřeba během transakcí zachovat záruky ACID typické pro relační databáze a zároveň zajistit škálovatelnost pro práci s velkým objemem dat, které relační databáze nezvládají, používají se NewSQL databáze. Jsou typické pro podnikové systémy, které jsou vytížené online transakcemi, jako jsou finanční systémy či zpracování objednávek. (13) (14)

2.2.8 Shrnuté rozdíly v tabulce

Tabulka 3: Rozdíly SQL a NoSQL databází

	SQL databáze	NoSQL databáze
Cíl vývoje	Vyvinuté v 70. letech 20. století s cílem řešit běžné ukládání dat	Vyvinuté v roce 2000 pro ukládání takových dat, které relační databáze neumožňovaly
Jazyk	Strukturovaný dotazovací jazyk SQL	Žádný či vlastní dotazovací jazyk
Schéma	Předdefinované schéma, jehož změna je pro systém náročná	Dynamické schéma, které lze měnit bez obtíží
Struktura	Tabulkové databáze, které ukládají data do řádků a sloupců	Různé technologie: key-value, grafové, dokumentové nebo sloupcové databáze
Škálovatelnost	Vertikálně rozšiřitelné specializovaným hardwarem a vysoce dostupnou sítí	Horizontálně rozšiřitelné komoditními servery i komoditní sítí
Hierarchie	Nejsou vhodné pro hierarchické ukládání dat	Díky metodě key-value se hodí pro hierarchické ukládání dat
Podpora	Velká komunitní i zákaznická podpora s řadou odborníků	Pouze komunitní podpora a externí odborníci
Transakce	ACID – kladen důraz především na konzistenci databáze	CAP – dává přednost rychlé dostupnosti dat před jejich konzistencí
Dotazy	Navrženo pro komplexní dotazy	Navrženo pro rychlé dotazování
Rychlost	S přibývajícími tabulkami se snižuje výkon databáze	Rozložení zatížení na více serverů pro lepší výkon
Náklady	Udržování špičkových systémů RDBMS je drahé	Levná údržba, nenáročné požadavky pro správu
Příklady	MySQL, SQLite, PostgreSQL	Redis, Cassandra, HBase, MongoDN, Neo4j, CouchDB

2.3 Samostatná produkční databáze nebo v kombinaci s analytickou databází?

Produkční databáze uchovávají data potřebná pro operační řízení, starají se o snadné ukládání dat v konzistentním prostředí a udržují je stále aktuální. Analytická databáze se hodí zejména pro analytické dotazování nad velkým množstvím dat. Vstupují do nich data z produkčních databází. Datové sklady kladou důraz na nahrávání dat v dávkách do logických celků a následném provádění složitých dotazů.

2.3.1 Srovnání účelů při využití technologií OLTP a OLAP

Pro ukládání dat do produkční databáze se využívá technologie OLTP, zatímco analytické databáze používají technologii OLAP. Rozdíl těchto technologií je v jejich orientaci na subjekt, sjednocení, proměnlivosti a stálosti dat.

- Orientace na subjekt: Relační produkční databáze se snaží o normalizaci dat do třetí normální formy tak, aby předcházela jejich redundanci, čímž se vyhne komplikacím v podobě nekonzistentní databáze, které by předcházely při změně redundantních dat. Analytická databáze klade důraz na uložení dat do přehledné struktury tak, aby bylo složitější dotazování na data provedeno co nejrychleji. Data jsou tak uložena do souvisejících celků nehledě na vznik redundantních záznamů a používají více indexů.
- Integrace a sjednocení dat: Produkční databáze ukládají data podle specifického okruhu, kdežto datový sklad shromažďuje data z různých zdrojů do logických celků tak, aby informace, které spolu funkčně souvisí, byly pro lepší dotazování u sebe.
- Časová proměnlivost dat: Produkční databáze je často vytížena zaznamenáváním transakcí a jejich modifikací, proto je i odpověď na složitější dotazy pomalejší. Do datových skladů jsou data nahrávána v off-line režimu ve větších dávkách v určitých časových intervalech podle potřeby analýz. Databáze tak odpovídá rychleji a v případě špatně zadaného dotazu neovlivní operativní řízení firmy. Data už většinou dále nejsou nijak upravována.
- Stálost dat: V běžné databázi jsou data stále aktuální, kdežto v datových skladech se data nemění, což je z analytického hlediska vhodné pro průběžné sledování a vývoje dat pro analýzu v čase.

Mezi špičkové analytické databáze využívající technologii OLAP se řadí např. HP Vertica, Teradata, Redshift nebo BigQuery. Open source databáze, které mohou posloužit jako datové sklady, jsou Apache Hadoop, MongoDB nebo PostgreSQL. (15)

2.4 Sloupcově orientované databázové systémy

Sloupcové databáze na rozdíl od tradičních relačních databází ukládají data podle sloupců. Počet sloupců se pro každý řádek může lišit, navíc každý řádek může obsahovat jiné sloupce s různou datovou strukturou. Každý sloupec obsahuje název, hodnotu a časové razítko, kdy byla data vložena. Při dotazování se na relační databáze se čtou celé řádky čili i sloupce, na které se dotaz neuplatní, kdežto sloupcové databáze načítají do cache pouze sloupec, se kterým se pracuje. Jelikož jsou dohromady serializovány hodnoty po sloupcích, jsou sloupcové databáze nejefektivnější pro operace náročné na dotazy nebo analytické operace.

Spíše než NoSQL databázím zaměřeným na key-value nebo dokumentově orientovaným databázím jsou sloupcové databáze velmi podobné relačním databázím. Mají podobnou strukturu, jsou s nimi dobře kompatibilní, při transakci zajišťují ACID operace a mají dotazovací jazyk podobný SQL. Nabízí ale lepší výkon.

2.4.1 Srovnání řádkových relačních databází se sloupcovými databázemi

V případě vytvoření různých typů adres u jednoho záznamu umožňují sloupcové databáze vytváření supersloupců složených z různých podsloupců nesoucí hodnoty. Pro každý záznam se tyto atributy mohou lišit. Do supersloupců lze přidávat další podsloupce nebo u záznamů celý sloupec s určitou adresou vynechat.

user_id (klíč řádku)	*název supersloupce*			*název supersloupce*			...
	název podsl.	*název podsl.*	...	*název podsl.*	*název podsl.*	...	...
	hodnota podsl.	*hodnota podsl.*	...	*hodnota podsl.*	*hodnota podsl.*	...	
1	home_address			work_address			
	city	street	...	city	street	...	
	Brno	Krásná 5	...	Praha	Pracovní 13	...	
4	home_address			temporary_address			
	city	street	...	city	PSČ		
	Plzeň	sady Pětatřicátníků 35	...	Praha	111 00		
...							

