Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Mav

Visoko skalabilna NewSQL relacijska podatkovna baza CockroachDB

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Matjaž Kukar

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Besedilo teme diplomskega dela študent prepiše iz študijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stavkih bo opisal, kaj pričakuje od kandidatovega diplomskega dela. Kaj so cilji, kakšne metode uporabiti, morda bo zapisal tudi ključno literaturo.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ACID	atomicity, consistency, isolation, durability	atomarnost, konsistentnost, izolacija, trajnost so transakcijske lastnosti katere zagotavljajo relacijske podatkovne baze
CSV	comma-seperated value	standardni format podatkov ločen z vejico
DBaaS	database as a service	podatkovna baza, ki je na voljo v oblaku in s katero v večini upravlja ponudnik oblačne storitve
HTAP	hybrid transaction/analytical processing	oznaka sistema, ki omogoča tako transakcijsko kakor tudi analitično obdelovanje
IMDB	in-memory database	podatkovna baza, ki shranjuje podatke v glavni pomnilnik
JDBC	Java Database Connectivity	javanski vmesnik za povezavanje s podatkovnimi bazami
JSON	Javascript object notation	standardna format za prenos podatkov v spletu
KV	key-value	ključ-vrednost

MVCC	multiversion concurrency control	mehanizem za nadzor sočasnosti, kateri hrani več verzij podatkov, omogoča izolacijo in konsistenco podatkov med transakcijami
NewSQL	new structured query language	nov strukturiran poizvedovalni jezik
NoSQL	not only structured query language	poizvedovalni jezik za delo z nerelacijskimi podatki v nekaterih primerih tudi relacijskimi
NTP	network time protocol	meržni protokol za sinhronizacijo ure med računalniškimi sistemi
OLAP	online analytical processing	oznaka sistema, ki omogoča analitično obdelovanje podatkov
OLTP	online transaction processing	oznaka sistema, ki omogoča obdelavo transakcij
ORM	object-relational mapper	programski vmesnik za pretvorbo ralacijskih podatkov v objekte in obratno
SQL	structured query language	strukturiran povpraševalni jezik za delo s relacijskimi podatkovnimi bazami
TPC	transaction processing performance council	organizacija, ki se ukvarja z primerjalno analizo podatkovnih baz v industriji
WAL	write-ahead logging	mehanizem zabeleži najprej vse spremembe v dnevnik šele na to pa jih izvede, mehanizem zagotavlja atomarnost in trajnost podatkov
XML	extensible markup language	format za izmenjavo strukturiranih podatkov v spletu

YCSB	Yahoo! Cloud Serving Benchmarking	enostavno orodje za izvedbo primerjalne zmogljivostne analize med različnimi podatkovnimi bazami
2PL	two-phase locking	mehanizem za nadzor sočasnosti, kateri preko zaklepanja zagotavlja serializabilnost transakcij

Povzetek

Naslov: Visoko skalabilna NewSQL relacijska podatkovna baza CockroachDB

Avtor: Matjaž Mav

Diplomsko delo obravnava NewSQL podatkovno bazo CockroachDB. Cilj diplomskega dela je opisati osnovne koncepte uporabljene v NewSQL podatkovnih bazah in izvesti primerjalno analizo zmogljivosti med novo podatkovno bazo CockraochDB ter že dobro uveljavljeno podatkovno bazo PostgreSQL. NewSQL podatkovne baze so prilagojene za porazdeljena okolja in združujejo lastnosti SQL in NoSQL podatkovnih baz. Uporabljajo standardni SQL poizvedovalni jezik za interakcijo s podatkovno bazo. Preko ACID transakcij zagotavljajo visoko konsistenco podatkov. Omogočajo enostavno horizontalno skaliranje, replikacijo, visoko razpoložljivost in avtomatsko obnovo ob izpadu. Rezultati enostavnih poizvedb so pokazali, da podatkovna baza CockroachDB v primerjavi z podatkovno bazo PostgreSQL na treh vozliščih dosega 5 krat manjšo prepustnos in 3 krat večjo latenco. Poleg tega pa ima trenutno podatkovna baza CockroachDB zelo slabo podporo za stične operacije.

Ključne besede: podatkovne baze, skaliranje, SQL, NewSQL, CockroachDB, PostgreSQL, Citus, YCSB.

Abstract

Title: Higly scalable NewSQL relational database CockroachDB

Author: Matjaž Mav

The thesis deals with NewSQL database called CockroachDB. The aim of the thesis is to describe key concepts used in NewSQL databases and then evaluate and compare performance between the new database CockroachDB and well-established PostgreSQL database. NewSQL databases are build for distributed environments and join properties from both SQL and NoSQL databases. NewSQL databases use standard SQL query language for interaction with a database. They use ACID transactions that guarantee high data consistency. They enable easier horizontal scaling, replication, high availability and automatic failover. The results of simple queries showed that CockroachDB in average achieves 3 times lower throughput and 5 times higher latency compared to PostgreSQL. Furthermore, CockroachDB provides only basic support for join operations.

Keywords: databases, scalability, SQL, NewSQL, PostgreSQL, Citus, YCSB.

Poglavje 1

Uvod

Trendi kažejo, da se vse več računalniške infrastrukture premika v oblak, s tem pa se prilagajajo in razvijajo tudi nove tehnologije, ki so temu bolj primerne. To se odraža tudi pri podatkovnih bazah. Iz starih monolitnih relacijskih podatkovnih baz, katere je še zlasti težko vzdrževati v porazdeljenih okoljih, so se razvile nerelacijske podatkovne baze in kasneje nove relacijske oziroma NewSQL podatkovne baze. NewSQL relacijske podatkovne baze so tako prilagojene za oblak. Zagotavljajo visoko konsistenco in razpoložljivost, poleg tega pa nudijo tudi boljši izkoristek računalniških virov.

Podatkovna baza CockroachDB [14] je ena izmed NewSQL podatkovnih baz. Z raziskovalnega področja je zanimiva, ker je v celoti odprtokodna, idejno pa izhaja iz Googlove podatkovne baze Spanner [8], je zelo enostavna za uporabo in je skoraj v celoti avtomatizirana. Na prvi pogled vsebuje vse, kar bi pričakovali od transakcijsko usmerjene (OLTP) podatkovne baze.

Diplomsko delo je razdeljeno na tri večja poglavja. V prvem poglavju bomo predstavil razlog za nastanek NewSQL podatkovnih baz, opisali njihove lastnosti, razdelitev in najpogosteje uporabljene mehanizme ter pristope.

V drugem poglavju bomo ~~opisal~~ opisali podatkovno bazo CockroachDB. ~~Opisal~~ Predstavili bomo njeno zgodovino in idejno osnovo. ~~Poglobil~~ Poglobili se bomo v arhitekturo podatkovne baze CockroachDB in ~~predstavil~~ predstavili nekaj ključnih mehanizmov kateri omogočajo standardni SQL ~~vmesnik~~jezik,

transakcije, konsistenco, replikacijo, visoko razpoložljivost in enostavnost z upravljanjem. Kasneje bomo predstavili še ostale lastnosti kot so SQL vmesnik, poslovna licenca ter podprta orodja, gonilniki in ORM-ji.

V tretjem poglavju bomo opisali celoten postopek izvedbe primerjalne analize zmogljivosti med podatkovno bazo CockroachDB in Citus ~~. Opisal~~ [6] . Opisali bomo hipoteze, testno infrastrukturo, izbiro orodij za izvedbo zmogljivostnih analiz, podatkovno bazo Citus in razlog za kaj smo CockroachDB primerjali ravno z njo, pripravo podatkov, izvedbo analize. Na koncu bomo predstavili rezultate jih komentirali.

Sledijo sklepne ugotovitve, kjer bomo povzel diplomsko delo in izpostavil nekatere zanimive ugotovitve. Našteli bomo nekaj produktov kjer je ta baza že v uporabi. Za zaključek bomo predlagali še nekaj odprtih vprašanj in ideje za nadaljnje raziskovanje.

Poglavje 2

NewSQL

NewSQL je oznaka za sodobne relacijske podatkovne baze, ki združujejo lastnosti tako relacijskih SQL, kakor tudi nerelacijskih NoSQL podatkovnih baze. Pojem NewSQL prvič omeni in definira Matthew Aslett aprila 2011 [45]. NewSQL podatkovne baze z relacijskih SQL podatkovnih baz prevzamejo standardni SQL vmesniki in ACID (angl. atomicity, consistency, isolation and durability) transakcijske lastnosti [44]. Kratica ACID stoji za:

- (A) atomarnost - transakcija se izvede v celoti ali pa se ne izvede ~~,~~
- (C) ~~konsistenco~~ konsistenca - transakcija z enega konsistentnega stanja preide v drugo konsistentno stanje ~~,~~
- (I) ~~izolacijo~~ izolacija - sočasne transakcije so med seboj izolirane ~~,~~spremembe nepotrjenih transakcij niso vidne navzven ~~in~~
- (D) trajnost - potrjene transakcije ostanejo potrjene tudi v primeru sistemskih napak ~~.~~

Z nerelacijskih NoSQL podatkovnih baz pa prilagojenost na porazdeljena okolja, enostavnost horizontalnega skaliranja in replikacije. NewSQL podatkovne baze nudijo primerljivo zmogljivost z NoSQL podatkovnimi bazami

[44]. Uporabljajo neblokirajoče mehanizme za nadzor sočasnosti [42]. NewSQL podatkovne baze so predvsem primerne za aplikacije tipa OLTP (angl. online transaction processing) z naslednjimi lastnostmi [45]:

- Izvajajo veliko število kratkotrajnih bralno-pisalnih transakcij.
- Večina transakcij preko indeksnih poizvedb uporabi le majhen del podatkov.
- Poizvedbe se ponavljajo, spreminjajo pa se samo njihovi vhodni parametri.

Lastnosti NewSQL podatkovnih baz so [36]:

- Uporabljajo standardni SQL vmenik kot primarni aplikacijski vmesnik za interakcijo.
- Podpirajo ACID transakcijske lastnosti.
- Uporabljajo neblokirajoče mehanizme za nadzor sočasnosti. Tako pisalni zahtevki ne povzročajo konfliktov z bralnimi zahtevki [42]. To smo podrobneje opisali v poglavju 2.2.
- Implementirajo ~~distribuirano~~ porazdeljeno ne deljeno (angl. shared-nothing) arhitekturo. Ta arhitektura zagotavlja horizontalno skalabilnost, podatkovna baza pa lahko teče na velikem številu vozlišč brez trpljenja ozkih grl [42].
- Omogočajo dosti boljšo zmogljivost v primerjavi z tradicionalnimi relacijskimi podatkovnimi bazami. Nekateri pravijo, da naj bi bile NewSQL podatkovne baze celo 50 krat bolj zmogljive [36].

2.1 Kategorizacija NewSQL arhitektur

Kategorizacija NewSQL podatkovnih baz deli implementacije proizvajalcev glede na njihove pristope kateri so uporabljeni, da sistem ohrani SQL vme-

snik, zagotavlja skalabilnost ter zmogljivost tradicionalnih relacijskih podatkovnih baz [42]. NewSQL podatkovne baze se v grobem delijo na štiri kategorije [40, 45]:

1. **Nove arhitekture:**
   V to kategorijo spadajo arhitekture, katere so optimizirane za več-vozliščna porazdeljena okolja. Te arhitekture so za nas najbolj zanimive in so implementirane od začetka. Omogočajo atomično sočasnost med več-vozliščnimi sistemi, odpornost na napake preko replikacije, nadzor pretoka ter porazdeljeno procesiranje poizvedb. ~~V to kategorijo spada tudi podatkovna baza CockroachDB .~~ Slabost teh arhitektur pa je predvsem slaba kompatibilnost z obstoječimi orodji. Primeri teh arhitektur so CockroachDB [14], NuoDB [43], VoltDB [30], MemSQL [39], ... Podatkovno bazo CockroachDB smo bolj podrobno opisali v poglavju 3.

2. **Storitve v oblaku:**
   V to kategorijo spadajo podatkovne baze z novo arhitekturo, ki so na voljo kot produkt oziroma storitev v oblaku (DBaaS). Za upravljanje je v celoti odgovoren ponudnik oblačne storitve. ~~V to kategorijo na primer sodi Amazon Aurora.~~ Slabost te arhitekture je, da nas tipično veže na enega od ponudnikov oblačnih storitev, poleg tega pa nimamo ~~popolenga~~ popolnega nadzora nad delovanjem podatkovne baze. V to kategorijo na primer sodi Amazon Aurora [1], Google Spanner[8], ClearDB [7], ...

3. **Transparentno drobljenje:**
   To so komponente, ki usmerjajo poizvedbe, koordinirajo transakcije, upravljajo s podatki, particijami ter replikacijo. Prednost teh komponent je, da največkrat ni potrebno prilagajati aplikacije novi arhitekturi, saj še vedno deluje kot ena logična podatkovna baza. Slabost teh arhitektur pa se odraža predvsem pri neenakomerno porazdeljenih podatkih ~~. Primer takega transparentnega vmesnika je MariaDB~~

MaxScale . in slabi izkoriščenosti računalniških resursov. Primeri teh arhitektur so MariaDB MaxScale [37], ScaleArc [18], AgilData Scalable Cluster [52], ...

