Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Mav

Visoko skalabilna "new SQL" 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
SQL	structured query language	strukturiran poizvedovalni jezik
NoSQL	not structured query language	ne strukturiran poizvedovalni jezik
YCSB	Yahoo! Cloud Serving Benchmark	Yahoo! Cloud Serving Benchmark
JDBC	Java Database Connectivity	javanski vmesnik za povezavo s podatkovnimi bazami
KV	key-value	ključ-vrednost

Povzetek

Naslov: Visoko skalabilna "new SQL" relacijska podatkovna baza CockroachDB

Avtor: Matjaž Mav

V vzorcu je predstavljen postopek priprave diplomskega dela z uporabo okolja LaTeX. Vaš povzetek mora sicer vsebovati približno 100 besed, ta tukaj je odločno prekratek. Dober povzetek vključuje: (1) kratek opis obravnavanega problema, (2) kratek opis vašega pristopa za reševanje tega problema in (3) (najbolj uspešen) rezultat ali prispevek magistrske naloge.

Ključne besede: podatkovne baze, skaliranje, new SQL, CockroachDB, Postgres.

Abstract

Title: Higly scalable "new SQL" relational database CockroachDB

Author: Matjaž Mav

This sample document presents an approach to typesetting your BSc thesis using LaTeX. A proper abstract should contain around 100 words which makes this one way too short.

Keywords: databases, scalability, new SQL, Postgres.

Poglavje 1

Uvod

večanje rač resoursev (cpu, mem) potreba po spremembi arhitekture. Motivacija...

Poglavje 2

NewSQL

NewSQL je oznaka za nove relacijske podatkovne baze, katere združujejo lastnosti tako relacijskih SQL kakor tudi ne-relacijskih NoSQL podatkovnih baze. Pojem NewSQL prvič omeni in definira Matthew Aslett ~~Aprila~~ aprila 2011 [23]. NewSQL podatkovne baze omogočajo poizvedovanje preko standardnega SQL vmesnika. So horizontalno skalabilne ter nudijo primerljivo zmogljivost z NoSQL podatkovnimi bazami. Poleg tega zagotavljajo ACID lastnosti tradicionalnih relacijskih podatkovnih baz [22]. Uporabljajo ne blokajoče mehanizme za nadzor sočasnosti [21]. NewSQL podatkovne baze so predvsem primerne za aplikacije tipa OLTP z naslednjimi lastnostmi [23]:

- Izvajajo veliko število kratkotrajnih bralno-pisalne transakcij.
- Vsaka transakcija se dotika majhnega dela podatkov z uporabo indeksnih poizvedb.
- Poizvedbe se ponavljajo, spreminjajo pa se samo vhodni parametri.

Lastnosti NewSQL podatkovnih baz so:

- Uporabljajo standardni SQL kot primarni aplikacijski vmesnik za interakcijo.
- Podpirajo ACID transakcijske lastnosti.

- Uporabljajo ~~ne-blokajoče~~ ne blokajoče mehanizme za nadzor sočasnosti. Tako pisalni zahtevki ne povzročajo konfliktov z bralnimi zahtevki.

- Implementirajo distribuirano ~~ne-deljeno~~ ne deljeno (angl. shared-nothing) arhitekturo. Ta arhitektura zagotavlja ~~zagotavlja~~ horizontalno skalabilnost, podatkovna baza pa lahko teče na velikem številu vozlišč brez trpljenja ozkih grl [21].

- Omogočajo dosti boljšo zmogljivost na enem vozlišču v primerjavi z tradicionalnimi relacijskimi podatkovnimi bazami. Nekateri pravijo, da naj bi bile NewSQL podatkovne baze celo 50 krat ~~boj~~ bolj zmogljive [18].