Obrázek 2: Vzor způsobu ukládání dat d sloupcově orientovaných databázových systémů

V případě, kdy by bylo potřeba u relační databáze vytvořit k záznamu více typů adres jako home_adress, work_adress nebo temporary_adress, musely by vzniknout atributy jako home_city, home_street, work_city, work_street, temporary_city, temporary_street, které by v případě nevyplnění u určitých záznamů obsahovaly hodnoty null. Dalším způsobem je vytvoření tabulky s adresami, které by nesly informaci o tom, zda se jedná o home_adress, work_adress nebo temporary_adress, což by v případě vyhledávání určitého typu adresy bylo podmínkou dotazu. (16)

Sloupcové databáze se také více hodí pro řídká data, která mají mnoho volitelných sloupců. U relačních databází by bylo potřeba vytvořit všechny požadované sloupce pro každý řádek a ty bez hodnoty naplnit hodnotou null, která by poté byla v databázi fyzicky uložena a zabírala další místo. Sloupcové databáze umí dobře pracovat s nedefinovanými hodnotami, jelikož se při zavolání atributu, který není u daného řádku definován, objeví hodnota null sama, aniž by byla někde uložena. Pro dosáhnutí dynamičnosti sloupců lze pomocí kolekcí, jako jsou set (množina), list (seznam) nebo map (asociativní pole).

```
10: 001,12: 002,11: 003,22: 004;
Smith: 001, Jones: 002, Johnson: 003, Jones: 004;
Joe: 001, Mary: 002, Cathy: 003, Bob: 004;
60000: 001,80000: 002,94000: 003,55000: 004;
```

Obrázek 3: Způsob serializace hodnot sloupce databází orientovaných na sloupce

```
…; Smith: 001; Jones: 002 004 ; Johnson: 003;…
```

Obrázek 4: Komprimace dvou row_id k jedné položce

Ve společné úpravě stejného úložiště jsou položky „Jones“ komprimovány do jediné položky se dvěma row_id. Všechny položky se stejnou hodnotou tohoto klíče tvoří jeden logický celek a jsou tak umístěny na stejném uzlu systému, jelikož se předpokládá, že většina dotazů se týkat záznamů se stejným row_id. Dotazy spadající do jedné rodiny záznamů, které spolu logicky souvisí a mají stejný typ, jsou vyhodnocovány efektivně.

Díky svojí struktuře dokáží sloupcové databáze rychle provádět agregační dotazy jako SUM, AVG nebo COUNT, protože jsou tyto hodnoty uloženy na jednom místě na rozdíl od relačních databází, které k získání hodnot těchto sloupců musí projít celý řádek záznamu.

U relačních databází probíhá komprese po řádcích, kde se v jednom bloku míchají různé typy a různé charaktery dat. U sloupcových databází je komprese probíhající po sloupcích se stejným datovým typem mnohem efektivnější, proto patří databáze orientované na sloupce k databázím s vysokou kompresí.

Díky ukládání dat do sloupců mají databáze lineární a modulární škálovatelnost, proto mohou ukládat i velké objemy dat. Sloupcové databáze mají dobrou odolnost vůči chybám jak na komoditním hardwaru, tak i na cloudové infrastruktuře, proto replikace napříč více datovými centry poskytují minimální latenci i během regionálních výpadků.

Zda je efektivnější řádková nebo sloupcová databáze v provozu do značné míry závisí na automatizované pracovní zátěži. U relačních databází zatížených interaktivními transakcemi, kdy je potřeba načtení všech dat z jednoho řádku, je pro minimalizaci vyhledávání na disku efektivnější, když jsou tato data umístěna na jednom místě, čímž jsou databáze orientované na řádky rychlejší. Sloupcové databáze jsou vhodné jako pracovní zátěž pro datové sklady sloužící k analýze v řádu až petabajtů dat, ale byly vyvinuty také jako hybridy, které jsou schopny zpracovávat OLTP i OLAP operace. (17) (18) (19) (20)

2.5 Možnost ukládání obrázků přímo do databáze

Zahrnutí obrázků či jiných souborů do databáze se obvykle řeší vytvořením dalšího atributu s textovým polem, kam se vkládá odkaz či cesta k daným obrázkům. Obrázky většinou bývají uloženy v souborovém systému či se průběžně ukládají na FTP server, odkud se nejprve musí ručně zpracovat. Obrázky je potřeba přesunout do souborového systému, zkomprimovat a následně doplňovat do databáze odkazy či názvy obrázků. Tento způsob řešení tak bývá časově náročnější a vyžaduje pravidelnou synchronizaci záznamů v databázi se souborovým systémem, aby byla zachována konzistence dat.

Druhou variantou je zahrnutí obrázků přímo do databázových tabulek, což s sebou nese řadu výhod. Data jsou spravována databází, zůstávají konzistentní a umožňují uzamknout metadata do obrázků, dají se snadno zálohovat, obnovit či zabezpečit spolu s ostatními záznamy v řádku, mají lepší verzování a žádná omezení souborového systému. Pro ukládání multimediálních položek, jako jsou obrázky, videa, audia a další soubory, se využívá datový typ BLOB (binárně velký objekt), který je součástí většiny databází. BLOB na rozdíl od ostatních datových typů, které uložené data konvertují do určité kódové stránky, ukládá data přesně tak, jak byly binárně uloženy do databáze, a stará se sám o jejich interpretaci.

Ukládání binárně velkých objektů do databáze může značně zpomalit výkon databáze a dojít až k velkému zatížení aplikace, která např. spravuje data z výroby. Databázový server tak musí ukládat velké množství dat, přitom by byl lépe využit ke zpracování vhodnějších a důležitějších dat. Proto je pro zachování výkonu vhodnější do databáze ukládat pouze objekty do pár kilobajtů a soubory o velikosti několika megabajtů už ukládat do souborového systému.

Nejčastěji se jedná o snímky z průmyslových kamer pořízené během výroby pro kontrolu správnosti určitých procesů. Průmyslové kamery mají za úkol nepřetržitě sledovat určený prostor a v případě, kdy výrobek, který běží po lince, zastaví ve stanici, udělat jeho snímek, který následně pošlou do záznamového zařízení, kde dojde k ověření správnosti. Obrazová data jsou obvykle komprimována do formátů se ztrátovou kompresí jako MPEG nebo H.264 určené pro kódování pohyblivých obrazů. (21)

2.5.1 Konverze obrázků a omezení jejich velikosti

Pokud už je ale potřeba mít uložené obrázky přímo v databázi tak, aby jejich uložení, zpracování a načtení minimálně omezovalo výkon databáze, určitě by stálo za to uvažovat nad nějakou variantou úpravy obrázků před jejich vložením do databáze. Pro pouhé nahlédnutí na obrázek pro lepší získání představy či uložení obrázku, u kterého není potřeba klást důraz na každý jeho pixel, se nabízí varianty jako omezení velikosti obrázku či jeho konverze.

Pro kvalitní fotografie se používá formát JPEG, který má přes 16 milionů barev, kde lze každou barvu vyjádřit jako kombinaci tří složek – červená, modrá, zelená v zastoupení každé barvy v rozmezí 0 až 255, proto je na vyjádření jedné libovolné barvy zapotřebí 24 bitů. To se značně projeví na jeho velikosti, navíc při velké kompresi se rozmazává a dochází ke ztrátě zobrazovaných dat. Na rozdíl od něj má bezztrátovou kompresi formát PNG, u kterého lze využít opět přes 16 milionů barev pro fotografie či 256 barev pro zobrazení jednoduchých ikon a tím zajistit nižší velikost obrázku.

