

Key-value databázové systémy

Key-value database systems

Bc. Jan Jedlička

Diplomová práce

Vedoucí práce: prof. Ing. Michal Krátký, Ph.D.

Ostrava, 2024

Zadání diplomové práce

Student:

Bc. Jan Jedlička

Studijní program:

N2647 Informační a komunikační technologie

Studijní obor:

2612T025 Informatika a výpočetní technika

Téma:

Key-value databázové systémy
Key-Value Database Management Systems

Jazyk vypracování:

čeština

Zásady pro vypracování:

Ačkoli relační SŘBD po mnoha letech vývoje poskytují dostatečný výkon pro velkou část aplikací, existují aplikace pro které jsou některé vlastnosti relačních SŘBD nevhodné. Key-value databázové systémy se snaží reagovat na specifické požadavky především v distribuovaných prostředích.

1. Nastudujte problematiku key-value databázových systémů.
2. Navrhněte a naimplementujte testovací prostředí pro porovnání těchto databázových systémů s ostatními SŘBD.
3. Vybrané databázové systémy otestujte a vyhodnoťte výsledky experimentů.

Seznam doporučené odborné literatury:

[1] predictiveanalyticstoday.com: Top NoSQL Key-value Databases. March 18 2022. <https://www.predictiveanalyticstoday.com/top-sql-key-value-store-databases/>

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **prof. Ing. Michal Krátký, Ph.D.**

Datum zadání: 01.09.2022

Datum odevzdání: 30.04.2024

Garant studijního oboru: prof. RNDr. Václav Snášel, CSc.

V IS EDISON zadáno: 09.08.2023 10:43:21

Abstrakt

Cílem diplomové práce je popsat Key-value DBS, ukázat výhody a nevýhody těchto systémů a představit některé významné zástupce. První část práce je zaměřena nejen na obecný popis Key-value DBS, ale i na podrobnější specifikace vybraných DBS a jejich vzájemné porovnání. Součástí práce je zmapování existujících testů DBS a následný výběr a zprovoznění testovacího prostředí pro testování těchto systémů a porovnání s ostatními DBS. Práce popisuje dva významné představitele testovacích prostředí DBS, YCSB a TPC. S prostředím YCSB se v práci následně pracuje a testují se v něm čtyři vybrané Key-value DBS: Redis, Aerospike, Riak KV a Memcached. Veškeré testy jsou spouštěny pomocí skriptů, které spustí a nastaví DBS v prostředí Docker. Připravené DBS jsou následně naplněny daty a otestovány za pomoci prostředí YCSB. Práce je zakončena reprezentací hodnot naměřených při testování, vzájemným porovnáním pomocí grafů a vyhodnocením výsledků těchto testů.

Klíčová slova

DBS; NoSQL; Key-value DBS; Benchmarking; YCSB; TPC; Redis; Riak KV; Aerospike; Memcached

Abstract

The aim of this thesis is to describe Key-value DBS, to show the advantages and disadvantages of these systems and to introduce some important representatives. The first part of the thesis focuses not only on a general description of Key-value DBSs, but also on more detailed specifications of selected DBSs and their comparison with each other. The work includes a mapping of existing DBSs and then the selection and operationalization of a testbed for testing these systems and comparing them with other DBSs. The thesis describes two prominent representatives of DBS test environments, YCSB and TPC. The YCSB environment is then used in the thesis to test four selected key-value DBSs: Redis, Aerospike, Riak KV and Memcached. All tests are run using scripts that start and set up the DBS in the Docker environment. The prepared DBSs are then populated with data and tested using the YCSB environment. The paper concludes by representing the values measured during testing, comparing them with each other using graphs and evaluating the results of these tests.

Keywords

DBS; NoSQL; Key-value DBS; Benchmarking; YCSB; TPC; Redis; Riak KV; Aerospike; Memcached

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu této práce prof. Ing. Michalovi Krátkému, Ph.D., za poskytnutí užitečných rad a pomoci vedením práce.

Obsah

Seznam použitých symbolů a zkratk	7
Seznam obrázků	8
Seznam tabulek	9
1 Úvod	10
2 Key-value databázové systémy	14
2.1 Amazon DynamoDB	15
2.2 Oracle NoSQL Database	16
2.3 Redis	17
2.4 Aerospike	17
2.5 Oracle Berkeley DB	18
2.6 Riak KV	19
2.7 Voldemort	20
2.8 InfinityDB	21
2.9 Memcached	21
2.10 Nezmíněné významné NoSQL databáze	22
3 Prostředí pro testování databázových systémů	24
3.1 YCSB	24
3.2 TPC	26
3.3 Relevance TPC a YCSB pro měření KDBS	27
4 Vyhodnocení výsledků testů	28
4.1 Testovací prostředí	28
4.2 Zprovoznění testů	28
4.3 Popis parametrů testů	29
4.4 Spouštění testů	29

4.5	Výsledky testů	32
5	Závěr	41
	Literatura	42

Seznam použitých zkratek a symbolů

DBS	– Databáze
NoSQL	– Not only Structured Query Language
Key-value DBS	– Klíč-hodnota DBS
KDBS	– Key-value DBS
RDBS	– Relační DBS
JSON	– JavaScript Object Notation
TTL	– Time to live
TPC	– Transaction Processing Performance Council
YCSB	– Yahoo! Cloud Serving Benchmark
Kiloops	– Tisíce operací

Seznam obrázků

1.1	Graf hodnot popularity RDBS Oracle a KDBS Redis [12]	13
1.2	Graf změn hodnot popularity RDBS a KDBS [8]	13
3.1	YCSB rámec testování funkčnosti [58]	26
4.1	Docker Redis, příkazy pro stažení, spuštění, zastavení a odstranění DBS	32
4.2	YCSB Redis, příkaz pro vložení dat (load)	33
4.3	YCSB Redis, příkaz pro spuštění testů (run)	34
4.4	Workloads A, B, C + Insert only - Throughput (kiloops/sec)	35
4.5	Workloads A, B, C + Insert only - Runtime (s)	35
4.6	Workloads A, B, C + Insert only - Max Latency (ms)	36
4.7	Workload A - Min Avg Latency (ms)	36
4.8	Workload A - Percentile Latency (ms)	37
4.9	Workload B - Min Avg Latency (ms)	37
4.10	Workload B - Percentile Latency (ms)	38
4.11	Workload C - Min Avg Latency (ms)	38
4.12	Workload C - Percentile Latency (ms)	39
4.13	Insert only - Min Avg Latency (ms)	39
4.14	Insert only - Percentile Latency (ms)	40

Seznam tabulek

2.1	Porovnání Key-value databází	23
3.1	TPC benchmarky	27
4.1	Specifikace stroje na kterém se spouštěly testy	28

Kapitola 1

Úvod

Key-value (neboli Klíč-hodnota) databázové systémy [1, 2], dále jen zkráceně KDBS, jsou jedním z paradigmat pro úložiště dat. Databáze je navržena pro rychlý přístup k datům a uživatelsky snadnou práci. Pro manipulaci s daty se zpravidla používá několik příkazů pro vložení, získání, aktualizaci a odstranění hodnot na základě klíčů. V KDBS se hodnota ukládá k danému klíči, přičemž klíč může být například "jmeno_uzivatele", a hodnota může být "Jan Novák".

KDBS fungují odlišně oproti tradičním relačním databázovým systémům. Datový model RDBS je relační [3], což znamená, že data jsou organizována do tabulek, kde každá tabulka představuje jednu relaci [4] a každý záznam v tabulce odpovídá jednomu řádku. Každá tabulka obsahuje sloupce, které předem pevně definují datové typy záznamů. Jedním ze sloupců je primární klíč, který jednoznačně identifikuje každý záznam. Tento relační model poskytuje strukturovaný a systematický přístup k ukládání a manipulaci s daty, což usnadňuje organizaci a vyhledávání informací. Díky definované struktuře záznamů může RDBS provádět řadu optimalizací, jako je například přidávání vlastních indexů nebo plánování a optimalizace dotazů [5]. Například pokud chceme do RDBS ukládat záznamy obsahující uživatele, musí mít každý záznam stejnou strukturu. Takže uživatel musí obsahovat všechny povinné atributy, jako jsou jméno a příjmení, a dodržet jejich předem stanovený datový typ a velikost, například počet číslic v telefonním čísle. Pokud by záznam obsahoval nějaké atributy navíc, například informaci o tom, zda je uživatel nemocný, nebo by jejich datový typ neodpovídal očekávanému, je zapotřebí upravit i tabulku, do které záznam vkládáme, a také záznamy, které se již v této tabulce nachází.

Na druhou stranu KDBS mohou mít pro každý klíč různě definované hodnoty, které mohou zahrnovat kolekce dat s odlišnými velikostmi. Tato vlastnost nabízí KDBS flexibilitu a možnost přiblížení se k objektově orientovanému programování. Například do KDBS s klíči jako ID uživatele a hodnotami reprezentujícími záznamy různých uživatelů můžeme ukládat JSON objekty. Tyto objekty reprezentují uživatele, kteří jsou definováni odlišnými, na sobě nezávislými třídami s různým počtem a datovým typem atributů, bez společného rodiče. Dále tyto databáze dosahují horizontální až lineární škálovatelnosti (viz kapitola 2). Protože KDBS nevyžaduje pevně nastavené datové typy

hodnot, jako je tomu u relační databáze, tak KDBS může vyžadovat méně paměti k uložení stejných dat [6], což vede k nárůstu výkonu v určitých případech.

Databáze se schématem [7] (schema-on-write) vyžadují definici datového schématu před tím, než jsou data uložena. To znamená, že musíte předem definovat strukturu dat včetně typů dat, a poté můžete data vložit do DBS. Na druhou stranu databáze bez schématu (schema-on-read, schema-less) umožňují ukládat data bez předchozí definice datového schématu. Schéma je definováno až v okamžiku, kdy data čtete z DBS. To umožňuje pružnější přístup k ukládání a zpracování dat, ale může vést k menší kontrole nad strukturou dat a k možným problémům s konzistencí dat.

Výkon a nedostatečná standardizace omezovaly KDBS pouze na specializovaná využití, ale díky rychlému přechodu na cloud computing, dochází v posledních letech k nárůstu popularity NoSQL databázových systémů, zatímco popularita RDBS je víceméně stejná [8]. Dle výzkumu z roku 2019 [9] byly RDBS využívány v 60,48 % případů napříč všemi databázovými systémy, zatímco NoSQL DBS pak zastávaly zbylých 39,52 %. Často se však pro využití výhod různých druhů DBS využívá kombinace RDBS a NoSQL DBS. Tato kombinace je proto využívána v 75,6 % případů. Čistě využívané RDBS poté zabírají 16,6 % a NoSQL DBS 9,8 %.

Webová stránka DB-Engines [10] porovnává DBS na základě popularity. Bodové skóre popularity je spočítáno na základě počtu zmínek na webových stránkách, frekvenci diskuzí na technických fórech, pracovních nabídek a poptávek, relevance DBS na sociálních sítích, počtu profilů na profesních sociálních sítích zmiňujících danou DBS a obecného zájmu o DBS [11]. Při sledování trendu DBS od roku 2013 do současnosti [12] (viz graf 1.1) dochází k výraznému nárůstu hodnocení popularity NoSQL DBS, konkrétně KDBS Redis [13] stoupla od roku 2013 z 25,973 bodů na 156,441 bodů, došlo tedy k nárůstu popularity o 502,68 %. Pro RDBS Oracle [14] došlo naopak k poklesu popularity z 1559,332 bodů na 1234,269 bodů, pokles činí 20,85 %. Trend změny hodnot popularity [8] (viz graf 1.2) pro RDBS obecně od roku 2013 se příliš nemění a dochází k poklesu v maximálně jednotkách bodů každý měsíc, KDBS naopak znatelně stoupały se svým hodnocením do konce roku 2022 a aktuálně dochází k poklesu popularity v řádu několika jednotek bodů každý měsíc.

Například databázový systém Redis je v současnosti jedním z deseti nejlépe hodnocených [15] databázových systémů napříč relačními i NoSQL databázovými systémy. V žebříčku porovnání téměř 70 čistě KDBS [16], je Redis na prvním místě, za ním následují Amazon DynamoDB [17], společně s Microsoft Azure Cosmos DB [18] a Memcached [19].

