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~~naslednje delo:

Tematika ~~naloge~~dela:

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~~ povezovanje s podatkovnimi bazami
JSON	Javascript object notation	~~standardna~~ standardni 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 z relacijskimi
NTP	network time protocol	~~meržni~~ mrežni 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~~ relacijskih podatkov v objekte in obratno
SQL	structured query language	strukturiran povpraševalni jezik za delo ~~s~~ z relacijskimi podatkovnimi bazami
TPC	transaction processing performance council	organizacija, ki se ukvarja ~~z~~ s primerjalno analizo podatkovnih baz v industriji
WAL	write-ahead logging	mehanizem zabeleži najprej vse spremembe v dnevnik~~šele na to pa jih izvede~~, šele nato 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~~ CockroachDB 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~~ s podatkovno bazo PostgreSQL na treh vozliščih dosega ~~5 krat~~ 5-krat manjšo ~~prepustnos in 3 krat~~ prepustnost in 3-krat večjo latenco. Poleg tega pa ima ~~trenutno~~ podatkovna baza CockroachDB trenutno 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 a 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~~ of the new database CockroachDB and the well-established PostgreSQL database. NewSQL databases are ~~build~~ built for distributed environments and ~~join properties from~~ integrate the properties of both SQL and NoSQL databases. NewSQL databases use the 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~~ achieves on average 5 times lower throughput and ~~5~~ 3 times higher latency compared to PostgreSQL on three node cluster. 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 [16] je ena izmed NewSQL podatkovnih baz. Z raziskovalnega področja je zanimiva, ker je v celoti odprtokodna, idejno pa izhaja iz Googlove podatkovne baze Spanner [10]~~, je~~. 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~~ predstavili razlog za nastanek NewSQL podatkovnih baz, opisali njihove lastnosti, razdelitev in najpogosteje uporabljene mehanizme ter pristope.

V drugem poglavju bomo opisali podatkovno bazo CockroachDB. Predstavili bomo njeno zgodovino in idejno osnovo. Poglobili se bomo v arhi-

tekturo podatkovne baze CockroachDB in predstavili nekaj ključnih mehanizmov, kateri omogočajo standardni SQL 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 [8]. 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 ~~,~~ ter izvedbo analize. Na koncu bomo predstavili rezultate ter jih komentirali.

Sledijo sklepne ugotovitve, kjer bomo ~~povzel~~ povzeli diplomsko delo in ~~izpostavil~~ izpostavili 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~~baz. Pojem NewSQL prvič omeni in definira Matthew Aslett aprila 2011 [52]. NewSQL podatkovne baze z relacijskih SQL podatkovnih baz prevzamejo standardni SQL vmesniki in ACID (angl. atomicity, consistency, isolation and durability) transakcijske lastnosti [50]. Kratica ACID stoji za:

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

Z nerelacijskih NoSQL podatkovnih baz pa prevzamejo prilagojenost na porazdeljena okolja, enostavnost horizontalnega skaliranja in replikacije. NewSQL podatkovne baze nudijo primerljivo zmogljivost z NoSQL podatkovnimi bazami [50]. Uporabljajo neblokirajoče mehanizme za nadzor sočasnosti [48]. NewSQL podatkovne baze so ~~predvsem primerne~~ primerne predvsem

za aplikacije tipa OLTP (angl. online transaction processing) z naslednjimi lastnostmi [52]:

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

Lastnosti NewSQL podatkovnih baz so [40]:

- ~~Uporabljajo standardni SQL vmenik~~ uporabljajo standardni SQL jezik kot primarni aplikacijski vmesnik za interakcijo~~.~~ ;
- ~~Podpirajo~~ podpirajo ACID transakcijske lastnosti~~.~~ ;
- ~~Uporabljajo~~ uporabljajo neblokirajoče mehanizme za nadzor sočasnosti. Tako pisalni zahtevki ne povzročajo konfliktov z bralnimi zahtevki [48]. To smo podrobneje opisali v poglavju 2.2~~.~~ ;
- ~~Implementirajo porazdeljeno~~ implementirajo porazdeljeno in ne deljeno (angl. shared-nothing) arhitekturo. Ta arhitektura zagotavlja horizontalno skalabilnost, podatkovna baza pa lahko teče na velikem številu vozlišč brez trpljenja ozkih grl [48]~~.~~ ;
- ~~Omogočajo~~ omogočajo dosti boljšo zmogljivost v primerjavi ~~z~~ s tradicionalnimi relacijskimi podatkovnimi bazami. Nekateri pravijo, da naj bi bile NewSQL podatkovne baze celo ~~50 krat~~ 50-krat bolj zmogljive [40].

2.1 Kategorizacija NewSQL arhitektur

Kategorizacija NewSQL podatkovnih baz deli implementacije proizvajalcev glede na njihove ~~pristope kateri so uporabljeni~~uporabljene pristope, da sistem

ohrani SQL vmesnik, zagotavlja skalabilnost ter zmogljivost tradicionalnih relacijskih podatkovnih baz [48]. NewSQL podatkovne baze se v grobem delijo na štiri kategorije [44, 52]:

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. Slabost teh arhitektur pa je predvsem slaba kompatibilnost z obstoječimi orodji. Primeri teh arhitektur so CockroachDB [16], NuoDB [49], VoltDB [34], MemSQL [43], ... 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. Slabost te arhitekture je, da nas tipično veže na enega od ponudnikov oblačnih storitev, poleg tega pa nimamo popolnega nadzora nad delovanjem podatkovne baze. V to kategorijo na primer ~~sodi~~ sodijo Amazon Aurora [2], Google Spanner[10], ClearDB [9] ~~,~~...

3. **Transparentno drobljenje:**
   To so komponente, ki usmerjajo poizvedbe, koordinirajo transakcije, upravljajo s podatki, participami 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 in slabi izkoriščenosti računalniških resursov. Primeri teh arhitektur so MariaDB MaxScale [41], ScaleArc [20], AgilData Scalable Cluster [47] ~~,~~...

4. **Novi shranjevalni pogoni in razširitve:**
   V to kategorijo spadajo novi shranjevalni pogoni in razširitve, kateri poizkušajo rešiti probleme skaliranja tradicionalnih relacijskih podatkovnih baz ~~. [40]~~[40]. Glede na [52] te rešitve ne sodijo v svet NewSQL podatkovnih baz, ampak le med razširitve relacijskih SQL podatkovnih baz. Slabost teh v primerjavi z novimi arhitekturami je predvsem slabša izraba računalniških resursov. Poleg tega pa so arhitekture veliko manj elastične in bolj funkionalno omejene. Primeri rešitev, ki sodijo v to kategorijo, so na primer MySQL NDB cluster [46], Citus [8] razširitev za PostgreSQL podatkovno bazo, Hekaton [70] 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~~ eden izmed najpomembnejših mehanizmov v podatkovnih bazah. Omogoča, da do podatkov dostopa več niti hkrati, ~~med tem~~ medtem pa ohranja atomičnost ter ~~garantira~~ jamči izolacijo [52].

Večina NewSQL podatkovnih baz implementira varianto mehanizma z urejenimi časovnimi oznakami (angl. timestamp ordering) oziroma TO. Najbolj priljubljen je decentraliziran več-verzijski ~~mehanizme~~ mehanizem za nadzor sočasnosti (angl. multi-version concurrency control) v nadaljevanju kar MVCC [52]. 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~~. Za vsako posodobitev 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 [27]. Ključna komponenta, da MVCC mehanizem deluje v porazdeljenih okoljih, je ~~čimbolj~~ čim bolj točna sinhronizacija ure med vozlišči. Na primer Google Cloud Spanner uporablja posebno infra-

strukturo, 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 uporabljajo hibridne protokole [52].

Poleg MVCC mehanizma nekatere implementacije podatkovnih baz uporabljajo kombinacijo MVCC ter dvo-fazno zaklepanje (angl. two-phase locking) oziroma 2PL [52, 27].

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~~ s 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 [53] 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)	...	T_1: zapiše novo vrednost A; T_2: čaka
U(A)	...	T_1: ~~sporsti~~ sprosti 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 [54] 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 kot predpomnilnik uporablja glavni pomnilnik ~~kot predpomnilnik~~ [52].

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. ~~Poslednično~~ Posledično 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) [26]. Glavna ideja pri tem ~~principu~~ načelu 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 [52]. Tako se v glavnem pomnilniku nahajajo vroči podatki ~~,~~ to so podatki, katerih se poizvedbe velikokrat dotaknejo, v sekundarnem pomnilniku pa se nahajajo mrzli podatki [26].