2.1 Kategorizacija

Kategorizacija NewSQL podatkovnih baz deli implementacije proizvajalcev glede na njihove pristope kateri so uporabljeni, da sistem ohrani SQL vmesnik, zagotavlja skalabilnost ter zmogljivost tradicionalnih relacijskih podatkovnih baz. [21] NewSQL podatkovne baze se v grobem delijo na štiri kategorije [20]:

1. **Nove arhitekture:**
   V to kategorijo spadajo arhitekture, ~~katera so optimizirane so~~ 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. [23]

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. [23]

3. **Transparentno drobljenje:**
   To so komponente, ki usmerjajo poizvedbe, koordinirajo transakcije, upravljajo ~~z~~ 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. Primer takega transparentnega vmesnika je MariaDB MaxScale. [23]

4. **Novi shranjevalni motorji in razširitve:**
   V to kategorijo spadajo novi shranjevalni motorji in razširitve, kateri poizkušajo rešiti probleme skaliranja tradicionalnih relacijskih podatkovnih baz. [18] Primer rešitev, ki sodijo v to kategorijo so na primer MySQL NDB cluster, Citus razširitev za ~~PostgreSQL~~ Postgres podatkovno bazo ter Hekaton OLTP motor za SQL Server. Glede na neke vire [23] te rešitve ne sodijo v svet NewSQL podatkovnih baz.

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 [23].

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~~ sočasnosti (angl. multi-version concurrency control) v nadaljevanju kar MVCC [23]. Glavna prednost MVCC mehanizma je, da omogoča sočasnost, saj bralni zahtevki nikoli ne blokirajo pisalnih. Poleg prednosti ima ta mehanizem tudi nekaj slabosti, za vsako posodobitev mora shraniti novo verzijo podatka ter poskrbeti, da so zastareli podatki odstranjeni ~~z~~ iz pomnilnika. Tako MVCC mehanizmi omogočajo večjo zmogljivost saj so bolj prilagojeni

na sočasnost, kar pa omogoča večjo izkoriščenost procesorskih virov [12]. Ključna komponenta, da MVCC mehanizem deluje, je čimbolj točna sinhronizacija ure. Na primer Google Cloud Spanner uporablja posebno infrastrukturo katera zagotavlja zelo točno sinhronizacijo ure z uporabo atomičnih ur. Podatkovne baze, ki pa niso vezane na infrastrukturo, kot na primer CockroachDB, pa uporabljajo hibridne protokole za sinhronizacijo ure [23].

Poleg MVCC mehanizma nekatere implementacije podatkovnih baz uporabljajo kombinacijo MVCC ter dvo-fazno zaklepanje (angl. two-phase locking) oziroma 2PL. V nekaterih primerih pa uporabljajo mehanizem razbitega sekvenčnega izvajanja (angl. partitioned serial execution). [23, 12].

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 ~~te~~ pomnilne enote relativno počasne, se v ta namen uporablja glavni pomnilnik kot predpomnilnik [23].

Historično je bil pomnilnik predrag, sedaj pa so cene pomnilnikov nižje tako, da v nekaterih primerih lahko celotno podatkovno bazo shranimo v glavni pomnilnik. S tem pa 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) [11]. Glavna ideja pri tem principu je, da vse ~~podatkov~~ podatke hranimo v pomnilniku. Poleg tega pa uporabljamo blokovno naslovljiv disk za varnost kopiranje glavnega pomnilnika. Ta ideja izvira že z leta 1980. NewSQL podatkovne baze pa dodajo mehanizem, da lahko v primeru prevelike količine podatkov, del podatkov shranijo na sekundarni pomnilnik [23]. Tako se v glavnem pomnilniku nahajajo vroči podatki, to so podatki katerih se ~~poizvedbe~~ poizvedbe velikokrat dotaknejo, v sekundarnem pomnilniku pa

se nahajajo mrzli podatki [11].