Cílem práce je prostudovat a podrobně popsat problematiku KDBS. Práce se zaměří na vysvětlení tohoto konceptu a identifikaci klíčových rozdílů mezi jednotlivými KDBS. Jednou z hlavních úloh práce je zmínit a detailně popsat některé z nejznámějších a nejčastěji využívaných databázových systémů v tomto odvětví. Tyto DBS budou vzájemně porovnány s důrazem na jejich vlastnosti, výhody a nevýhody. Dalším úkolem práce je příprava a zprovoznění testovacího prostředí, které umožní měření propustnosti a odezvy na dotazy těchto vybraných KDBS. Návrh a implementace testovacího prostředí zahrnuje konfiguraci prostředí a zprovoznění instancí databázových systémů, ať už lokálně nebo v cloudovém prostředí. Následně budou vybrané DBS podrobeny testování a budou

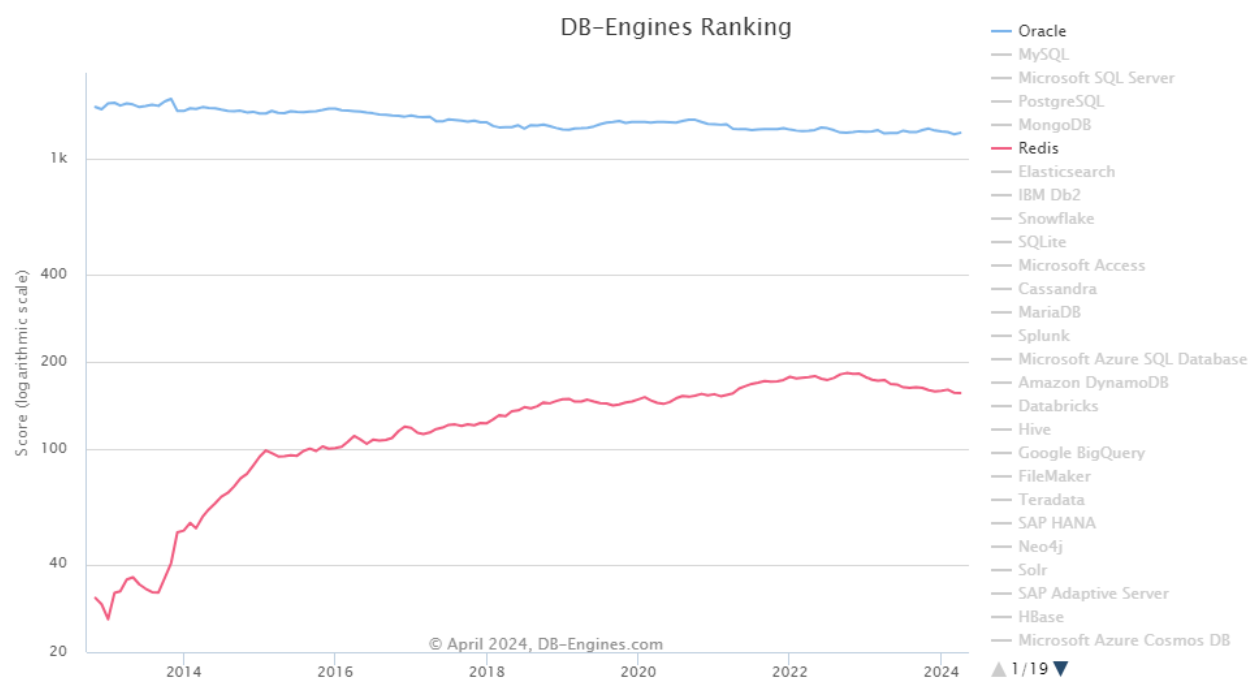
měřeny jejich propustnosti a odezva v reálném provozu. Naměřené hodnoty budou prezentovány a analyzovány s cílem porovnat chování jednotlivých DBS v různých scénářích. Konečné výsledky budou porovnány, což umožní vyvodit závěry o tom, jak se jednotlivé DBS osvědčily v praxi při manipulaci s daty.

Struktura práce je následující. První část práce (viz kapitola 2) se zaměřuje na detailní představení několika klíčových KDBS (např. Redis, Riak KV, Aerospike). Každá z těchto DBS je popsána, a to včetně analýzy jejich charakteristických vlastností. V závěru této části je provedeno srovnání jednotlivých databází na základě jejich vlastností, například škálovatelnost, dotazovací jazyk a odezva.

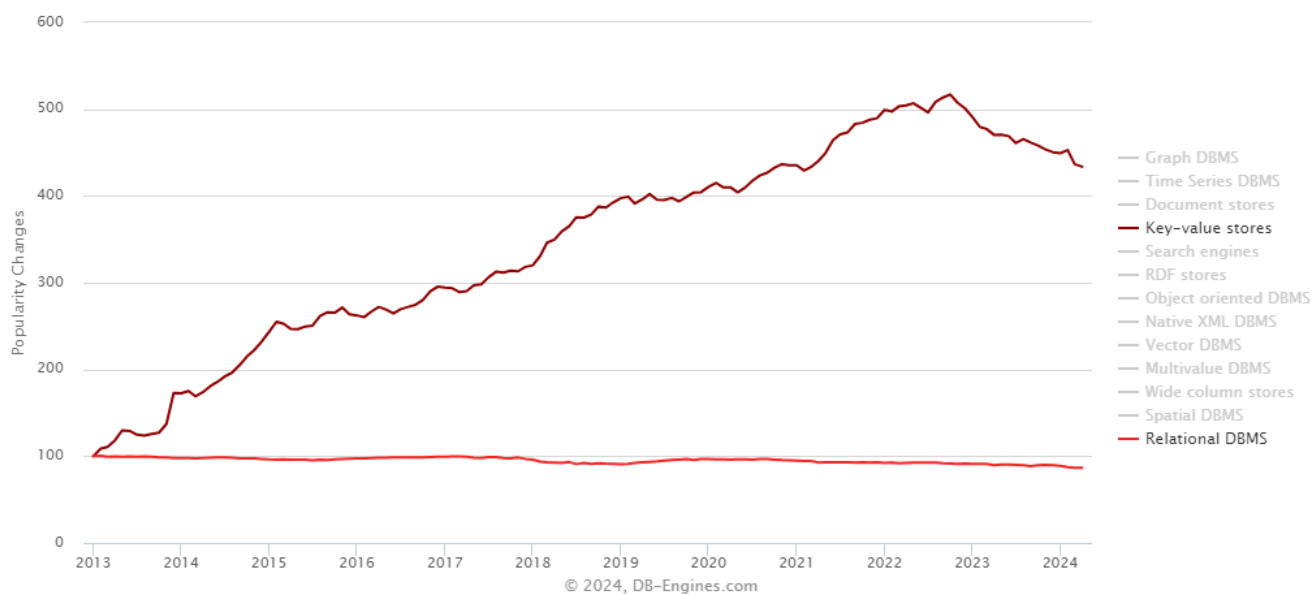
Kapitola 3 (viz kapitola 3) obsahuje popis a využitelnost testovacích prostředí databázových systémů. V kapitole jsou uvedena dvě existující testovací prostředí TPC [20] a YCSB [21]. Testovací prostředí TPC je využíváno především pro relační databázové systémy, zatímco prostředí YCSB je připraveno a vyvíjeno zejména pro NoSQL DBS.

Kapitola 4 (viz kapitola 4) je věnována testování a vyhodnocení výsledků čtyř vybraných KDBS (Redis [13], Riak KV [22], Aerospike [23] a Memcached [19]). Tyto DBS byly vybrány pro analýzu z důvodu své relevance podle internetových žebříčků [15, 24]. Kapitola popisuje stroj, na kterém se KDBS testovaly, a potřebnou přípravu zvoleného testovacího prostředí YCSB a databází v Dockeru [25]. Konkrétně se zde popisují konfigurace jednotlivých testů, kroky pro úspěšné spuštění testů a samotné otestování. Kapitola je zakončena prezentací výsledků naměřených hodnot při testování a vzájemným porovnáním výsledků jednotlivých KDBS.

V závěru práce, kapitole 5 (viz kapitola 5), jsou vyhodnoceny výsledky srovnání KDBS. Součástí kapitoly je zhodnocení samotné práce.



Obrázek 1.1: Graf hodnot popularity RDBS Oracle a KDBS Redis [12]



Obrázek 1.2: Graf změn hodnot popularity RDBS a KDBS [8]

Kapitola 2

Key-value databázové systémy

KDBS [1, 2, 26, 27] jsou typem databázového systému, který se specializuje na ukládání dat ve formě jednoduchých párování klíč-hodnota. Každá hodnota v této databázi je spojena s unikátním klíčem, který slouží k identifikaci daného záznamu (hodnoty). Tyto databáze jsou často preferované pro svou schopnost rychle a efektivně ukládat a získávat data, což je zvláště důležité pro aplikace, které potřebují vysoký výkon a škálovatelnost.

Jedna z hlavních vlastností KDBS spočívá v tom, že ukládají data v jednoduchém formátu klíč-hodnota. Klíč slouží jako identifikátor, který umožňuje rychlé vyhledávání a přístup k datům. Data jsou obvykle indexována podle klíče, což zajišťuje rychlý přístup k nim i při velkém objemu dat.

Důležitým rysem těchto databází je také jejich škálovatelnost. Mohou zpracovávat velké objemy dat a udržovat vysoký výkon i při zvyšujícím se počtu uživatelů nebo objemu dat. Některé systémy KDBS nabízejí distribuované ukládání dat, což umožňuje rozložení dat mezi různé servery a zvýšení odolnosti a výkonu celého systému.

Pro uživatele je klíčovým faktorem také jednoduchost použití. KDBS poskytují jednoduché rozhraní pro práci s daty, které zahrnuje základní operace jako vložení, aktualizaci a vyhledávání dat pomocí klíče. Tato jednoduchost usnadňuje integraci DBS do různých aplikací a snižuje nároky na vývojáře.

Vzhledem k těmto vlastnostem jsou KDBS vhodné pro širokou škálu aplikací, včetně webových aplikací, analytických systémů, cache pamětí a dalších. Jsou ideální pro situace, kde je potřeba rychle a efektivně ukládat a získávat data a zároveň udržovat vysoký výkon a škálovatelnost systému.

Jelikož KDBS jsou primárně určeny pro distribuované nasazení, nyní popíšeme tento pojem. Distribuovaný databázový systém [28] je typ databázového systému, který ukládá data na několik počítačů nebo uzlů v rámci sítě. Tento přístup umožňuje efektivnější zpracování velkých objemů dat a zlepšuje odolnost systému vůči výpadkům jednotlivých uzlů. Distribuovaná architektura umožňuje systému růst s rostoucím objemem dat a zároveň snižuje riziko selhání jednoho centrálního bodu, což zvyšuje celkovou spolehlivost a dostupnost systému.

Škálovatelnost [29] je schopnost systému přizpůsobit se zvýšené zátěži nebo rostoucímu objemu dat a zachovat požadovanou úroveň výkonu a dostupnosti. Existují různé druhy škálovatelnosti. Horizontální škálovatelnost zahrnuje rozložení zátěže a zvýšení celkové kapacity přidáním dalších uzlů nebo serverů k distribuovanému systému. Vertikální škálovatelnost zlepšuje výkon jednotlivých uzlů nebo serverů zvětšením jejich výpočetní nebo paměťové kapacity. Lineární škálovatelnost je ideální situace, kdy se se zvýšením zdrojů (buď horizontálně nebo vertikálně) zvyšuje výkon systému lineárně.

V současné době existuje celá řada KDBS, od malých open-source projektů po velké komerční cloudové služby. Různé systémy disponují odlišnými vlastnostmi, jako je propustnost, škálovatelnost, uživatelská přívětivost skrze dotazovací jazyk a podpora atd. Dle průzkumu [24, 30, 15] bylo vybráno 9 významných KDBS, snažících se o jednoduchý popis, porovnání (viz tabulka 2.1) a konečný výběr KDBS pro testování a porovnání.

2.1 Amazon DynamoDB

Amazon DynamoDB [17] je v současné době největší a nejvyužívanější KDBS. Jedná se o serverless cloud systém s odezvou v řádu jednotek mikrosekund a využitím v oblastech jako je web, technologie IoT, mobilní aplikace a herní průmysl. DynamoDB je plně a automaticky spravovatelná, multi-master databáze zaměřená na vysoké využití horizontální škálovatelnosti. Unikátní primární klíče umožňují identifikaci jednotlivých záznamů v tabulkách a sekundární indexy zlepšují dotazovací flexibilitu. Primární klíč slouží jako vstup do hashovací funkce a výsledná hashovací hodnota určuje fyzickou pozici uloženého záznamu. DynamoDB poskytuje silnou konzistenci při čtení hodnot od poslední aktualizace. Atomické čítače umožňují automatické změny hodnot číselných atributů. Pro expirované záznamy v tabulkách využívá tzv. TTL (Time To Live) [31] k označení doby, po kterou má záznam zůstat platný v databázi. Archivace dat je umožněna díky full backupu. Amazon DynamoDB rovněž nabízí VPC [32] pro soukromou komunikaci za využití vlastní privátní síťové prostředí v rámci AWS cloudu.

Databázový systém disponuje konzolovým API pro správu DBS a práci s daty, ale nabízí také možnost využití jazyka PartiQL [33], který je vhodný pro kompatibilní SQL dotazy na databázích bez schématu. DynamoDB API je rozděleno do čtyř hlavních částí. Kontrolní plán zahrnuje funkce spojené s vytvářením, úpravami, mazáním a získáním jmen všech tabulek. Dále umožňuje výpis podrobných specifikací dané tabulky, jako jsou primární klíče, indexy a nastavení propustnosti. Následuje datový plán, který poskytuje CRUD operace pro data v dané tabulce. S daty lze pracovat buď jednotlivě po záznamech, nebo pomocí Batch funkcí, které umožňují provést stejnou operaci nad desítkami záznamů najednou a dosáhnout tak vyšší propustnosti, než při volání stejných funkcí pro jednotlivé záznamy opakovaně. Následně je možné provést Scan pro získání všech záznamů dané tabulky nebo indexu, případně Query pro získání hledané části dat. Třetí částí je DynamoDB Streams pro práci s časovými sekvencemi a práci s logy za posledních 24 hodin. Stream API poskytuje

funkce pro výpis všech streamů, konkrétní popis daného streamu, získání iterátoru pro daný stream a nakonec získání jednoho záznamu z daného streamu. Poslední částí API jsou ACID transakce, které jsou rozděleny do dvou částí. První část je určena pro batch vkládání, úpravu a mazání záznamů a druhá část slouží k batch získání záznamů.