4. **Novi shranjevalni pogoni in razširitve:**
   V to kategorijo spadajo novi shranjevalni pogoni in razširitve, kateri poizkušajo rešiti probleme skaliranja tradicionalnih relacijskih podatkovnih baz. [36] Primer rešitev, ki sodijo v to kategorijo so na primer MySQL NDB cluster, Citus razširitev za PostgreSQL podatkovno bazo ter Hekaton OLTP motor za SQL Server. Podatkovno bazo Citus smo bolj podrobno opisali v poglavju 4.3. Glede na [45] te rešitve ne sodijo v svet NewSQL podatkovnih baz, ampak le med razširitve repacijskih relacijskih SQL podatkovnih baz. Slabost teh v primerjavi z novimi arhitekturami je predvsem slabša izraba računalniških resursov. Poleg tega pa so arhitekture veliko manjn manj elastične in bollj bolj funkionalno omejene. Primeri rešitev, ki sodijo v to kategorijo so na primer MySQL NDB cluster [41], Citus [6] razširitev za PostgreSQL podatkovno bazo, Hekaton [58] OLTP pogon za SQL Server, ... Podatkovno bazo Citus smo bolj podrobno opisali v poglavju 4.3.

2.2 Nadzor sočasnosti

Nadzor sočasnosti je en izmen najpomembnejših mehanizmov v podatkovnih bazah. Omogoča, da do podatkov dostopa več niti hkrati, med tem pa ohranja atomičnost ter garantira izolacijo [45].

Večina NewSQL podatkovnih baz implementira varianto mehanizma z urejenimi časovnimi oznakami (angl. timestamp ordering) oziroma TO. Najbolj priljubljen je decentraliziran več-verzijski mehanizme za nadzor sočasnosti (angl. multi-version concurrency control) v nadaljevanju kar MVCC [45]. Glavna prednost MVCC mehanizma je, da omogoča sočasnost, saj bralni zahtevki nikoli ne blokirajo pisalnih in pisalni nikoli ne blokirajo bralnih. Poleg prednosti ima ta mehanizem tudi nekaj slabosti, za vsako posodobi-

tev mora shraniti novo verzijo podatka ter poskrbeti, da so zastareli podatki odstranjeni iz pomnilnika. Tako MVCC mehanizmi omogočajo večjo zmogljivost saj so bolj prilagojeni na sočasnost, kar pa omogoča večjo izkoriščenost procesorskih virov [24]. Ključna komponenta, da MVCC mehanizem deluje v porazdeljenih okoljih, je čimbolj točna sinhronizacija ure med vozlišči. Na primer Google Cloud Spanner uporablja posebno infrastrukturo katera zagotavlja zelo točno sinhronizacijo ure z uporabo atomskih ur. Podatkovne baze, ki pa niso vezane na infrastrukturo, kot na primer CockroachDB, pa uporabljajo hibridne protokole za sinhronizacijo ure [45].

Poleg MVCC mehanizma nekatere implementacije podatkovnih baz uporabljajo kombinacijo MVCC ter dvo-fazno zaklepanje (angl. two-phase locking) oziroma 2PL [45, 24].

T_1@1	**T_2@2**	**komentar**
BEGIN		T_1: začne transakcijo
R(A)		T_1: prebere vrednost A@0
W(A)	BEGIN	T_1: zapiše vrednost A@1; T_2: začne transakcijo
	R(A)	T_2: prebere vrednost A@0
	W(A)	T_2: konflikt z T_1 (čaka, da T_1 konča transakcijo)
R(A)	...	T_1: prebere vrednost A@1; T_2: čaka
COMMIT	...	T_1: potrdi transakcijo; T_2: čaka
	COMMIT	T_2: zapiše vrednost A@2 in potrdi transakcijo

Tabela 2.1: Primer [46] konflikta, ki se zgodi ob sočasnem pisanju (W(A)) dveh transakcij T_1@1 in T_2@2 pri uporabi MVCC mehanizma. V tem primeru MVCC uporablja inkrementalne oznake predstavljene z @0 @1 @2.

T_1	**T_2**	**komentar**
BEGIN		T_1: začne transakcijo
XL(A)	BEGIN	T_1: zahteva in pridobi ekskluzivno ključavnico nad A
R(A)	SL(A)	T_1: prebere vrednost A; T_2 zahteva deljeno ključavnico nad A (čaka, da jo T_1 sprosti)
W(A)	..~~V nekaterih primerih pa uporabljajo mehanizem razbitega sekvenčnega izvajanja (angl.partitioned serial execution)~~ . ~~[45, 24]~~ ~~.~~	T_1: zapiše novo vrednost A; T_2: čaka
U(A)	...	T_1: sporsti ključavnico nad A; T_2: čaka
COMMIT	R(A)	T_1: potrdi transakcijo; T_2: pridobi deljeno ključavnico in prebere novo vrednost A
	U(A)	T_2: sprosti ključavnico nad A
	COMIT	T_2: potrdi transakcijo

Tabela 2.2: Primer [47] blokiranja, ki se zgodi ob sočasnem pisanju (W(A)) in branju (R(A)) dveh transakcij T_1 in T_2 pri uporabi 2PL protokola. Primer predpostavlja, da gre za stopnjo izolacije READ UNCOMMITED.

2.3 Glavni pomnilnik

Večina tradicionalnih relacijskih podatkovnih baz uporablja diskovno usmerjeno arhitekturo za shranjevanje podatkov. V teh sistemih je primarna lokacija za shranjevanje največkrat kar blokovno naslovljiv disk kot na primer HDD oziroma SSD. Ker so bralne in pisalne operacije v te pomnilne enote relativno počasne, se v ta namen uporablja glavni pomnilnik kot predpomnilnik [45].

Zgodovinsko je bil glavni pomnilnik predrag, danes pa so cene pomnilnikov nižje, tako, da v nekaterih primerih lahko celotno podatkovno bazo shranimo v glavni pomnilnik. Posledničnо so tudi nekatere nove arhitekture NewSQL podatkovnih baz posvojile glavni pomnilnik kot primarno lokacijo za shranjevanje podatkov, te podatkovne baze označimo z IMDB (angl. in-memory database) [23]. Glavna ideja pri tem principu je, da vse podatke hranimo v glavnem pomnilniku. Poleg tega pa uporabljamo blokovno naslovljiv disk za varnostno kopiranje glavnega pomnilnika. Ta ideja izvira že iz leta 1980. NewSQL podatkovne baze pa dodajo mehanizem, da lahko, v primeru prevelike količine podatkov, del podatkov shranijo na sekundarni pomnilnik [45]. Tako se v glavnem pomnilniku nahajajo vroči podatki, to so podatki katerih se poizvedbe velikokrat dotaknejo, v sekundarnem pomnilniku pa se nahajajo mrzli podatki [23].

Tukaj se pojavi vprašanje kateri pristop je boljše izbrati, (1) primarno shranjevanje v glavni pomnilnik ali (2) priamrno shranjevanje v blokovno naslovljivi disk z velikim glavnim pomnilnikom, kjer podatkovna baza lahko predpomni vse, oziroma skoraj vse podatke. Res je, da bo podatkovna baza pri obeh pristopih delovala bistveno hitreje. Vendar pa bo drugi pristop počasnejši od prvega, saj podatkovna baza še vedno predvideva, da se podatki nahajajo na blokovno naslovljivem disku in so nekatere operacije ne optimizirane za tako delovanje [26].

Ker glavni pomnilnik ni odporen na napake ter izpade energije, IMDB podatkovne baze uporabljajo posebne postopke, da zagotovijo obstojnost podatkov ter odpornost celotnega sistema. To dosežejo z beleženjem do-

godkov v dnevnik, ki se nahaja na obstojnem pomnilniku, največkrat kar SSD. Beleženje dogodkov v dnevnik največkrat povzroči ozko grlo, zato ~~tei~~ ti sistemi poizkušajo minimizirati število teh operacij [24]. Obstaja nekaj pristopov kako IMDB podatkovne baze beležijo dogodke v dnevnik:

- Ključavnica nad dnevnikom se odklene čim prej. S tem se zmanjša čas blokiranja ter omogoči, da se ~~ostale sočasne transakcije~~ ostali dogodki hitreje zapišejo v dnevnik ~~. [23]~~[23].
- Beleženje samo dogodkov, kateri spreminjajo stanje podatkov, saj so samo ti dogodki potrebni za obnovo podatkov ~~. [24]~~[24].
- ~~Grupiranje dogodkov ter periodično~~ Periodično beleženje dogodkov v dnevnik. Pri tem pristopu lahko ob morebitni napaki pride do izgube transakcij ~~. [23]~~[23].
- Asinhrono potrjevanje ne čaka potrditve transakcije. Pri tem pristopu lahko ob morebitni napaki pride do izgube transakcij ~~. [23]~~[23].

Ker so bralne in pisalne operacije pri IMDB podatkovnih bazah, v primerjavi z tradicionalnimi relacijskimi podatkovnimi bazami, relativno veliko hitrejše, lahko te bolje izkoristijo procesorsko moč. Posledično se lahko uporabijo za poslovno analitiko na živih podatkih ter hkrati omogočajo zelo visoko pisalno prepustnost. Te procese označimo kot hibridno transakcijsko analitične procese oziroma HTAP (angl. hybrid transaction/analytical processing) [23].

2.4 Drobljenje

Drobljenje (angl. sharding) je pristop s katerim večina NewSQL podatkovnih baz dosega horizontalno skalabilnost [45]. Drobljenje je vrsta horizontalnega particioniranja (angl. horizontal partitioning) podatkov, pri katerem se vsaka particija hrani na ločenem strežniku (logičnem ali fizičnem) [54]. Podatki so razdeljeni horizontalno (po vrsticah) na več delov glede na vrednosti

v izbranih stolpcev v tabeli, te stolpce imenujemo delitveni atributi (angl. partitioning attributes). Najbolj poznana sta dva tipa delitve:

1. **Delitev glede na interval:**
   Glede na postavljene povezane ne prekrivajoče intervale se zapisi delijo na particije. Ta pristop je uporaben predvsem kjer so podatki logično strukturirani v intervale, na primer po letih, kvartalih, mesecih...

2. **Razpršena delitev:**
   Ta delitev za delovanje uporablja razpršitveno funkcijo (angl. hash function), katera podatke deli enakomerno preko vseh particij.

Idealno podatkovna baza iz poizvedbe prepozna katerih particij se poizvedba ~~dotika~~ tiče ter nato izvede poizvedbo porazdeljeno preko teh particij. Dobljene rezultate združi in vrne en rezultat [45].

Nekatere izmed NewSQL podatkovnih baz omogočajo migracijo podatkov med particijami med izvajanjem podatkovne baze. Ta mehanizem je potreben, ker nekatere particije rastejo hitreje kot druge. Če želimo, da podatkovna baza deluje optimalno, potrebujemo podatke ponovno razdeliti enakomerno. Ta pristop ni nov, uporablja ga večina NoSQL podatkovnih baz, je pa bolj kompleksen, saj NewSQL podatkovne baze potrebujejo zagotavljati ACID lastnosti [45].

2.5 Replikacija

Najboljši način za doseganje visoke razpoložljivosti je uporaba replikacije. Večina sodobnih podatkovnih baz podpira nek pristop replikacije [45]. V osnovi poznamo dva pristopa repliciranja:

1. **Sinhrona replikacija**:
   Imenujemo jo tudi aktivno-aktivna (angl. active-active) replikacija. Pri tem pristopu vse replike sočasno obdelajo zahtevek [45].

2. **Asinhrona replikacija**:
   Imenujemo jo tudi aktivno-pasivna (angl. active-passive) replikacija. Pri tem pristopu se zahtevek najprej obdela na eni repliki, nato pa se sprememba posreduje ostalim replikam. Ta pristop implementira večina NewSQL podatkovnih baz, saj zaradi ne deterministične sočasnosti ni mogoče izvesti zahtevkov v pravilnem vrstnem redu na vseh replikah [45, 29].

V visoko konsistentnih podatkovnih bazah morajo biti replikacija potrjene na vseh replikah, šele takrat se smatra, da je replikacija uspešna [23]. Pri ~~distribuiranih~~ porazdeljenih podatkovnih bazah, katere uporabljajo soglasni algoritem (angl. consensus algorithm), je dovolj, da replikacijo potrdi večina replik.

Ker so NewSQL podatkovne baze prilagojene za oblak in namestitve med katerimi so velike geografske razlike, podpirajo tudi optimizirano replikacijo preko WAN (angl. wide-area network) omrežji.

2.6 Obnova

Pomembna lastnost podatkovne baze, ki zagotavlja toleranco na napake, je tudi obnova podatkovne baze po izpadu. Poleg same obnove sistema, kot ga poznamo pri tradicionalnih podatkovnih bazah, NewSQL podatkovne baze poizkušajo tudi minimizirati sam čas obnove [45]. Pričakuje se, da bo podatkovna baza na voljo skoraj ves čas, saj že kratkotrajni izpadi lahko pomenijo velike finančne izgube. Tukaj ponavadi govorimo o tako imenovanih petih devetkah (angl. five nines) oziroma 99,999% razpoložljivostjo, to označimo z pojmom visoka razpoložljivost.