Tukaj se pojavi vprašanje kateri pristop je boljše izbrati, glavni pomnilnik ali blokovno naslovljivi disk z velikim predpomnilnikom. Res je, da bo podatkovna baza katera nosi vse podatke v predpomnilniku delovala bistveno hitreje, kljub temu pa tak sistem ni optimiziran za tak scenarij [14].

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 te sistemi poizkušajo minimizirati število [12] teh operacij. ~~Obstajajo~~ Obstaja nekaj pristopov kako IMDB podatkovne baze beležijo dogodke v dnevnik:

- Ključavnica nad dnevnikom se odkleni čim prej. S tem se zmanjša čas blokiranja ter omogoči, da se ostale sočasne transakcije hitreje ~~pišejo~~ zapišejo v dnevnik. [11]
- Beleženje samo dogodkov, kateri spreminjajo stanje podatkov, saj so samo te dogodki potrebni za obnovo podatkov. [12]
- Grupiranje dogodkov ter periodično beleženje v dnevnik. Pri tem pristopu lahko ob morebitni napaki pride do izgube transakcij. [11]
- Asinhrono potrjevanje ne čaka potrditve transakcije. Pri tem pristopu lahko ob morebitni napaki pride do izgube transakcij. [11]

Ker ~~imajo IMDB podatkovne baze zelo majhne stroške operacij, lahko~~ 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~~in se~~. 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) [11].

2.4 Drobljenje

Drobljenje (angl. sharding) je pristop s katerim večina NewSQL podatkovnih baz dosega horizontalno skalabilnost [23]. Drobljenje je vrsta horizontalnega particioniranja (angl. horizontal partitioning) podatkov, pri katerem se vsaka particija hrani na ločenem strežniku (logičnem ali fizičnem) [27]. 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~~ ne prekrivajoče intervale se zapisi delijo na ~~particij~~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 ~~z~~ iz poizvedbe prepozna katerih particij se poizvedba dotika ter nato izvede poizvedbo porazdeljeno preko teh particij. Dobljene rezultate združi in vrne en rezultat [23].

Nekatere izmed NewSQL podatkovnih baz omogočajo migracijo podatkov med particijami med izvajanjem podatkovne baze. Ta mehanizem je potreben, ker nekatere ~~particij~~ 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 [23].

2.5 Replikacija

Najboljši način za doseganje visoke razpoložljivosti je uporaba replikacije. Večina sodobnih podatkovnih baz podpira ~~neko vrsto~~ nek pristop replikacije [23]. V osnovi ~~obstaja dva pristopa replikacije~~popznamo dva pristopa repliciranja:

1. ~~**Aktivna** **Sinhrona** **replikacija**~~:
   Imenujemo jo tudi aktivno-aktivna (angl. active-active) ~~oziroma več-gospodarska (angl. multi-master) replikacija[17]~~replikacija. Pri tem pristopu vse replike sočasno obdelajo zahtevek [23].

2. ~~**Pasivna** **Asinhrona** **replikacija**~~:
   Imenujemo jo tudi aktivno-pasivna (angl. active-passive) ~~oziroma gospodar-sužnj (angl. master-slave) replikacija[17]~~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~~ sočasnosti ni mogoče izvesti zahtevkov v pravilnem vrstnem redu na vseh replikah ~~[23]~~[23, 16]
   .

~~Sinhrona in asinhrona replikacija~~

V visoko konsistentnih podatkovnih bazah morajo biti ~~transakcije~~ replikacija potrjene na vseh replikah, šele takrat se smatra, da je ~~transakcija potrjena [11].~~ replikaija uspešna [11]. Pri distribuiranih 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 [23]. 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~~ 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. [23]

Pri NewSQL podatkovnih bazah, algoritem ARIES, ki ga uporabljajo tradicionalne podatkovne baze ni direktno uporabljen. NewSQL podatkovne baze so distribuirane, 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 ~~z~~ iz ostalih vozlišč pridobiti vse spremembe, ki so se zgodile v času, ko je bilo vozlišče nedosegljivo [23]. Poleg tega je algoritem ARIES zasnovan za diskovno orientirane podatkovne baze, kar pa ni optimalno za nove arhitekture NewSQL podatkovnih baz [36].