2.2 Oracle NoSQL Database

Oracle NoSQL Database [34] je databázová cloud služba vhodná pro práci s velkými objemy dat a odhadovatelnou nízkou odezvou v řádu jednotek milisekund. Služba je postavena na enginu z Oracle Berkeley DB. Databáze je plně spravovatelná, flexibilní, škáluje horizontálně, dynamicky a dosahuje vysokých výkonů. Mimo Key-value data se jedná i o spolehlivé úložiště pro dokumenty a data s pevně daným schématem. Vzhledem k tomu, že databázový systém je plně spravovaný společností Oracle, tak je pro vývojáře rychlé a snadné začít tuto službu využívat a soustředit se pouze na vývoj aplikací, neboť není potřeba se obtěžovat se správou základní infrastruktury databáze, softwaru, zabezpečení atp. Jedná se o Single Master, Multi Replica grafový systém. Pokud dojde k chybě na masteru, je master automaticky nahrazen jednou z replik. Pro Key-value ukládání s kapacitu jednotek terabytů využívá systém velký počet Storage uzlů, které je možno skupinově konfigurovat. Pro udržení konzistence jsou Storage uzly replikovány. Uzly a hrany v grafu reprezentují entity které vytvářejí vztahy a propojení. Sdílený systém, uniformně alokuje data okolo ostatních částí skupin. Databáze obsahuje i SQL Query s jazykem pro import, export a přenos dat mezi různými Oracle NoSQL databázemi. Mimo jiné je zde podpora i pro Failover, SwitchOver, Bulk Get API, Off Heap Cache a podpora Big Data SQL.

Restové API pro Oracle NoSQL Database je rozděleno do pěti částí. Správa indexů, která dovoluje vytvářet a mazat indexy pro danou tabulku. Tato část API také umožňuje zobrazit všechny indexy, které jsou pro danou tabulku vytvořeny a společně s detailním popisem každého indexu. Druhá část API se věnuje dotazům, umožňuje tedy syntaktickou kontrolu daného SQL dotazu, před připravením a spuštěním dotazu. Třetí část je zaměřena na správu záznamů, obsahuje tedy CRUD funkce pro jednotlivé záznamy. Tato část ale neobsahuje funkci pro úpravu existujícího záznamu a ani neumožňuje správu mnoha záznamů najednou, pro úpravu je tedy nutno provést funkci odstranění záznamu a vložení nového a všechny záznamy je tedy také potřeba spravovat jednotlivě a postupně. Čtvrtá část je zaměřena správě tabulek, obsahuje možnost vytvoření, upravování, a mazání tabulek. Tato část také umožňuje výpis všech tabulek, informace o dané tabulce a využívání dané tabulky. Poslední část API se věnuje správě pracovních požadavků, lze zde zobrazit stav jednotlivých požadavků, mazat požadavky, získat chyby či log daného požadavku a list všech požadavků.

2.3 Redis

Redis [13] je in-memory úložiště pro datové struktury, využívané jako KDBS, cache, streaming engine nebo zprostředkovatel zpráv. Toto datové úložiště má skvělé využití pro klíče v podobě hashe a hodnoty jako velký JSON objekt. Pro persistenci dat můžeme ukládání dat na disk provádět po nastavitelných pravidelných intervalech, nebo je možné data logovat vždy při vykonávání operací. Pokud nemáme zájem o trvanlivost dat, je možné ukládání dat úplně vypnout a datové úložiště využít čistě jako cache. Úložiště škáluje horizontálně. Redis podporuje datové struktury jako řetězce, hashe, listy, množiny, bitmapy, hyperloglog a geospatial indexy. Nad datovými typy Redis umožňuje rychlé atomické operace, jako je rozšíření řetězců, přidání prvků na začátek a konec listů, atd. Datové úložiště také poskytuje seřazené množiny pro vytváření indexů dle ID nebo jiného číselného atributu. Redis hashing ukládá data jako klíč a mapu. Keyspace notifikace dovoluje klientům odebírat Publisher-Subscriber kanály. Pro práci s dotazy na souřadnice a geometrii je možné využívat Geo API. Redis umožňuje provádět transakce, volat Lua skripty a nastavovat různé úrovně TTL pro záznamy. Redis podporuje Trivial-to-setup Master-Slave asynchronního replikování, společně s rychlou neblokující se prvotní synchronizací. Struktura pro ukládání dat je single-rooted replikovaný strom. Redis má vlastní API pro práci s daty pro populární programovací jazyky jako C, Python, Java a JavaScript.

S Redis databází lze pracovat například pomocí konzolového rozhraní, toto CLI [35] poskytuje řadu jednoduše čitelných, ale netradičních příkazů pro práci s daty. Vždy potřebujeme specifikovat klíč, se kterým chceme v databázi pracovat. Pomocí příkazu SET a DEL vkládáme do DBS nebo mažeme jednotlivé hodnoty pro zvolený klíč. Příkazem GET získáme hodnoty pro daný klíč, případně můžeme zjistit, zda již existuje záznam pro daný klíč příkazem EXISTS. Pokud vyžadujeme práci s poli, tak můžeme pro daný klíč zleva i zprava vkládat hodnoty zřetězené v poli díky příkazům LPUSH a RPUSH. Obdobně odebíráme hodnoty z pole pomocí LPOP a RPOP, příkazem LRange vypíšeme hodnoty z pole a příkazem LLen zjistíme počet jeho záznamů. Místo jednoduchých polí je možno pracovat i s množinami pomocí příkazů SADD, SREM, SISMEMBER a obdobně. Množiny mohou být i seřazené a pro ně se využívají příkazy jako ZADD. Pro práci se záznamy strukturovanými jako kolekce párů atribut-hodnota se využívá datový typ Hash, umožňuje nám pro daný klíč uložit záznam obsahující názvy atributů a jednotlivé hodnoty pro ně. Opět se zde využívají příkazy jako HSET a HGETALL pro nastavení a získání daného záznamu, případně HGET pro získání hodnoty daného atributu pro záznam na zadaném klíči. API obsahuje také příkazy pro ostatní datové typy, jako jsou bitmapy, geografické prostory, HyperLogLog a další.

2.4 Aerospike

Aerospike [23] je KDBS využívající Hybrid Memory architekturu [36], která umožňuje odezvu v jednotkách milisekund a vysokou propustnost v řádech stovek tisíc až milionů operací za sekundu.

Hybrid Memory architektura od Aerospike je implementována tak, že index je čistě In-Memory, tím pádem není index perzistentní (vhodné například pro uživatelské cache sessions), a data jsou uložena čistě perzistentně na SSD disku a čtou se přímo z něj. Díky tomu, že je Aerospike jako KDBS naprosto schéma-less, je možné definovat Sets a Bins za běhu pro maximální flexibilitu aplikací. Databáze škáluje lineárně a poskytuje silnou konzistenci, nízkou cenu a korektnost. Umožňuje real-time analýzu pro rychlé rozhodování a dynamickou optimalizaci pro vhodné využívání zdrojů dat, proto je databáze vhodná pro velké a stále aktualizované DBS. Poskytuje server-side clustering a bezpečnost na transportní vrstvě. Databáze také umožňuje customer deployment s nulovým downtime. V praxi se Aerospike díky svým vlastnostem využívá například pro banking, telekomunikace, adtech a gaming. Aerospike poskytuje vlastní silný dotazovací jazyk AQL [37], který má prakticky shodnou syntaxi jako SQL (i když se o SQL nejedná). Vlastní vytvořitelné agregační funkce pomocí Lua jazyka jsou flexibilní pro agregační algoritmy.

Dotazovací jazyk AQL se snaží zachovat standardní SQL syntaxi, obvyklé příkazy SELECT, INSERT, DELETE jsou tedy zachovány. Je možné vytvářet vlastní indexy nad tabulkami pomocí CREATE INDEX a provádět agregace pomocí AGGREGATE. Pro dotazy nad konkrétním záznamem specifikovaným pomocí hexadecimálního řetězce či Base64 lze v podmínce dotazu použít porovnání hodnoty s DIGEST nebo EDIGEST. Dotazování můžeme provádět i standardně nad primárním klíčem a ostatními atributy. Při vkládání záznamů lze specifikovat speciální datové typy atributů, jako je LIST, MAP, GEOJSON a další.

2.5 Oracle Berkeley DB

Oracle Berkeley DB [38] je rodina vestavěných Key-value databázových knihoven. Jedná se o čistě In-memory databázi, díky čemuž dosahuje vysokého výkonu a odezvy v jednotkách mikrosekund. Databáze škáluje horizontálně. Data jsou replikována pro vysokou dostupnost z více zdrojů a dobrou toleranci chybovosti. Oracle Berkeley DB využívá vhodné datové struktury pro práci s daty, jako jsou B-strom, hash table indexy nebo fronta. Databáze využívá obnovitelné ACID transakce a poskytuje několik různých úrovní izolace (včetně MVCC [39]). Data jsou dělena do oddílů dle key ranges. Umožňuje komprimaci dat. Databáze je Single-master, Multi-replica, tedy je vysoce dostupná a umožňuje dobrou konfigurovatelnost. Repliky umožňují čtecí škálovatelnost, rychlý fail-over, hot-standby a další distribuované konfigurace, dodávající podnikové prostředky v malém, vestavěném balíčku. Pro přístup k datům a nastavení databáze se využívá jednoduché volání funkcí API. Mnoho moderních programovacích jazyků, jako například C++, C#, Java, Python atd., podporuje tyto knihovny. Data mohou být ukládána v nativním formátu aplikace, XML, SQL nebo jako Java objekty. Oracle Berkeley DB je vhodný nástroj pro vše od lokálního úložiště po world-wide distribuovanou databázi (od kilobytů po petabyty).

Interakce s Berkeley DB SQL API je prakticky identická jako s SQLite [40]. Pro práci s databází vytvořenou rozhraním BDB SQL [41] používáte stejná rozhraní API, stejné Shell prostředí, stejné

příkazy SQL a stejné PRAGMA, jako se využívá u SQLite. BDB SQL rozšiřuje standardní SQLite PRAGMA o možnosti nastavení velikosti alokované paměti sdílených zdrojů, nastavení počtu bucketů v hashovací tabulce objektů zámek, zvolení soukromého prostředí místo sdíleného, přesměrování logování chyb do vlastního souboru, nastavení příznaku, který způsobí, že sdílené prostředky databáze budou vytvořeny ve sdílené paměti systému a další. Dalším drobným rozdílem je, že BDB SQL rozhraní nepodporuje klíčové slovo IMMEDIATE.

2.6 Riak KV

Riak KV [22] je distribuovaná KDBS s pokročilou lokální a multi-cluster replikací, která garantuje čtení a zápis i v případě selhání hardwaru nebo síťových oddílů. Riak využívá bezkonfliktní replikované datové typy (CRDT [42]), které umožňují nezávisle a souběžně aktualizovat jakoukoliv repliku v distribuované databázi se zajištěním sjednocení hodnot pomocí algoritmu, který je součástí samotného datového typu (flagy, registry, čítače, množiny a mapy). Poskytuje konfiguraci aktivního clusteru a dosahuje nízké latence v řádech jednotek milisekund díky dodávání dat z nejbližšího datacentra. Databáze rozděluje data z clusterů pro své dostupné zóny, má multi-cluster repliky a využívá redundance dat v geografickém regionu. Riak tedy automaticky distribuuje data skrz cluster pro robustnost a vysoký výkon. KDBS poskytuje flexibilní datový model bez předem definovaného schématu. Databáze má vylepšené logování chyb a reporty. Data jsou automaticky komprimována pomocí Snappy kompresní knihovny [43]. Databáze využívá master-less architekturu, je vysoce dostupná a má design horizontální škálovatelnosti. Škálovatelnost je téměř lineární při využití snadného přidání hardwarové kapacity bez nutnosti mnoha operací. Riak KV dovoluje zpracování dat pro analýzu a vyvození závěrů pro zlepšení chodu DBS. Riak KV je navržen pro nulové restrikce na hodnoty, takže session data mohou být enkódována mnoha způsoby a nevyžadují změnu schématu. Během nejvyšší zátěže nezhoršuje databáze zápis a horizontální škálovatelnost, uživatelé jsou stále obsluhováni bez problémů. Databáze je vhodná pro ukládání velkého množství nestrukturovaných dat, také pro big-data aplikace, ukládání dat z připojených zařízení a replikaci dat do okolí. Díky nízké latenci je DBS vhodná i pro chat/messaging aplikace. Riak KV exceluje v soukromém, veřejném či hybridním cloud nasazení.

Riak KV API obsahuje všechny potřebné CRUD operace pro správu objektů. Při vytváření nových objektů je potřeba nastavit typ a název bucketu, který skladuje klíče a data do něj vložena. Bucket má také vlastní indexy pro vyhledávání dat uvnitř něj. Dva různé buckety mohou uchovávat stejnou hodnotu klíče, ale jeden bucket obsahuje pouze unikátní klíče. Klíč pro data lze specifikovat explicitně vlastní při vytváření objektu pomocí parametru nebo při jeho absenci je datům přiřazen náhodný klíč. Při vkládání dat do DBS můžeme jednoduše nastavit parametr TTL daného objektu a také počet jeho replik. Při čtení dat můžeme před získáním výsledku zadat minimální počet replik, které se musí shodnout na stejných datech pro zvolený klíč. Pro efektivnější dotazy lze vytvořit vlastní indexy pro výchozí nebo námi zvolená datová schémata. Lze se dotazovat

na data pro zvolený klíč nebo provádět fulltextové vyhledávání. Databáze poskytuje i funkce pro tvorbu sekundárních indexů a následné dotazy nad nimi. Riak API také umožňuje hlubší nastavení autorizace a bezpečnosti, práci s replikami a řešení konfliktů.