Pro získání nejnižší velikosti obrázku, který bude minimálně zatěžovat databázi, se nejvhodnějším řešením nabízí obrázkové soubory před vložením do databáze konvertovat do formátu GIF. Má pouze 256 barev, čili pro jeden pixel je využito pouze 8 bitů. Velikost výsledného obrázku se tak pohybuje v řádu jednotek kilobajtů. Navíc po kompresi zůstává kvalita obrázku stále stejná a nedochází ke ztrátě zobrazovaných dat. Pro rychlé nahlédnutí na obrázek, či přidání jednoduchých grafických objektů do databáze je GIF ideálním řešením. (22)

3 Vybrané databáze k testování a jejich implementace

Na základě kritérií by pro databáze byly výhodou tolerantnější licenční podmínky jako tříbodová BSD licence nebo Apache Licence 2.0. Databáze, které jsou vyvíjené komunitou a nemají restriktivnější licenční podmínky, často bývají NoSQL databáze. Při zpracování velkého objemu dat se dají levně škálovat, což se projeví i na lepším výkonu zpracování dat. NoSQL databáze mohou data ukládat do různých datových struktur a ukládat objekty jako jsou dokumenty, grafy atd. Ukládání do sloupcové struktury u nerelačních databází bývá poněkud rychlejší než u řádkových relačních databází. Při výběru správné varianty databáze je velmi důležité porozumět jejímu případu použití.

3.1 Cassandra

Apache Cassandra spadá do sloupcové rodiny databází, jejímž záměrem je fyzické ukládání všech dat do sloupců, čímž se minimalizuje doba jejich vyhledávání. Byla navržena především pro určitý případ použití, a to vypořádání se velkými objemy dat v co nejlepším čase. Jedná se o nejrychlejší open source databázi s extrémně rychlým výkonem pro zápis velkého množství dat. Vyniká neomezenou lineární škálovatelností, čímž je možné do klastru přidávat libovolné množství uzlů, aniž by docházelo ke zvýšení složitosti klastru. Její dotazovací jazyk CQL (Cassandra Query Language) je velmi podobný SQL, čímž odpadá nutnost učit se nový dotazovací jazyk. Uplatnění najde především v protokolování, analytice a zpracovávání stovek petabajtů dat denně. Cassandra byla původně vyvinuta pro Facebook v roce 2008, ale o 2 roky později se stala jedním z vrcholových projektů a dnes na ni spoléhají společnosti jako Reddit, Twitter nebo Cisco. Spadá mezi databáze využívající CAP teorém, proto se nehodí použít v systémech, které požadují vysokou konzistenci dat. Na rozdíl od ostatních vysoce výkonných databází je charakteristickou vlastností databáze Cassandra její dostupnost bez licenčních poplatků. Pro maximální využití všech předností této databáze je potřeba znát velmi dobře všechny možnosti a funkce Cassandry. (23)

Při implementaci Apache Cassandra na Raspberry Pi je potřeba dodržet hned několik zásad. Tou první je problém v tom, že Cassandra je stavěná na zařízení s lepším výkonem, aby byl zajištěn její plynulý provoz. Oficiální balíček Cassandra nepodporuje architekturu „armhf“, která se v Raspberry Pi 2 používá, proto je databázi potřeba nainstalovat přímo z tarballu, neboli balíčku, ve kterém jsou archivovány potřebné soubory pro konfiguraci a běh databáze.

Jelikož je Cassandra napsaná v jazyce Java, je pro její spuštění zapotřebí mít nainstalovanou a defaultně nastavenou Javu verze 8, nikoli vyšší. Vzhlede k tomu, že pro komunikaci s databázemi byl zvolen programovací jazyk Python, je opět zapotřebí pro práci s databází Cassandra dodržet verzi Pythonu, se kterou je kompatibilní, čímž je Python verze do 2.7. Ten pro komunikaci s databází Cassandra používá vysoce laděnou a funkční knihovnu Cassandra-Driver.

3.2 HBase

Apache HBase je distribuované uložiště dat orientované na sloupce. To znamená, že data jsou uložena po jednotlivých sloupcích a indexována klíčem řádku. Díky této architektuře je skenování přes jednotlivé sloupce v tabulce efektivnější. Veškerá data a požadavky na ně jsou uložena na všech serverech klastru HBase, což umožňuje dotazovat výsledky během milisekund s minimální latencí. K vlastnostem HBase také patří vysoká škálovatelnost a tolerance k chybám.

Databázi Apache HBase je možné implementovat ve třech různých režimech v závislosti na požadavcích na databázi. První možností je instalace samotného režimu, který není závislý na systému Hadoop a funguje v lokálním systému souborů, pro spuštění využívá pouze deamona HMaster a běží v jednom JVM. Další možností je instalace pseudo-distribuovaného režimu, který běží na jednom uzlu Hadoop a na rozdíl od předchozího řešení je už vhodný pro výrobní prostředí. Poslední, nejvhodnější variantou pro výrobní prostředí, je instalace plně distribuovaného režimu HBase, který běží na systému Hadoop a všechny deamony využívají veškeré uzly přítomné v clusteru, proto dokáží rychle zpracovávat velké objemy dat. Jelikož testování probíhalo na zařízení s nízkým výkonem, byl instalován pouze samostatný režim HBase bez využití systému Hadoop. HBase nepodporuje strukturovaný dotazovací jazyk, ale dotazování na data probíhá pomocí implementace JRuby, která je napsaná v Jave a umožňuje komunikaci programovacího jazyka Ruby na JVM. Pro vysoce výkonnou koordinaci HBase spoléhá na ZooKeeper, který poskytuje informace o konfiguraci, pojmenování, synchronizaci a skupinových službách pro usnadnění správy HBase. Programovací jazyk Python používá pro komunikaci s databází HBase vývojářskou knihovnu HappyBase, které se připojuje k databázi pomocí brány Thrift.

3.3 MariaDB

Poté, co v roce 2010 převzala MySQL společnost Oracle, byl zahájen nový projekt, jehož základem se staly původní zdrojové kódy MySQL. Vytvoření forku neboli rozvětvení těchto kódů vedlo k vytvoření nové databáze MariaDB s otevřeným zdrojovým kódem. Za tu dobu byla MariaDB doplněna o mnoho nových funkcí usnadňující práci a vylepšující výkon oproti původní MySQL. Takovým příkladem může být Spider engine pro práci s transakcemi nebo podpora ColumnStore pro masivní ukládání dat. Instalace MariaDB pro lokální použití je velmi snadná a hotová během pár minut.

3.4 PostgreSQL

PostgreSQL je správnou volbou pro ty, kteří kromě ukládání dat do relačního datového modelu budou chtít ukládat i specifické požadavky. Podporuje totiž i ukládání XML, key-value nebo dokumentů ve stylu JSON. To se ale také promítá v její náročnosti na čtení dat v náročném schématu. Ovšem primárně je PostgreSQL relační databází, proto při ukládání velkých datových sad jako dokumentů se její struktura může začít rozpadat. Python pro komunikaci s PostgreSQL používá databázový adaptér Psycopg.

3.5 MongoDB

NoSQL databáze MongoDB na rozdíl od relačních nebo sloupcových databází ukládá data pomocí polostrukturovaného schématu jako dokumenty ve formě JSON, kdy jsou související data společně shlukována do stejných bloků. Všechna data, která souvisí s určitým objektem, jsou uložená přímo uvnitř objektu, nikoli v tabulce, se kterou by jinak byla propojena, proto načtení jednoho objektu neboli dokumentu automaticky načte i všechna data uvnitř. Díky tomuto způsobu ukládání lze rychle provádět změny ve struktuře. S tím ale také přichází nevýhody v podobě datového skladu s velkým množstvím neshodných dat v případě, kdy uživatel nerozumí přesné struktuře.