Tradicionalne podatkovne baze ponavadi tečejo na enem vozlišču in največkrat uporabljajo neko implementacijo algoritma ARIES (angl. Algorithms for Recovery and Isolation Exploiting Semantics) za obnovo ob izpadu. ARIES za svoje delovanje uporablja WAL (angl. Write-Ahead Log), to je dnevnik, ki je shranjen na obstojnem pomnilniku, vsaka sprememba pa se

najprej zapiše v dnevnik in šele nato izvede. Algoritem ARIES je sestavljen iz treh faz, analize (angl. analysis), ponovitve (angl. redo) in razveljavitev (angl. undo). V fazi analize algoritem pripravi podatkovne strukture. V fazi ponovitve ponovi vse spremembe, zabeležene v WAL dnevniku, v tej točki je podatkovna baza točno v takem stanju kakor je bila pred izpadom. V fazi razveljavitve pa razveljavi vse nepotrjene transakcije. Po končanem postopku je podatkovna baza v konsistentnem stanju. [45]

Pri NewSQL podatkovnih bazah ~~,~~ algoritem ARIES, ki ga uporabljajo tradicionalne podatkovne baze ni direktno uporabljen. NewSQL podatkovne baze so ~~distribuirane~~porazdeljene, kar pomeni, ko eno vozlišče odpove, si ostala vozlišča avtomatsko razdelijo obremenitev, sistem kot celota pa deluje naprej. Ko pride vozlišče nazaj, mora obnoviti stanje v času izpada, poleg tega pa se mora sinhronizirati in iz ostalih vozlišč pridobiti vse spremembe, ki so se zgodile v času, ko je bilo vozlišče nedosegljivo [45]. Poleg tega je algoritem ARIES zasnovan za diskovno orientirane podatkovne baze, kar pa ni optimalno za nove pomnilniško usmerjene arhitekture NewSQL podatkovnih baz [66].

Poglavje 3

CockroachDB

~~CockroachDB je distribuirana~~ CockroachDB je porazdeljena SQL podatkovna baza, temelji na transakcijski in visoko konsistenti ključ-vrednost shranjevalnem mehanizmu. Zagotavlja ACID transakcijske lastnosti, kot primarni vmesnik za interakcijo pa uporablja standardni SQL. Je horizontalno skalabilna in je odporna na napake strežnika ali pa celotnega podatkovnega centra. Prilagojena je za izvajanje v oblak in namenjena globalnim oblačnim storitvam. Je transakcijska podatkovna baza in ni primerna za večje analitične obremenitve. Je zelo enostavna za upravljanje ter uporabo. [25]

Leta 2015 so Spencer Kimball, Ben Darnell in Peter Mattis ustanovili podjetje CockroachLabs. Njihov glavni produkt je v celoti odprtokodna podatkovna baza CockroachDB [14]. Vsi trije ustanovitelji so bili predhodno inženirji v podjetju Google. Podatkovna baza Google Spanner pa je bila navdih za začetek podatkovne baze CockroachDB. [51]

Podatkovni bazi CockroachDB in Google Spanner sta si arhitekturno različni. Podatkovna baza Google Spanner deluje le v oblaku na Googlovi infrastrukturi, saj se za svoje delovanje zanaša na TrueTime API. TrueTime API je vmesnik, ki s pomočjo GPS in atomskih ur, skrbi za zelo natančno sinhronizacijo ure na Googlovi infrastrukturi [17]. CockroachDB za svoje delovanje ne potrebuje tako točne sinhronizacije ur, v splošnem je priporočena uporaba Googlove zunanje NTP (angl. Network Time Protocol) storitve [19].

Podatkovna baza CockroachDB je odprto kodna in lahko deluje v oblaku ali pa lokalno na operacijskih sistemih Linux ter OS X.

3.1 Arhitektura

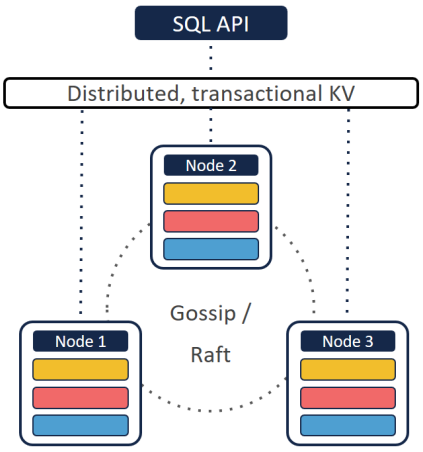
CockroachDB je odprtokodna, avtomatizirana, porazdeljena, skalabilna in konsistentna SQL podatkovna baza. Večino upravljanja je avtomatiziranega, kompleksnost samega sistema pa je prikrita končnemu uporabniku.

CockroachDB v grobem sestavlja pet funkcionalnih plasti. Zaradi lažjega razumevanja večslojne arhitekture bom najprej predstavil najbolj pogoste termine in koncepte uporabljene v podatkovni baze CockroachDB. Kasneje bom povzel samo arhitekturo in delovanje podatkovne, kar je bolj podrobno opisa v uradni dokumentaciji [10] in prvotnih zasnutkih delovanja CockroachDB podatkovne baze [11].

Terminologija in koncepti

- **Gruča (angl. cluster):** Predstavlja en logični sistem.
- **Vozlišče (angl. node):** Posamezni strežnik, ki poganja CockroachDB, več vozlišč se poveže in tvori gručo.
- **Obseg (angl. range)** Nabor urejenih in sosednjih podatkov.
- **Replika (angl. replica):** Kopija obsega, ki se nahaja na vsaj treh vozliščih.
- **Najem obsega (angl. range lease):** Za vsako območje obstaja ena replika (angl. leaseholder) katera ima obseg v najemu. Ta replika sprejme in koordinira vse bralne in pisalne zahtevke za določen obseg.
- **Konsistenca (angl. consistency):** Podatki v podatkovni bazi so vedno v končnem veljavnem stanju in nikoli v vmesnem stanju. CockroachDB ohranja konsistenco preko ACID transakcij ter CAP teorema.

- **Soglasje (angl consensus):** To je postopek, preko katerega ~~distribuirani~~ porazdeljeni sistemi pridejo do soglasja o vrednosti enega podatka. CockroachDB ob pisalnem zahtevku čaka, da večina replik potrditi, da je podatek uspešno zapisan. S tem mehanizmom se izognemo izgubi podatkov ter ohranimo konsistentnost podatkovne baze v primeru, da odpove eno od vozlišč.
- **Replikacija (angl. replication):** Je postopek kopiranja in ~~distribuiranja~~ porazdelitve podatkov med vozlišči, tako da podatki ostanejo v konsistentnem stanju. CockroachDB uporablja sinhrono replikacijo. To pomeni, da morajo vsi pisalni zahtevki najprej dobiti soglasje kovruma, preden se sprememba smatra za potrjeno.
- **Transakcija (angl. transaction):** Množica operacij izvršenih na podatkovni bazi, katere ohranjajo ACID lastnosti.
- **Visoka razpoložljivost (angl. high availability):** CockroachDB zagotavlja visoko razpoložljivost, ki ji pravimo pet devetk (angl. five nines), kar pomeni da je sistem dosegljiv vsaj 99,999% časa. CockroachDB ne zagotavlja razpoložljivosti preko CAP teorema [25].



Slika 3.1: Arhitekturni pregled [51]

Plasti

Podatkovna baza CockroachDB je sestavljena iz petih plasti. Plasti med seboj delujejo kot črne škatle. Vsaka plast pa sodeluje le z plastjo direktno nad in pod seboj.

1. **SQL plast:** Prevede SQL poizvedbe v ~~KV operacije~~ operacije tipa ključ-vrednost (KV).
2. **Transakcijska plast:** Omogoča atomične spremembe nad večjim številom KV operacij.
3. **Porazdelitvena plast:** Predstavi replicirana KV območja kot eno entiteto.
4. **Replikacijska plast:** Konsistentno in sinhrono replicira KV obsege v gruči.
5. **Shranjevalna plast:** Izvaja bralne in pisalne operacija na disku.

3.1.1 SQL plast

SQL plast predstavlja vmesnik med podatkovno bazo ter ostalimi aplikacijami. Podatkovna baza CockroachDB implementira velik del SQL standarda [20]. Zunanje aplikacije komunicirajo preko PostgreSQL žičnega protokola. To omogoča enostavno povezavo med zunanjimi aplikacijami in kompatibilnost z obstoječimi gonilniki, orodji ter ORM-ji.

Poleg tega so vsa vozlišča v CockroachDB gruči simetrična, kar pomeni, da se lahko aplikacija poveže na katero koli vozlišče. To vozlišče obdela zahtevek, oziroma ga preusmeri na vozlišče katero zna obdelati zahtevek. To omogoči enostavno porazdelitev bremena.

Ko vozlišče prejme SQL zahtevek, ga najprej razčleni v abstraktno sintaktično drevo. Potem CockroachDB začne s pripravo poizvedovalnega plana. V tem koraku CockroachDB preveri tudi sintaktično pravilnost poizvedb ter

nato vse operacije prevedene v KV operacije ter transformira podatke v binarno obliko. S pomočjo poizvedovalnega plana transakcijska plast nato izvede vse operacije.

ključ	**vrednost**
/system/databases/mydb/id	51
/system/tables/customer/id	42
/system/desc/51/42/address	69
/system/desc/51/42/url	66
/51/42/Apple/69	1 Infinite Loop, Cupertino, CA
/51/42/Apple/66	http://apple.com/

Tabela 3.1: Poenostavljen primer preslikave SQL v KV model [11]. V podatkovni bazi `mydb` se nahaja tabela `customer`, katera ima poleg primarnega ključa še dva stolpca `address` in `url`. Zadnja dva zapisa v tabeli predstavljata en vrstico v tabeli `customer` z primarnim ključem `Apple`.

3.1.2 Transakcijska plast

Podatkovna baza CockroachDB je konsistentna, to dosega tako da, transakcijska plast implementira celotno semantiko ACID transakcij. Vsak stavek predstavlja svojo transakcijo, transakcije pa niso omejene samo na določeno tabelo, obseg ali vozlišče in delujejo preko celotne gruče. To dosežejo z dvofaznim potrditvenim postopkom (angl. two-phase commit):

1. **Faza 1:** Vsaka transakcija najprej kreira transakcijski zapis s statusom v teku (angl. pending). To je podatkovna struktura katera nosi transakcijski ključ in status transakcije. Sočasno, se za pisalne operacije kreira pisalni namen (angl. write intent). Pisalni namen je v osnovi MVCC vrednost označena z zastavico `<intent>` in kazalcem na transakcijski zapis. Primer MVCC shrambe je prikazan v tabeli 3.2.

2. **Faza 2:** V kolikor je transakcijski zapis v status prekinjeno (angl.

aborted), se transakcija ponovno izvrši.

Če pa so izpolnjeni vsi pogoji, se transakcija potrdi. Status v transakcijskem zapisu se spremeni v potrjeno (angl. commited). Sočasno se vsem pisalnim namenom povezanim s trenutno transakcijo odstrani zastavico `<intent>`.

3. **Faza 3 (asinhrono):** Ko se transakcija konča, se vsem potrjenim pisalnim namenom odstrani zastavico `<intent>` in kazalec na transakcijski zapis. Nepotrjeni pisalni nameni se samo izbrišejo. To se izvede asinhrono, zato vse operacije, preden kreirajo pisalni namen preverijo obstoječi pisalni namen s transakcijskim zapisom in ga ustrezno upoštevajo.

ključ	**čas**	**vrednost**
A<intent>	500	nepotrjena vrednost
A	400	trenutna vrednost
A	322	stara vrednost
A	50	prvotna vrednost
B	100	vrednost B

Tabela 3.2: Primer MVCC shrambe z pisalnim namenom na ključu A [53].

Podatkovna baza CockroachDB privzeto podpira najvišji standardni ~~ANSI~~ SQL izolacijski nivo, to je *serializable*. Ta nivo ne dopušča nikakršnih anomalij v podatkih, če obstaja možnost anomalije se transakcija ponovno izvede. Podpira pa tudi zastareli nestandardni izolacijski nivo *snapshot*.

3.1.3 Porazdelitvena plast

Vsi podatki v gruči so na voljo preko kateregakoli vozlišča. CockroachDB shranjuje podatke v urejeno vrsto tipa ključ-vrednost. Ta podatkovna struktura je namenjena shranjevanju sistemskih kakor tudi uporabniških podatkov. Z nje CockroachDB razbere kje se nahaja podatek in vrednost samega

podatka. Podatki so razdeljeni na koščke, katere imenujemo obsegi (angl. ranges). Obsegi so urejeni koščki podatkov preko katerih CockroachDB lahko hitro in učinkovito izvede *lookup* in *scan* operacije.

Lokacija vseh obsegov je shranjena v dvo-nivojskem indeksu. Ta indeks na prvem nivoju sestavlja meta obseg (angl. meta range) imenovan `meta1`, kateri kaže na meta obseg na drugem nivoju imenovan `meta2`. Meta obseg `meta2` pa kaže na podatkovne obsege.

Ko vozlišče prejme zahtevek, v meta obsegih poišče vozlišče katero hrani obseg v najemu (angl. range lease) ter preko tega vozlišča izvede zahtevek. Meta obsegi so predpomnjeni, zato se lahko zgodi, da kažejo na napačno vozlišče. V tem primeru se vrednosti meta obsegov posodobijo.