~~Tukaj~~ Pri tem se pojavi vprašanje, kateri pristop je ~~boljše izbrati,~~ bolje izbrati: (1) primarno shranjevanje v glavni pomnilnik ali (2) ~~priamrno~~ primarno shranjevanje v blokovno naslovljivi disk z velikim glavnim pomnilnikom, kjer ~~podatkovna baza lahko predpomni vse ,~~ lahko podatkovna baza 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~~ neoptimizirane za tako delovanje [30].

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 dogodkov v dnevnik, ki se nahaja na obstojnem pomnilniku, največkrat kar SSD. Beleženje dogodkov v dnevnik največkrat povzroči ozko grlo, zato ti sistemi poizkušajo minimizirati število teh operacij [27]. Obstaja nekaj pristopov, kako IMDB podatkovne baze beležijo dogodke v dnevnik:

- ~~Ključavnica~~ ključavnica nad dnevnikom se odklene čim prej. S tem se zmanjša čas blokiranja ter omogoči, da se ostali dogodki hitreje zapišejo v dnevnik [26]~~.~~;
- ~~Beleženje samo~~ beleženje zgolj dogodkov, kateri spreminjajo stanje podatkov, saj so samo ti dogodki potrebni za obnovo podatkov [27]~~.~~;
- ~~Periodično~~ periodično beleženje dogodkov v dnevnik. Pri tem pristopu lahko ob morebitni napaki pride do izgube transakcij [26]~~.~~;
- ~~Asinhrono~~ asinhrono potrjevanje ne čaka potrditve transakcije. Pri tem pristopu lahko ob morebitni napaki pride do izgube transakcij [26].

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) [26].

2.4 Drobljenje

Drobljenje (angl. sharding) je pristop, s katerim večina NewSQL podatkovnih baz dosega horizontalno skalabilnost [52]. Drobljenje je vrsta horizontalnega particioniranja (angl. horizontal partitioning) podatkov, pri katerem se

vsaka particija hrani na ločenem strežniku (logičnem ali fizičnem) [66]. Podatki so razdeljeni horizontalno (po vrsticah) na več delov glede na vrednosti v izbranih ~~stolpcev~~ stolpcih v tabeli, ~~te stolpce~~ katere imenujemo delitveni atributi (angl. partitioning attributes). Najbolj poznana sta dva tipa delitve:

1. **Delitev glede na interval:**
   Glede na postavljene intervale se zapisi delijo na particije. Ta pristop je uporaben predvsem tam, kjer so podatki logično strukturirani v intervale, delitveni atribut pa ima veliko kardinalnost, majhno frekvenco in ni ~~naraščujoč~~ naraščajoč ali padajoč.

2. **Razpršena delitev:**
   Ta delitev za delovanje uporablja razpršitveno funkcijo (angl. hash function), katera ~~podatke~~ poizkuša ~~deliti čimbolj~~ podatke deliti čim bolj enakomerno preko vseh particij.

Idealno podatkovna baza iz poizvedbe prepozna, katerih particij se poizvedba tiče, ter nato izvede poizvedbo, porazdeljeno preko teh particij. Dobljene rezultate združi in vrne en rezultat [52].

Nekatere izmed NewSQL podatkovnih baz omogočajo migracijo podatkov med participami 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~~ moramo podatke ponovno enakomerno razdeliti. Ta pristop ni nov, uporablja ga večina NoSQL podatkovnih baz, je pa bolj kompleksen, saj morajo NewSQL podatkovne baze ~~potrebujejo zagotavljati ACID~~ zagotavljati ACID transakcijske lastnosti [52].

2.5 Replikacija

Najboljši način za doseganje visoke razpoložljivosti je uporaba replikacije. Večina sodobnih podatkovnih baz podpira nek pristop replikacije [52]. 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 [52].

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 [52, 33].

V visoko konsistentnih podatkovnih bazah morajo biti ~~replikacija~~ replikacije potrjene na vseh replikah, šele takrat se ~~smatra~~šteje, da je replikacija uspešna [26]. Pri porazdeljenih podatkovnih bazah, katere uporabljajo soglasni algoritem (angl. consensus algorithm), je dovolj, da replikacijo potrdi večina replik. Od soglasnih algoritmov najpogosteje zasledimo algoritma Paxos in Raft [51]. Algoritem Raft je nekoliko enostavnejši za razumevanje in implementacijo ter je v porazdeljenih sistemih pogosteje uporabljen [51].

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~~omrežij.

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 [52]. Pričakuje se, da bo podatkovna baza na voljo skoraj ves čas, saj že kratkotrajni izpadi lahko pomenijo velike finančne izgube. Tukaj ~~ponavadi~~ po navadi govorimo o tako imenovanih petih devetkah (angl. five nines) oziroma 99,999 % ~~razpoložljivostjo, to~~ razpoložljivosti, kar označimo ~~z~~ s pojmom visoka razpoložljivost.

Tradicionalne podatkovne baze ponavadi po navadi tečejo na enem vozlišču in največkrat uporabljajo neko implementacijo algoritma ARIES (angl. Algorithms for Recovery and Isolation Exploiting Semantics) [45] za obnovo ob izpadu. ARIES za svoje delovanje uporablja WAL (angl. Write-Ahead Log), to . 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 razveljavitve (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 . [52][52].

Pri NewSQL podatkovnih bazah algoritem ARIES, ki ga uporabljajo tradicionalne podatkovne bazeni direktno , ni neposredno uporabljen. NewSQL podatkovne baze so porazdeljene, kar pomeni, ko eno vozlišče odpove, si da si ob odpovedi enega vozlišča 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 [52]. Poleg tega je algoritem ARIES zasnovan za diskovno orientirane podatkovne baze, kar pa ni optimalno za nove pomnilniško usmerjene arhitekture NewSQL podatkovnih baz [80].

Poglavje 3

CockroachDB

CockroachDB je porazdeljena SQL podatkovna baza, ~~temelji na transakcijski in visoko konsistenti~~ ki temelji na transakcijskem in visoko konsistentnem ključ-vrednost shranjevalnem mehanizmu. Zagotavlja ACID transakcijske lastnosti, kot primarni jezik 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~~ oblaku in namenjena globalnim oblačnim storitvam. Je transakcijska podatkovna baza in ni primerna za večje analitične obremenitve. Je zelo enostavna za upravljanje ter uporabo ~~. [29]~~[29].

Leta 2015 so Spencer Kimball, Ben Darnell in Peter Mattis ustanovili podjetje CockroachLabs. Njihov glavni produkt je v celoti odprtokodna podatkovna baza CockroachDB [16]. Vsi trije ustanovitelji so bili predhodno inženirji v podjetju Google. Podatkovna baza Google Spanner pa je bila navdih za začetek podatkovne baze CockroachDB ~~. [63]~~[63].

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 [19]. CockroachDB za svoje delovanje ne potrebuje tako točne sinhronizacije ur, v splošnem je priporočena

uporaba Googlove zunanje NTP (angl. Network Time Protocol) storitve [22]. Podatkovna baza CockroachDB je odprto kodna in lahko deluje v oblaku ali pa lokalno na operacijskih sistemih Linux, Windows in OS X.

3.1 Arhitektura

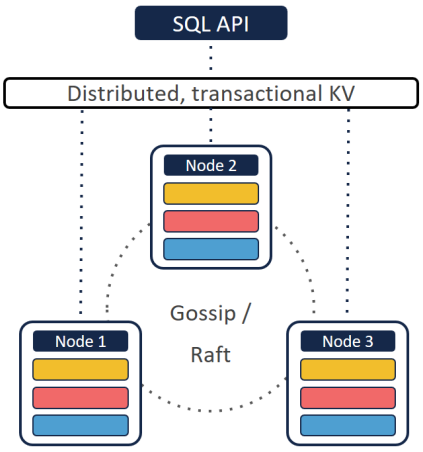
CockroachDB je odprtokodna, avtomatizirana, porazdeljena, skalabilna in konsistentna SQL podatkovna baza. Večino upravljanja je avtomatiziranega, kompleksnost samega sistema pa je prikrita končnemu uporabniku.

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 baze, kar je bolj podrobno ~~opisa~~ opisano v uradni dokumentaciji [12] in prvotnih zasnutkih delovanja CockroachDB podatkovne baze [13].

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.