Při implementaci databáze MongoDB na Raspberry Pi lze použít pouze starší verzi 2.4.14, protože novější verze vyžadují architekturu ARM64, ale Raspbian má pouze 32bitový operační systém. Tento požadavek s sebou hned přináší problém v podobě komunikace Pythonu s databází, kdy je použit interaktivní nástroj PyMongo. Ten ale podporuje MongoDB verze 2.6 a vyšší. Naštěstí první verze tohoto modulu je kompatibilní i s verzí 2.4.

4 Způsoby testování databází a příprava testů

Pro připojení k databázím a přípravě testů byl zvolen programovací jazyk Python verze 3.7.5, pro databázi Cassandra pak Python verze 2.7.5., jelikož vyšší verzi nepodporovala. Dále byla přidána Java 8 JDK a sada knihoven, které umožňují propojení programovacího jazyka Python s jednotlivými databázemi.

4.1 Platforma Raspberry Pi 2

Testy probíhaly na platformě Raspberry Pi 2 Model B, která vstoupila na trh roku 2015. Tento jednočipový počítač používá mikroprocesory z rodiny ARM, výstup pro monitor HDMI a USB porty pro připojení klávesnice a myši. Přesné parametry hardwaru u Raspberry Pi 2 Modelu B jsou uvedeny v tabulce.

Tabulka 4: Hardware pro Raspberry Pi 2 Model B

Architektura	ARMv7 (32-bit)
SoC	Broadcom BCM2836
Procesor	32-bitový čtyřjádrový procesor ARM Cortex-A7 s taktem 900 MHz
GPU	Broadcom VideoCore IV, podporující OpenGL ES 2.0, 1080p30, MPEG-4
Paměť	1 GB RAM sdílená s GPU
Porty	Čtyři porty USB 2.0, MicroUSB port, Full Size HDMI, kompozitní video, displej rozhraní MIPI (DSI), MIPI kamerový CSI konektor
Uložiště	Micro SD slot
Síť	Ethernetový adaptér 10/100 Mbit/s s konektorem RJ-45
GPIO	40 pinové vývody GPIO

Standartním operačním systémem pro Raspberry Pi je Raspbian odvozený od linuxové distribuce Debian. Raspberry Pi 2 umožňuje používat i další alternativní operační systémy, kam spadají různé distribuce Linuxu, operační systém NetBSD nebo Microsoft Windows 10 IoT Core bez grafického uživatelské rozhraní ovládaný pouze přes příkazový řádek. Pro testování byl použit původní operační systém Raspbian.

4.2 Způsoby testování databází

Testování se provádí v několika krocích:

1. Plánování zátěžových testů shromážděním systémových dat, analýzou systému a definování cílů zátěžových testů.
2. Vytvoření skriptů a vygenerování testovacích dat.
3. Spuštění skriptů a uložení výsledku zátěžových testů.
4. Analýza výsledků a nalezení slabších míst systému.
5. Vyladění systému změnou konfigurace či optimalizací kódu pro splňování požadavků.

4.2.1 Zátěžové testování

Zátěžové testování se provádí při testování výkonu systému během běžného zatížení ke zjištění chování aplikace při přístupu více uživatelů zároveň. Obvykle se provádí před ostrým provozem aplikace, aby se zjistilo, zda je pro reálný provoz dostačující. Cílem je ověřit dobu odezvy pro transakce, výkon při různých zatížených nebo problémy s hardwarovým omezením (CPU, paměť,…). (24)

4.2.2 Stresové testování

Stresové testování se používá především pro ověření stability a spolehlivosti systému. Snaží se o testování při extrémních zátěžových podmínkách a provádí se až do selhání systému. Tento bod zlomu má ukázat, kolik systém zpracuje dat před selháním, jak velký provoz zvládne a zda se účinně dokáže zotavit po selhání. Součástí testování je pak příslušná chybová zpráva, která má analyzovat chování systému v době selhání. Příkladem stresového testování může být snaha o přizpůsobení se neobvyklým zátěžím, jako je online prodej lístků na koncert v době specifického času, kdy je zaznamenán velký nárůst provozu na stránce. (25)

4.2.3 Testování stability

Testování stability se provádí za účelem kontroly, zda databáze bude schopná pokračovat v činnosti velmi dlouhou dobu bez jejího selhání. Aplikace, které takto běží i dlouhé měsíce, se mohou časem zpomalit, narazit na problémy s funkčností nebo se jim zhroutí celý systém. (26)

4.2.4 Testování škálovatelnosti

Při testování škálovatelnosti se měří výkon sítě nebo systému, během kterého se zvyšují nebo snižují požadavky uživatelů. Účelem tohoto testování je zajistit, aby aplikace zvládla zvýšený počet uživatelů na sítí či zpracování většího objemu dat nebo počtu transakcí. Testování škálovatelnosti odhalí přizpůsobení databáze na rostoucí pracovní zatížení a zjistí uživatelský limit. Test spadající pod škálovatelnost především měří okamžik, během kterého přestane aplikace škálovat, a identifikování příčiny. Mezi parametry, které se při měření škálovatelnosti sledují, jsou doba odezvy, propustnost, čas provedení určitých relací nebo měření výkonu s počtem uživatelů. Při testování škálovatelnosti se předpokládá, zda software testovaný na zátěž je schopný rozložit zátěž na více strojů. (26)

4.2.5 Testování objemu

Testováním objemu se zjišťuje výkon databáze vystavené velkému množství dat. Lze studovat chování systému a dobu jeho odezvy. Tímto typem testování se zjistí, zda je databáze schopna běžet v reálném světě, nebo zda existují úzká místa, na kterých by se databáze zpomalila. S postupným zvyšováním objemu dat je možné kontrolovat výkon systému, identifikovat problém, který by nastal u velkého množství dat, zjistit bod, ve kterém se zhoršuje stabilita systému, nebo určit maximální kapacitu databáze. (27)

4.2.6 Vytrvalostní testování

Vytrvalostní testování je posledním testem aplikace při testování jejího výkonu. Používá se pro vyhodnocení chování systému se zátěží po dlouhou dobu běhu, aby se zajistila dostatečná schopnost aplikace zvládnout větší zatížení bez zkrácené doby odezvy. Průběh tohoto testování může trvat i roky. Cílem je zjistit fungování systému při trvalém používání, kontrolovat únik paměti a čas, během kterého zůstane doba odezvy stejná jako na začátku. (28)

4.2.7 Testování komprese

Při testování komprese databáze je třeba se zaměřit na velikost kompresního bloku. Jelikož platí, že čím větší kompresní blok, tím lepší kompresní poměr a optimalizace pro větší pracovní zátěž. Velikost komprese také ovlivňují různé datové typy. Komprimace dat často probíhá na pozadí databáze, kdežto dekomprimace se provádí v popředí, což může zvyšovat latenci.

4.3 Typy testů pro testování výkonu databáze

Pro získání validního a relevantního výsledku je před každým spuštěním zátěžového testu potřeba provádět testování na jednotném základu, tj. prázdné databázi implementované na čistý operační systém bez dalších procesů spuštěných na pozadí. To se zajistí vytvořením image operační systému po jeho instalaci a konfiguraci. Tento image se použije pokaždé, kdy bude testována nová databáze. Tak se zajistí stejné podmínky pro testování každé databáze a tím i validní výsledky. Pro všechny databáze budou použity stejné datové sety, které se do databáze budou vkládat skriptem. Jednotlivé kroky je třeba pak kontrolovat přes logovací soubory pro odhalení případných problému během vykonávání skriptu. Pro zátěžové testy je možné i připravit určitý počet virtuálních uživatelů, délku běhu testu a samozřejmě i desetitisíce záznamů, které naplní databází v řádu několika jednotek až desítek MB dat.