2.7 Voldemort

Project Voldemort [44] je distribuovaná KDBS založena na Amazon DynamoDB. Škáluje horizontálně pro čtení i zápis. Umožňuje zapojení storage-enginu (MySQL, Read-Only). Databáze automaticky replikuje data napříč servery pro dostupnost a bezpečnost jednotlivých oddílů při vysoké propustnosti, nicméně každý server obsahuje pouze část z celkových dat. Databáze je decentralizovaná z pohledu uzlů, každý uzel je samostatný a nezávislý, nenachází se zde žádný centrální řídicí uzel nebo uzel řídicí řešení chyb. Voldemort má výkonost desítek tisíc operací za sekundu na jeden uzel (1 op. za 50 mikrosekund), samozřejmě závisí na hardwaru, síti, systému disku atp. Konzistence dat je nastavitelná (přísné kvórum nebo případná konzistence). Selhání serverů jsou ošetřována transparentně, pro lepší viditelnost, interní monitorování a validaci dat lze využívat JMX [45]. Data jsou verzována pro maximální integritu i během poruch. In-Memory caching pro eliminaci oddělených částí cache, jednoduché a rychlé in-memory testování (např. pro unit testy). Databáze umožňuje jednoduchou distribuci dat skrz stroje, data mohou být rozdělována například dle primárních klíčů. Databáze má hashovatelné schéma, vyhledávání dle primárního klíče a možnost modifikace jednotlivých hodnot. Voldemort poskytuje široké možnosti pro klíče i hodnoty díky serializaci včetně listů a tuplů s pojmenovanými poli. Pro serializaci (Java Serialization, Thrift, Avro) se využívá JSON datový model v kompaktním bytovém formátu, probíhá zde typová kontrola dat dle očekávaného schématu. Pomocí API je možné rozhodovat o replikování a místech ukládání dat, nastavení různé strategie pro specifické aplikace a možnost distribuce dat skrz data centra která jsou mezi sebou geologicky velice vzdálená. Databáze neposkytuje trigger, cizí klíče ani komplexní filtry pro dotazy.

Práce s Voldemort databází z pohledu klienta je přímočará, API se skládá pouze z pár základních funkcí pro správu dat. Tyto funkce jsou Put, Get a Del pro nastavení, získání a odstranění hodnot pro explicitně specifikovaný klíč. Funkce GetAll umožňuje obdržet více hodnot pro více specifikovaných klíčů pomocí volání pouze jedné funkce, GetAll dosahuje vyšší propustnosti než zřetěžené volání samostatné funkce Get. Pro připojení k Voldemort databázi a nastavení výchozího uzlu úložiště se využívá funkce Bootstrap, bez nastavení výchozího uzlu je potřeba specifikovat uzel explicitně před každým voláním funkce Get a dalších. Pro funkci Bootstrap je také možné nastavovat serializer pro klíče i hodnoty, čas spojení klienta se serverem a interval automatické změny uzlu v rámci clusteru na ten nejvhodnější. Pro ukončení komunikace se využívá jednoduše funkce Close.

2.8 InfinityDB

InfinityDB [46] je NoSQL hierarchicky tříděná KDBS implementovaná v jazyce Java. Databáze má možnost využít čistě In-Memory ukládání dat, která je vhodné pro cache, nebo naopak se mohou data ukládat i perzistentně na disk do souboru, přičemž je možné měnit nastavení bez zasahování do kódu. Přístup k datům v cache je plně vícevláknový, využívá se většina jader, a data, která nejsou často využívána, jsou stránkována na disk. Databáze dosahuje výkonu v jednotkách milionů operací za sekundu pro více vláknové operace v cache. Veškerá data a informace o databázi jsou uložena na disku v jednom souboru, což zajišťuje jejich aktuálnost a zároveň maximalizuje bezpečnost a korektnost. Databáze je designována právě pro použití jednoho kompletního souboru s okamžitým zotavením a nevyžaduje proto administraci. Databáze neobsahuje dodatečné konfigurační nebo dočasné soubory, upgrade skriptů ani logy. Zotavení je bez logů o transakcích okamžité ihned po restartu. V databázi není potřeba dělat čištění junk souborů po operacích, když zde nejsou žádné zanechány. InfinityDB podporuje ACID pro vlákna a ACD pro bulk operace. Databáze poskytuje prostor pro ukládání strukturovaných, polostrukturovaných a nestrukturovaných dat. Tento jednoduchý model umožňuje ukládání vnořených Multi-values a je možné reprezentovat různé datové struktury, jako jsou stromy, grafy, key/value mapy, dokumenty, velká řádká pole a tabulky. Schema je možné měnit za běhu pro zpětnou i následující kompatibilitu. Data dotazů lze dynamicky sledovat pomocí set logic views, delta views a ranges. Databáze se využívá pro servery, pracovní stanice a příruční zařízení.

InfinityDB poskytuje základní jednoduché API o deseti hlavních voláních. Funkcionalitu pro vkládání, úpravu a mazání hodnot zajišťují funkce Insert, Update a Delete. Funkce Delete je rozšířena o funkci Delete-suffixes, která umožňuje odstranit více hodnot v jednom volání. Pro získávání hodnot se využívá kurzoru, jehož pohyb v obou směrech zajišťují funkce First, Next, Last a Previous. Nakonec jsou k dispozici také potřebné funkce Commit a Rollback pro možnost využívání transakcí.

2.9 Memcached

Memcached [19] je open-source, distribuované, in-memory key-value úložiště, navržené pro rychlý a efektivní caching. Primárním účelem Memcached je uchovávání často používaných dat v paměti, aby se snížila latence a zvýšila rychlost přístupu k nim. Jeho jednoduchý a efektivní přístup k ukládání a získávání dat ho činí populární volbou pro webové servery, kde je potřeba rychle cachovat často používané informace jako například HTML stránky, databázové dotazy nebo výpočty. Memcached funguje jako distribuovaná cache, díky čemuž může být nasazen na více serverech, a data jsou mezi nimi rovnoměrně distribuována. Tento přístup umožňuje horizontální škálování, takže kapacitu a výkon Memcached lze jednoduše rozšiřovat přidáním dalších serverů.

Jednou z klíčových vlastností této KDBS je jeho jednoduché API, které podporuje základní operace s key-value páry, jako je ukládání, získávání a mazání dat. Tato funkcionalita je dostupná pomocí funkcí SET, GET, ADD, REPLACE a DELETE. Díky tomu je integrace Memcached do existujících aplikací relativně snadná a není vyžadována žádná složitá konfigurace. Memcached také poskytuje možnost nastavení expirace dat, což umožňuje automatické odstranění zastaralých dat z cache a uvolnění paměťových prostředků. Tato funkce je užitečná zejména pro udržování čerstvých a aktuálních dat v cache. Další významnou vlastností Memcachedu je jeho podpora distribuovaných transakcí, která umožňuje atomické operace nad více key-value páry. Pro práci s touto KDBS jsou k dispozici knihovny pro různé programovací jazyky, což usnadňuje jeho integraci do široké škály aplikací a systémů. Díky své jednoduchosti, efektivitě a škálovatelnosti je Memcached oblíbenou volbou pro mnoho webových aplikací a služeb.

2.10 Nezmíněné významné NoSQL databáze

Do práce nebyly záměrně zahrnuty databázové systémy MongoDB a Couchbase [47, 48]. I když se jedná o známé a hojně využívané NoSQL databáze, byly obě záměrně vynechány z práce, protože mají Key-value model až jako sekundární datový model. Primárně jsou určeny pro ukládání dokumentově orientovaných informací [49]. Další často využívanou a nezmíněnou NoSQL databází je Cassandra [50], která udává jako datový model wide-column store [51]. Z tohoto důvodu byla i tato DBS vyřazena z testování.

Tabulka 2.1: Porovnání Key-value databází

Databáze	Amazon DynamoDB	Oracle NoSQL DB	Redis	Aerospike	Oracle Berkeley DB	Riak KV	Voldemort	InfinityDB
Čistě cloud	ano	ne	ne	ne	ne	ne	ne	ne
Schéma dat	ne	ano i ne	ne	ne	ne	ne	ne	ano
Licence	komerční	open source	open source	open source	open source	open source	open source	komerční
Server OS	hostovaná	Linux, Solaris	Linux, Windows, OS X, BSD	Linux	Linux, Windows, OS X, Android ad.	Linux, OS X	Linux, Windows	Linux, Windows, OS X, Solaris
Napsáno v	-	Java	C	C	C, C++, Java	Erlang	Java	Java
Sekundární	ano	ano	ano	ano	ano	omezené	ne	ne
indexy								
Koncept	ACID	ACID	atomické,	atomické	ACID	ne	ne	ACID
transakcí		v rámci uzlu	izolované					
Triggery	ano	ne	pub/sub	ne	ano	ano	ne	ne
Dělení	sdílení	sdílení	sdílení	sdílení	ne	sdílení	ne	ne
metody								
Replikační	ano	source-replica	source-replica,	volitelná	source-replica	volitelný	ne	ne
metody		multi-region	multi-source	faktor repl.		faktor repl.		
Administrace	vysoká	nízká	vysoká	vysoká	vysoká	vysoká	vysoká	ne
Škálovatelnost	horizontální	horizontální	horizontální	lineární	horizontální	lineární	horizontální	horizontální
Odezva	mikrosekundy	milisekundy	milisekundy	milisekundy	mikrosekundy	milisekundy	milisekundy	milisekundy
Dotazovací	PartiQL	Omezený SQL	Redis	AQL	SQL	Riak	Voldemort	InfinityDB
jazyk			query			query	query	query

Kapitola 3

Prostředí pro testování databázových systémů

Různé databázové systémy mohou přistupovat k řešení jednotlivých problémů odlišně. Pokud chceme rozhodnout, který z těchto systémů je nejvhodnější pro určité úkoly, musíme provést řadu testů a porovnání. Je prakticky nemožné nalézt ideální databázový systém, který by exceloval ve všech aspektech pro všechna data a případy využití. Testování nám však umožní odhalit, který systém vyniká a naopak zaostává pro konkrétní operace nad určitými daty. Proto je důležité najít ideální prostředí pro možnost měření a porovnání vlastností vybraných KDBS v kapitole 2.

Existuje celá řada nástrojů pro měření výkonu databázových systémů. Mezi dva dosti známé a dostupné nástroje se řadí například TPC [20] a YCSB [21]. TPC benchmarky od Transaction Processing Performance Council se dělí do mnoha kategorií. Například TPC-H je považován spíše za benchmark pro systémy pro podporu rozhodování [52], zatímco TPCx-BB je benchmark pro Big Data. Obecně se TPC benchmarky využívají spíše pro typické relační databázové systémy. Na druhou stranu Yahoo! Cloud Serving Benchmark (dále jen YCSB) od společnosti Yahoo! je open-source specifikace a sada programů pro vyhodnocování možností vyhledávání a údržby počítačových programů. Často se ale právě YCSB používá k porovnání relativního výkonu NoSQL databázových systémů, což je pro tuto práci zaměřenou na KDBS ideální. Proto byla tato technologie zvolena pro měření výkonu jednotlivých databázových systémů [53, 54].

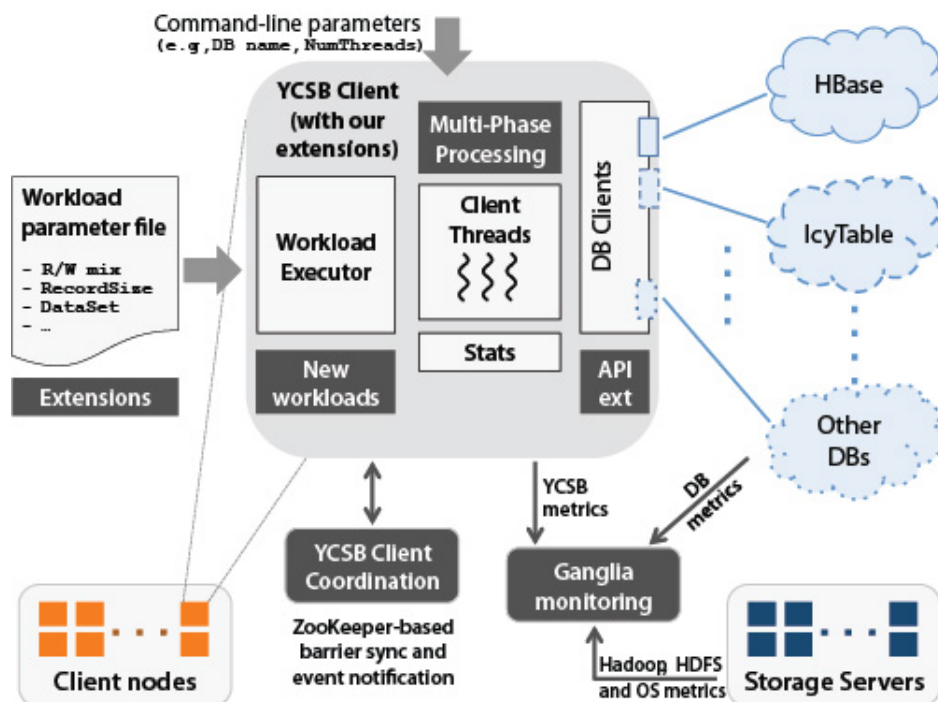
3.1 YCSB

YCSB [21, 55] architektura je založena na pluginech a poskytuje snadnou rozšiřitelnost pomocí skriptů. Pro značnou část významných databázových systémů existuje podpora v podobě bindingů. Samotný benchmark se skládá ze dvou fází. První z nich je Loading fáze zaměřená na vložení dat do DBS a následně druhá je Running fáze, ve které se spouští daný test (viz schéma 3.1).

Při spouštění každého testu je možné nastavit určité parametry pro lepší konkretizaci měřeného scénáře [56]. První a druhý parametr slouží pro specifikaci loading nebo running fáze a výběr testovaného databázového systému. Následně se vybírá testovaný scénář (Workload), počet záznamů v databázi, počet atributů daného záznamu, bytovou velikost každého atributu v záznamu, počet vláken, umístění serveru databáze a nakonec distribuci dotazů (uniformní, exponenciální, sekvenční, nejnovější, hotspot, definované).