- **Soglasje (angl consensus):** To je postopek, preko katerega porazdeljeni sistemi pridejo do soglasja o vrednosti enega podatka. Podatkovna baza CockroachDB za doseganje soglasja ~~uporabja~~ uporablja algoritem Raft. 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 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~~kvoruma, preden se sprememba ~~smatra~~ šteje za potrjeno.
- **Transakcija (angl. transaction):** Množica ~~operacij~~ izvršenih operacij na podatkovni bazi, katere ohranjajo ACID transakcijske 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 [29].



Slika 3.1: Arhitekturni pregled [63]

Plasti

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

1. **SQL plast:** Prevede SQL poizvedbe v 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~~ operacije 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 [23]. Zunanje aplikacije komunicirajo preko PostgreSQL žičnega protokola (angl. PostgreSQL wire protocol) [58]. 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~~, katero ga zna obdelati. To omogoči enostavno porazdelitev bremena.

Ko vozlišče prejme SQL zahtevek, ga najprej razčleni v abstraktno ~~sitaksno~~ sintaksno drevo. Potem CockroachDB začne s pripravo poizvedovalnega

plana. V tem koraku CockroachDB preveri tudi semantično pravilnost poizvedbter , nato vse operacije prevedene v KV operacije ter transformira podatke v binarno obliko. S pomočjo poizvedovalnega plana transakcijska plast nato izvede vse operacije. Kot primer poglejmo poenostavljeno stanje KV hrambe (tabela 3.1) po izvedbi naslednjih SQL stavkov:

```
\DIFdelbegin \DIFdel{CREATEA }\DIFdelend \DIFaddbegin \DIFadd{CREATE }\DI
    name STRING PRIMARY KEY,
    address STRING,
    url STRING
);
INSERT INTO customer VALUES (
    'Apple',
    '1 Infinite Loop, Cupertino, CA',
    'http://apple.com/'
);
```

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 [13]. 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 eno vrstico v tabeli `customer` z s primarnim ključem `Apple`.

3.1.2 Transakcijska plast

Podatkovna baza CockroachDB je konsistentna~~, to~~. To dosega tako, da transakcijska plast implementira celotno semantiko ACID transakcij. Vsak stavek je obravnavan kot svoja transakcija~~za katerim~~, za katero stoji `COMMIT`~~, temu~~. Temu pravimo način avtomatskega potrjevanja transakcij (angl. autocommit mode). Transakcije pa niso omejene samo na en stavek, 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 spremenjen v status prekinjeno (angl. aborted), se transakcija ponovno izvrši~~. Če~~, če pa so izpolnjeni vsi pogoji, se transakcija potrdi. Status v transakcijskem zapisu se spremeni v potrjeno (angl. commited).

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~~ s pisalnim namenom na ključu A [65].

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

3.1.3 Porazdelitvena plast

Vsi podatki v gruči so na voljo preko ~~kateregakoli~~ katerega koli vozlišča. CockroachDB shranjuje podatke v urejeno vrsto tipa ključ-vrednost. Ta podatkovna struktura je namenjena shranjevanju ~~sistemskih~~ tako 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~~, preko katerih lahko CockroachDB 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 pa obseg, 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~~64 MiB. 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~~64 MiB, se razdeli v dva ~~32MiB~~ 32 MiB podatkovna obsega. Ta dvo-nivojska indeksna struktura omogoča, da naslovimo do $2^{(18+18)} = 2^{36}$ obsegov. Vsak obseg pa privzeto naslavlja ~~$2^{26}B = 64MiB$~~ $2^{26}B = 64MiB$ pomnilniške prostora. Teoretično lahko podatkovna baza CockroachDB s privzetimi nastavitvami ~~lahko naslovi do $2^{(36+26)}B = 4EiB$~~ naslovi do $2^{(36+26)}B = 4EiB$ 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 [51]. 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~~ne,~~ , ne 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 oz. liho število, večje ali enako tri in manjše ali enako številu vseh vozlišč v gruči~~. Ter~~ ; ter f maksimalno število vozlišč, ki še lahko odpove~~brez,~~ , brez da bi vplivalo na delovanje podatkovne baze. Na primer če je replikacijski faktor $r = 3$, pomeni, da za vsak obseg obstajajo 3 replike, shranjene na treh različnih vozliščih. Gruča posledično lahko tolerira odpoved enega vozlišča $f = 1$.

Za vsak obseg obstaja Raft skupina, kjer je eno vozlišče, ki vsebuje repliko, ~~označena~~ označeno 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 [64]. Ker KV shramba RocksDB omogoča transakcije, to bistveno olajša implementacijo ACID transakcij v podatkovni bazi CockroachDB.

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 podatkov pretečejo po 24 urah in so počiščene iz shrambe. Primer poizvedbe po zgodovinskih podatkih:

```
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~~ šele kasneje v optimizacije. Tako je verzija 1.0.0 (maj 2017) omogočila razvijalcem večino potrebnih funkcionalnosti, kot so poizvedovalni jezik SQL porazdeljene poizvedbe, ACID transakcije, visoka razpoložljivost preko ~~avtomatkse~~ avtomatske replikacije in enostavno ~~namestitve~~namestitev. Verzija 2.0.0 (april 2018) pa je prinesla zmogljivostne ~~izbolšave,~~ izboljšave ter boljšo kompatibilnost ~~z~~ s podatkovno bazo PostgreSQL.

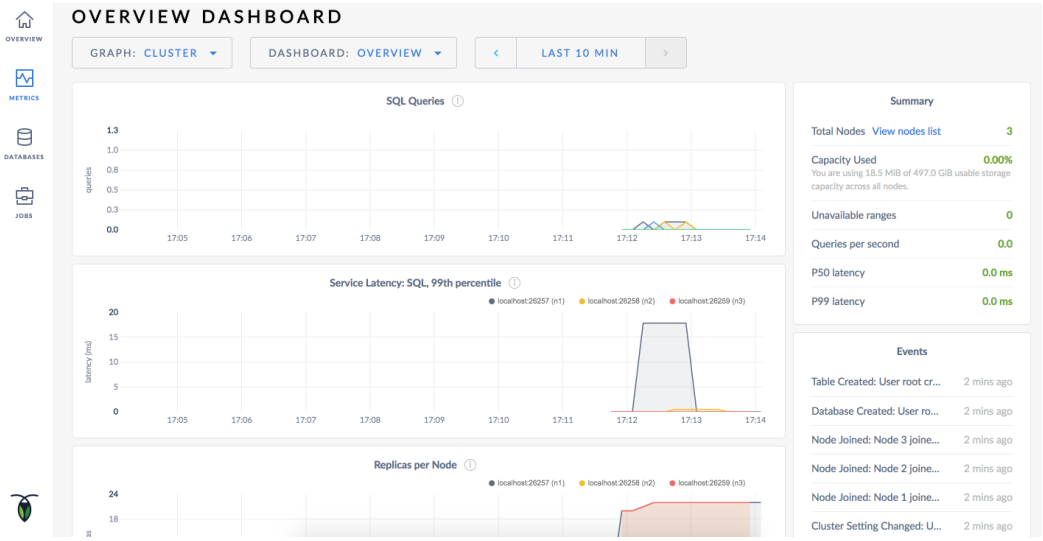
3.2.1 Enostavnost

Podatkovna baza CockroachDB stremi k enostavnosti upravljanja in vzdrževanja. Večina kompleksnosti glede zagotavljanja visoke razpoložljivosti, avtomatske obnove vozlišč, usmerjanja porazdeljenih poizvedb in izenačevanja

obremenitve je skoraj v celoti skrita končnemu uporabniku. Izogiba se zunanjim odvisnostim in je na voljo kot ena sama binarna datoteka. Minimalno je za zagon gruče ~~je~~ potrebno na vsakem vozlišču zagotoviti ~~sinhronizacije~~ sinhronizacijo ure, priporočena je uporaba zunanje Google NTP storitve, ter kopirati 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~~ vmesnik 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~~ administratorskega vmesnika podatkovne baze CockroachDB.

3.2.3 Podpora standardom SQL

CockroachDB, kakor večina relacijskih podatkovnih baz, podpira podmnožico SQL standarda [23]. Poleg tega implementira nekatere najpogostejše razširitve, razširitve, specifične za PostgreSQL, ter svoje razširitve. V spodnjih podpoglavjih ~~bom v grobem opisal~~bomo v grobem opisali, kaj od jezika SQL ponuja CockroachDB, bolj podroben opis pa se nahaja v uradni dokumentaciji [69].

Podatkovni tipi

CockroachDB podpira naslednje podatkovne tipe:

- **Celoštevilčni**: `INT` in `SERIAL` (avtomatsko kreirana unikatna številka)
- **Decimalni**: `DECIMAL` in `FLOAT`
- **Textovni**: `STRING` in `COLLATE` (podobno kot `STRING`, vendar upošteva specifike jezika)
- **Datumski in časovni**: `DATE`, `TIME`, `INTERVAL` (časovni interval) in `TIMESTAMP` (datum in čas)
- **Ostali**: `ARRAY`, `BOOL`, `BYTES`, `INET` (IPv4 in IPv6 naslovi), `JSONB` in `UUID` (~~128 bitna~~ 128-bitna heksadecimalna vrednost)

Poleg zgoraj naštetih podatkovnih tipov ~~,~~ima vsak podatkovni tip zaradi ~~kompatibilnost~~ kompatibilnosti s standardnim SQL jezikom še nekaj psevdonimov. CockroachDB ne podpira podatkovnih tipov, kot so `XML`, `UNSIGNED INT` ter `SET` in `ENUM`.

Shema

CockroachDB podpira vse standardne ukaze za kreiranje, spreminjanje in odstranjevanje baz, tabel, stolpcev, omejitev, indeksov in pogledov. Od podatkovnih omejitev podpira vse standardne, to so: `PRIMARY KEY`, `NOT NULL`, `UNIQUE`, `CHECK`, `FOREIGN KEY` in `DEFAULT`. Omejitev `PRIMARY KEY` ~~,~~ 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~~ s tipom `UUID`, saj je ta pomemben za samo delovanje podatkovne baze.

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

Prožilci (angl. triggers) še niso podprti in tudi ne načrtovani za naslednje verzije.

Transakcije

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

Tabelarični izrazi

CockroachDB podpira referenciranje tabel, pogledov, poizvedb, tabelaričnih funkcij in CTE-jev (angl. common table expressions). Podpora za CTE-je je le osnovna, referenciramo jih lahko samo enkrat, poleg tega pa jih ni mogoče uporabiti v pogledih.

Od pod-poizvedb (angl. sub-queries) omogoča le nekorelirane pod-poizvedbe. Nekorelirane pod-poizvedbe so poizvedbe, katere ne referencirajo nadrejenih poizvedb. Običajno je možno korelirano pod-poizvedbo spremeniti v nekorelirano z uporabo stične operacije:

```
# Z \DIFdelbegin \DIFdel{uporabno }\DIFdelend \DIFaddbegin \DIFadd{uporab
SELECT c.name
FROM customers c
WHERE EXISTS(
    SELECT *
    FROM orders o
    WHERE o.customer_id = c.id
);

# Z uporabo stične operacije (podprto)
SELECT DISTINCT ON(c.id) c.name
FROM customers c CROSS JOIN orders o
WHERE c.id = o.customer_id;
```

Od stičnih operacij CockroachDB omogoča `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL JOIN` in `CROSS JOIN`. Stične operacije še niso optimizirane. Da bi ~~le~~ te dosegle optimalno delovanje, je zelo pomemben optimalni zapis poizvedbe, ustrezni filtri in morebitna uporaba eksperimentalne zastavice `SET experimental_force_lookup_join = true`.

Operatorji, skalarni izrazi, logični izrazi in funkcije

CockroachDB implementira večji del standardnih operatorjev, skalarnih izrazov, logičnih izrazov ter funkcij. Podpira poizvedovanje po tipu JSON, omogoča iskanje ~~z~~ s POSIX regularnimi izrazi ter implementira agregacijske funkcije (funkcije, katere se izvedejo nad 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 Licenca

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

- svetovanje~~,~~;
- tehnično pomoč~~,~~;
- geografsko porazdelitev podatkov na nivoju ene vrstice (podatki so blizu uporabnika)~~,~~;
- vizualizacijo geografsko porazdelitve gruče na zemljevidu~~,~~;
- nadzor dostopa na nivoju skupin in
- porazdeljeno kreiranje in ~~obnova~~ obnovo varnostnih kopij.

3.2.5 Podprta orodja, gonilniki~~in~~ , ORM-ji in skupnost

Podatkovna baza CockroachDB implementira standardni SQL [23], 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 vse gonilnike CockroachLabs v času pisanja zagotavlja le beta podporo [68].

Od orodij za upravljanje ~~z~~ s podatkovno bazo nam CockroachLabs ponuja konzolno aplikacijo `cockroach sql`. Od grafičnih orodij nudijo beta podporo [67] za pgweb [55], dbglass [21], Postico [61], PSequel [62], TablePlus [72] in Valentina studio [77].

Orodje pgweb, ~~katero~~ ki je bilo primarno razvito za podatkovno bazo PostgreSQL, ima trenutno najbolj aktivno skupnost, je odprtokodno in podpira

vse glavne operacijske sisteme (Windows, OSX in Linux). Do aplikacije dostopamo preko spletnega brskalnika. Orodje pgweb 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` [16]. Projekt ima aktivno skupnost ~~, kateremu~~ in mu trenutno sledi približno 14 tisoč navdušencev. Poleg GitHub repozitorija, kjer se odvija ves razvoj, vodenje nalog in pomoč, CockroachLabs vodi še spletno stran [12] (dokumentacija, form, blog, mediji), Gitter kanal [31] in Docker repozitorij [18].

Poglavje 4

Primerjalna analiza zmogljivosti CockroachDB

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

Za primerjavo ~~z~~ s Citus smo 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~~ implementirata standardni SQL ter komunicirata preko PostgreSQL žičnega protokola. Podatkovno bazo smo nekoliko bolj podrobno opisali v poglavju 4.3.

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

- **TPC:**
  TPC (angl. transaction processing performance council) [75] je ~~ne profitna organizacija, katera~~ neprofitna organizacija, ki nudi preverljive podatke TPC zmogljivostnih analiz podatkovnih baz. TPC definira več različnih standardnih testov, ~~kateri~~ ki simulirajo različne realne obre-

menitve. ~~Te testi so točno~~ Ti testi so natančno opisani, strogo omejeni in kompleksni. Primer testa za transakcijske obremenitve je test TPC-C~~kateri~~ , ki simulira obremenitve v dobavnem skladišču. Primer za testiranje analitičnih ~~obremenitve~~ obremenitev je test TPC-H~~kateri~~ , ki izvaja kompleksne poizvedbe za podporo pri odločanju.

TPC testi so standardizirani in simulirajo realne obremenitve za določene problemske domene. ~~Te~~ Ti testi bi bili najbolj primerni, vendar pa so zelo točno definirani, kompleksni in težki za izvedbo, zato jih v ~~diplomski nalogi~~ diplomskem delu nismo izvedli. Uradnih TPC testov za podatkovno bazo CockroachDB še ni.

- **pgbench:**
  Je enostavno orodje, namenjeno zmogljivostni analizi PostgreSQL podatkovnih baz [57]. Simulira transakcijsko obremenitev, podobno zastarelemu testu TPC-B. Test TPC-B simulira obremenitve v bančništvu.

  Orodje pgbench v času testiranja še ni bilo kompatibilno ~~z~~ s podatkovno bazo CockroachDB.

- **cockroachdb/loadgen:**
  Je skupek ~~orodj~~ orodij, namenjenih zmogljivostni analizi CockroachDB podatkovne baze [17]. Orodje vsebuje nabor testov, ki generirajo TPC-C, TPC-H, YCSB in KV obremenitve. Te orodja CockroachLabs interno uporablja za zmogljivostno primerjavo med posameznimi verzijami podatkovne baze CockroachDB. Kasneje je bil objavljen članek, kjer so primerjali rezultate TPC-C obremenitve med podatkovnima bazama CockroachDB in Amazon Aurora [14].

  Orodje je enostavno, vendar pa je slabo dokumentirano in ne omogoča zmogljivostne analize podatkovne baze PostgreSQL.

- **Apache JMeter:**
  Je orodje, namenjeno izvajanju raznih obremenitvenih testov [4]. Orod-

je omogoča izvajanje poljubnih obremenitev podatkovnih baz preko JDBC vmesnika.

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

- **YCSB:**
  YCSB (angl. Yahoo! Cloud Serving Benchmark) je orodje, namenjeno za primerjavo zmogljivostnih metrik med raznimi podatkovnimi bazami [7]. Orodje smo podrobneje opisali v poglavju 4.4.

V naslednjih podpoglavjih bomo opisali točen postopek, s katerim smo izvedli primerjalno analizo. Opisali bomo strojno arhitekturo, pripravo posameznih konfiguracij, pripravo podatkov in samo testiranje. Na koncu bomo predstavili rezultate zmogljivostne analize ter naše ugotovitve.

4.1 Hipoteze

Pred začetkom izvajanja primerjalne analize smo postavili 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~~ trikrat bolj zmogljiv.

4.2 Testno okolje

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 [22].

Veliko časa smo vložili v samo postavitev testnega okolja. Zaradi natančnosti testov smo bili pozorni, da smo odpravili čim več spremenljivk, ki

bi lahko ~~uplivale~~ vplivale na končne rezultate. Vsa štiri vozlišča so med seboj povezana preko gigabitnega Ethernet omrežja v Docker Swarm [71] gručo. Vozlišče `n0` je v vlogi vodje, vozlišča `n1`, `n2` in `n3` pa so v vlogi delavcev. Za uporabo Docker Swarm tehnologije smo se ~~odločil~~ odločili zaradi enostavnosti postavitve testnega okolja, avtomatizacije in izvedbe samih testov ter lažje ponovljivosti testov.