Pro jednotlivé databáze bylo navrženo sedm různých typů testů, které měly za úkol zjistit především čas potřebný k uložení dat, spočítat počet zpracovaných operací za sekundu, čas potřebný k provedení různých dotazů na databázi a porovnání funkcí pro komprese dat. Pro prozkoumání času na dotazování na data byly připraveny tři různé dotazy. První vybíral všechna data v tabulce, tudíž se testovala rychlost, kdy si databáze stihne připravit data z tabulky do struktury vhodné k vypsání. Druhým typem byly dotazy na určité sloupce v tabulce. Zde by měl být vidět rozdíl zpracování dotazu mezi sloupcovými a relačními databázemi. Jelikož databáze orientované na sloupec ukládají data po sloupcích, ze kterých následně tvoří logické celky, kdy jsou data uložena fyzicky u sebe, mělo by být dotazování na sloupce u těchto databází rychlejší na rozdíl od relačních databází, které při dotazování na konkrétní sloupce musí pokaždé načíst všechny řádky, ve kterých postupně vyhledávají požadované hodnoty atributů. Posledním dotazem je obtížnější dotaz s podmínkami, který umožňují zpracovat i NoSQL databáze.

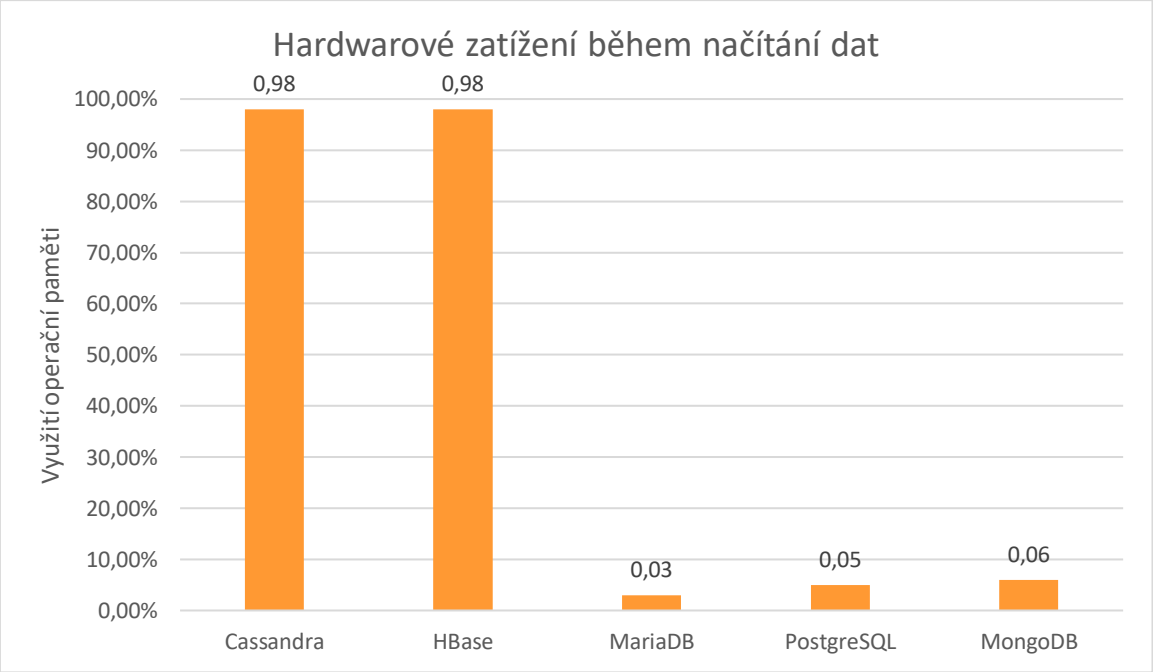
5 Výsledky a návrh řešení

Pro otestování rychlosti načtení dat a zpracování dotazů jednotlivých databází byl využit 10MB soubor, který obsahoval 80 125 záznamů a 16 atributů různých datových typů. Každý jednotlivý test byl na jedné databázi proveden dvanáctkrát a výsledky byly uloženy do log souboru každé databáze. Z těchto 12 hodnot byla odstraněna maximální a minimální hodnota jako odchylka a zbylé hodnoty byly zprůměrovány k dosažení co nejpřesnějších výsledků.

5.1 Načítání dat do databáze

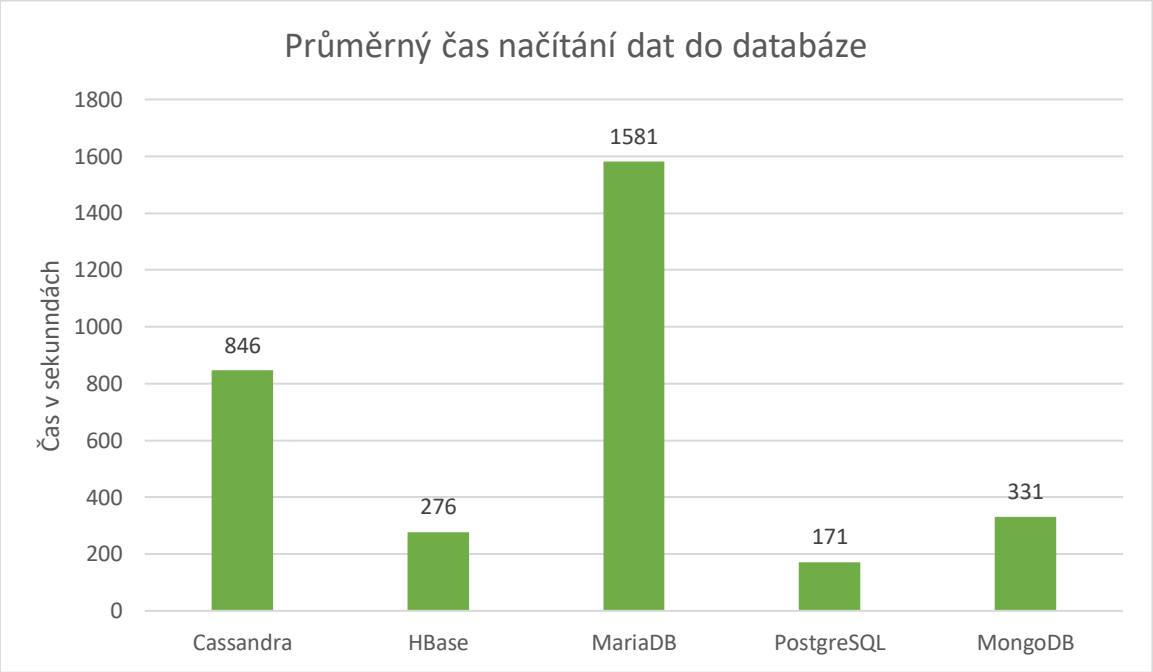
Pro spuštění a běh databází MariaDB, PostgreSQL a MongoDB bylo zapotřebí pouze pár jednotek MB operační paměti RAM a nedocházelo k zatěžování systému. Při zápisu dat do databáze tak byl pro jejich zpracování využit pouze omezený výkon i při potřebě uložit velké množství dat. Tyto databáze se vždy snaží o konzistentní zapsání dat bez ohledu na to, jak dlouho jejich zpracování potrvá.