Privzeto je velikost podatkovnega obsega omejena na 64MiB. To omogoča lažji prenos obsega med vozlišči, poleg tega pa je obseg dovolj velik, da lahko hrani nabor urejenih podatkov, kateri so bolj verjetno dostopani skupaj. Če obseg preseže velikost 64MiB, se razdeli v dva 32MiB podatkovna obsega. Ta dvo-nivojska indeksna struktura omogoča, da naslovimo do $2^{(18+18)} = 2^{36}$ obsegov. Vsak obseg pa privzeto naslavlja $2^{26}B = 64MiB$ pomnilniške prostora. Teoretično podatkovna baza CockroachDB s privzetimi nastavitvami lahko naslovi do ~~$2^{(36+26)}B = 4EB$~~ $2^{(36+26)}B = 4EiB$ podatkov.

3.1.4 Replikacijska plast

Replikacijska plast skrbi za kopiranje podatkov med vozlišči in jih ohranja v konsistentnem stanju. To doseže preko soglasnega algoritma (angl. consensus algorithm) Raft. Ta algoritem zagotovi, da se večina vozlišč v gruči strinja z vsako spremembo v podatkovni bazi. S tem, kljub odpovedi posameznega vozlišča, ohrani podatke v konsistentnem stanju poleg tega pa zagotovi nemoteno delovanje podatkovne baze (visoko razpoložljivost).

Število vozlišč, ki lahko odpove brez, da bi s tem vplivalo na delovanje podatkovne baze je enako:

$$(r-1)/2 = f, \text{ če } r = 3, 5, 7, ... \text{ in } r <= N$$

Kjer je N število vseh vozlišč v gruči, r faktor replikacije liho število večje ali enako tri in manjše ali enako številu vseh vozlišč v gruči. Ter f maksimalno število vozlišč, ki še lahko odpove brez, da bi vplivalo na delovanje podatkovne baze. Na primer če je replikacijski faktor $r = 3$, gruča lahko tolerira odpoved enega vozlišča $f = 1$.

Za vsak obseg obstaja Raft skupina, kjer je eno vozlišče, ki vsebuje repliko, označena kot "vodja". Vodja je izvoljen in koordinira vse pisalne zahtevke za določen obseg. V idealnih pogojih je vodja Raft skupine tudi najemnik obsega (angl. leaseholder).

3.1.5 Shranjevalna plast

CockroachDB za shranjevanje uporablja KV shrambo RocksDB. RocksDB zagotavlja atomične skupine pisalnih zahtevkov, kar CockroachDB potrebuje za zagotavljanje transakcij. RocksDB preko kompresije ključev zagotavlja učinkovito shrambo podatkov.

CockroachDB uporablja MVCC pristop in hrani več verzij vsakega podatka. Podatkovna baza CockroachDB preko MVCC pristopa omogoča poizvedbe v zgodovino `SELECT...AS OF SYSTEM TIME`. Privzeto stare verzije podatka pretečejo po 24 urah in so počiščene s shrambe. Primer poizvedbe po zgodovinskih podatkih:

```
\DIFadd{SELECT name, balance
FROM accounts
    AS OF SYSTEM TIME '2016-10-03 12:45:00'
WHERE name = 'Edna Barath';
```

3.2 Lastnosti

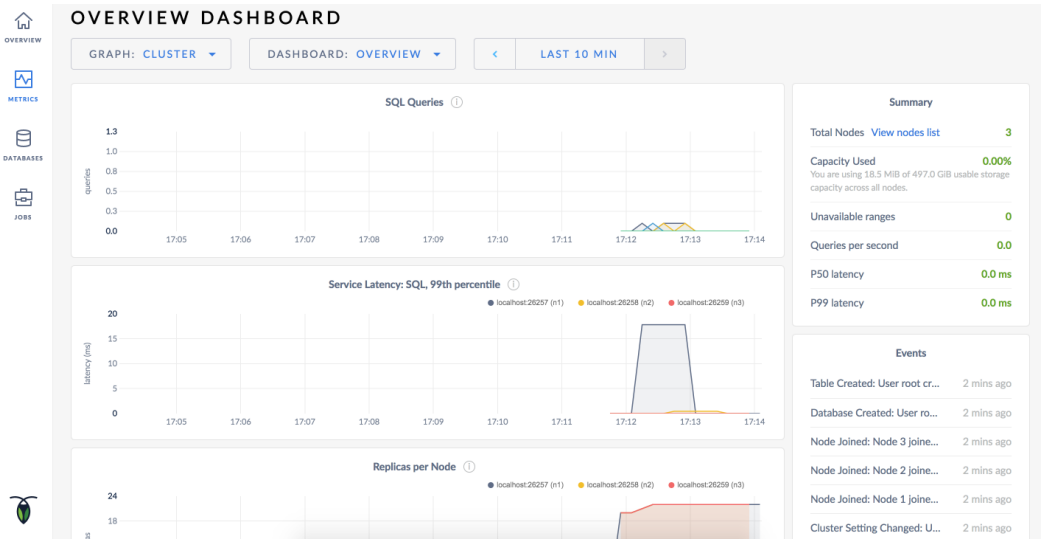
Razvoj podatkovne baze CockroachDB je bil usmerjen prvotno v funkcionalnosti in še le kasneje v optimizacije. Tako verzija 1.0.0 omogoča razvijalcem večina potrebnih funkcionalnosti, verzija 2.0.0 pa je poleg manjših funkcionalnosti prinesla predvsem optimizacije.

3.2.1 Enostavnost

Podatkovna baza CockroachDB strmi k enostavnosti upravljanja in vzdrževanja. Večina kompleksnosti je skrite končnemu uporabniku. Izogiba se zunanjim odvisnostim in je na voljo kot ena sama binarna datoteka. Minimalno za zagon gruče je potrebno na vsakem vozlišču zagotoviti sinhronizacije ure, priporočena je uporaba zunanje Google NTP storitve ter namestiti CockroachDB binarno datoteko na vsako vozlišče in jo zagnati.

3.2.2 Uporabniški vmesnik in nadzorovanje

Poleg podatkovne baze CockroachDB vsebuje tudi spletni administratorski vmesnik, ki je prikazan na sliki 3.2. Administratorski vmesniki nudi vizualizacije raznih časovnih metrik o delovanju posameznih vozlišč, kakor tudi celotne gruče. Omogoča pregled dnevnikov in pripomore k lažjemu odkrivanju težav v gruči. CockroachDB omogoča tudi izvoz metrik v odprtokodno rešitev Prometheus, katera nam omogoča shrambo, obdelavo, vizualizacije in obveščanje nad časovnimi vrstami.



Slika 3.2: Primer spletnega administratorski vmesnika podatkovne baze CockroachDB.

3.2.3 SQL

CockroachDB, kakor večina relacijskih podatkovnih baz, podpira podmnožico SQL standarda [20]. Poleg tega implementira nekatere najpogostejše razširitve, razširitve specifične za PostgreSQL ter svoje razširitve. V spodnjih podpoglavjih bom v grobem opisal, kaj od jezika SQL ponuja CockroachDB, bolj podroben opis pa se nahaja v uradni dokumentaciji [57].

Podatkovni tipi

CockroachDB podpira podatkovne tipe kot so `ARRAY`, `BOOL`, `BOOL`, `BYTES`, `COLLATE` (podobno kot `STRING` vendar upošteva specifike jezika), `DATE`, `DECIMAL`, `FLOAT`, `INET` (IPv4 in IPv6 naslovi), `INTERVAL` (časovni interval), `JSONB`, `SERIAL` (`INT` z avtomatsko kreirano številko, priporočena uporaba `UUID`), `STRING`, `TIME`, `TIMESTAMP` in `UUID`. CockroachDB ne podpira podatkovnih tipov `XML`, `UNSIGNED INT` ter `SET` in `ENUM`.

Shema

CockroachDB podpira vse standardne ukaze za kreiranje, spreminjanje in odstranjevanje tabel, stolpcev, omejitev in indeksov. Od podatkovnih omejitev podpira vse standardne, to so: `NOT NULL`, `UNIQUE`, `CHECK`, `FOREIGN KEY` in `DEFAULT`. Izjema je `PRIMARY KEY`, to omejitev lahko nastavimo samo ob kreiranju tabele in je kasneje ne moremo spreminjati. Če omejitve `PRIMARY KEY` ne nastavimo v času kreiranja tabele, CockroachDB v ozadju sam kreira skriti stolpec z tipom `UUID`, saj je ta pomemben za samo delovanje podatkovne baze.

Poizvedovanje sheme je na voljo na standardni način, preko virtualne sheme `informationq_schema`.

Prožilci še niso podprti in tudi ne načrtovani za naslednje verzije.

Transakcije

CockroachDB podpira ACID transakcije, katere so lahko ~~distribuirane~~ porazdeljene preko celotne gruče. Privzeto transakcije uporabljajo najvišji izolacijski nivo `SERIALIZABLE`, omogoča pa še zastareli izolacijski nivo `SNAPSHOT`. Ostali standardni izolacijski nivoji niso podprti.

Tabelarični izrazi

CockroachDB podpira referenciranje tabel, pogledov, poizvedb, tabelaričnih funkcij. Od `JOIN` operacij omogoča `INNER`, `LEFT`, `RIGHT`, `FULL`, `CROSS`, vendar pa te operacije še niso optimizirane. Da bi dosegli optimalno delovanje je potrebno `JOIN` operacije pravilno omejiti.

Operatorji, skalarni izrazi, logični izrazi in funkcije

CockroachDB implementira večji del standardnih operatorjev, skalarnih izrazov, logičnih izrazov ter funkcij. Slabo podporo zagotavlja konsturktu `EXISTS` in skalarnim pod-poizvedbam. Podpira poizvedovanje po tipu JSON, omogoča iskanje z POSIX regularnimi izrazi ter implementira ~~okenske~~ agregacijske

funkcije (funkcije katere se izvedejo nad ~~podmnožico, tipično~~ posamezno skupino ob uporabi `GROUP BY`) in okenske funkcije (`OVER()`).

Uporabniške funkcije niso podprte in tudi ne načrtovane za naslednje verzije. Shranjene procedure so načrtovane za eno od naslednjih verzij.

3.2.4 ~~Poslovna licenca~~Licenca

CockroachDB je odprtokodna in zastonjska podatkovna baza. Večina potrebnih funkcionalnosti omogoča zastonjska različica, poleg tega pa ponuja tudi poslovno licenco. Poslovna licenca omogoča strankam:

- svetovanje,
- tehnično pomoč,
- geografsko ~~distribucijo~~ porazdelitev na nivoju ene vrstice,
- vizualizacijo geografsko ~~distribuirane~~ porazdelitve gruče na zemljevidu,
- nadzor dostopa na nivoju skupin in
- ~~distribuirano~~ porazdeljeno kreiranje in obnova varnostnih kopij.

3.2.5 Podprta orodja, gonilniki in ORM-ji in skupnost

Podatkovna baza CockroachDB implementira standardni SQL [20], za komunikacijo med odjemalcem in strežnikom pa uporablja PostgreSQL žični protokol, kar omogoča dobro kompatibilnost z obstoječimi PostgreSQL komponentami.

Na uradni spletni dokumentaciji je za jezike Go, Java, .NET, C++, NodeJS, PHP, Python, Ruby, Rust, Clojure ter Elixir objavljen seznam podprtih gonilnikov in ORM-jev. Za te CockroachLabs v času pisanja zagotavlja le beta podporo [56].

Od orodij za upravljanje z podatkovno bazo nam CockroachLabs ponuja konzolno aplikacijo `cockroach sql`. Od grafičnih orodij nudijo beta podporo za pgweb, dbglass, Postico, PSequel, TablePlus in Valentina studio [55].

Orodje pgweb ima trenutno najbolj zdravo in aktivno skupnost, je odprtokodno in podpira vse glavne operacijske sisteme (Windows, OSX in Linux). Aplikacijo dostopamo preko spletnega brskalnika in podpira funkcionalnosti kot so:

- pregled podatkovne baze in sheme
- izvedba in analiza SQL poizvedb
- zgodovina poizvedb
- izvoz podatkov v formatu CSV, JSON in XML

Izvirna koda podatkovne baze CockroachDB je javno dostopna preko GitHub repozitorija `cockroachdb/cockraoch` [14]. Projekt ima zdravo skupnost, kateremu trenutno sledi približno 14 tisoč navdušencev. Poleg GitHub repozitorija, kjer se odvija ves razvoj, vodenje nalog in pomoč, CockroachLabs vodi še spletno stran [10] (dokumentacija, form, blog, mediji), Gitter kanal [27] in Docker repozitorij [16].

Poglavje 4

Primerjalna analiza zmogljivosti CockroachDB

Primerjalna analiza zmogljivosti (angl. performance benchmarking) je postopek za primerjavo enega sistema z ostalimi podobnimi sistemi. V mojem primeru bom z orodjem YCSB primerjal podatkovni bazi CockroachDB ter PostgreSQL z nameščeno razširitvijo Citus (v nadaljevanju samo Citus).

Za primerjavo z Citus sem se odločil zaradi lažje izvedbe ter bolj primerljivih rezultatov. Obe podatkovni bazi CockroachDB in Citus sta po nekaterih lastnostih zelo podobni. Obe podatkovni bazi implementirat standardni SQL ter komunicirata preko PostgreSQL žičnega protokola.