Ker tehnologije Docker še nismo dobro poznali, smo imeli v fazi prototipiranja težave pri zmogljivosti. Izkazalo se je, da je bil ključen problem v tem, da nismo ~~uporabi podatkovnih nosilcov~~ uporabili podatkovnih nosilcev (angl. data volumes) [1].

	procesor	**pomnilnik**	**trdi disk**
n0	Intel Core2 Quad CPU Q9400 št. jeder: 4 frekvenca: 2,66 GHz predpomnilnik: 6 MB	4 GB	SAMSUNG HD753LJ velikost: 750 GB frekvenca: 7200 RPM predpomnilnik: 32 MB
n1	Intel Core i5 CPU 650 št. jeder: 4 frekvenca: 3,20 GHz predpomnilnik: 4 MB	4 GB	WDC WD10EARS-22Y5B1 velikost: 1 TB frekvenca: 5400 RPM predpomnilnik: 64 MB
n2	Intel Core i7-3770 št. jeder: 8 frekvenca: 3,40 GHz predpomnilnik: 8 MB	8 GB	ST500DM002-1BD142 velikost: 500 GB frekvenca: 7200 RPM predpomnilnik: 16 MB
n3	Intel Core i5-2400 št. jeder: 4 frekvenca: 3,10 GHz predpomnilnik: 6 MB	4 GB	Hitachi HDS721050CLA662 velikost: 500 GB frekvenca: 7200 RPM predpomnilnik: 16 MB

Tabela 4.1: Specifikacije strojne opreme, ~~katere~~ ki 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 Razširitev Citus

Citus je razširitev za podatkovno bazo PostgreSQL [8]. Omogoča enostavno horizontalno skaliranje podatkov za več najemniške (angl. multi tenant) aplikacije in obdelavo analitičnih podatkov v realnem času (angl. real time analytics). Uporaba razširitve Citus ni primerna, če:

- nam zadošča le eno vozlišče~~,~~ ;
- poizvedbe vračajo ogromno podatkov in
- ne potrebujemo analitike v realnem času.

Razširitev Citus nam je voljo kot odprtokodna razširitev, plačljiva različica in storitev (DBaaS). Plačljiva različica poleg fukcionalnosti odprtokodne razširitve ponuja še nekaj dodatne funkcionalnosti in ~~24 urno~~ 24-urno podporo strankam.

Za primerjavo s Citus smo se odločili predvsem zaradi enostavnosti izvedbe primerjalne zmogljivostne analize. Obe podatkovni bazi, CockroachDB in Citus, temeljita na podatkovni bazi PostgreSQL. Obe komunicirata preko PostgreSQL žičnega protokola. Prav tako pa enostavna YCSB shema, katero smo opisali v naslednjem poglavju 4.4, dobro ~~vpada~~ spada v kontekst več najemniških aplikacij.

4.4 Orodje YCSB

YCSB [7] 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 vmesnik JDBC, preko katerega smo ~~primerjal~~ primerjali podatkovno bazo CockroachDB ter Citus. YCSB nudi nekaj ~~v naprej definiranih obremenitev~~vnaprej definiranih obremenitev, katere smo tudi uporabili v naši primerjalni analizi zmogljivosti [78]:

- **A** - Obremenitev je sestavljena ~~z~~ iz 50 % `SELECT` in 50 % `UPDATE` operacij.
- **B** - Obremenitev je sestavljena ~~z~~ iz 95 % `SELECT` in 5 % `UPDATE` operacij.
- **C** - Obremenitev je sestavljena ~~z~~ iz 100 % `SELECT` operacij.
- **D** - Obremenitev je sestavljena ~~z~~ iz 95 % `SELECT` in 5 % `INSERT` operacij, pri čemer bere vedno zadnje vstavljene vrstice.
- **F** - Obremenitev je sestavljena ~~z~~ iz 50 % `SELECT` in 50 % `SELECT` - `UPDATE` operacij.

Izbira vrstice je pri A, B, C in F obremenitvah porazdeljena preko Zipf porazdelitvene funkcije. Zipfov zakon pravi, da je najbolj verjetna vrednost približno ~~dva krat~~ dvakrat bolj verjetna od druge najbolj verjetne vrednosti in trikrat bolj verjetna od tretje najbolj verjetne ~~verdnosti~~ vrednosti [28].

Orodje YCSB vrača rezultate v enostavni tekstovni obliki. Od rezultatov vrne skupni čas trajanja, prepustnosti (enačba 4.1) ter agregirano število vseh operacij in različne podatke o latencah (enačba 4.2). Od latence vrača povprečno vrednost, minimalno in maksimalno vrednost ~~,~~ ter 95 in 99 percentil.

$$throughput = 1000 * total\_operations/total\_duration\_ms \tag{4.1}$$

$$latency = operation\_end\_ms - operation\_start\_ms \tag{4.2}$$

4.5 Izvedba testiranja YCSB obremenitev

Testiranje je potekalo iz odjemalca, torej vozlišča `n0`. Meritev je bilo veliko, zato smo pripravil program, s katerim smo izvedbo testiranja avtomatizirali. Program, s katerim smo testirali, je enostaven in ni prenosljiv. Izvorna koda in rezultati meritev so na voljo na GitHub repozitoriju [42]. V tem poglavju bomo opisal vse korake, ki so bili potrebni za izvedbo meritev.

4.5.1 Namestitev programske opreme

Na vozlišču `n0` smo namestili naslednjo programsko opremo:

- YCSB (verzija 0.12.0) [7] je orodje ~~namenjeno za izvedbo~~ , namenjeno izvedbi zmogljivostne analize ter ~~primerjavo~~ primerjavi med različnimi podatkovnimi bazami. Samo orodje smo bolj natančno opisali v poglavju 4.4.

- Ansible (verzija 2.5.0) [3] je orodje za avtomatizacijo, uporabili smo ga zaradi lažjega konfiguriranja gruče preko SSH povezave.

- Go (verzija 1.10.1) [74] je programski jezik, v katerem je napisan program za avtomatizacijo testiranja. Za programski jezik Go smo se odločili ~~,~~ zaradi njegove enostavnosti.

4.5.2 Priprava podatkov

Program za avtomatizacijo testiranja 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 smo pripravili podatke za vsako bazo ter za vsako konfiguracijo (eno vozlišče in tri vozlišča)~~pripravili podatke~~. Vse konfiguracijske datoteke, ki so uporabljene v spodnjih primerih, so na voljo na GitHub repozitoriju `matjazmav/diploma-ycsb` [42].

Citus

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

2. Na vozlišču `n2` smo ročno kreirali shemo~~katero potrebuje YCSB~~ , katero za svoje delovanje potrebuje orodje YCSB.

```
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 smo generirali podatke. V bazo na vozlišču `n2` smo z YCSB orodjem naložili 5 milijonov zapisov, kar na disku zasede približno 6 GB prostora.

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

4. Podatkovno bazo smo varno ustavili ter naredili varnostno kopijo fizične podatkovne baze na disku.

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

6. Na vozliščih `n1` in `n3` smo kreirali shemo, definirano v točki 2.

7. Nato smo na vozlišču `n2` povezali še vozlišči `n1` ter `n3` in kreirali 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 smo bazo varno ustavili ter naredili varnostno kopijo podatkov na disku.