Datové sklady jako Apache Cassandra nebo Apache HBase využívají k co nejrychlejšímu ukládání dat do databáze maximální dostupný výkon. Při testování těchto dvou databází na Raspberry Pi 2 s RAM o velikosti 1 GB sdíleném s GPU bylo už jen při spuštění samotných databází, deamonů pro připojení k databázi a frameworků potřebných pro komunikaci s programovacím jazykem využito téměř veškeré operační paměti, které Raspberry Pi nabízí. Při následném spuštění testů byly databáze značně omezeny výkonem hardwaru a v mnoha případech docházelo ke zpomalení nebo přerušení právě vykonávaného testu z důvodu nedostatku operační paměti. Zatížení hardwaru při načítání dat do jednotlivých databází je znázorněno na Graf 1.



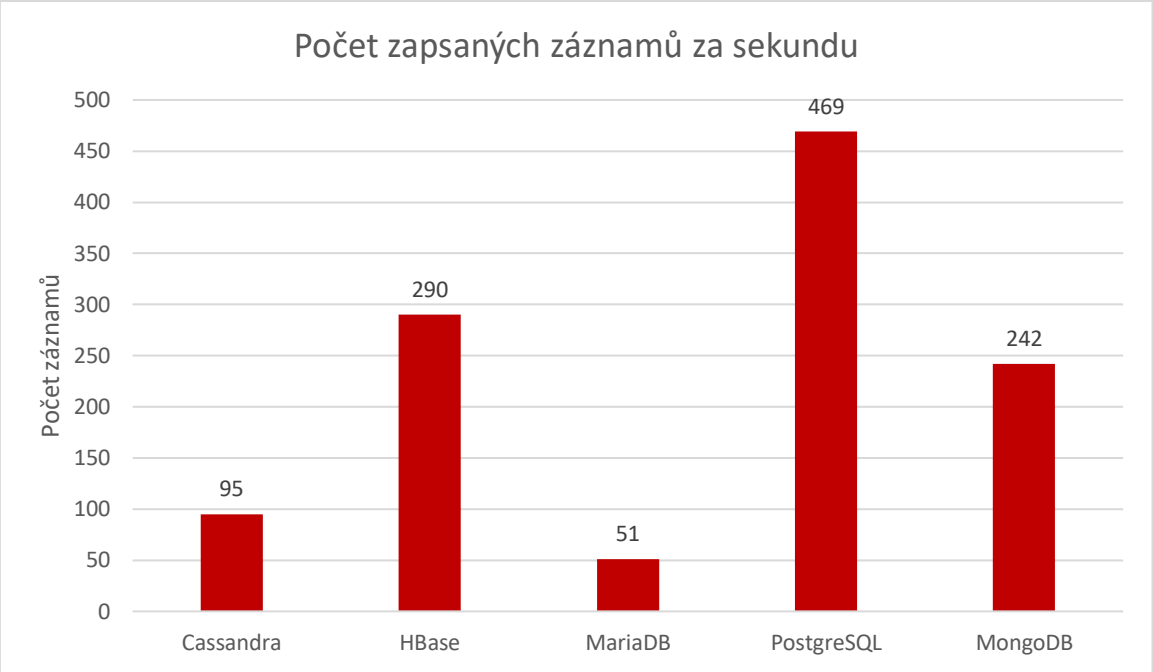
Graf 1: Hardwarové zatížení Raspberry Pi během načítání dat do databáze

Při zpracování datového setu o velikosti 10 MB, který byl použit pro testování všech databází, se databázím Cassandra ani HBase nepovedlo ani jednou zpracovat všechna data a testování končilo přerušením spojení s databází z důvodu nedostatku paměti. Pro přibližnou představu rychlosti načítání dat byl pro tyto dvě databáze použit soubor desetkrát menší, na jehož zpracování těmto dvěma databázím již postačila operační paměť. Výsledek v podobě počtu sekund potřebných pro zpracování 1MB souboru byl následně vynásoben deseti pro přibližnou představu trvání v případě, kdy by tyto databáze měly dostatek paměti ke zpracování původního 10MB souboru. Ovšem tento vynásobený čas se nemůže považovat za validní výsledek, ale slouží pouze pro přibližné porovnání s rychlostí načtení dat na rozdíl od ostatních databází. Výsledky jsou zobrazeny v Graf 2.



Graf 2: Průměrný čas v sekundách potřebný k uložení dat do databáze

Cassandra a HBase sice patří k databázím, které dokáží zpracovat velké množství dat mnohonásobně rychleji než běžné relační databáze, ovšem potřebují k tomu mít k dispozici dostatečný výkon. Se zvyšujícím se počtem dat mohou provádět operace rychleji. Avšak zpracování malého množství dat může NoSQL sloupcovým databázím trvat déle než běžným relačním databázím. Jelikož testování databází na rychlost ukládání dat bylo testováno na souboru s daty o velikosti pouze 10 MB a na hardwaru s velmi omezeným výkonem, nebyla možnost nechat databáze Cassandra a HBase ukázat svůj výkon při velkém pracovním zatížení, proto jejich výsledky nejsou zrovna nejoptimálnější.