Za orodje YCSB sem se odločil, saj podpira vmesnik JDBC, katerega podpirata tudi obe podatkovni bazi poleg tega pa je enostaven za uporabo. Orodje YCSB sem bolj podrobno opisal v poglavju 4.4. Ob iskanju najprimernejšega orodja sem pregledal naslednja orodja:

- **TPC:**
  TPC (angl. transaction processing performance council) [62] je ne profitna organizacija, katera nudi preverljive podatke TPC zmogljivostnih analiz podatkovnih baz. TPC definira več različnih standardnih testov, kateri simulirajo različne realne obremenitve. Te testi so točno opisani in strogo omejeni. Najbolj poznan tip testa za transakcijske

obremenitve je TPC-C, za analitične pa TPC-H.

To so standardizirani testi in bi bili najbolj primerni, vendar pa so zelo kompleksni. Uradnih TPC testov za podatkovno bazo CockroachDB še ne obstaja.

- **pgbench:**
  Je enostavno orodje namenjeno PostgreSQL podatkovni bazi. Simulira ne standardno TPC-C obremenitev.

  Orodje pgbench v času izvajanja testov še ni bilo kompatibilno z podatkovno bazo CockroachDB.

- **cockroachdb/loadgen:**
  Je orodje namenjeno CockroachDB podatkovni bazi. Orodje vsebuje nabor testov kot so TPC-C, TPC-H in YCSB. Te orodja interno uporabljajo za zmogljivostno primerjavo med verzijami CockroachDB podatkovne baze. Kasneje je bil objavljena tudi objava na njihovem blogu, kjer so primerjali rezultate TPC-C obremenitve med podatkovnima bazama CockroachDB in Amazon Aurora.

  Orodje je enostavno, vendar ne omogoča zmogljivostne analize PostgreSQL podatkovne baze.

- **Apache JMeter:**
  Je orodje primarno namenjeno izvajanju obremenitvenih testov. Tega orodje omogoča izvajanje obremenitvenih testov preko JDBC vmesnika.

  Orodje je zelo konfigurabilno vendar pa ne omogoča v naprej definiranih obremenitvenih testov in je težko za uporabo.

- **YCSB:**
  YCSB (angl. Yahoo! Cloud Serving Benchmark) je orodje namenjeno za primerjavo zmogljivostnih metrik med raznimi podatkovnimi bazami.

V naslednjih podpoglavjih bom opisal točen postopek s katerim sem izvedel primerjalno analizo. Opisal bom arhitekturo, pripravo posameznih konfiguracij, pripravo podatkov in samo testiranje. Na koncu bom predstavil rezultate zmogljivostne analize ter ugotovitve.

4.1 Hipoteze

Pred začetkom izvajanja primerjalne analize sem postavil naslednje hipoteze:

1. CockroachDB bo na enem vozlišču nekoliko počasnejši od PostgreSQL podatkovne baze.

2. CockroachDB bo zaradi linearne skalabilnosti na treh vozliščih skoraj tri krat bolj zmogljiv.

4.2 Arhitektura

Testno okolje sestavljajo štiri vozlišča označena z `n0`, `n1`, `n2` in `n3`. Specifikacije strojne opreme so opisane v tabeli 4.1. Vsa vozlišča imajo nameščen Ubuntu 16.04 LTS, Docker 18.03.0-ce ter ntpd, ki je konfiguriran glede na produkcijska priporočila CockroachDB podatkovne baze [19].

Vozlišča so med seboj povezana preko gigabitnega Ethernet omrežja v Docker Swarm [59] gručo. Vozlišče `n0` je v vlogi vodje, vozlišča `n1`, `n2` in `n3` pa so v vlogi delavcev. Za uporabo Docker Swarm tehnologije sem se odločil zaradi enostavnosti postavitve okolja ter lažje ponovljivosti testov.

	procesor	**pomnilnik**	**trdi disk**
n0	Intel Core2 Quad CPU Q9400	4GB	SAMSUNG HD753LJ
n1	Intel Core i5 CPU 650	4GB	WDC WD10EARS-22Y5B1
n2	Intel Core i7-3770	8GB	ST500DM002-1BD142
n3	Intel Core i5-2400	4GB	Hitachi HDS721050CLA662

Tabela 4.1: Specifikacije strojne opreme, katere se razlikujejo med posameznimi vozlišči.

4.2.1 Odjemalec

Vozlišče n0 je v vlogi odjemalca. Na njem teče program, ki zažene podatkovno bazo, obnovi podatke, izvaja teste in beleži rezultate. Podroben opis delovanja odjemalca se nahaja v poglavju 4.5.

4.2.2 Strežnik

V vlogi strežnika so vozlišča `n1`, `n2` in `n3`. Na njih teče ali CockroachDB ali PostgreSQL z nameščeno razširitvijo Citus. V primeru, da gre za konfiguracijo z enim vozliščem, podatkovna baza teče na vozlišču `n2`.

4.3 Citus

TODO

4.4 YCSB

YCSB [5] je razširljiva in odprtokodna rešitev za primerjalno analizo zmogljivosti. Največkrat se uporablja za zmogljivostno analizo različnih NoSQL podatkovnih baz. Podpira veliko število različnih podatkovnih baz kot so Mongo, Couchbase, S3, Redis, itd. Poleg tega pa podpira tudi JDBC

vmesnik, preko katerega sem primerjal CockroachDB ter PostgreSQL podatkovni bazi. YCSB nudi šest enostavnih predefiniranih tipov obremenitev [64] označenih z A, B, C, D, E in F.

4.5 Testiranja YCSB obremenitev

Testiranje je potekalo iz odjemalca, torej vozlišča `n0`. Meritev je bilo veliko, zato sem pripravil program s katerim sem si pomagal. Program je enostaven in ni prenosljiv. Izvorna koda in rezultati meritev so na voljo na GitHub repozitoriju [38]. V tem poglavju bom opisal vse korake kateri so bili potrebni za izvedbo meritev.

4.5.1 Namestitev programske opreme

Na vozlišču `n0` sem namestil naslednjo programsko opremo:

- YCSB (verzija 0.12.0) [5] je orodje namenjeno za izvedbo zmogljivostne analize ter primerjavo med različnimi podatkovnimi bazami. Samo orodje sem bolj natančno opisal v poglavju 4.4.
- Ansible (verzija 2.5.0) [2] je orodje za avtomatizacijo, uporabil sem ga zaradi lažjega konfiguriranja gruče preko SSH povezave.
- Go (verzija 1.10.1) [61] je programski jezik, v katerem je napisan program za avtomatizacijo testiranja. Za programski jezik Go sem se odločil, zaradi njegove enostavnosti.

4.5.2 Priprava podatkov

Program za avtomatizacijo predpostavlja, da imajo vsa vozlišča na točno določeni lokaciji pripravljene podatke za obnovo fizične podatkovne baze. Pred vsakim testom se podatki kopirajo v začasno mapo, nad katero kasneje baza izvaja operacije. Po končanem testu se začasna mapa izbriše.

V spodnjih korakih je opisan postopek, po katerem sem za vsako bazo ter za vsako konfiguracijo (eno vozlišče in tri vozlišča) pripravil podatke. Vse konfiguracijske datoteke, ki so uporabljene v spodnjih primerih so na voljo na GitHub repozitoriju `matjazmav/diploma-ycsb` [38].

Citus

1. Z Docker Swarm konfiguracijsko skripto (`stacks/postgres-n1.yml`) sem pognal Citus podatkovno bazo na vozlišču `n2`.

2. Na vozlišču `n2` sem ročno kreiral shemo katero potrebuje YCSB za svoje delovanje.

```
CREATE DATABASE ycsb;
\c ycsb;
CREATE TABLE usertable (
    YCSB_KEY VARCHAR(255) PRIMARY KEY,
    FIELD0 TEXT, FIELD1 TEXT,
    FIELD2 TEXT, FIELD3 TEXT,
    FIELD4 TEXT, FIELD5 TEXT,
    FIELD6 TEXT, FIELD7 TEXT,
    FIELD8 TEXT, FIELD9 TEXT
);
```

3. Nato sem generiral podatke. V bazo na vozlišču `n2` sem z YCSB orodjem naložil 5 milijonov zapisov, kar na disku zasede približno 6GB prostora.

```
ycsb load jdbc \
    -P workloads/workloada \
    -P configs/postgres-n1.properties \
    -p threadcount=16 \
    -p recordcount=5000000
```

4. Podatkovno bazo sem varno ustavil ter naredil varnostno kopijo fizične podatkovne baze na disku.

5. Nato sem Docker Swarm konfiguracijsko skripto (`stacks/postgres-n3.yml`) pognal Citus podatkovno bazo na treh vozliščih.

6. Na vozliščih `n1` in `n3` sem kreiral shemo definirano v točki 2.

7. Nato sem na vozlišču `n2` povezal še vozlišči `n1` ter `n3` in kreiral ~~distribuirano~~ porazdeljeno tabelo. Podatki v tabeli `usertable` so se enakomerno porazdelili med vsa tri vozlišča.

```
\c ycsb;
SELECT * FROM master_add_node('<n1 ip addr>', 5432);
SELECT * FROM master_add_node('<n2 ip addr>', 5432);
SELECT create_distributed_table('usertable', 'ycsb_key');
```

8. Po končani konfiguraciji sem bazo varno ustavil ter naredil varnostno kopijo podatkov na disku.

CockroachDB

1. Z Docker Swarm konfiguracijsko skripto (`stacks/cockroachdb-n1.yml`) sem pognal CockroachDB bazo na vozlišču `n2`.

2. Na vozlišču `n2` sem ročno kreiral shemo katero potrebuje YCSB za svoje delovanje.

```
CREATE DATABASE ycsb;
USE ycsb;
CREATE TABLE usertable (
    YCSB_KEY VARCHAR(255) PRIMARY KEY,
    FIELD0 TEXT, FIELD1 TEXT,
    FIELD2 TEXT, FIELD3 TEXT,
    FIELD4 TEXT, FIELD5 TEXT,
    FIELD6 TEXT, FIELD7 TEXT,
    FIELD8 TEXT, FIELD9 TEXT
);
```

3. Nato sem generiral podatke. V bazo na vozlišču `n2` sem z YCSB orodjem naložil 5 milijonov zapisov, kar na disku zasede približno 6GB prostora.

```
ycsb load jdbc \
    -P workloads/workloada \
    -P configs/cockroachdb-n1.properties \
    -p threadcount=16 \
    -p recordcount=5000000
```

4. Podatkovno bazo sem varno ustavil ter naredil varnostno kopijo fizične podatkovne baze na disku.

5. Nato sem Docker Swarm konfiguracijsko skripto (`stacks/cockroachdb-n3.yml`) pognal CockroachDB bazo na treh vozliščih.

6. Baza je sama zaznala dve novi vozlišči in pričela avtomatsko z replikacijo podatkov na ostala dva vozlišča. Ko se je replikacija končala sem bazo varno ustavil ter naredil varnostno kopijo fizične podatkovne baze na disku.

4.5.3 Program za avtomatizacijo

Vozlišče `n0` je v vlogi odjemalca. Na njem teče program, ki za vse kombinacije parametrov (baza, število vozlišč, število sočasnih povezav) izvaja teste in beleži rezultate. Program predpostavlja, da obstajajo za vsako konfiguracijo v naprej pripravljeni podatki na točno določenem mestu. Postopek za pripravo podatkov sem opisal v poglavju 4.5.2. Program v grobem za vsako kombinacijo parametrov izvede naslednje koraka:

1. Obnovi vnaprej definirane podatke (`cp -a`)
2. Zažene podatkovno bazo (`docker stack up`)
3. Izvede YCSB test (`ycsb run jdbc ...`)
4. Zabeleži rezultat v datoteko CSV
5. Ustavi podatkovno bazo (`docker stack rm`)
6. Počisti podatke (`rm -rf`)

Koraki 2, 3, 4 in 5 se ponovijo za vsak tip obremenitve v točno določenem vrstnem redu (A, B, C, F, D). Vrstni red obremenitev je pomemben [64], ker obremenitve tipa A, B, C in F ne vstavljajo novih zapisov. Obremenitev tipa D pa vstavlja nove zapise, zato je po vsaki izvedbi potrebno obnoviti bazo na prvotno stanje.

Zaradi morebitnih odstopanj sem vse teste ponovil trikrat.