YCSB poskytuje 5 různých scénářů označených A až F pro testování propustnosti, odezvy a škálovatelnosti jednotlivých databázových systémů. Tyto pracovní scénáře, neboli Workloads [53, 57], napodobují různé chování požadavků webových aplikací, jako jsou scénáře zaměřené výhradně na čtení, zápis nebo kombinace obojího. Konkrétní počet zvolených operací dle procentuální definice je vypočítán na základě parametru určujícího celkový počet operací daného scénáře. Při sekvenčním skenování ve Workloadu E je maximální počet skenovaných záznamů v jedné operaci definován jako 5% z celkového počtu záznamů. Takže při počtu záznamů 1000 bude každá operace skenování číst právě 1 až 50 záznamů.

- Workload A (Update heavy)
 - 50% operací zaměřených na čtení a 50% operací zaměřených na úpravu
- Workload B (Read mostly)
 - 95% operací zaměřených na čtení a 5% operací zaměřených na úpravu
- Workload C (Read only)
 - 100% operací zaměřených na čtení
- Workload D (Read latest)
 - 95% operací zaměřených na čtení, 5% operací zaměřených na vkládání a poslední vložené záznamy jsou čteny přednostně
- Workload E (Short ranges)
 - 95% operací zaměřených na sekvenční skenování nízkého počtu záznamů a 5% operací zaměřených na vkládání
- Workload F (Read-modify-write)
 - každá operace se skládá z čtení daného záznamu, úpravy záznamu a následného vložení změněného záznamu zpět



Obrázek 3.1: YCSB rámec testování funkčnosti [58]

3.2 TPC

Transaction Processing Performance Council [20], dále jen TPC, je společnost spravující software pro vytváření kvalitních benchmarků výkonnosti systémů pro online zpracování transakcí (OLTP) [59] a možnosti jejich následného monitoringu a porovnávání. TPC benchmarky poskytují spolehlivé testy pro velké firmy s průměrnou zátěží, výsledkem benchmarků je počet transakcí za minutu (tpm).

TPC benchmarky jsou rozděleny do více modelů pro různě specifikované testy. Prvním z modelů pro OLTP byl TPC Benchmark A (TPC-A), který následně nahradil benchmark TPC-B a aktuálně se v tomto odvětví využívá poslední generace OLTP benchmarků TPC-C a TPC-E. Například modely TPC-DS/DI a TPC-H jsou uzpůsobeny pro benchmark pro systémy pro podporu rozhodování [52]. TPC benchmarky jsou přizpůsobeny i pro virtualizaci, IoT [60] a další (viz tabulka TPC benchmarků 3.1).

Model TPC-C [61] simuluje velkoobchodní provoz s více sklady, známý jednoduše jako "společnost". V minimálním testu má společnost deset skladů, každý s deseti uživatelskými terminály. Každý sklad obsluhuje deset definovaných prodejních okrsků, každý s 3000 zákazníky, kteří objednávají podle katalogu výrobků o 100 000 položkách. Nejčastějšími transakcemi jsou objednávky zákazníků, přičemž každá objednávka obsahuje v průměru 10 položek, a platby zákazníků. Méně časté požadavky se dotazují na stav objednávek a skladových zásob, expedují objednávky a doplňují zásoby, které se snížily. Pro testování výkonnosti daného systému se počet skladů zvyšuje tak, aby

TPC benchmark	využití
TPC-C, TPC-E	zpracovávání transakcí
TPC-H, TPC-DS, TPC-DI	podpora rozhodování
TPC _x -V, TPC _x -HCI	virtualizace
TPC _x -HS, TPC _x -BB	velká data
TPC _x -IoT	IoT
TPC _x -AI	umělá inteligence
TPC-Energy, TPC-Pricing	běžné specifikace
TPC-A, TPC-B, TPC-APP, TPC-D	zastaralé benchmarky
TPC-R, TPC-W, TPC-VMS	

Tabulka 3.1: TPC benchmarky

splňoval požadované minimum potřebné k měření cílové úrovně výkonnosti.

Výsledky srovnávacího testu se měří v transakcích za minutu, známých jako tpmC. První výsledek tpmC byl zveřejněn v září 1992 pro IBM AS/400 a přinesl výsledek 54 tpmC. V roce 2000 byl průměrný výsledek pro špičkové stroje 2,4 milionu tpmC a společnosti ve snaze získat rekord stavěly systémy velmi velkých rozměrů. Současný rekord byl stanoven v roce 2020 pomocí cloud computingu, který poskytl 707,3 milionu tpmC [62]. Nedávné výsledky pro menší lokální systémy se zaměřily na snížení nákladů na tpmC.

3.3 Relevance TPC a YCSB pro měření KDBS

Pokud jde o relevanci TPC pro KDBS, je potřeba brát v úvahu, že KDBS se liší od tradičních relačních databází. Zatímco TPC benchmarky se zaměřují na transakční zpracování a operace typické pro RDBS, KDBS se často používají pro rychlý přístup k datům pomocí jednoduchého klíč-hodnota rozhraní a jsou často součástí aplikací s vysokým objemem operací typu čtení a zápisu.

Z tohoto důvodu by benchmarky navržené specificky pro KDBS byly relevantnější pro měření jejich výkonnosti a charakteristik. Nicméně některé aspekty TPC benchmarků, jako je schopnost zpracovávat vysoký objem transakcí nebo zátěžové testování, mohou poskytnout užitečné informace i pro KDBS, i když nejsou primárně určeny pro tuto kategorii databází.

Pokud jde o relevanci YCSB pro KDBS, můžeme říci, že je velmi relevantní. To je způsobeno tím, že YCSB je zaměřen na testování databází pomocí jednoduchého klíč-hodnota rozhraní, což je přesně ten způsob, jakým komunikují KDBS.

YCSB poskytuje sadu připravených scénářů a operací, které simulují reálné aplikace a zátěžové podmínky. To umožňuje uživatelům testovat výkonnost KDBS v různých situacích a porovnávat je s jinými NoSQL DBS. Celkově lze říci, že YCSB je relevantní nástroj pro testování KDBS a poskytuje užitečné informace o jejich výkonu a škálovatelnosti v scénářích podobných reálnému použití.

Kapitola 4

Vyhodnocení výsledků testů

4.1 Testovací prostředí

Veškeré testy byly spouštěny na vlastním stroji, domácím počítači. Konkrétní specifikace tohoto stroje se nachází v tabulce (viz tabulka 4.1).

4.2 Zprovoznění testů

Pro rozsáhlé otestování byly vybrány čtyři vhodné KDBS. A to konkrétně Redis (2.3), Riak KV (2.6), Aerospike (2.4) a Memcached. Všechny tyto zvolené databáze jsou v aktuálním roce 2024 hodnoceny jako jedny z nejlepších podle žebříčku na webu DB-Engines Ranking [15] právě pro testovaný model Key-value. Tento web přiřazuje databázím bodové hodnocení na základě četnosti nových článků o dané databázi na internetu, obecného zájmu, četnosti diskuzí na fórech, množství pracovních nabídek a poptávek a relevanci na sociálních sítích.

Ve snaze o možnost replikace testů byly všechny databáze instalovány a spouštěny pomocí open-source platformy Docker [25]. Bylo tedy nutné najít vhodné a kompatibilní docker images pro každou z testovaných databází. V kontextu této práce Docker pomáhá zrychlit zdlouhavou fázi instalování

Tabulka 4.1: Specifikace stroje na kterém se spouštěly testy

komponent	název	podrobnosti
OS	Microsoft Windows 10 PRO	x64
CPU	Intel Core i5 4590	3,3GHz (Boost 3,7GHz), core/thread 4, Haswell
GPU	NVIDIA GeForce GTX 1660 SUPER	6GB, 1530MHz (Boost 1785MHz)
RAM	Crucial Ballistix Sport	8GB (2x4GB), 1600MHz, DDR3
SSD	Samsung 870 EVO	R/W 560/530MB/s, 1TB, TLC, SATA 6Gb/s
Základní deska	GIGABYTE GA-H81M-H - Intel H81	1150 socket, DDR3 DIMM

a nastavení počátečního stavu databází, udržení funkčnosti vybrané verze instalovaného softwaru a odstínění od stavu stroje, na kterém DBS spouštíme.

Veškeré testy byly vytvářeny a spouštěny pomocí frameworku YCSB (3.1). YCSB framework v první části, Load, do databáze vložil data, a následně v druhé části, Run, spustil testy a vrátil hodnoty výsledků. Pomocí přidání volitelných parametrů [63] bylo možné testy upravit podle vlastních potřeb.

Po spuštění databáze v Dockeru se k ní připojil YCSB framework, který následně prováděl testování nad zvolenou připojenou databází. Pro možnost komunikace bylo zapotřebí zprovoznit YCSB binding pro každou z databází, aby YCSB framework mohl úspěšně komunikovat se zvolenou databází, vložit data, spustit testy a vrátit patřičné výsledky.

4.3 Popis parametrů testů

Veškeré testy pro každou z testovaných databází byly spuštěny třikrát, a finální výsledek byl tedy průměrem ze všech tří testů pro každou databázi v dané testovací kategorii. Každý test byl spuštěn paralelně na čtyřech vláknech.

Do databáze bylo vždy vloženo 100 000 záznamů a následující test prováděl 1 000 000 dotazů nad danou naplněnou databází. Následně byla DBS vyprázdněna a celý proces se opakoval ještě dvakrát.

Testy byly prováděny ve třech YCSB kategoriích (tzv. YCSB Workload [57]). Workload A (Update-heavy: 50% read, 50% update), Workload B (Read-mostly: 95% read, 5% update) a Workload C (Read-only) (viz YCSB Workloads 3.1).

Pro každý Workload a jednotlivé databáze byla vytvořena tabulka výsledků jednotlivých testů a výsledný průměr těchto testů. Mezi nejdůležitější výsledky testů patří celková doba trvání testu, propustnost a 95/99 percentil odezvy na operaci. V tabulce jsou také data o počtu provedených operací, průměrné odezvě na operaci, a také minimální, průměrná a maximální doba odezvy na operaci.

Určité databáze vyžadují i další parametry pro spuštění testování, nejčastěji to jsou host IP adresy nebo porty dané DBS pro možné spojení. Dále je například možné specifikovat cestu k souborům, se kterými pracuje testovací prostředí, nebo stanovit TTL [31] testovaných dat, aby bylo možné vyhnout se expiraci dat při dlouhotrvajících nebo vzájemně opožděných testech.

4.4 Spouštění testů

Pro spuštění testů je zapotřebí nejprve nainstalovat, nastavit a spustit vybranou databázi. Následně se musíme k DBS připojit testovacím prostředím YCSB pro možnou komunikaci s DBS a naplnit ji daty. Nakonec jsou spuštěny požadované testy, po dokončení testů nám prostředí YCSB vrátí tabulku výsledků v textové podobě.

Nainstalujeme platformu Docker [64] pro snadnou práci s instalovanými databázemi. Pokud chceme využívat grafického rozhraní pro prostředí Docker je možnost i instalace platformy Docker Desktop [65, 66] (případně si samozřejmě můžeme DBS instalovat i jiným pro nás vhodnějším způsobem). Docker Desktop bych doporučil pro práci s DBS u kterých se plánuje větší úprava konfiguračních souborů, případně pro sledování chování databáze za běhu prostřednictvím logování.

Práce s Docker Desktop je velice intuitivní, proto se zaměříme na práce s Docker CLI [66]. Po zvolení vhodné databáze si musíme stáhnout Docker Image [67] této DBS [68] pomocí příkazu `"docker image pull [OPTIONS] NAME[:TAG|@DIGEST]"` [69]. Druhým krokem je nastavení a spuštění DBS, vytvoříme tedy pomocí Docker Image náš Docker Container [67] který spustíme příkazem `"docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]"` [70]. Nesmíme zapomenout nastavit správné čísla portů pro Docker Container, jinak nebude testovací prostředí YCSB schopno komunikovat s Docker Image.

Testovací prostředí YCSB je možno získat z oficiálního Yahoo! Cloud Serving Benchmark GitHubu [21]. Pokud nechceme manuálně překládat zdrojové kódy projektu pro možnost nasazení vlastních změn, je vhodné stáhnout funkční verzi na odkazu zmíněném v README souboru [71]. Pro úspěšné sestavení projektu ze zdrojových kódů je zapotřebí mít nainstalovaný Apache Maven [72] verze 3, pokud je na stroji Maven verze 2 bude docházet k chybám (používanou verzi je možné zkontrolovat pomocí příkazu `"mvn -version"`). Dále je zapotřebí mít zprovozněné Java JDK 8 (1.8) [73] a výše. Na OS Windows doporučuji zkontrolovat hodnotu systémové proměnné [74] `"%JAVA_HOME%"` zda je nastavena správně příkazem `"echo %JAVA_HOME%"`, případně ji manuálně změnit (správně by nám měl příkaz vrátit cestu k nainstalovanému Java JDK s verzí minimálně 1.8 např. `"C:\Program Files\Java\jdk1.8.0_231"`).

Po úspěšném zprovoznění testovacího prostředí YCSB a nastavení DBS můžeme přejít na poslední fázi testování. Nejprve musíme DBS naplnit daty a následně spustit testy. Veškeré testy byly spouštěny prostřednictvím skriptů vložených do příkazového prostředí Windows PowerShell [75], nicméně není problém skripty vkládat i do obyčejné příkazové řádky pokud se vymění obrácená lomítka za obyčejná lomítka a vynechá se tečka na začátku každého příkazu. Všechny příkazy byly spouštěny v adresáři `"bin"`, uvnitř kořenového adresáře projektu `"ycsb-0.17.0"`.