CockroachDB

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

2. Na vozlišču `n2` smo ročno kreirali shemo ~~katero potrebuje YCSB~~, katero za svoje delovanje potrebuje orodje YCSB.

```
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 smo generirali podatke. V bazo na vozlišču `n2` smo z YCSB orodjem naložili 5 milijonov zapisov, kar na disku zasede približno 6 GB prostora.

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

4. Podatkovno bazo smo varno ustavili ter naredili varnostno kopijo fizične podatkovne baze na disku.

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

6. Baza je samodejno zaznala dve novi vozlišči in pričela avtomatsko z replikacijo podatkov na ostali dva vozlišči. Ko se je replikacija končala, smo bazo varno ustavili ter ~~naredil~~ naredili 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~~ vnaprej pripravljeni podatki na točno določenem mestu. Postopek za pripravo podatkov smo opisali v poglavju 4.5.2. Program v grobem za vsako kombinacijo parametrov izvede naslednje ~~koraka~~korake:

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

Koraki 2, 3, 4 in 5 se ponovijo za vsak tip obremenitve v ~~točno~~ natančno določenem vrstnem redu (A, B, C, F, D). Vrstni red obremenitev je pomemben [78], 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 smo vse teste ~~ponovil~~ ponovili trikrat.

4.6 Izvedba testiranja stičnih operacij

Ker orodje YCSB izvaja le enostavne operacije nad eno tabelo, smo kasneje sami ročno preverili podporo za stične operacije.

4.6.1 Priprava podatkov

Osnova za te teste so nam bili podatki, katere smo predhodno pripravili. Priprava je opisana v poglavju 4.5.2. Poleg teh podatkov, ki vsebujejo le tabelo `usertable` (5 milijonov vrstic), smo kreirali še tabelo `ext` in vanjo zapisali 100 vrstic. Sam postopek kreiranja tabele `ext` je bil za Citus na treh vozliščih nekoliko drugačen, za ostale tri konfiguracije pa je bila uporabljena naslednja skripta:

```
CREATE TABLE ext (
    ycsb_key VARCHAR(255) PRIMARY KEY,
    value int not null );

INSERT INTO ext
SELECT
    ycsb_key,
    LTRIM(RIGHT(ycsb_key, 5), '0')::int % 10 AS value
FROM usertable ORDER BY ycsb_key LIMIT 100;
```

Za Citus podatkovno bazo je postopek nekoliko daljši. Ker ne moremo ~~direkno~~ neposredno s porazdeljene tabele `usertable` kopirati v lokalno tabelo `ext`, moramo najprej kreirati začasno tabelo, podatke napolniti ~~, in potem z~~ in potem iz začasne tabele podatke prestaviti v novo tabelo `ext`. To smo ~~stori~~ storili z naslednjo skripto:

```
CREATE TABLE ext (
    ycsb_key VARCHAR(255) PRIMARY KEY,
    value int not null );

CREATE TEMPORARY TABLE tmp AS
SELECT
    ycsb_key,
    LTRIM(RIGHT(ycsb_key, 5), '0')::int % 10 AS value
FROM usertable ORDER BY ycsb_key LIMIT 100;

INSERT INTO ext SELECT * FROM tmp;

SELECT create_distributed_table('ext', 'ycsb_key');
```

4.6.2 Izvedba testiranja in rezultati

Testna poizvedba, ki smo jo uporabili na vseh štirih konfiguracijah, vrača 11 vrstic in 2 stolpca (`ycsb_key` ter `field4`). Testna poizvedba je sledeča:

```
SELECT u.ycsb_key, u.field4
FROM usertable u
INNER JOIN ext e ON e.ycsb_key = u.ycsb_key
WHERE e.value = 4;
```

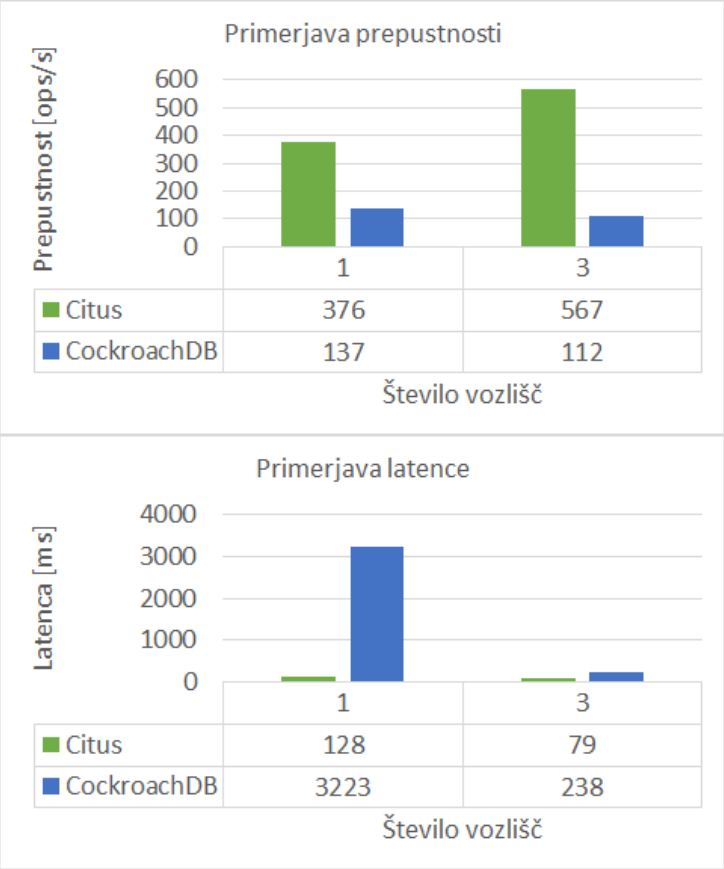
Rezultati za podatkovno bazo CockroachDB so zelo slabi. Rezultati so ~~redstavljeni~~ predstavljeni v tabeli 4.2. Podatkovna baza ne ugotovi, da tabela `ext` ~~z~~ s filtrirnim pogojem omeji rezultat na 11 vrstic, zato začne združevati obe tabeli na 5 milijonih vrsticah. Obe poizvedbi ~~sem~~ smo po 1 minuti ~~prekinil~~prekinili, po nekaj deset minutah pa je ukazna vrstica spet postala odzivna.

konfiguracija	**čas [ms]**
Citus 1 vozlišče	66,336
Citus 3 vozlišče	163,049
CockroachDB 1 vozlišče	(prekinjeno po 1 minuti)
CockroachDB 3 vozlišče	(prekinjeno po 1 ~~minut u~~minuti)

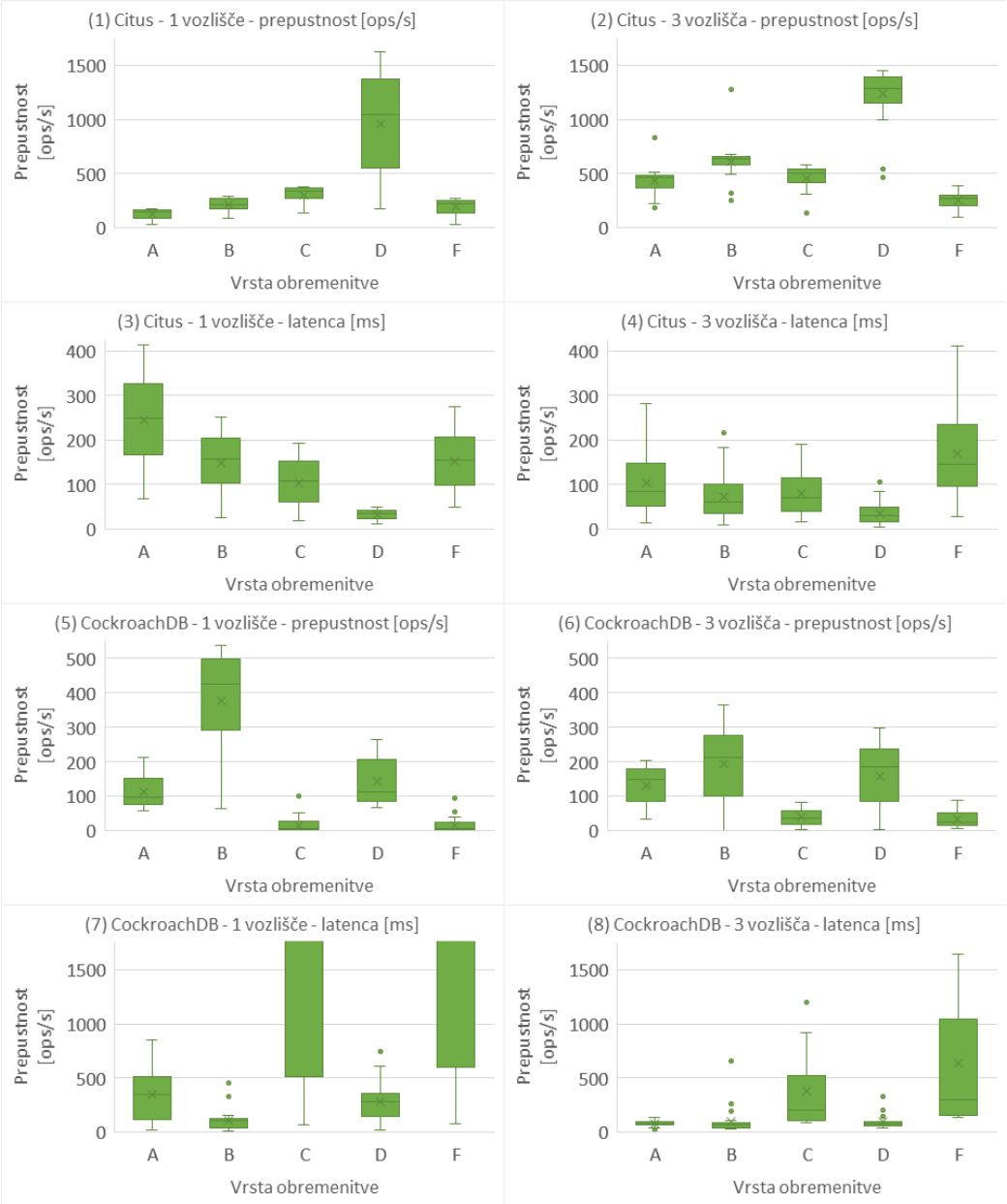
Tabela 4.2: Čas trajanja testne poizvedbe ~~z~~ s stično operacijo glede na konfiguracijo.

4.7 Rezultati

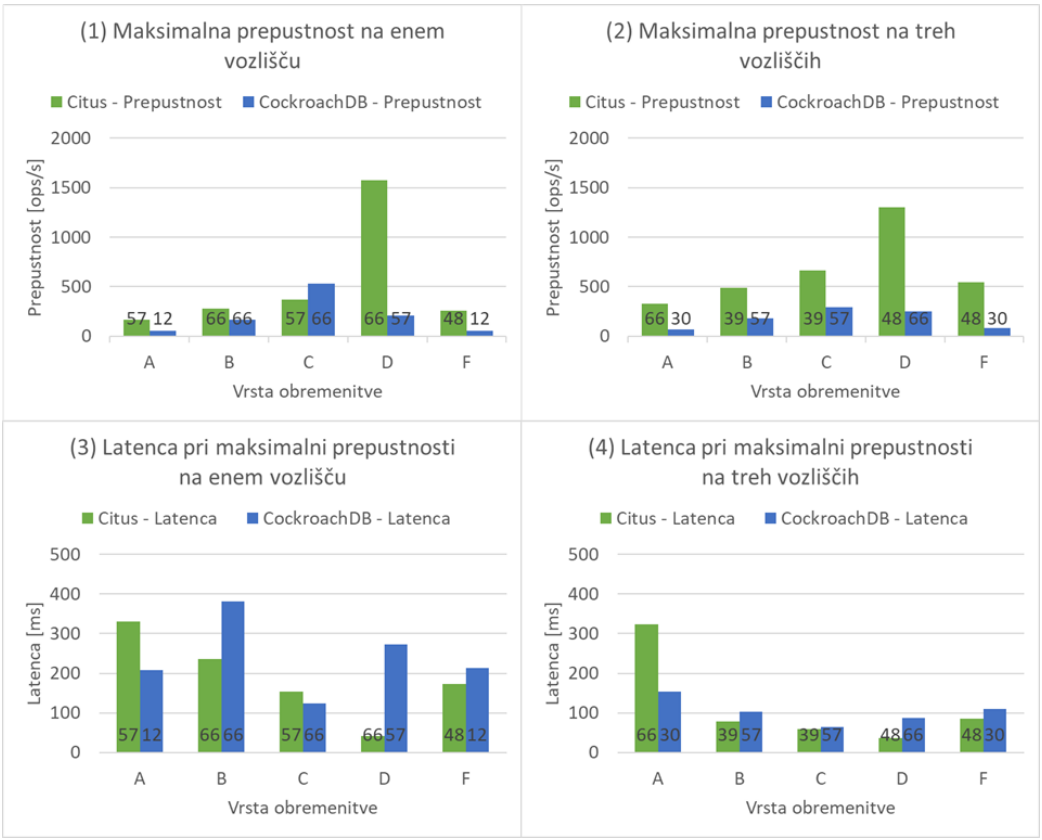
Vsi rezultati so na voljo na GitHub repozitoriju `matjazmav/diploma-ycsb` [42]. V mapi `results` se nahajajo datoteke z neobdelani podatki. Analizo rezultatov ~~pa sem izvedel~~ smo izvedli v Excel datoteki `results.xlsx`. Po analizi ~~sem prišel~~ pa smo prišli do spodnjih rezultatov.



Slika 4.1: Groba primerjava povprečne prepustnosti in latence med obema podatkovnima bazama na enem in treh vozliščih. ~~Z~~ Iz grafov je razvidno, da podatkovna baza CockroachDB dosega bistveno manjšo prepustnost, poleg tega pa ima večjo latenco.



Slika 4.2: Prikazuje primerjavo prepustnosti in latenc glede na vrsto obremenitve med podatkovno bazo Citus in CockroachDB. Graf (7) je zaradi lažje primerjave odrezan. ~~Z~~ Iz grafov ~~se vidi~~je razvidno, da je podatkovna baza Citus v primerjavi ~~z~~ s CockroachDB veliko bolj stabilna, kar se odraža v razponu med največjo in najmanjšo vrednostjo.



Slika 4.3: Prikazuje primerjavo prepustnosti in latence za vzorec z ~~maksimalna prepustnost~~maksimalno prepustnostjo. Številke v spodnjem delu serije predstavljajo število sočasnih niti, pri katerih je bila dosežena maksimalna prepustnost. ~~Z~~ Iz grafov lahko razberemo, da se podatkovna baza CockroachDB z vidika sočasnih obremenitev skalira bolje kot Citus.

4.8 Ugotovitve

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

S slike 4.1 je razvidno, da CockroachDB v primerjavi s Citus dosega 2,~~7 krat~~ 7-krat manjšo prepustnost na enem vozlišču in kar ~~5 krat~~ 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~~ 2-krat večjo latenco, na treh vozliščih pa samo še ~~3 krat~~ 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~~, saj so rezultati meritev pri podatkovni bazi CockroachDB veliko bolj razpršeni.

Če primerjamo spremembe sočasnih niti, pri katerih sta bazi dosegali maksimalno prepustnost (slika 4.3), opazimo naslednje. Podatkovna baza CockroachDB ob skaliranju na tri vozlišča lahko v povprečju ~~lahko~~ obdela 5,4 več sočasnih niti, podatkovna baza Citus pa kar 10,8 manj.

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 [59] 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 [63], druga verzija (2.0.0) pa je bila objavljena aprila 2018. Vsaka verzija je dodala veliko novih funkcionalnosti.

~~Hipotezo sem potrdil~~Hipotezo smo potrdili, kar je razvidno tudi na sliki 4.1. Podatkovna baza CockroachDB v primerjavi ~~z~~ s PostgreSQL dosega

približno 2,~~7 krat~~ 7-krat nižjo prepustnost.

Podobno zmogljivostno primerjavo so decembra 2017 izvedli na zasebni raziskovalni univerzi v Bruslju [63]. 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~~ s PostgreSQL (verzija 9.6.5) odziva približno ~~10 krat~~ 10-krat slabše, kakor PostgreSQL. V njihovem primeru so testno okolje postavili v Amazon oblaku, vsako bazo na treh vozliščih.

4.8.2 Hipoteza 2

CockroachDB bo zaradi linearne skalabilnosti na treh vozliščih skoraj ~~tri krat~~ 3-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 [13].

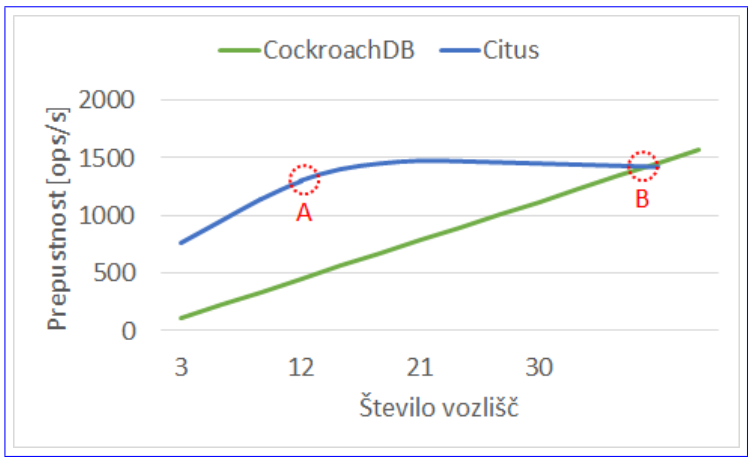
~~To hipotezo sem ovrgel~~To hipotezo smo ovrgli. Meritve so pokazale, da podatkovna baza CockroachDB doseže celo slabšo prepustnost na treh vozliščih. To ~~razvidno na sliki~~ je razvidno iz slike 4.1.

Razlog za tak rezultat naj bi bila primerjava med konfiguracijo na enem in treh vozliščih [79]. CockroachDB na treh vozliščih s privzetimi nastavitvami začne ~~z~~ s 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 [15]. Ob skaliranju ~~z~~ iz 3 na 30 vozlišč ter ob sorazmernem povečanju obremenitve so dosegli tudi sorazmerno večjo prepustnost.

~~Prišel sem~~ Prišli smo 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~~ smo pripravili optimistično oceno rasti povprečne prepustnosti. Obe bazi se v idealnih pogojih in do neke mere skalirata linearno, kar je prikazano na sliki 4.4. Vendar pa se pri PostgreSQL konfiguraciji vsi odjemalci ~~direktno~~ neposredno povezujejo na eno vozlišče (koordinatorja), ~~to vozlišče~~ ki je ozko grlo (točka A), saj bo ob ~~večanjem~~ večanju v neki točki prišlo do zasičenosti resursov [11]. V neki točki (točka B) pa bo CockroachDB presegel prepustnost podatkovne baze Citus.



Slika 4.4: 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 [12]. Podatkovno bazo CockroachDB smo kasneje praktično testirali. ~~Izvedeli~~ Izvedli smo primerjalno analizo zmogljivosti med podatkovno bazo CockroachDB in PostgreSQL [60] (z nameščeno razširitvijo Citus [8]). 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~~ iz štirih starejših računalnikov, ~~kateri~~ ki so bili povezani preko gigabitnega omrežja, na njih pa je tekel Ubuntu Server [76] in Docker [25]. Testno okolje smo v začetku nadzirali preko Telegraf agenta [73], InfluxDB podatkovne baze za shranjevanje časovnih podatkov [35] in Grafane za vizualizacijo podatkov [32]. 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 [7] 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~~ 3-krat večjo latenco in ~~5 krat~~ 5-krat manjšo prepustnost kakor podatkovna baza PostgreSQL na treh vozliščih. 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 [37, 6, 14, 63], do Jepsen testov konsistence in obnašanja sistema med napakami [36, 24].

Kljub temu, da je CockroachDB relativno nova podatkovna baza, jo podjeta že uporabljajo. Na primer podjetje Baidu [5] ~~,~~ uporablja CockroachDB pri dveh novih aplikacijah, ki sta predhodno uporabljale MySQL podatkovno bazo. Ti dve aplikaciji obsegata približno ~~2TB~~ 2 TB podatkov in ustvarita približno ~~50M~~ 50 M zapisov na dan. Podjetje Kindred [38] ~~, ki se ukvarja z~~ se ukvarja s spletnim igralništvom~~. Z~~, z letom 2014 pa so začeli s prehodom na globalni trg. Imajo kompleksen ekosistem z več kot 200 ~~mikrostoritev~~mikrostoritvami. 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~~ oblaku, kakor tudi na svoji infrastrukturi.

Glede na znanje~~katerega smo pridobil~~, katerega smo pridobili skozi diplomsko delo, menimo, da je podatkovna baza CockroachDB primerna za