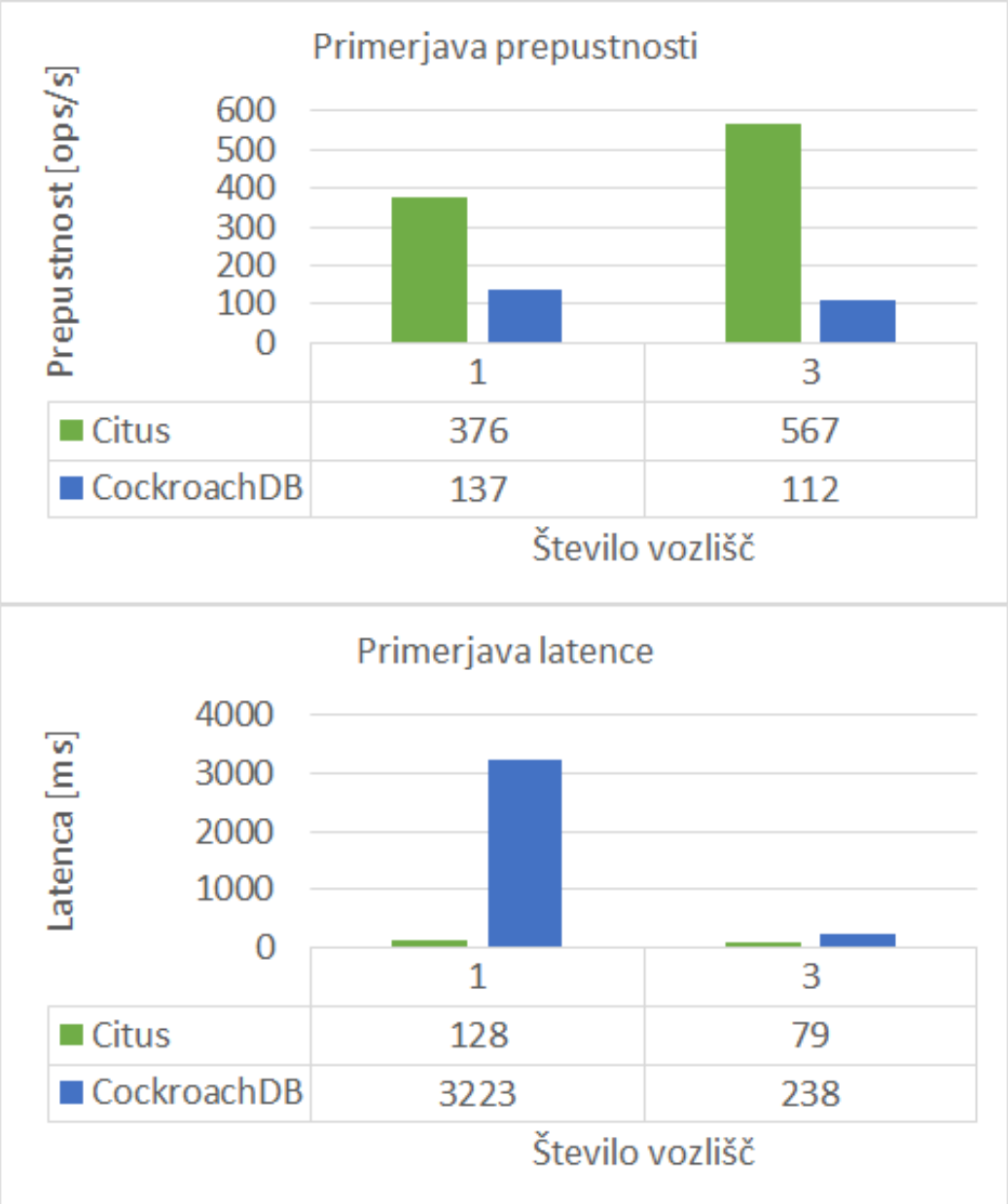
4.6 Testiranje stičnih operacij

TODO

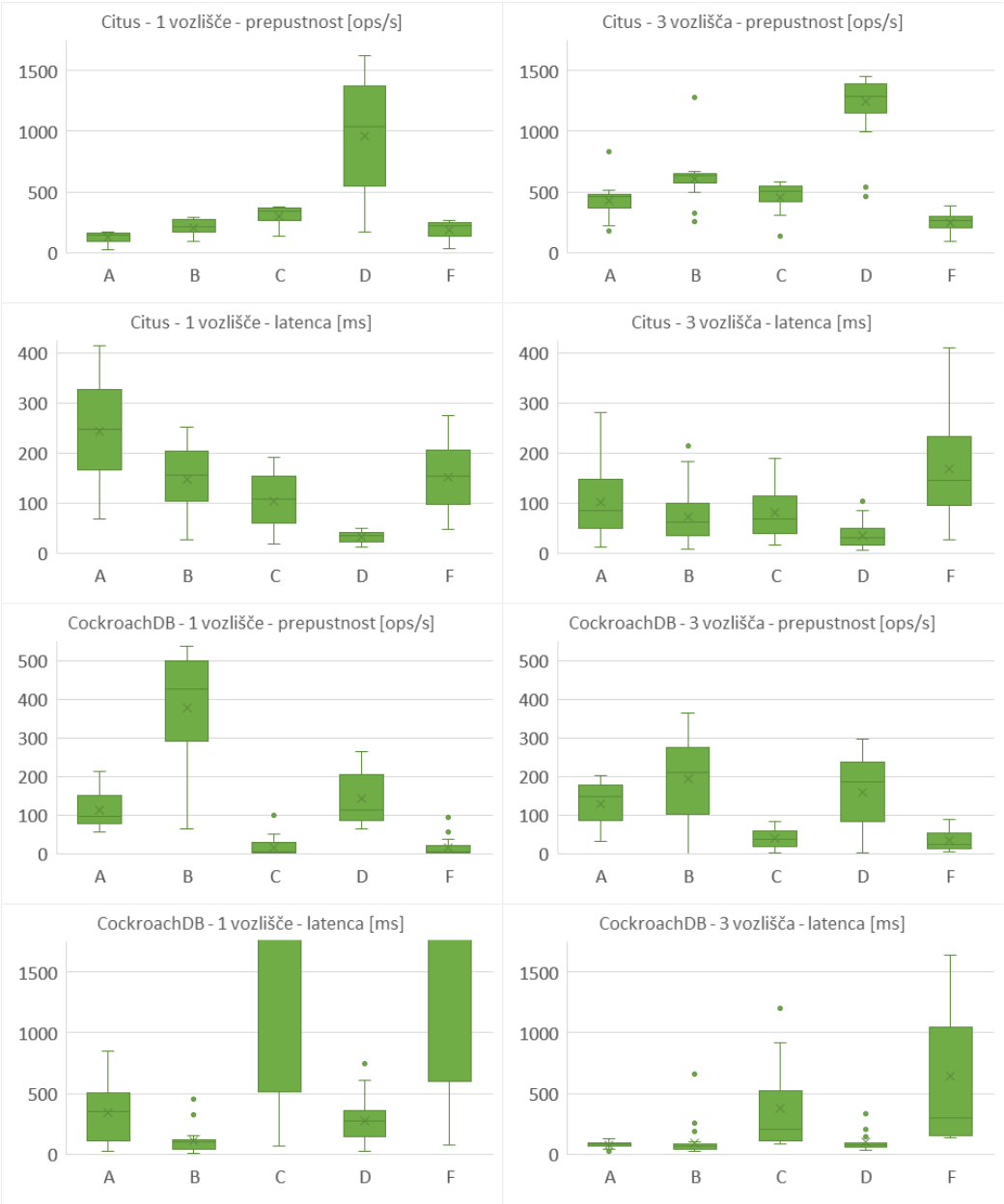
4.7 Rezultati

Vsi rezultati so na voljo na GitHub repozitoriju `matjazmav/diploma-ycsb` [38]. V mapi `results` se nahajajo datoteke z neobdelani podatki. Analizo

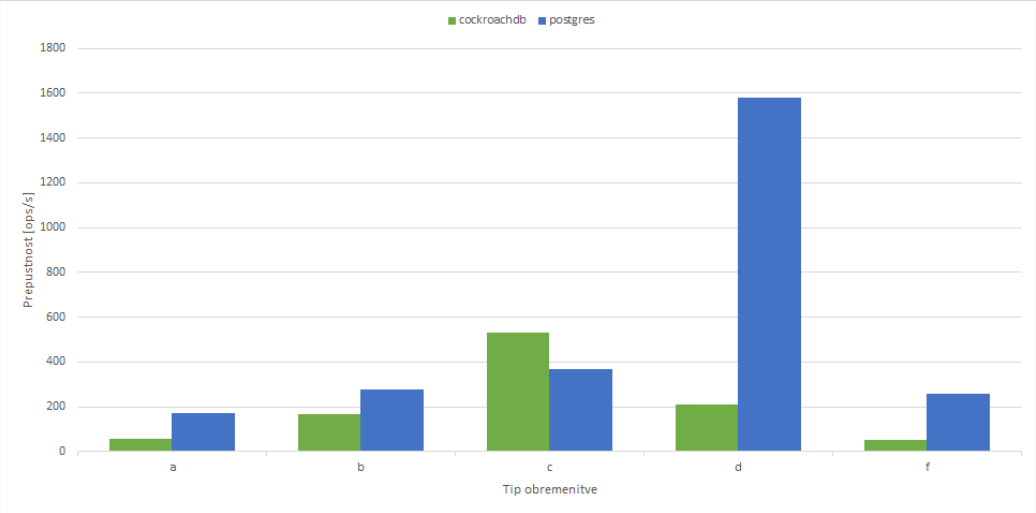
rezultatov pa sem izvedel v ~~excel~~ Excel datoteki `results.xlsx`. Po analizi sem prišel do spodnjih rezultatov.



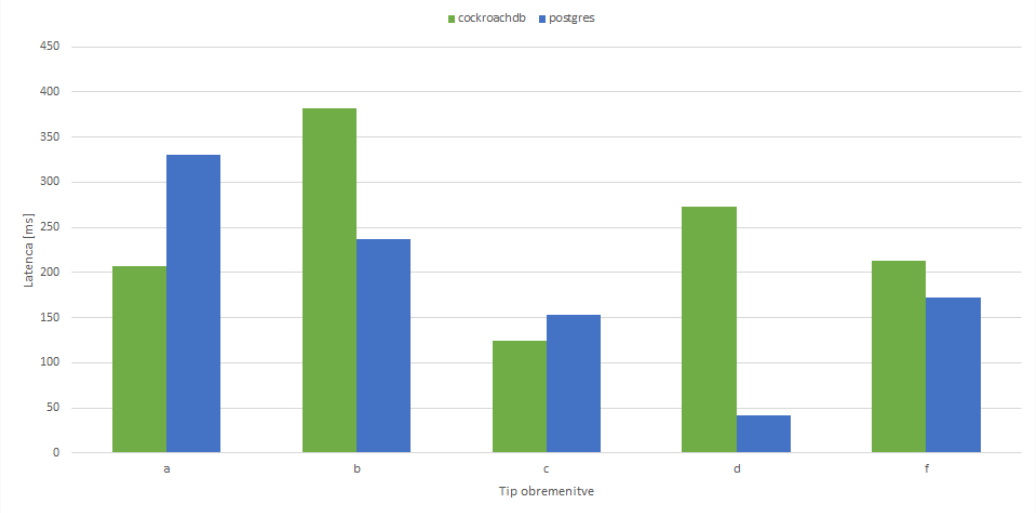
Slika 4.1: Groba primerjava povprečne prepustnosti in latence med obema podatkovnima bazama na enem in treh vozliščih.



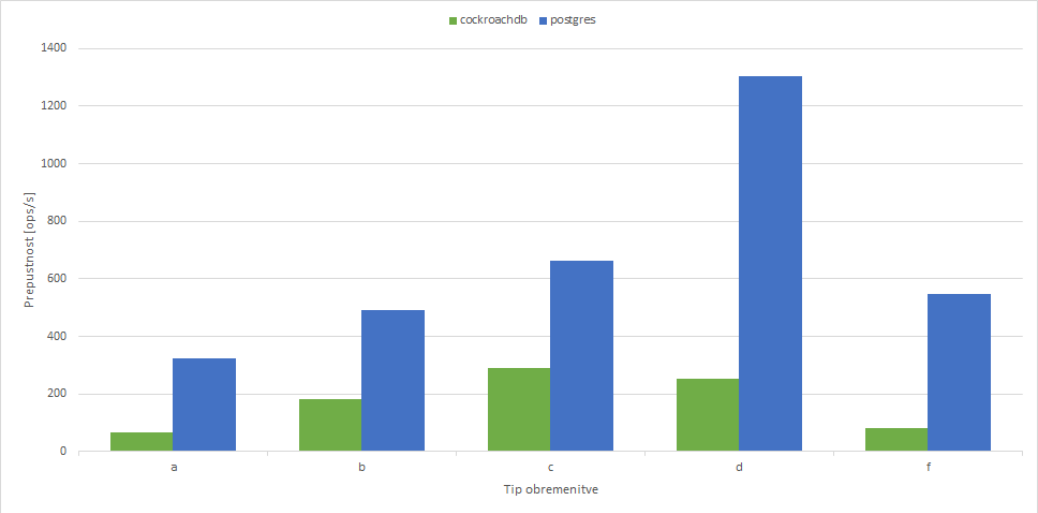
Slika 4.2: Prikazuje primerjavo prepustnosti in latenc glede na vrsto obremenitve med podatkovno bazo Citus in CockroachDB. Graf **CockroachDB - 1 vozlišče - latenca [ms]** je zaradi lažje primerjave odrezan.