V první části testování spustíme příkaz `"ycsb load"` pro připojení k databázi a vložení dat do DBS. Nejprve musíme specifikovat název testované DBS a následně zvolený Workload [57] (viz popis YCSB Workloads 3.1). Pomocí volitelných parametrů [63] na konci každého příkazu specifikujeme námi zvolené požadavky. Například počet záznamů které chceme do DBS vložit parametrem `"-p recordcount=NUM"`, počet vláken `"-p threadcount=NUM"` a nebo nejčastěji IP adresu či port DBS `"-p DBS_NAME.host=IP_ADR -p DBS_NAME.port=NUM"` pro spojení s DBS. Pomocí dodatku `"-s"` po názvu DBS dosáhneme podrobnějšího výstupu při testování včetně chybových hlášek a logů. Celý příkaz pro vložení dat do DBS pak vypadá takto `"ycsb load DBS_NAME -s -P WORKLOAD [-p PARAMETERS]"`. Po dokončení vkládání dat získáme mimo data v DBS i textový výstup popisující průběh zápisu dat do DBS, nejpodstatnější je zde mimo jiné propustnost

vkládání a čas odezvy na operace rozdělen do sekce minimum, průměr, maximum a percentily (95%, 99%).

Druhá část testování se věnu spuštění operací pro čtení a úpravu vložených dat z předešlého kroku pomocí příkazu "ycsb run". Opět musíme zadat název testované DBS a Workload [57] který nám specifikuje poměr operací pro čtení a úpravy dat. Pomocí volitelných parametrů [63] určíme počet prováděných operací "-p operationcount=NUM". Kombinace celkového počtu operací a Workloadu nám udá konkrétní počet operací čtení a úprav které se během testu vykonají. Pokud pracujeme s Workloadem D nebo E (viz popis YCSB Workloads 3.1) tak dochází i k zápisu nových hodnot do DBS a počet klíčů v DBS stoupá, ostatní Workloady provádějí pouze úpravy již existujících prvků a počet klíčů v DBS je během testu stále stejný. Stejným způsobem jako v první fázi testování i nyní můžeme specifikovat IP adresu a port DBS, případně počet vláken, podrobnější výstup testu atd. Celý příkaz pro spuštění testu DBS pak vypadá takto "ycsb run DBS_NAME -s -P WORKLOAD [-p PARAMETERS]". Po dokončení testu získáme opět textový výstup v podobě tabulky popisující průběh testu, nejpodstatnější je zde mimo jiné propustnost operací a čas odezvy na operace rozdělen do sekce minimum, průměr, maximum a percentily (95%, 99%).

Před reálným testováním můžeme vyzkoušet zda vše správně nastaveno pomocí spuštěný rychlého demo. Demo se skládá ze dvou příkazů, naplnění databáze tisíci záznamy a následného otestování, provedení přibližně 500 operací čtení a 500 úprav na databázi BasicDB [76]. Po spuštění obou příkazů by nám měly testy na konci vrátit status "Return=OK".

- `.\ycsb load basic -P ..\workloads\workloada`
- `.\ycsb run basic -P ..\workloads\workloada`

Pro názorný příklad si popíšeme a ukážeme příkazy pro testování databáze Redis [13] (viz obrázky 4.1, 4.2, 4.3). Test poběží na 4 vláknech. Testovat se bude případ Workload A [57] (50% Read, 50% Update). Do databáze se vloží 10 000 záznamů, a provede se 10 000 operací při testování (přibližně 5 000 čtení a 5 000 úprav dat).

- `docker pull redis:latest`
- `docker run --name my-redis -p 6379:6379 -d redis:latest`
- `.\ycsb load redis -P ..\workloads\workloada -p redis.host=127.0.0.1 -p redis.port=6379 -p recordcount=10000 -p threadcount=4`
- `.\ycsb run redis -P ..\workloads\workloada -p redis.host=127.0.0.1 -p redis.port=6379 -p operationcount=10000 -p threadcount=4`
- `docker stop aba5fab123example4568aa97`
- `docker rm aba5fab123example4568aa97`

```
PS E:\ycsb-0.17.0\bin> docker pull redis:latest
latest: Pulling from library/redis
Digest: sha256:f14f42fab123example456c7e824b9
Status: Image is up to date for redis:latest
docker.io/library/redis:latest

PS E:\ycsb-0.17.0\bin> docker run --name my-redis -p 6379:6379 -d redis:latest
aba5fab123example4568aa97

PS E:\ycsb-0.17.0\bin> docker stop aba5fab123example4568aa97
aba5fab123example4568aa97

PS E:\ycsb-0.17.0\bin> docker rm aba5fab123example4568aa97
aba5fab123example4568aa97
```

Obrázek 4.1: Docker Redis, příkazy pro stažení, spuštění, zastavení a odstranění DBS

4.5 Výsledky testů


```

PS E:\ycsb-0.17.0\bin> .\ycsb load redis -P ..\workloads\workloada
-p redis.host=127.0.0.1 -p redis.port=6379 -p recordcount=10000
-p threadcount=4
YCSB Client 0.17.0
Loading workload...
Starting test.
[OVERALL], RunTime(ms), 4211
[OVERALL], Throughput(ops/sec), 2374.7328425552128
[TOTAL_GCS_PS_Scavenge], Count, 4
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 6
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.14248397055331277
[TOTAL_GCS_PS_MarkSweep], Count, 0
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 0
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.0
[TOTAL_GCS], Count, 4
[TOTAL_GC_TIME], Time(ms), 6
[TOTAL_GC_TIME_%], Time(%), 0.14248397055331277
[CLEANUP], Operations, 4
[CLEANUP], AverageLatency(us), 333.25
[CLEANUP], MinLatency(us), 80
[CLEANUP], MaxLatency(us), 1060
[CLEANUP], 95thPercentileLatency(us), 1060
[CLEANUP], 99thPercentileLatency(us), 1060
[INSERT], Operations, 10000
[INSERT], AverageLatency(us), 1647.61
[INSERT], MinLatency(us), 904
[INSERT], MaxLatency(us), 13895
[INSERT], 95thPercentileLatency(us), 2481
[INSERT], 99thPercentileLatency(us), 3389
[INSERT], Return=OK, 10000

```

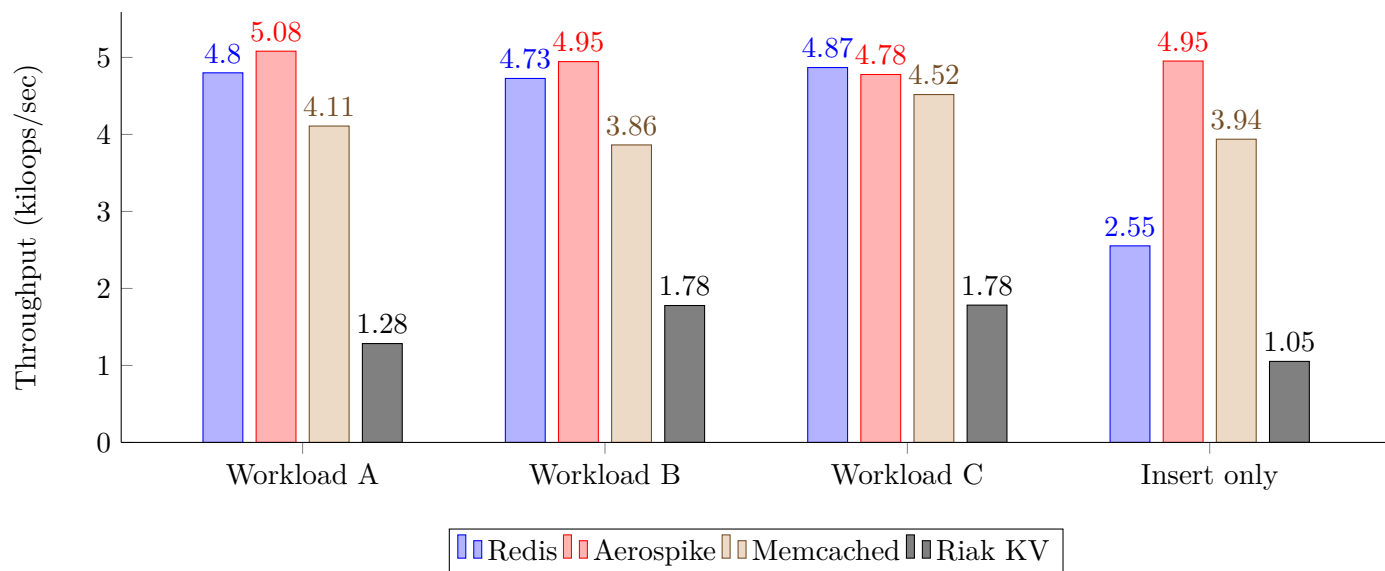
Obrázek 4.2: YCSB Redis, příkaz pro vložení dat (load)

```

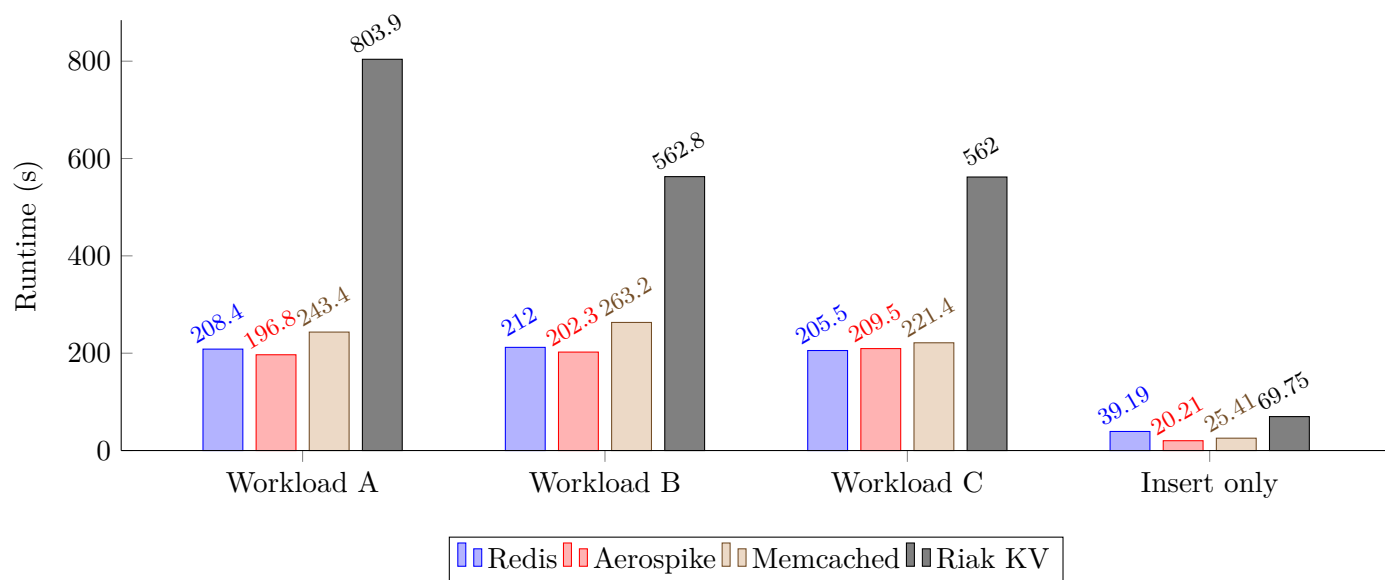
PS E:\ycsb-0.17.0\bin> .\ycsb run redis -P ..\workloads\workloada
-p redis.host=127.0.0.1 -p redis.port=6379 -p operationcount=10000
-p threadcount=4
YCSB Client 0.17.0
Loading workload...
Starting test.
[OVERALL], RunTime(ms), 2560
[OVERALL], Throughput(ops/sec), 3906.25
[TOTAL_GCS_PS_Scavenge], Count, 1
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 1
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.0390625
[TOTAL_GCS_PS_MarkSweep], Count, 0
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 0
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.0
[TOTAL_GCs], Count, 1
[TOTAL_GC_TIME], Time(ms), 1
[TOTAL_GC_TIME_%], Time(%), 0.0390625
[READ], Operations, 5157
[READ], AverageLatency(us), 998.8885010665116
[READ], MinLatency(us), 389
[READ], MaxLatency(us), 17695
[READ], 95thPercentileLatency(us), 1692
[READ], 99thPercentileLatency(us), 3479
[READ], Return=OK, 5157
[CLEANUP], Operations, 4
[CLEANUP], AverageLatency(us), 221.75
[CLEANUP], MinLatency(us), 60
[CLEANUP], MaxLatency(us), 638
[CLEANUP], 95thPercentileLatency(us), 638
[CLEANUP], 99thPercentileLatency(us), 638
[UPDATE], Operations, 4843
[UPDATE], AverageLatency(us), 985.7646087136072
[UPDATE], MinLatency(us), 364
[UPDATE], MaxLatency(us), 17567
[UPDATE], 95thPercentileLatency(us), 1744
[UPDATE], 99thPercentileLatency(us), 3675
[UPDATE], Return=OK, 4843