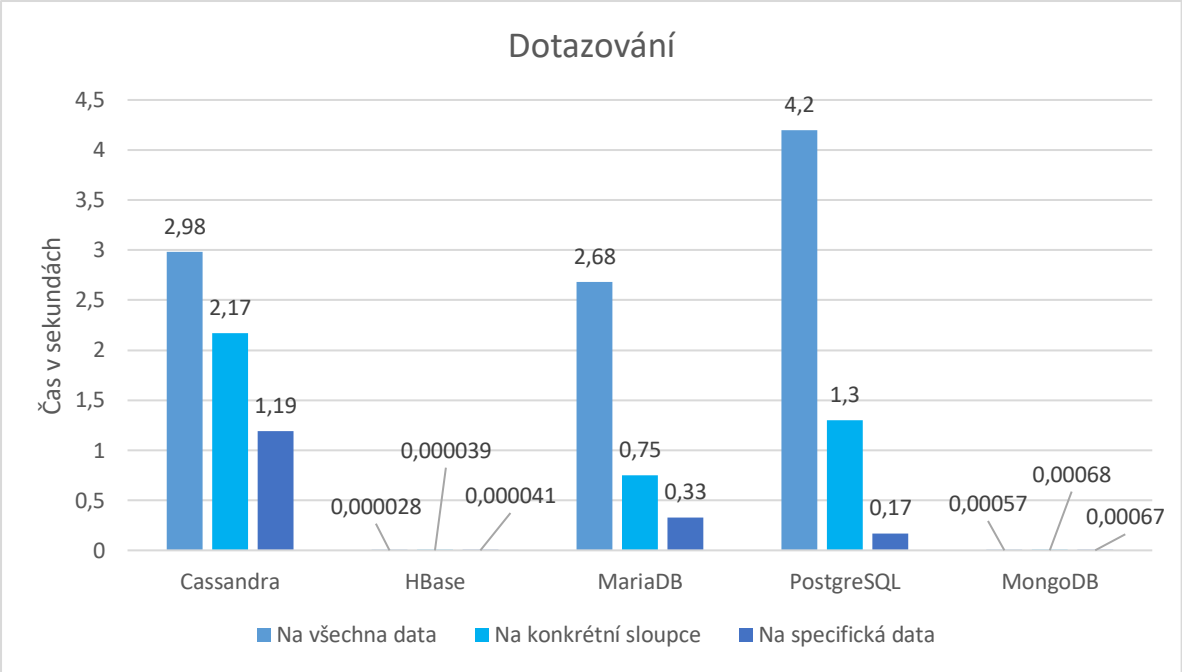


Graf 3: Počet záznamů uložených do jednotlivých databází během sekundy

5.2 Zpracování dotazů

Hlavním úkolem databáze Cassandra je především rychlé uložení a následně rychlé vyhledání základních dat pomocí snadných dotazů. Oproti tomu mají SQL databáze výhodu ve snadnějším a rychlejším zpracování komplexnějších dotazů, jako jsou operace nad kartézským součinem JOIN a dalšími podobnými dotazy. Cassandra podobné dotazy neumí, ale využívá řadu dalších snadno dostupných externích nástrojů, které kompenzují tyto nedostatky v dotazování. Z Graf 4 je patrné, že v případě, kdy probíhá složitější dotaz na specifická data, Cassandra zaostává v rychlosti zpracování dotazu na rozdíl od databází MariaDB a PostgreSQL s dotazovacím jazykem SQL.

Pro databáze HBase a MongoDB je typické, že dotaz zpracují velmi rychle, ale pokud dojde k latenci (zpomalení), přičemž musí okamžitě zpracovat dotaz, vrátí na dotaz raději chybnou hlášku, než aby narušili konzistenci dat. V Graf 4 jsou rychlosti zpracování dotazů databází HBase a MongoDB znázorněny jako jedny z nejnižších. Oproti tomu se databáze Cassandra snaží o co nejrychlejší zpracování dotazu a zajištění jeho dostupnosti a to i s rizikem nesrovnalosti dat bez záruk jejich konzistence. Relační databáze MariaDB a PostgreSQL vrátí na dotaz vždy správnou odpověď, ovšem pokud nastane určitá latence, bude systém pokračovat až po zajištění správnosti dat.



Graf 4: Čas v sekundách potřebný ke zpracování různých typů dotazů jednotlivých databází

5.3 Komprese dat

Výhodou komprese je snížení velikosti dat uložených v databázi i o třetinu původní velikosti. Čtení dat pak probíhá mnohem rychleji, protože fyzické množství dat, které je potřeba přesunout z disku na paměť, je rychlejší. Na druhou stranu je ale potřeba data dekomprimovat. Míra komprese závisí na typu dat, kdy je např. obtížné komprimovat datový typ float a naopak velmi jednoduché zkomprimovat anglický text.

Komprese se nevyplatí na příliš malé množství dat, kdy z důvodu vytvoření slovníku pro správu kompresních klíčových slov mohou být komprimované soubory větší než data nekomprimované.

Pro kompresi NoSQL databází Cassandra a HBase se vhodným řešením nabízí knihovny Snappy nebo LZ4. Pro MongoDB dobře poskytuje kompresi FlashArray. PostgreSQL využívá k zmenšení fyzického objemu dat TOAST. Databázi MariaDB je možné komprimovat ve třech podobách – po řádcích pomocí COMPRESSED, po sloupcích díky funkci ColumnStore nebo jako celá stránka díky InnoDB.

5.4 Vnější faktory ovlivňující výkon databáze

Samotné testování databáze může být ovlivněno řadou dalších faktorů. Jedním z nich je i nedostatečný výkon hardwaru na provedení plnohodnotných testů, což je i případ testování databází Cassandra a HBase na platformě Raspberry Pi. Testování databází na výkonnějším zařízení by zobrazilo i jiné výsledky. Existují speciální disky (SAS) určené přímo pro databáze a serverová disková pole, které zvyšují výkon.

Dalším rozdílem je způsob, kterým Python komunikuje s databází. Pro komunikaci s každou databází používá odlišnou knihovnu, model či rozhraní, které se liší i v rychlosti připojení k databázi a následně konverzi dotazů z prostředí Pythonu přímo na syntaxi databáze.

Každá databáze zpracovává různé objemy dat jinou rychlost. Zatímco zápis malého množství dat bude rychleji probíhat do SQL databáze, NoSQL databáze může být při zápisu pomalejší. To se ovšem mění se zvyšujícím se objemem dat, kdy NoSQL databáze orientované na sloupce postupně výkonově předbíhají relační databáze.

Sloupcově orientované databáze jsou rychlé při zpracování sloupcových dotazů. Relační databáze mohou svou rychlost získávání dat z konkrétních sloupců vylepšit přidáním indexů na sloupce. Indexování využívá hashovacích algoritmů pro rychlejší získání hodnot ze sloupců.

Dalším faktorem ovlivňující výkon databáze je i rychlost připojení k serveru, kdy jinak velmi výkonnou databázi může značně zpomalovat slabé připojení.

Komprese je sice způsob jak zvýšit místo na disku, ale při dotazování na data, která jsou archivovaná, dochází k mírnému zpoždění, jelikož je potřeba archiv dat nejdřív rozbalit. To je důvod, proč by data, se kterýma se aktivně pracuje, neměla být komprimována. Kdežto pro data, která jsou uložena v datovém skladu a slouží pouze pro analýzu, je komprimace vhodným řešením.

Závěr

Cílem bakalářské práce bylo implementovat zvolené databáze na vybraný hardware a operační systém, otestovat jejich výkon pomocí vlastních navržených zátěžových testů připravených ve zvoleném programovacím jazyce, porovnat výsledky z validity log souboru a navrhnout a zdůvodnit vybraný databázový systém.

Nejprve jsem prozkoumala běžně dostupné databázové systémy s permisívnějšími licenčními podmínkami, porovnala databáze SQL s NoSQL a vyzdvihla rozdíly mezi relačními databázemi a databázemi orientovanými na sloupce. Na základě této analýzy jsem zvolila pět open source databází, kterými jsou Cassandra, HBase, MariaDB, PostgreSQL a MongoDB, které jsem následně implementovala na zařízení Raspberry Pi s operačním systémem Raspbian. Připravila jsem sadu testů napsanou v programovacím jazyce Python a vytvořila validity log soubor pro ukládání výsledků zátěžových testů prováděných na vybraných databázích.

Během provádění zátěžových testů na sloupcově orientovaných databázích Cassandra a HBase jsem narazila na problém s nedostatečnou operační pamětí na zařízení Raspberry Pi, proto nebylo možné tyto testy uskutečnit s připraveným datovým setem najednou. Pro simulaci přibližných výsledků jsem musela použít desetkrát menší datové sety, při jejichž postupném ukládání do databáze byla operační paměť dostačující. Výsledky jsem následně opět desetkrát vynásobila, abych získala přibližnou hodnotu výsledků z důvodu porovnání hodnot s ostatními databázemi. Ovšem tyto hodnoty již nebyly plně validní, jelikož některé databáze dosahují s přibývajícím množstvím dat i vyšší rychlosti ukládání dat do databáze. Vzhledem k tomu, že Cassandra a HBase patří mezi nejvýkonnějších dostupné databáze, byly předem považovány za databáze s nejlepšími hodnotami při provádění zátěžových testů. Ovšem tyto dvě databáze byly značně omezeny výkonem hardwaru, proto se databází, s nejrychlejším ukládáním záznamů, stala PostgreSQL. Tento databázový systém ale nepovažuji za finální výsledek této práce. Jelikož byl výkon použitého hardwaru nedostatečný pro cílové užití databází, bude potřeba do budoucna tyto databázové systémy opět otestovat na silnějším hardwaru pro získání přesnějších výsledků.