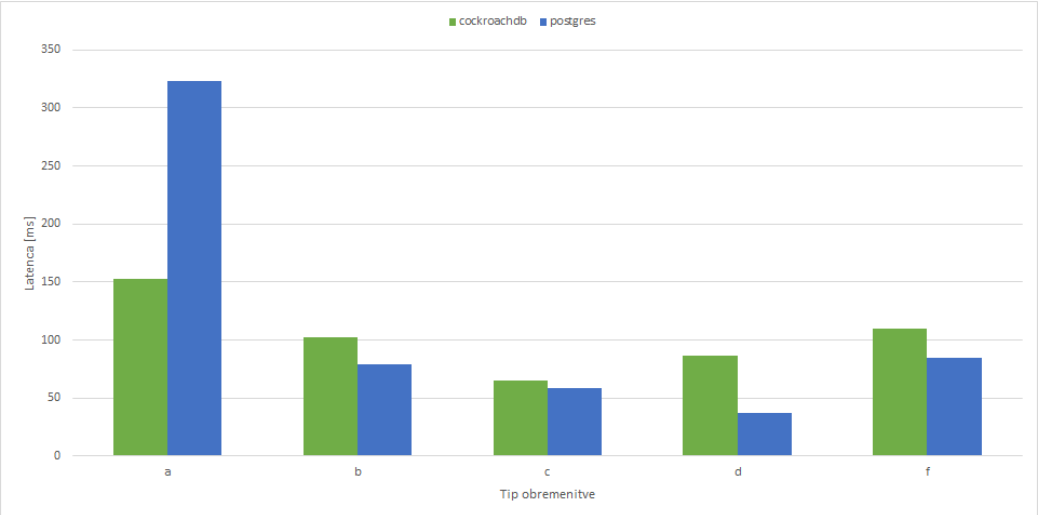
Slika 4.3: Primerjava maksimalne prepustnosti glede na tip obremenitve pri enem vozliščih.



Slika 4.4: Latenca pri maksimalni obremenitvi glede na tip obremenitve pri enem vozliščih.



Slika 4.5: Primerjava maksimalne prepustnosti glede na tip obremenitve pri treh vozliščih.



Slika 4.6: Latenca pri maksimalni obremenitvi glede na tip obremenitve pri treh vozliščih.

4.8 Ugotovitve

Zmogljivostna analiza je pokazala, da se podatkovna baza Citus pri večini obremenitev, katere sem testiral, odziva bitveno bolje kot podatkovna baza CockroachDB.

S slike 4.1 je razvidno, da CockroachDB v primerjavi s Citus dosega 2,7 krat manjšo prepustnost na enem vozlišču in kar 5 krat manjšo prepustnost na treh vozliščih. Pri primerjavi latence pa je CockroachDB v primerjavi s Citus na enem vozlišču dosegel približno 25,2 krat večjo latenco, na treh vozljiščih pa samo še 3 krat večjo.

S slike 4.2 je razvidno, da je podatkovna baza CockroachDB v primerjavi s podatkovno bazo Citus manj stabilna. To se odraža na posameznih grafih saj so podatki pri podatkovni bazi CockroachDB veliko bolj razpršeni.

4.8.1 Hipoteza 1

CockroachDB bo na enem vozlišču nekoliko počasnejši od PostgreSQL podatkovne baze.

Razlog za to hipotezo je, da je podatkovna baza PostgreSQL že dolgo na trgu [49] in je uporabljena na mnogih projektih. Zato je zmogljivostno bolje optimizirana kakor CockroachDB. CockroachDB pa je na trgu od leta 2015. Prvo stabilno verzijo (1.0.0) so objavili maja 2017 [51], druga verzija (2.0.0) pa je bila objavljena aprila 2018. Vsaka verzija je dodala veliko novih funkcionalnosti.

Hipotezo sem potrdil, kar je razvidno tudi na sliki 4.1. Podatkovna baza CockroachDB v primerjavi z PostgreSQL dosega približno 2,7 krat nižjo prepustnost.

Podobno zmogljivostno primerjavo so decembra 2017 izvedli na zasebni raziskovalni univerzi v Bruslju [51]. Z orodjem YCSB (verzija 0.12.0) so primerjali obremenitve tipa A, B in C. Ugotovili so, da se podatkovna baza CockroachDB (verzija 1.1.3) v primerjavi z PostgreSQL (verzija 9.6.5) odziva

približno 10 krat slabše kakor PostgreSQL. V njihovem primeru so testno okolje postavili v Amazon oblaku, vsako bazo na treh ~~vozljiščih~~vozliščih.

4.8.2 Hipoteza 2

CockroachDB bo zaradi linearne skalabilnosti na treh vozliščih skoraj tri krat bolj zmogljiv.

Vsa vozlišča povezana v CockroachDB gručo so simetrična, kar pomeni, da vsa vozlišča opravljajo enake naloge. Ob predpostavki, da se uporabniki enakomerno razporedijo preko vseh vozlišč, ter, da je obremenitev enakomerna, naj bi bila rast prepustnosti linearna [11].

To hipotezo sem ovrgel. Meritve so pokazale, da podatkovna baza CockroachDB doseže celo slabšo prepustnost na treh vozliščih. To razvidno na sliki 4.1.

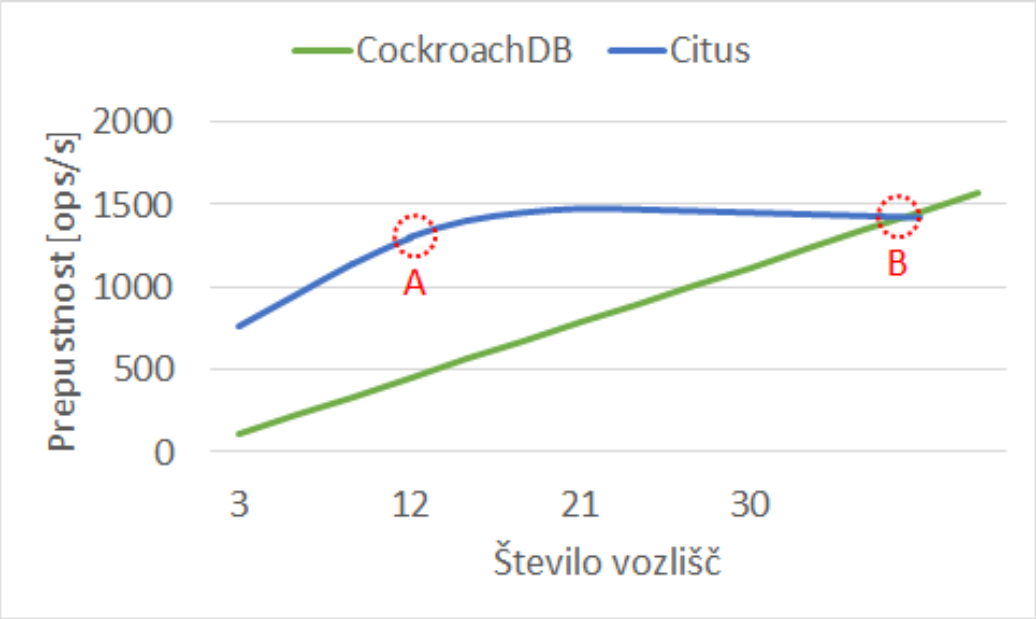
Razlog za tak rezultat naj bi bila primerjava med konfiguracijo na enem in treh vozliščih [65]. CockroachDB na treh vozliščih s privzetimi nastavitvami začne z postopkom replikacije podatkov. Privzeto je vsak podatek repliciran trikrat, enkrat na vsakem vozlišču. CockroachDB tako zagotavlja konsistenco in visoko razpoložljivost.

Rezultati neuradne TPC-C zmogljivostne analize, katero so izvedli v CockroachLabs, kažejo, da je CockroachDB linearno skalabilen [13]. Ob skaliranju z 3 na 30 vozlišč ter ob sorazmernem povečanju obremenitve so dosegli tudi sorazmerno večjo prepustnost.

Prišel sem do ugotovitve, da je CockroachDB linearno skalabilna podatkovna baza, (1) če je obremenitev enakomerno porazdeljena med vsa vozlišča in (2) faktor replikacije ob skaliranju ostane konstanten.

Na podlagi teh ugotovitev sem pripravil optimistično oceno rasti povprečne prepustnosti. Obe bazi se v idealnih pogojih in do neke mere skalirata linearno, kar je prikazano na sliki 4.7. Vendar pa se pri PostgreSQL konfiguraciji vsi odjemalci direktno povezujejo na eno vozlišče (koordinatorja), to vozlišče je ozko grlo (točka A), saj bo ob večanjem v neki točki prišlo do

zasičenosti resursov [9]. V neki točki (točka B) pa bo CockroachDB presegel prepustnost podatkovne baze Citus.



Slika 4.7: Optimistična ocena rasti povprečne prepustnosti. Pri podatkovni bazi Citus, se v točki A linearna rast konča. V točki B pa je ima CockroachDB že večjo prepustnost. Graf je le simboličen in ne predstavlja realnih meritev.

Poglavje 5

Sklepne ugotovitve

V diplomskem delu smo pregledali lastnosti ter najpogosteje uporabljene mehanizme in pristope pri implementacijah novih arhitektur NewSQL podatkovnih baz. Opisali smo arhitekturo in lastnosti izbrane NewSQL podatkovne baze CockroachDB [10]. Podatkovno bazo CockroachDB smo kasneje praktično testirali. Izvedeli smo primerjalno analizo zmogljivosti med podatkovno bazo CockroachDB in PostgreSQL [50] (z nameščeno razširitvijo Citus [6]). Veliko časa smo vložili v postavitev testnega okolja in bili pozorni, da smo odpravili čim več spremenljivk, ki bi lahko vplivale na testne scenarije. Testno okolje je bilo sestavljeno s štirih starejših računalnikov, kateri so bili povezani preko gigabitnega omrežja, na njih pa je tekel Ubuntu Server [63] in Docker [22]. Testno okolje smo v začetku nadzirali preko Telegraf agenta [60], InfluxDB podatkovne baze za shranjevanje časovnih podatkov [31] in Grafane za vizualizacijo podatkov [28]. Kljub trudu, ki smo ga vložili v postavitev testnega okolja, bi le tega lahko še izboljšali. Za boljšo primerljivost zmogljivostnih metrik, bi morala imeti vsa vozlišča enako strojno opremo, testirati pa bi moral še konfiguracije z več kot tremi vozlišči. Izvedbo testiranja smo avtomatizirali. Z orodjem YCSB [5] smo izvajali različne zmogljivostne teste, rezultate pa shranjevali v CSV datoteko in jih kasneje analizirali.

Primerjalna analiza zmogljivosti je pokazala, da je podatkovna baza CockroachDB kljub enostavnim obremenitvam, katere vrši orodje YCSB, v večini

primerov bistveno slabša od PostgreSQL podatkovne baze. V povprečju je imela podatkovna baza CockroachDB 3 krat večjo latenco in 5 krat manjšo prepustnost kakor podatkovna baza PostgreSQL na treh vozljiščih. Kljub slabšim rezultatom zmogljivostne analize, se moramo zavedati, da je težko pošteno primerjati dve zelo različni podatkovni bazi. Podatkovna baza PostgreSQL je bistveno starejša in zato bolj stabilna ter optimizirana. CockroachDB pa je na trgu šele dobri dve leti in je trenutno še vedno bolj funkcionalno usmerjena. Za smiselno odločitev pri izbiri podatkovne baze moramo poleg zmogljivostnih metrik upoštevati tudi druge, kot so poizvedovalni jezik, podpora transakcijam, zrelost sistema, skupnost, zmogljivost, težavnost postavitve ter vzdrževanja, ostale lastnosti specifične za vsako podatkovno bazo, ...

Podatkovna baza CockroachDB ima močno in aktivno skupnost. Na spletu najdemo že kar nekaj vrednotenj drugih avtorjev, od raznih primerjalnih analiz [33, 4, 12, 51], do Jepsen testov konsistence in obnašanja sistema med napakami [32, 21].

Kljub temu, da je CockroachDB relativno nova podatkovna baza, jo podjeta že uporabljajo. Na primer podjetje Baidu [3], uporablja CockroachDB pri dveh novih aplikacijah, ki sta predhodno uporabljale MySQL podatkovno bazo. Ti dve aplikaciji obsegata približno 2TB podatkov in ustvarita približno 50M zapisov na dan. Podjetje Kindred [34], ki se ukvarja z spletnim igralništvom. Z letom 2014 pa so začeli s prehodom na globalni trg. Imajo kompleksen ekosistem z več kot 200 mikrostoritev. Ta arhitektura omogoča elastičnost vendar pa mora biti skoraj v celoti avtonomna. CockroachLabs poizkuša s podjetji sodelovati, nekatera podjetja so tudi prispevala del funkcionalnosti. Podjetja katera so se odločila za uporabo podatkovne baze CockroachDB, ciljajo globalni trg, poleg tega pa želijo poenostaviti postavitev in vzdrževanje, tako v oblak, kako tudi na svoji infrastrukturi.

Glede na znanje katerega smo pridobil skozi diplomsko delo, menimo, da je podatkovna baza CockroachDB primerna za nove transakcijsko usmerjene (OLTP) aplikacije, katere zahtevajo konsistenco in visko razpoložljivost

hkrati pa ciljajo hitrorastoč globalni trg. Za aplikacije kjer je čas do trga zelo pomemben in kjer si ne moremo privoščiti velikih stroškov vzdrževanja. Pred odločitvijo moramo oceniti še podprto sintakso SQL jezika [57] in morebitne ostale lastnosti [35]. Podatkovna baza CockroachDB trenutno ni primerna za scenarije kateri zahtevajo kompleksne stične operacije, nizko latenco in ~~aplikacije katere so~~ analitično usmerjene aplikacije (OLAP) [25].