```

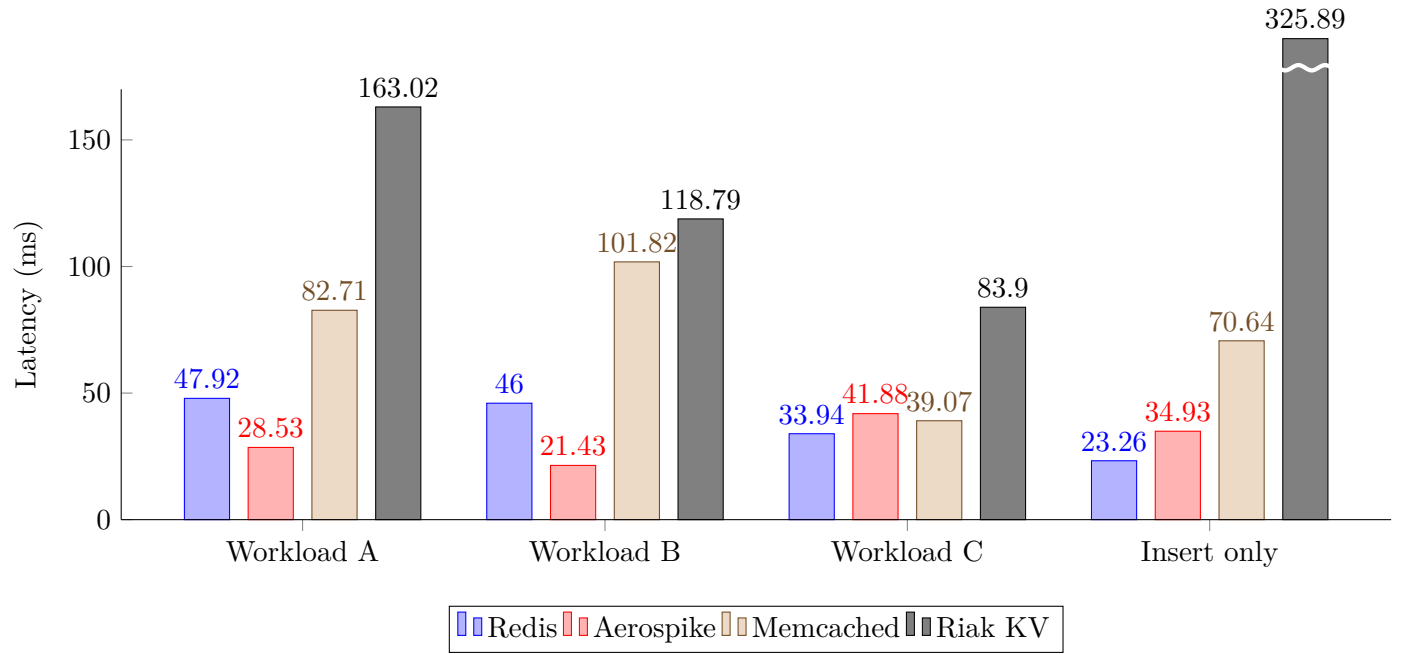
Obrázek 4.3: YCSB Redis, příkaz pro spuštění testů (run)



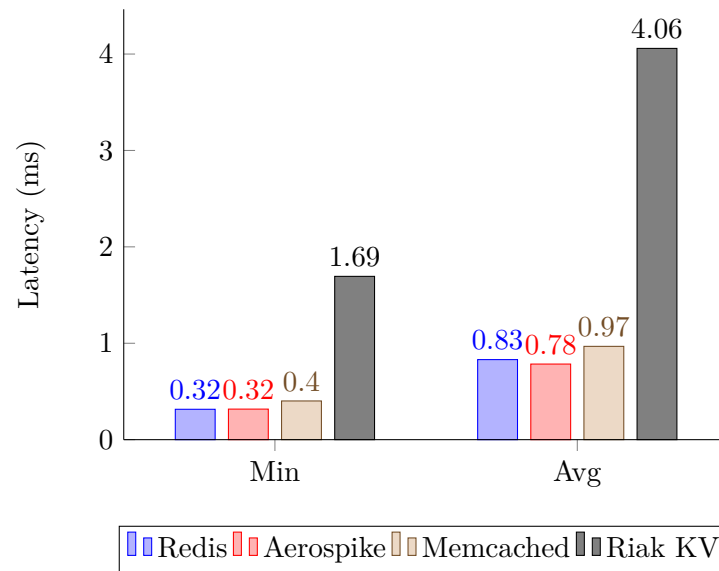
Obrázek 4.4: Workloads A, B, C + Insert only - Throughput (kiloops/sec)



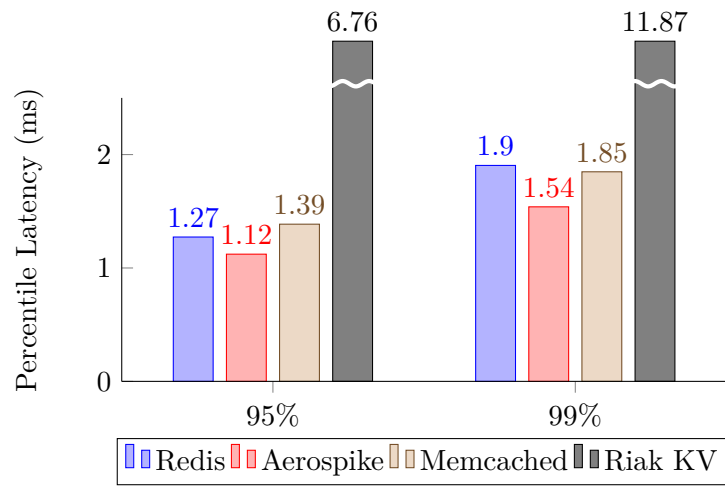
Obrázek 4.5: Workloads A, B, C + Insert only - Runtime (s)



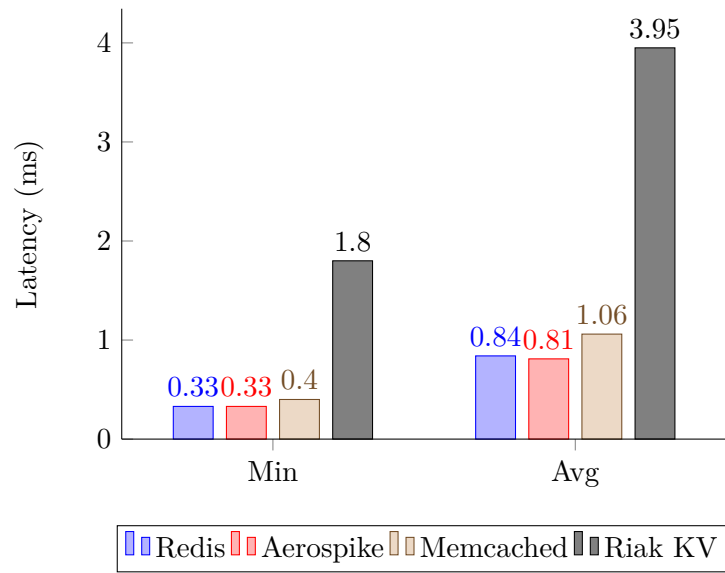
Obrázek 4.6: Workloads A, B, C + Insert only - Max Latency (ms)



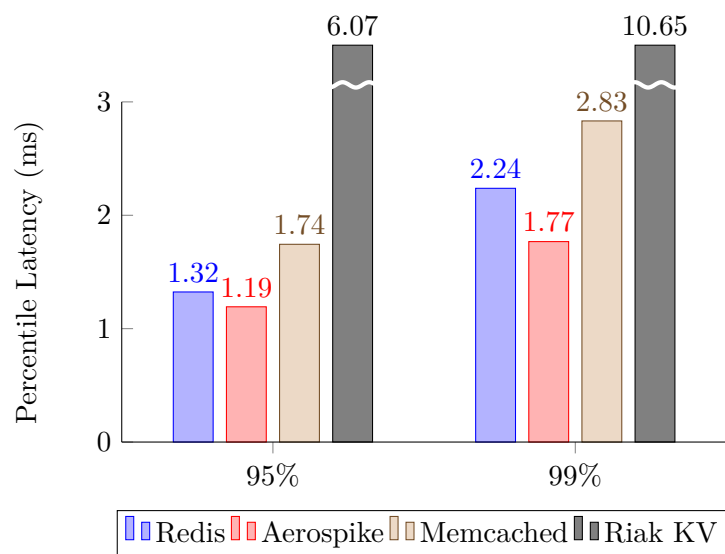
Obrázek 4.7: Workload A - Min Avg Latency (ms)



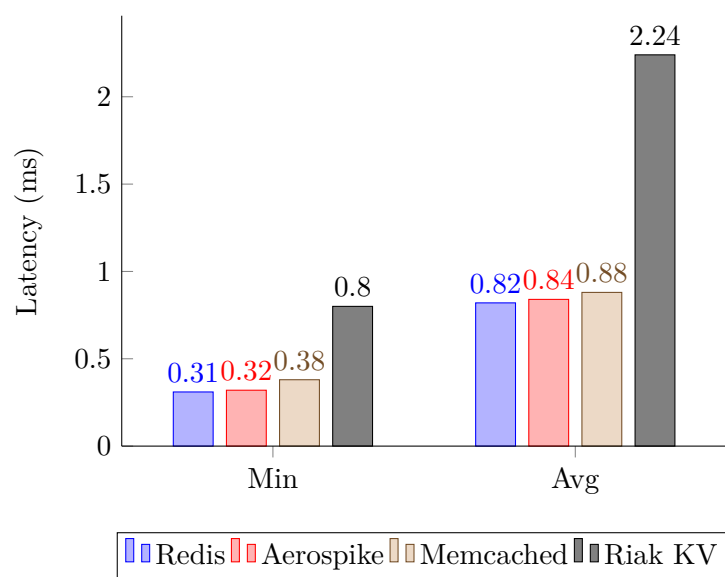
Obrázek 4.8: Workload A - Percentile Latency (ms)



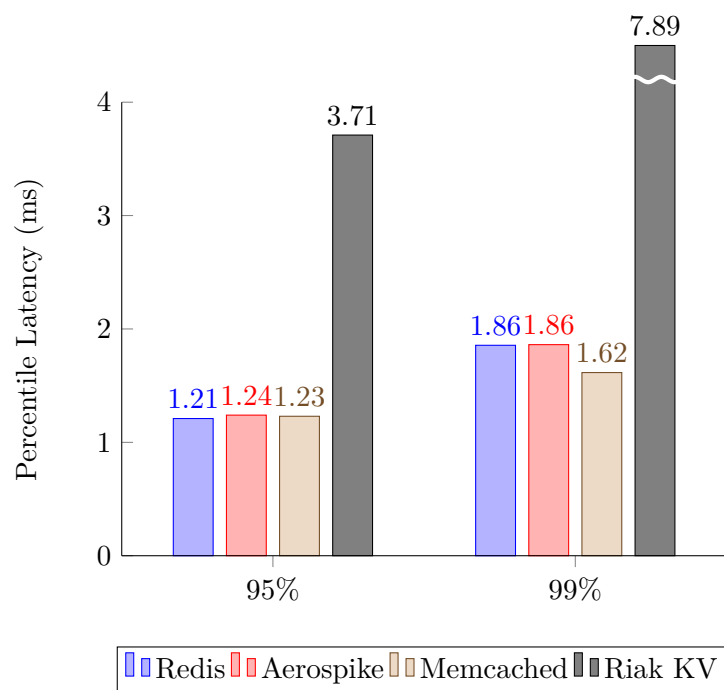
Obrázek 4.9: Workload B - Min Avg Latency (ms)



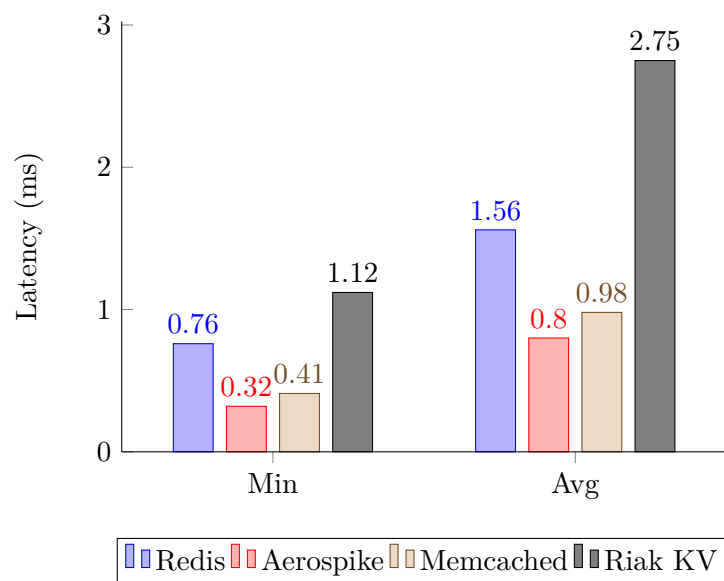
Obrázek 4.10: Workload B - Percentile Latency (ms)



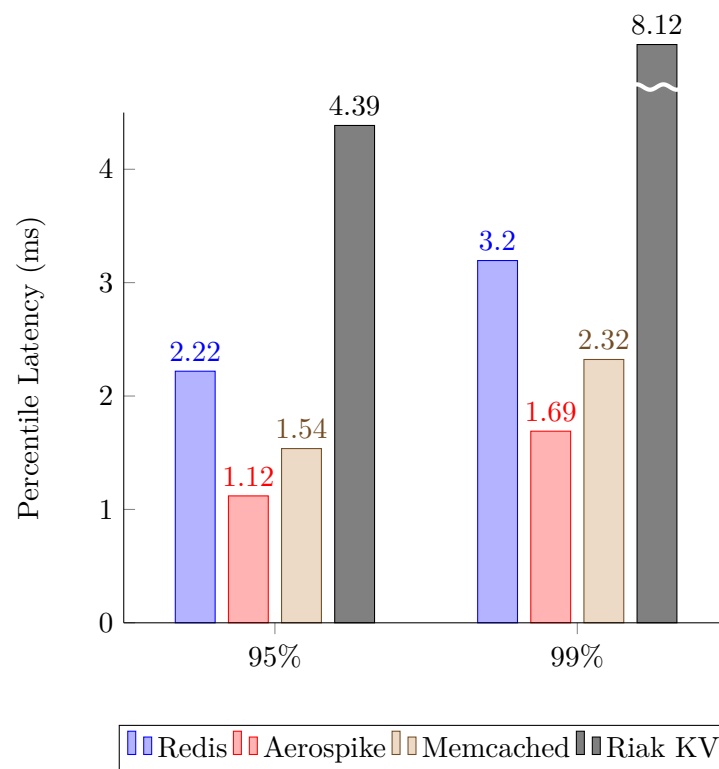
Obrázek 4.11: Workload C - Min Avg Latency (ms)