Poglavje 3

CockroachDB

CockroachDB je distribuirana 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 ANSI 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. [13]

Leta 2015 so Spencer Kimball, Ben Darnell in Peter Mattis ustanovili podjetje CockroachLabs~~njihov glavni produkt pa~~ . Njihov glavni produkt je v celoti odprto kodna podatkovna baza CockroachDB [7]. Vsi trije ustanovitelji so bili ~~prej~~ predhodno inženirji ~~zaposleni v Googlu~~v podjetju Google. Podatkovna baza Google Spanner pa je bila navdih za začetek podatkovne baze CockroachDB. [25]

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 atomičnih ur, skrbi za zelo natančno sinhronizacijo ure na Googlovi infrastrukturi [9]. CockroachDB za svoje delovanje ne potrebuje tako točne sinhronizacije ur, generalno je priporočena

uporaba Googlove zunanje NTP (angl. Network Time Protocol) storitve [10]. Podatkovna baza CockroachDB je ~~odprto kodna~~ odprtokodna in lahko deluje v oblaku ali pa lokalno na operacijskih sistemih Linux ter OS X.

3.1 Arhitektura

CockroachDB je odprto kodna, avtomatizirana, distribuirana, 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 ~~pa bom predstavil samo arhitekturo po posameznih plasteh~~bom povzel samo arhitekturo in delovanje podatkovne, kar je bolj podrobno opisa v uradni dokumentaciji [4] in prvotnih zasnutikh delovanja CockroachDB podatkovne baze [5].

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.

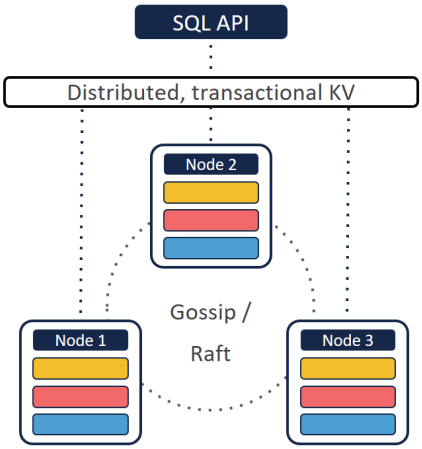
- **Konsistenca (angl. consistency):** Podatki v podatkovni bazi so vedno v končnem veljavnem stanju in nikoli v vmesnem stanju. CockroachDB ohranja konsistenco preko ACID transakcij ter CAP teorema.

- **Soglasje (angl consensus):** To je postopek, preko katerega distribuirani 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 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 dostopnost (angl. high availability):** CockroachDB zagotavlja visoko dostopnost, ki ji pravimo pet devetk (angl. five nines), kar pomeni da je sistem dosegljiv vsaj 99,999% časa. CockroachDB ne zagotavlja dostopnosti preko CAP teorema [13].



Slika 3.1: Arhitekturni pregled [25]

Plasti

Podatkovna baza CockroachDB je sestavljena ~~z~~ 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.

2. **Transakcijska plast:** Omogoča atomične spremembe nad večjim ~~številom~~ številom KV operacij.

3. **Porazdelitvena plast:** Predstavi replicirana KV območja kot eno entiteto.

4. **Replikacijska plast:** Konsistento 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 ANSI SQL standarda. Zunanje aplikacije komunicirajo preko PostgreSQL žičnega protokola. To omogoča enostavno povezavo med zunanjimi aplikacijami in kompatibilnost z obstoječimi gonilniki, orodji ter ORMji.