V prihodnje bi bilo zanimivo raziskati in primerjati še zelo podobno in prav tako odprtokodno podatkovno bazo TiDB [48]. TiDB je trenutno še zelo mlad produkt, z verzijo 1.0.0, ki je izšla sredi oktobra 2017 in verzijo 2.0.0 iz aprila 2018. TiDB prav tako idejno temelji na podatkovni bazi Google Spanner in se v nekaterih arhitekturnih lastnostih zelo ujema s podatkovno bazo CockroachDB. Najbolj očitna razlika je, da je arhitektura ločena na tri večje komponente, poleg tega za sinhronizacijo ure uporablja drugačen pristop. Razvijalci podatkovne baze TiDB obljubljajo, da je podatkovna baza primerna za hibridne transakcijske in analitične obremenitve.

Zanimivo bi bilo tudi razširiti zmogljivostno primerjalno analizo izvedeno v tej diplomski nalogi. ~~Distribuirane~~ Porazdeljene podatkovne baze kot je CockroachDB so primerne za okolja z vsaj tremi vozlišči, zanimivo bi bilo ugotoviti kaj se dogaja, ko v gručo povežemo še več vozlišč. Poleg osnovnih YCSB obremenitev, bi bilo zanimivo preveriti še druge, kot sta TCP-C in TCP-H. V tej diplomski nalogi smo se tem obremenitvam izognil, saj orodje [15] še ni bilo podprto za primerjavo s podatkovno bazo PostgreSQL.

Literatura

[1] Amazon Aurora – Relational Database Built for the Cloud - AWS. Dosegljivo: https://aws.amazon.com/rds/aurora. [Dostopano: 22. 8. 2018].

[2] Red Hat Ansible. Ansible is Simple IT Automation. Dosegljivo: https://www.ansible.com, May 2018. [Dostopano: 27. 5. 2018].

[3] Baidu. Dosegljivo: https://www.cockroachlabs.com/customers/baidu. [Dostopano: 17. 8. 2018].

[4] Benchmarking Google Cloud Spanner, CockroachDB, and NuoDB. Dosegljivo: https://www.nuodb.com/techblog/benchmarking-google-cloud-spanner-cockroachdb-nuodb, Sep 2017. [Dostopano: 20. 5. 2018].

[5] brianfrankcooper/YCSB. Dosegljivo: https://github.com/brianfrankcooper/YCSB, May 2018. [Dostopano: 27. 5. 2018].

[6] Citus Community is Open Source, Scale-Out, Worry-Free Postgres. Dosegljivo: https://www.citusdata.com/product/community. [Dostopano: 17. 8. 2018].

[7] ClearDB – The Ultra Reliable, Geo Distributed Data Services Platform. Dosegljivo: http://w2.cleardb.net. [Dostopano: 22. 8. 2018].

[8] Cloud Spanner | Automatic Sharding with Transactional Consistency at Scale | Cloud Spanner. Dosegljivo: https://cloud.google.com/spanner. [Dostopano: 21. 8. 2018].

[9] Cluster Management — Citus Docs 7.4 documentation. Dosegljivo: https://docs.citusdata.com/en/v7.4/admin_guide/cluster_management.html#adding-a-coordinator, Jun 2018. [Dostopano: 10. 6. 2018].

[10] Cockroach Labs. Dosegljivo: https://www.cockroachlabs.com. [Dostopano: 3. 8. 2018].

[11] CockroachDB Design. Dosegljivo: https://github.com/cockroachdb/cockroach/blob/master/docs/design.md, Jun 2018. [Dostopano: 3. 6. 2018].

[12] CockroachDB is 10x more scalable than Amazon Aurora for OLTP workloads. Dosegljivo: https://www.cockroachlabs.com/blog/performance-part-two. [Dostopano: 12. 8. 2018].

[13] CockroachDB TPC-C Performance | Cockroach Labs Guides. Dosegljivo: https://www.cockroachlabs.com/guides/cockroachdb-performance, Jun 2018. [Dostopano: 3. 6. 2018].

[14] cockroachdb/cockroach. Dosegljivo: https://github.com/cockroachdb/cockroach, Jul 2018. [Dostopano: 16. 7. 2018].

[15] cockroachdb/loadgen. Dosegljivo: https://github.com/cockroachdb/loadgen. [Dostopano: 12. 8. 2018].

[16] cockroachdb's Profile - Docker Store. Dosegljivo: https://store.docker.com/profiles/cockroachdb. [Dostopano: 3. 8. 2018].

[17] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eu-

gene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google’s globally distributed database. *ACM Trans. Comput. Syst.*, 31(3):8:1–8:22, August 2013.

[18] Database Performance & Load Balancing Software | ScaleArc. Dosegljivo: `http://www.scalearc.com`. [Dostopano: 22. 8. 2018].

[19] Deploy CockroachDB On-Premises | CockroachDB. Dosegljivo: `https://www.cockroachlabs.com/docs/stable/deploy-cockroachdb-on-premises.html#step-1-synchronize-clocks`, May 2018. [Dostopano: 20. 5. 2018].

[20] Detailed SQL Standard Comparison | CockroachDB. Dosegljivo: `https://www.cockroachlabs.com/docs/stable/detailed-sql-support.html`. [Dostopano: 17. 8. 2018].

[21] DIY Jepsen Testing CockroachDB. Dosegljivo: `https://www.cockroachlabs.com/blog/diy-jepsen-testing-cockroachdb/#from-si-to-linearizability`. [Dostopano: 8. 8. 2018].

[22] Docker. Dosegljivo: `https://www.docker.com`. [Dostopano: 17. 8. 2018].

[23] Siddhartha Duggirala. Chapter Two - NewSQL Databases and Scalable In-Memory Analytics. *Advances in Computers*, 109:49–76, Jan 2018.

[24] Franz Faerber, Alfons Kemper, Per-Åke Larson, Justin Levandoski, Thomas Neumann, Andrew Pavlo, et al. Main memory database systems. *Foundations and Trends® in Databases*, 8(1-2):1–130, 2017.

[25] Frequently Asked Questions | CockroachDB. Dosegljivo: `https://www.cockroachlabs.com/docs/stable/frequently-asked-questions.html`, Jul 2018. [Dostopano: 16. 7. 2018].

[26] Hector Garcia-Molina and Kenneth Salem. Main memory database systems: An overview. *IEEE Transactions on knowledge and data engineering*, 4(6):509–516, 1992.

[27] Gitter - cockroachdb/cockroach. Dosegljivo: `https://gitter.im/cockroachdb/cockroach`. [Dostopano: 3. 8. 2018].

[28] Grafana - The open platform for analytics and monitoring. Dosegljivo: `https://grafana.com`. [Dosegljivo: 17. 8. 2018].

[29] Rachael Harding, Dana Van Aken, Andrew Pavlo, and Michael Stonebraker. An evaluation of distributed concurrency control. *Proc. VLDB Endow.*, 10(5):553–564, January 2017.

[30] In-Memory Database | VoltDB. Dosegljivo: `https://www.voltdb.com`. [Dostopano: 22. 8. 2018].

[31] InfluxDB | The Time Series Database in the TICK Stack | InfluxData. Dosegljivo: `https://www.influxdata.com/time-series-platform/influxdb`. [Dostopano: 17. 8.. 2018].

[32] Jepsen: CockroachDB beta-20160829. Dosegljivo: `https://jepsen.io/analyses/cockroachdb-beta-20160829`. [Dostopano: 8. 8. 2018].

[33] Karambir Kaur and Monika Sachdeva. Performance evaluation of newsql databases. In *Inventive Systems and Control (ICISC), 2017 International Conference on*, pages 1–5. IEEE, 2017.

[34] Kindred. Dosegljivo: `https://www.cockroachlabs.com/customers/kindred`. [Dostopano: 17. 8. 2018].

[35] Known Limitations in | CockroachDB. Dosegljivo: `https://www.cockroachlabs.com/docs/stable/known-limitations.html`. [Dostopano: 11. 8. 2018].

[36] Rakesh Kumar. NewSQL Databases: Scalable RDBMS for OLTP Needs to Handle Big Data. Dosegljivo: https://www.academia.edu/13363206/NewSQL_Databases_Scalable_RDBMS_for_OLTP_Needs_to_Handle_Big_Data, Jun 2018. [Dostopano: 23. 6. 2018].

[37] MariaDB MaxScale | Database Proxy - Database Security, HA. Dosegljivo: https://mariadb.com/products/technology/maxscale. [Dostopano: 22. 8. 2018].

[38] matjazmav/diploma-ycsb. Dosegljivo: https://github.com/matjazmav/diploma-ycsb, May 2018. [Dostopano: 27. 5. 2018].

[39] MemSQL is the No-Limits Database Powering Modern Applications and Analytical Systems. Dosegljivo: https://www.memsql.com. [Dostopano: 22. 8. 2018].

[40] Marko Mikuletič. Comparison of relational, NoSQL and NewSQL databases. Dosegljivo: http://eprints.fri.uni-lj.si/2925, Feb 2015.

[41] MySQL :: MySQL 5.7 Reference Manual :: 21 MySQL NDB Cluster 7.5 and NDB Cluster 7.6. Dosegljivo: https://dev.mysql.com/doc/refman/5.7/en/mysql-cluster.html. [Dostopano: 22. 8. 2018].

[42] Newsql — the new way to handle big data. Dosegljivo: https://opensourceforu.com/2012/01/newsql-handle-big-data, Jan 2012. [Dostopano: 23. 6. 2018].

[43] NuoDB. Dosegljivo: https://www.nuodb.com. [Dostopano: 22. 8. 2018].

[44] João Oliveira and Jorge Bernardino. Newsql databases. 2017.

[45] Andrew Pavlo and Matthew Aslett. What's Really New with NewSQL? *SIGMOD Rec.*, 45(2):45–55, Sep 2016.

[46] Andy Pavlo. Multi-version concurrency control. Dosegljivo: https://15445.courses.cs.cmu.edu/fall2017/slides/20-multiversioning.pdf, 2017. [Dostopano: 22. 8. 2018].

[47] Andy Pavlo. Two-phase locking. Dosegljivo: https://15445.courses.cs.cmu.edu/fall2017/slides/17-twophaselocking.pdf, 2017. [Dostopano: 22. 8. 2018].

[48] PingCAP - TiDB. Dosegljivo: https://www.pingcap.com/en. [Dostopano: 12. 8. 2018].

[49] PostgreSQL: Happy Birthday, PostgreSQL! Dosegljivo: https://www.postgresql.org/about/news/978, Jun 2018. [Dostopano: 3. 6. 2018].

[50] PostgreSQL: The world's most advanced open source database. Dosegljivo: https://www.postgresql.org. [Dostopano: 17. 8. 2018].

[51] Kashif Rabbani and Ivan Putera MASLI. CockroachDB - NewSQL Distributed, Cloud Native Database. Dosegljivo: http://cs.ulb.ac.be/public/_media/teaching/cockroachdb_2017.pdf, Dec 2017. [Dostopano: 20. 5. 2018].

[52] SCALING MYSQL AND BIG DATA SERVICES. Dosegljivo: http://www.agildata.com. [Dostopano: 22. 8. 2018].

[53] Serializable, Lockless, Distributed: Isolation in CockroachDB. Dosegljivo: https://www.cockroachlabs.com/blog/serializable-lockless-distributed-isolation-cockroachdb. [Dostopano: 24. 7. 2018].

[54] Shard (database architecture) - Wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/Shard_%28database_architecture%29, Jun 2018. [Dostopano: 28. 6. 2018].

[55] sql: Database Visualizers List · Issue #25467 · cockroachdb/cockroach. Dosegljivo: https://github.com/cockroachdb/cockroach/issues/25467. [Dostopano: 3. 8. 2018].

[56] sql: Driver & ORM List · Issue #25468 · cockroachdb/cockroach. Dosegljivo: https://github.com/cockroachdb/cockroach/issues/25468. [Dostopano: 3. 8. 2018].

[57] SQL Feature Support in CockroachDB v2.0 | CockroachDB. Dosegljivo: https://www.cockroachlabs.com/docs/v2.0/sql-feature-support.html. [Dostopano: 29. 7. 2018].

[58] SQL Server In-Memory OLTP Internals for SQL Server 2016. Dosegljivo: https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/sql-server-in-memory-oltp-internals-for-sql-server-2016?view=sql-server-2017. [Dostopano: 22. 8. 2018].

[59] Swarm mode overview. Dosegljivo: https://docs.docker.com/engine/swarm, May 2018. [Dostopano: 20. 5. 2018].

[60] Telegraf | Agent for Collecting & Reporting Metrics & Data | InfluxData. Dosegljivo: https://www.influxdata.com/time-series-platform/telegraf. [Dostopano: 17. 8. 2018].

[61] The Go Programming Language. Dosegljivo: https://golang.org, May 2018. [Dostopano: 27. 5. 2018].

[62] TPC-Homepage V5. Dosegljivo: http://www.tpc.org/. [Dostopano: 6. 8. 2018].

[63] Ubuntu Server - for scale out workloads | Ubuntu. Dosegljivo: https://www.ubuntu.com/server. [Dostopano: 17. 8. 2018].

[64] YCSB - Core Workloads. Dosegljivo: https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads, May 2018. [Dostopano: 21. 5. 2018].

[65] YCSB performance analisis · Issue #26137 · cockroachdb/cockroach. Dosegljivo: https://github.com/cockroachdb/cockroach/issues/26137, Jun 2018. [Dostopano: 3. 6. 2018].

[66] Wenting Zheng. *Fast checkpoint and recovery techniques for an in-memory database*. PhD thesis, Massachusetts Institute of Technology, 2014.