Citovaná literatura

1. **Krishna.** Database(Data) Testing Tutorial. *Guru99.* [Online] ©2020. [Citace: 21. 5 2020.] https://www.guru99.com/data-testing.html#2.

2. **Malý, Martin.** Softwarové licence. *Zdroják.* [Online] 9. Únor 2011. [Citace: 6. Květen 2020.] https://www.zdrojak.cz/clanky/softwarove-licence-uvod-pro-obycejne-lidi/.

3. **Krčmář, Petr.** Affero GPLv3: Vydejte zdrojové kódy síťových aplikací! *Root.* [Online] 8. Červen 2007. [Citace: 9. Květen 2020.] https://www.root.cz/clanky/affero-gplv3-vydejte-zdrojove-kody-sitovych-aplikaci/.

4. **Aujezdský, Josef.** Obecné závěry k licenčním podmínkám BSD. *Root.cz.* [Online] 2019. [Citace: 28. Duben 2020.] https://www.root.cz/specialy/licence/obecne-zavery-k-licencnim-podminkam-bsd/.

5. **The Apache Software Foundation.** APACHE LICENSE, VERSION 2.0. *Apache.* [Online] Leden 2004. [Citace: 2. Květen 2020.] https://www.apache.org/licenses/LICENSE-2.0.

6. **Burom, Barna.** NoSql vs relational database file storing . *Coding sans.* [Online] 9. Září 2017. [Citace: 1. Červen 2020.] https://codingsans.com/blog/nosql-vs-relational-database.

7. **Guru99 Tech Pvt Ltd.** SQL vs NoSQL: What's the difference? *Guru99.* [Online] 27. Březen 2019. [Citace: 15. Květen 2020.] https://www.guru99.com/privacy-policy.html.

8. **Krishna.** SQL vs NoSQL: What's the difference? *Guru99.* [Online] ©2020. [Citace: 21. 5 2020.] https://www.guru99.com/sql-vs-nosql.html.

9. **MongoDB.** NoSQL vs SQL Databases. *mongoDB.* [Online] 2020. [Citace: 12. Květen 2020.] https://www.mongodb.com/nosql-explained/nosql-vs-sql.

10. **MongoDB, Inc.** NoSQL vs Relational Databases. *mongoDB.* [Online] 2020. [Citace: 10. Květen 2020.] https://www.mongodb.com/scale/nosql-vs-relational-databases.

11. **Sharma, Ayush.** Difference between SQL and NoSQL. *GeeksforGeeks.* [Online] 2020. [Citace: 28. Duben 2020.] https://www.geeksforgeeks.org/difference-between-sql-and-nosql/.

12. **Smallcombe, Mark.** SQL vs NoSQL: 5 Critical Differences. *Xplenty.* [Online] 19. Květen 2020. [Citace: 25. Květen 2020.] https://www.xplenty.com/blog/the-sql-vs-nosql-difference/.

13. **Team LoginRadius.** Relational Database Management System (RDBMS) vs noSQL. *loginradius.* [Online] 28. Duben 2015. [Citace: 11. Květen 2020.] https://www.loginradius.com/engineering/blog/relational-database-management-system-rdbms-vs-nosql/.

14. **Tol, Sandeep.** Relational vs NoSQL and RDBMS to NoSQL Migration. *Databaze Zone.* [Online] 21. Leden 2020. [Citace: 8. Květen 2020.] https://dzone.com/articles/relational-vs-nosql-databases-and-rdbms-db-to-nosq.

15. **Reed, Paula.** Analytics Databases: The Best And The Fastest. *panoply blog.* [Online] 2. Květen 2019. [Citace: 3. Květen 2020.] https://blog.panoply.io/analytics-databases-the-best-and-the-fastest.

16. **Holubová, Irena, a další.** *Big Data a NoSQL databáze.* místo neznámé : Grada, 2015. 978-80-247-5466-6.

17. **Bigdata Software.** Top column-oriented databases. *Predictive analytics today.* [Online] 2020. [Citace: 2. Květen 2020.] https://www.predictiveanalyticstoday.com/top-wide-columnar-store-databases/.

18. **Chand, Mahesh.** What Is A Column Store Database. *c-sharp corner.* [Online] 23. Červenec 2019. [Citace: 1. Květen 2020.] https://www.c-sharpcorner.com/article/what-is-a-column-store-database/.

19. **Ian.** What is a Column Store Database? *Database guide.* [Online] 23. Červen 2016. [Citace: 20. Květen 2020.] https://database.guide/what-is-a-column-store-database/.

20. **Stěhule, Pavel.** Několik poznámek ke sloupcovým databázím. *Root.* [Online] 30. Březen 2015. [Citace: 25. 4 2020.] https://www.root.cz/clanky/nekolik-poznamek-ke-sloupcovym-databazim/.

21. **ATEsystem s.r.o.** Průmyslová kamera. *atesystem.* [Online] 2019. [Citace: 24. Květen 2020.] http://kamery.atesystem.cz/know-how/slovnik-pojmu-ve-strojovem-videni/prumyslova-kamera/.

22. **Skřivan, Jaromír.** GIF, JPEG a PNG – jak a kdy je použít? *interval.* [Online] 16. Květen 2002. [Citace: 10. Duben 2020.] https://www.interval.cz/clanky/gif-jpeg-a-png-jak-a-kdy-je-pouzit/.

23. **Bártík, František.** Cassandra: Databáze pro skutečně velké projekty. *LinuxExpress.* [Online] 16. 5 2013. [Citace: 9. 10 2019.] https://www.linuxexpres.cz/software/cassandra-databaze-pro-skutecne-velke-projekty.

24. **Krishna.** Load Testing Tutorial: What is? How to? *Guru99.* [Online] ©2020. [Citace: 21. 5 2020.] https://www.guru99.com/load-testing-tutorial.html.

25. —. What is STRESS Testing in Software Testing? *Guru99.* [Online] ©2020. [Citace: 21. 5 2020.] https://www.guru99.com/stress-testing-tutorial.html.

26. —. What is Scalability Testing? *Guru99.* [Online] ©2020. [Citace: 21. 5 2020.] https://www.guru99.com/scalability-testing.html.

27. —. What is Volume Testing? *Guru99.* [Online] ©2020. [Citace: 21. 5 2020.] https://www.guru99.com/volume-testing.html.

28. —. What is Endurance Testing in Software Testing? *Guru99.* [Online] ©2020. [Citace: 21. 5 2020.] https://www.guru99.com/endurance-testing.html.

29. **bogotobogo.** Algorithms & data structures - introduction. *bogotobogo.* [Online] 2029. [Citace: 10. Květen 2020.] https://www.bogotobogo.com/Algorithms/algorithms.php.

30. **Krishna.** Stability Testing in Software Testing. *Guru99.* [Online] ©2020. [Citace: 21. 5 2020.] https://www.guru99.com/stability-testing.html.

31. —. Performance Testing Tutorial. *Guru99.* [Online] ©2020. [Citace: 21. 5 2020.] https://www.guru99.com/performance-testing.html.

32. —. Load Testing vs Stress Testing vs Performance Testing: Difference Discussed. *Guru99.* [Online] ©2020. [Citace: 21. 5 2020.] https://www.guru99.com/performance-vs-load-vs-stress-testing.html.