nove transakcijsko usmerjene (OLTP) aplikacije, katere zahtevajo konsistenco in ~~visko~~ visoko razpoložljivost, hkrati pa ciljajo hitrorastoč globalni trg ~~. Za aplikacije~~ter 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 [69] in morebitne ostale lastnosti [39]. Podatkovna baza CockroachDB trenutno ni primerna za scenarije~~kateri~~, ki zahtevajo kompleksne stične operacije, nizko latenco in analitično usmerjene aplikacije (OLAP) [29].

V prihodnje bi bilo zanimivo raziskati in primerjati še zelo podobno in prav tako odprtokodno podatkovno bazo TiDB [56]. 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~~, izvedeno v tem diplomskem delu. Porazdeljene podatkovne baze, kot je CockroachDB, so primerne za okolja z vsaj tremi vozlišči~~, zanimivo~~. 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 TPC-C in TPC-H.

Literatura

[1] About storage drivers. Dosegljivo: https://docs.docker.com/storage/storagedriver. [Dostopano: 26. 8. 2018].

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

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

[4] Apache JMeter - Apache JMeter™. Dosegljivo: https://jmeter.apache.org. [Dostopano: 26. 8. 2018].

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

[6] 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].

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

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

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

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

[11] 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].

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

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

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

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

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

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

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

[19] 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, Eugene 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.

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

[21] dbglass. Dosegljivo: `http://dbglass.web-pal.com`. [Dostopano: 25. 8. 2018].

[22] 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].

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

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

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

[26] Siddhartha Duggirala. NewSQL Databases and Scalable In-Memory Analytics. In *Advances in Computers*, volume 109, pages 49–76. Elsevier, 2018.

[27] 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.

[28] Fagan, Stephen and Gençay, Ramazan. An introduction to textual econometrics. *Handbook of empirical economics and finance*, pages 133–154, 2011.

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

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

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

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

[33] 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.

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

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

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

[37] 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.

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

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

[40] 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].

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

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

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

[44] Marko Mikuletič. Comparison of relational, NoSQL and NewSQL databases. Bachelor's thesis, University of Ljubljana, Faculty of Computer and Information Science, Feb 2015.

[45] C Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems (TODS)*, 17(1):94–162, 1992.

[46] 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].

[47] MySQL Scaling and Big Data Services. Dosegljivo: http://www.agildata.com. [Dostopano: 22. 8. 2018].

[48] NewSQL — The New Way to Handle Big Data. Dosegljivo: https://opensourceforu.com/2012/01/newsql-handle-big-data, Jan 2012. [Dostopano: 23. 6. 2018].

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

[50] João Oliveira and Jorge Bernardino. NewSQL Databases - MemSQL and VoltDB Experimental Evaluation. In *KEOD*, pages 276–281, 2017.

[51] Diego Ongaro and John K Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, pages 305–319, 2014.

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

[53] Andy Pavlo. Multi-Version Concurrency Control. Dosegljivo: https://15445.courses.cs.cmu.edu/fall2017/slides/20-multiversioning.pdf, 2017. [Dostopano: 22. 8. 2018].

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

[55] Pgweb - Cross-platform client for PostgreSQL databases. Dosegljivo: http://sosedoff.github.io/pgweb. [Dostopano: 25. 8. 2018].

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

[57] PostgreSQL: Documentation: 10: pgbench. Dosegljivo: https://www.postgresql.org/docs/10/static/pgbench.html. [Dostopano: 26. 8. 2018].

[58] PostgreSQL: Documentation: 9.3: Frontend/Backend Protocol. Dosegljivo: https://www.postgresql.org/docs/9.3/static/protocol.html. [Dostopano: 23. 8. 2018].

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

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

[61] Postico – a modern PostgreSQL client for the Mac. Dosegljivo: https://eggerapps.at/postico. [Dostopano: 25. 8. 2018].

[62] PSequel, a PostgreSQL GUI Tool for macOS. Dosegljivo: http://www.psequel.com. [Dostopano: 25. 8. 2018].

[63] 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].

[64] RocksDB | A persistent key-value store. Dosegljivo: https://rocksdb.org. [Dostopano: 25. 8. 2018].

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

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

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

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

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

[70] 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].

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

[72] TablePlus | Modern, Native Tool for Database Management. Dosegljivo: https://tableplus.io. [Dostopano: 25. 8. 2018].

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

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

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

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

[77] Valentina Studio. Dosegljivo: https://valentina-db.com/en/valentina-studio-overview. [Dostopano: 25. 8. 2018].

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

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

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