Obrázek 4.12: Workload C - Percentile Latency (ms)



Obrázek 4.13: Insert only - Min Avg Latency (ms)



Obrázek 4.14: Insert only - Percentile Latency (ms)

Kapitola 5

Závěr

TODO

Literatura

1. *What Is a Key-Value Database?* [online]. 2022. [cit. 2022-11-18]. Dostupné z: <https://aws.amazon.com/nosql/key-value/>.
2. *How do NoSQL databases work? Simply Explained! youtube* [online]. 2021. [cit. 2022-11-18]. Dostupné z: <https://www.youtube.com/watch?v=0buKQHokLK8>.
3. *What is a Relational Database (RDBMS)?* [online]. 2024. [cit. 2024-04-23]. Dostupné z: <https://www.oracle.com/cz/database/what-is-a-relational-database/>.
4. *Relace* [online]. 2024. [cit. 2024-04-23]. Dostupné z: <https://portal.matematickabiologie.cz/index.php?pg=zaklady-informatiky-pro-biology--teoreticke-zaklady-informatiky--teorie-mnozin--relace>.
5. *Efektivnější SQL dotazy pomocí plánování a optimalizace* [online]. 2024. [cit. 2024-04-23]. Dostupné z: <https://cs.khanacademy.org/computing/computer-programming/sql/relational-queries-in-sql/a/more-efficient-sql-with-query-planning-and-optimization>.
6. FOOTE, Keith D. *Key-Value Databases Demystified* [online]. 2023. [cit. 2024-04-23]. Dostupné z: <https://www.dataversity.net/understanding-key-value-databases/>.
7. PODETI, Suresh. *Schema-on-read vs Schema-on-write* [online]. 2023. [cit. 2024-04-23]. Dostupné z: <https://medium.com/@sureshpodeti/schema-on-read-vs-schema-on-write-624a1f1343e4>.
8. *DBMS popularity broken down by database model* [online]. 2024. [cit. 2024-04-23]. Dostupné z: https://db-engines.com/en/ranking_categories.
9. *2019 Database Trends – SQLTM vs. NoSQL, Top Databases, Single vs. Multiple Database Use* [online]. 2019. [cit. 2024-04-23]. Dostupné z: <https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/>.
10. *DB-Engines* [online]. 2024. [cit. 2024-04-23]. Dostupné z: <https://db-engines.com/en/>.
11. *Method of calculating the scores of the DB-Engines Ranking* [online]. 2024. [cit. 2024-04-23]. Dostupné z: https://db-engines.com/en/ranking_definition.

12. *DB-Engines Ranking - Trend Popularity* [online]. 2024. [cit. 2024-04-23]. Dostupné z: https://db-engines.com/en/ranking_trend.
13. *Redis* [online]. 2022. [cit. 2022-11-18]. Dostupné z: <https://redis.io/>.
14. *Oracle - Database* [online]. 2024. [cit. 2024-04-23]. Dostupné z: <https://www.oracle.com/database/>.
15. *DB-Engines Ranking* [online]. 2022. [cit. 2022-11-18]. Dostupné z: <https://db-engines.com/en/ranking>.
16. *DB-Engines Ranking of Key-value Stores* [online]. 2024. [cit. 2024-04-22]. Dostupné z: <https://db-engines.com/en/ranking/key-value+store>.
17. *Amazon DynamoDB* [online]. 2022. [cit. 2022-11-18]. Dostupné z: <https://aws.amazon.com/dynamodb/>.
18. *Azure Cosmos DB* [online]. 2024. [cit. 2024-04-22]. Dostupné z: <https://azure.microsoft.com/cs-cz/products/cosmos-db>.
19. *What is Memcached?* [online]. 2024. [cit. 2024-04-17]. Dostupné z: <https://memcached.org/>.
20. *TPC* [online]. 2023. [cit. 2023-02-11]. Dostupné z: <https://www.tpc.org/>.
21. *Yahoo! Cloud Serving Benchmark (YCSB)* [online]. 2019. [cit. 2024-04-22]. Dostupné z: <https://github.com/brianfrankcooper/YCSB>.
22. *Riak KV* [online]. 2022. [cit. 2022-11-21]. Dostupné z: <https://riak.com/products/riak-kv/index.html>.
23. *Aerospike* [online]. 2022. [cit. 2022-11-20]. Dostupné z: <https://aerospike.com/>.
24. *Top NoSQL Key Value store Databases: Predictiveanalyticstoday* [online]. 2022. [cit. 2022-11-13]. Dostupné z: <https://www.predictiveanalyticstoday.com/top-sql-key-value-store-databases/>.
25. *Docker Builds: Now Lightning Fast* [online]. 2024. [cit. 2024-03-19]. Dostupné z: <https://www.docker.com/>.
26. *Introduction to Oracle NoSQL Database* [online]. 2024. [cit. 2024-04-21]. Dostupné z: https://docs.oracle.com/cd/E26161_02/html/GettingStartedGuide/introduction.html.
27. *WHAT IS A KEY-VALUE DATABASE?* [online]. 2024. [cit. 2024-04-21]. Dostupné z: <https://redis.io/nosql/key-value-databases/>.
28. *Distribované databázové systémy (DDBS)* [online]. 2024. [cit. 2024-04-17]. Dostupné z: http://langer.zam.slu.cz/teaching/dbaii/distribuvane_db_systemy.pdf.
29. ANANDHI, R.; CHITRA, K. *A Challenge in Improving the Consistency of Transactions in Cloud Databases - Scalability* [online]. 2012. [cit. 2024-04-17]. Dostupné z: <https://research.ijcaonline.org/volume52/number2/pxc3881485.pdf>.

30. *Best Document Databases: G2* [online]. 2022. [cit. 2022-11-13]. Dostupné z: <https://www.g2.com/categories/document-databases>.
31. *Using Time to Live (TTL)* [online]. 2024. [cit. 2024-04-25]. Dostupné z: https://docs.oracle.com/cd/E17277_02/html/GettingStartedGuide/timetolive.html.
32. *Amazon Virtual Private Cloud (VPC)* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://aws.amazon.com/vpc/>.
33. *PartiQL* [online]. 2023. [cit. 2023-01-09]. Dostupné z: <https://partiql.org/docs.html>.
34. *Oracle NoSQL Database* [online]. 2022. [cit. 2022-11-19]. Dostupné z: <https://www.oracle.com/database/nosql/technologies/nosql/>.
35. *Redis CLI* [online]. 2023. [cit. 2023-01-14]. Dostupné z: <https://redis.io/docs/manual/cli/>.
36. *Hybrid Memory Architecture* [online]. 2022. [cit. 2022-11-20]. Dostupné z: <https://aerospike.com/products/features/hybrid-memory-architecture/>.
37. *Aerospike Quick Look (AQL)* [online]. 2022. [cit. 2022-11-20]. Dostupné z: <https://docs.aerospike.com/tools/aql>.
38. *Oracle Berkeley DB* [online]. 2022. [cit. 2022-11-21]. Dostupné z: <https://www.oracle.com/database/technologies/related/berkeleydb.html>.
39. *Multiversion concurrency control* [online]. 2022. [cit. 2022-11-21]. Dostupné z: <https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/What-is-MVCC-How-does-Multiversion-Concurrencty-Control-work>.
40. *SQLite* [online]. 2023. [cit. 2023-01-15]. Dostupné z: <https://www.sqlite.org/index.html>.
41. *Getting and Installing BDB SQL* [online]. 2024. [cit. 2024-04-08]. Dostupné z: https://docs.oracle.com/cd/E17276_01/html/bdb-sql/buildinstall.html.
42. SHAPIRO, Marc; PREGUIÇA, Nuno; BAQUERO, Carlos; ZAWIRSKI, Marek. *Conflict-free Replicated Data Types* [online]. 2014. [cit. 2022-11-21]. Dostupné z: https://inria.hal.science/hal-00932836/file/CRDTs_SSS-2011.pdf.
43. *Snappy* [online]. 2022. [cit. 2022-11-21]. Dostupné z: <https://www.solvusoft.com/cs/file-extensions/software/google/snappy/>.
44. *Project Voldemort* [online]. 2022. [cit. 2022-11-22]. Dostupné z: <https://www.project-voldemort.com/voldemort/>.
45. *Java Management Extensions* [online]. 2022. [cit. 2022-11-22]. Dostupné z: <https://www.oracle.com/technical-resources/articles/javase/jmx.html>.
46. *InfinityDB Java NoSQL Database: Boiler Bay Software* [online]. 2022. [cit. 2022-11-22]. Dostupné z: <https://boilerbay.com/>.

47. *MongoDB* [online]. 2023. [cit. 2023-01-28]. Dostupné z: <https://www.mongodb.com/>.
48. *Couchbase* [online]. 2023. [cit. 2023-01-28]. Dostupné z: <https://www.couchbase.com/>.
49. *Document-oriented database* [online]. 2019. [cit. 2023-01-28]. Dostupné z: <https://web.archive.org/web/20190813163612/https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models>.
50. *Cassandra* [online]. 2023. [cit. 2023-01-28]. Dostupné z: https://cassandra.apache.org/_/index.html.
51. *Wide Column Stores* [online]. 2023. [cit. 2023-01-28]. Dostupné z: <https://db-engines.com/en/article/Wide+Column+Stores>.
52. *Systém pro podporu rozhodování* [online]. 2023. [cit. 2023-02-11]. Dostupné z: <https://storm.fsv.cvut.cz/data/files/p%C5%99edm%C4%9Bty/RPZ/08-DSS.pdf>.
53. OSE, Omoruyi; OKOKPUJIE, Kennedy; NDUJIUBA, Nsikan Nkordeh Charles; JOHN, Samuel; UZAIRUE, Idiake Stanley. Performance Benchmarking of Key-Value Store NoSQL Databases [online]. 2023 [cit. 2023-02-12]. Dostupné z: https://www.researchgate.net/publication/330653733_Performance_Benchmarking_of_Key-Value_Store_NoSQL_Databases.
54. AHAMED, Athiq. Benchmarking Top NoSQL Databases [online]. 2023 [cit. 2023-02-12]. Dostupné z: https://www.researchgate.net/publication/303485422_Benchmarking_Top_NoSQL_Databases.
55. COOPER, Brian F.; SILBERSTEIN, Adam; TAM, Erwin; RAMAKRISHNAN, Raghu; SEARS, Russell. *Benchmarking Cloud Serving Systems with YCSB* [online]. 2023. [cit. 2023-02-11]. Dostupné z: <https://courses.cs.duke.edu/fall13/cps296.4/838-CloudPapers/ycsb.pdf>.
56. *5/ Yahoo Cloud Serving Benchmark(YCSB): Knobs and Tunes* [online]. 2023. [cit. 2023-02-12]. Dostupné z: <https://www.youtube.com/watch?v=ZJPTgzFXTKo&list=PL9Bv8oH2HsjyOnAOYYLEPUAKq-2HoUWoA&index=5>.
57. *Core Workloads - YCSB* [online]. 2023. [cit. 2023-02-12]. Dostupné z: <https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads>.
58. GIBSON, Garth. *YCSB++: PARALLEL DATA LAB, CMU* [online]. 2023. [cit. 2023-02-24]. Dostupné z: <https://www.pdl.cmu.edu/ycsb++/>.
59. *Online transaction processing (OLTP)* [online]. 2023. [cit. 2023-03-19]. Dostupné z: <https://www.oracle.com/cz/database/what-is-oltp/>.
60. *IoT definition in detail* [online]. 2023. [cit. 2023-03-19]. Dostupné z: <https://www.sap.com/insights/what-is-iot-internet-of-things.html>.
61. *TPC-C* [online]. 2023. [cit. 2023-03-19]. Dostupné z: <https://www.tpc.org/tpcc/>.

62. *TPC-C Top Performance Results* [online]. 2023. [cit. 2023-03-19]. Dostupné z: https://www.tpc.org/tpcc/results/tpcc_perf_results5.asp?resultttype=all.
63. *Core YCSB properties* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://github.com/brianfrankcooper/YCSB/wiki/Core-Properties>.
64. *Get Docker* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://docs.docker.com/get-docker/>.
65. *Docker Desktop* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://www.docker.com/products/docker-desktop/>.
66. *Docker Docs - CLI* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://docs.docker.com/reference/cli/docker/>.
67. *What's the Difference Between Docker Images and Containers?* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://aws.amazon.com/compare/the-difference-between-docker-images-and-containers/>.
68. *Docker Hub* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://hub.docker.com/>.
69. *docker image pull* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://docs.docker.com/reference/cli/docker/image/pull/>.
70. *docker container run* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://docs.docker.com/reference/cli/docker/container/run/>.
71. *Download the latest release of YCSB* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://github.com/brianfrankcooper/YCSB/releases/download/0.17.0/ycsb-0.17.0.tar.gz>.
72. *Apache Maven Project* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://maven.apache.org/download.cgi>.
73. *Oracle - Java Downloads* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://www.oracle.com/java/technologies/downloads/>.
74. *Windows Environment Variables* [online]. 2024. [cit. 2024-04-26]. Dostupné z: <https://ss64.com/nt/syntax-variables.html>.
75. *Co je PowerShell?* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://learn.microsoft.com/cs-cz/powershell/scripting/overview?view=powershell-7.4>.
76. *com.yahoo.ycsb.BasicDB* [online]. 2024. [cit. 2024-04-25]. Dostupné z: <https://github.com/TSDBBench/YCSB-TS/blob/master/core/src/main/java/com/yahoo/ycsb/BasicDB.java>.