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 ~~z~~ 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 [5]. V podatkovni bazi `mydb` se nahaja tabela `customer`, katera ima pog primarnega ključa še dva stolpca `address` in `url`. ~~Zadnji zva~~ 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 ~~z~~ 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 ~~"namen"~~ <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 ~~z~~ s trenutno transakcijo odstrani zastavico ~~"namen"~~<intent>.

3. **Faza 3 (asinhrono):** Ko se transakcija konča, se vsem potrjenim pisalnim namenom odstrani zastavico ~~"namen"~~<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 ~~z~~ s transakcijskim zapisom in ga ustrezno upoštevajo.

KLJUČ	ČAS	VREDNOST
A<~~namen~~intent>	500	~~"nepotrjena vrednost "~~nepotrjena vrednost
A	400	~~"trenutna vrednost "~~trenutna vrednost
A	322	~~stara vrednost "~~stara vrednost
A	50	~~"prvotna vrednost "~~prvotna vrednost
B	100	~~"vrednost B "~~vrednost B

Tabela 3.2: Primer MVCC shrambe z pisalnim namenom na ključu A [26].

Podatkovna baza CockroachDB privzeto podpira najvišji standardni ~~ANSI~~ 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 ~~katerega koli~~ kateregakoli vozlišča. CockroachDB shranjuje podatke v urejeno vrsto tipa ključ-vrednost. Ta podatkovna ~~struktura~~ 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 ~~z~~ s privzetimi nastavitvami lahko naslovi do $2^{(36+26)}B = 4EB$ 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 dostopnost).

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, ~~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.

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.

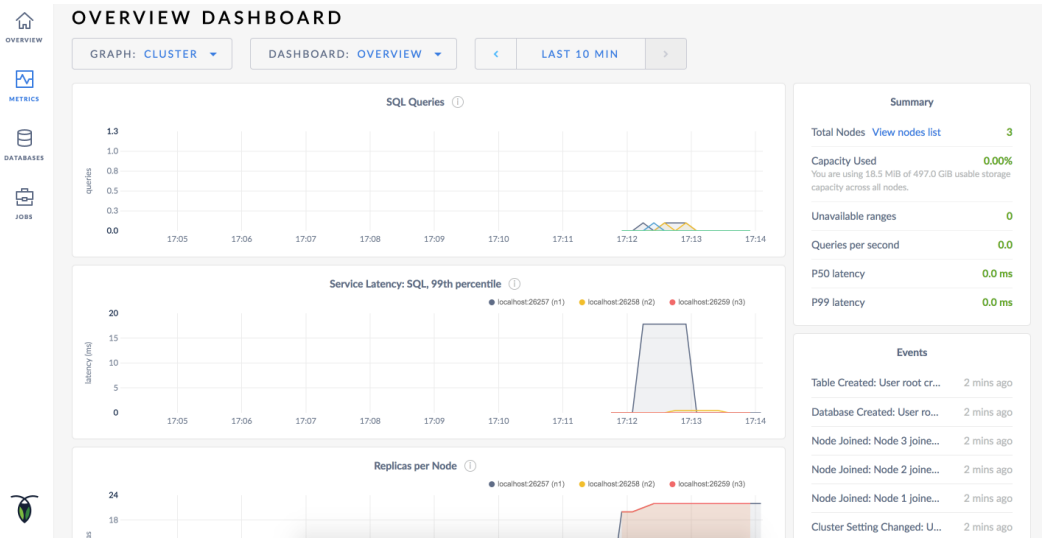
3.2.1 Enostavnost

Podatkovna baza CockroachDB strmi k enostavnosti upravljanja in ~~vzdrževanja~~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 odprto kodno rešitev Prometheus, katera nam omogoča shrambo, obdelavo, vizualizacije in obveščanje nad časovnimi vrstami.



Slika 3.2: Primer spletnega administratorski vmesnika.

3.2.3 SQL

CockroachDB, kakor večina relacijskih podatkovnih baz, podpira podmnožico ANSI SQL standarda. Poleg tega implementira nekatere najpogostejše ~~razširitve~~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 [30].

Podatkovni tipi

CockroachDB podpira podatkovne tipe kot so `ARRAY`, `BOOL`, `BOOL`, `BYTES`, `COLLATE` (podobno kot `STRING` vendar upošteva specifike jezika), `DATE`, ~~`DECIMAL`~~`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 ~~planirani~~ načrtovani za naslednje verzije.

Transakcije

CockroachDB podpira ACID transakcije, katere so lahko distribuirane preko celotne gruče. Privzeto transakcije uporabljajo najvišji izolacijski nivo ~~SERIALIZABLE~~SERIALIZABLE, omogoča pa še zastareli izolacijski nivo ~~SNAPSHOT~~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~~. 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 funkcije (funkcije katere se izvedejo nad podmnožico, tipično ob uporabi `GROUP BY`).

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

3.2.4 Poslovna licenca

CockroachDB je odprto kodna 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 na nivoju ene vrstice,
- vizualizacijo geografsko distribuirane gruče na zemljevidu,
- nadzor dostopa na nivoju skupin
- ~~in~~ in
- distribuirano kreiranje in obnova varnostnih kopij.

3.2.5 Podprta orodja, gonilniki in ORMji in skupnost

Podatkovna baza CockroachDB implementira standardni ANSI SQL, 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 ORMjev. Za te CockroachLabs v času pisanja zagotavlja le beta podporo [29].

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 [28] . 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

~~Drivers ORMs: https://github. com/cockroachdb/cockroach/issues/25468~~
Izvirna koda podatkovne baze CockroachDB je javno dostopna preko GitHub repozitorija `cockroachdb/cockraoch` [7]. 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 [4] (dokumentacija, form, blog, mediji), Gitter kanal [15] in Docker repozitorij [8].

~~DB Visualizers: https://github.com/cockroachdb/cockroach/issues/25467~~

~~skupnost, sam sem uporabil pgweb in gorm~~

Poglavje 4

Primerjalna analiza zmogljivosti CockroachDB z Citus

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 Postgres ~~(z~~ z nameščeno razširitvijo Citus (v nadeljevanju samo Citus).

Za primerjavo z ~~Postgres~~ Citus sem se odločil zaradi lažje izvedbe ter bolj primerljivih rezultatov~~, saj je CockroachDB v nekaterih vidikih zelo podoben Postgres podatkovni bazi~~ . Obe podatkovni bazi CockroachDB in Citus sta po nekaterih lastnostih zelo podobeni. Obe podatkovni bazi implementirat ANSI SQL ter komunicirata preko PostgreSQL žičnega protokola.

Za orodje YCSB sem se odločil, ~~ker~~ saj podpira vmesnik JDBC, katerega podpirata tudi obe ~~bazi~~podatkovni bazi poleg tega pa je enostaven za uporabo. Orodje YCSB sem bolj podrobno opisal v poglavju 4.3. Ob iskanju najprimernejšega orodja sem pregledal naslednja orodja:

- **TPC:**
  TPC (angl. transaction processing performance council) [33] je neprofitna 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 pozan 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 CockraochDB še ne obstaja.

- **pgbench:**
  Je enostavno orodje namenjeno Postgres 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 Postgres 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~~ postavill naslednje hipoteze:

1. CockroachDB bo na enem vozlišču nekoliko počasnejši od Postgres 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 [10].

Vozlišča so med seboj povezana preko gigabitnega Ethernet omrežja v Docker Swarm [31] 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~~ PROCESOR	~~pomnilnik~~ POMNILNIK	~~trdi disk~~ 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 4.4.

4.2.2 Strežnik

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

4.3 YCSB

~~katere so še ostale metode kater sem pregledal (pgBench, TPC-C, CRDB tpcc, JMeter,...)~~

YCSB [2] je razširljiva in odprto kodna 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 Postgres podatkovni bazi. YCSB nudi šest enostavnih predefiniranih tipov obremenitev [34] označenih z A, B, C, D, E in F.

4.4 Postopek testiranja

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 [19]. V tem poglavju bom opisal vse korake kateri so bili potrebni za izvedbo meritev.

4.4.1 Namestitev programske opreme

Na vozlišču `n0` sem namestil naslednjo programsko opremo:

- YCSB (verzija 0.12.0) [2] je orodje namenjeno za izvedbo zmogljivostne analize ter primerjavo med različnimi podatkovnimi bazami. Samo orodje sem bolj natančno opisal v poglavju 4.3.

- Ansible (verzija 2.5.0) [1] je orodje za avtomatizacijo, uporabil sem ga zaradi lažjega konfiguriranja gruče preko SSH povezave.

- Go (verzija 1.10.1) [32] je programski jezik, v katerem je napisan program za avtomatizacijo testov.

4.4.2 Priprava podatkov

Program za avtomatizacijo predpostavlja, da imajo vsa vozlišča na točno določeni lokaciji pripravljene podatke za obnovo podatkovne baze. Pred vsakim testom se podatki skopirajo 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` [19].

~~Postgres z razširitvijo~~ Citus

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

2. Na primarnem 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 primarnem 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. Varno sem ustavil bazo ter naredil varnostno kopijo podatkov na disku.

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

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

7. Nato sem na primarnem vozlišču `n2` povezal obe sekunarni vozlišči `n1` ter `n3` in kreiral distribuirano 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. Varno sem ustavil bazo ter naredil varnostno kopijo podatkov 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 podatkov na disku.

4.4.3 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 4.4.2. Program v grobem za vsako kombinacijo parametrov izvede naslednje koraka:

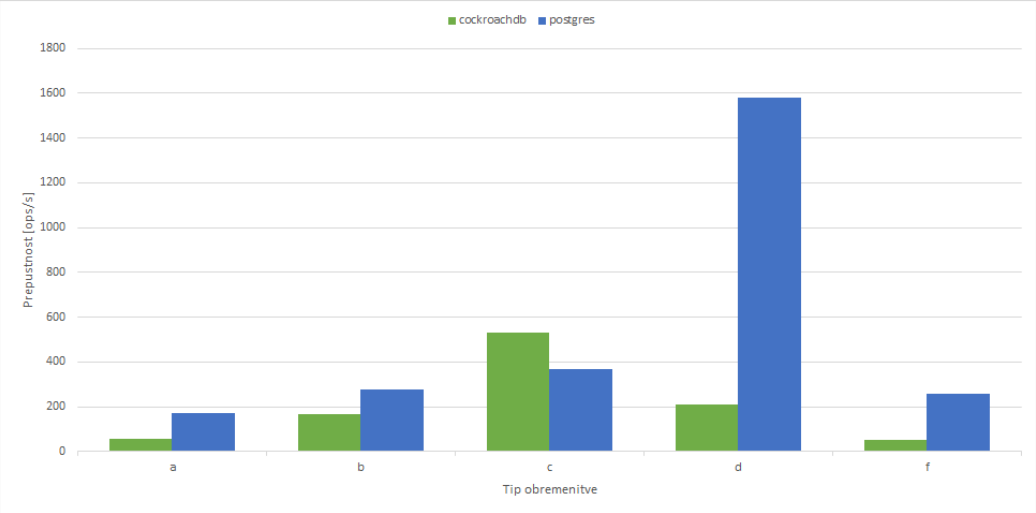
1. Obnovi v naprej definirane podatke (`cp -a`)
2. Zažene podatkovno bazo (`docker stack up`)
3. Izvede YCSB test (`ycsb run jdbc ...`)
4. Zabeleži rezultat v CSV datoteko
5. Ustavi podatkovno bazo (`docker stack rm`)
6. Počisti podatke (`rm -rf`)

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 [34], 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.

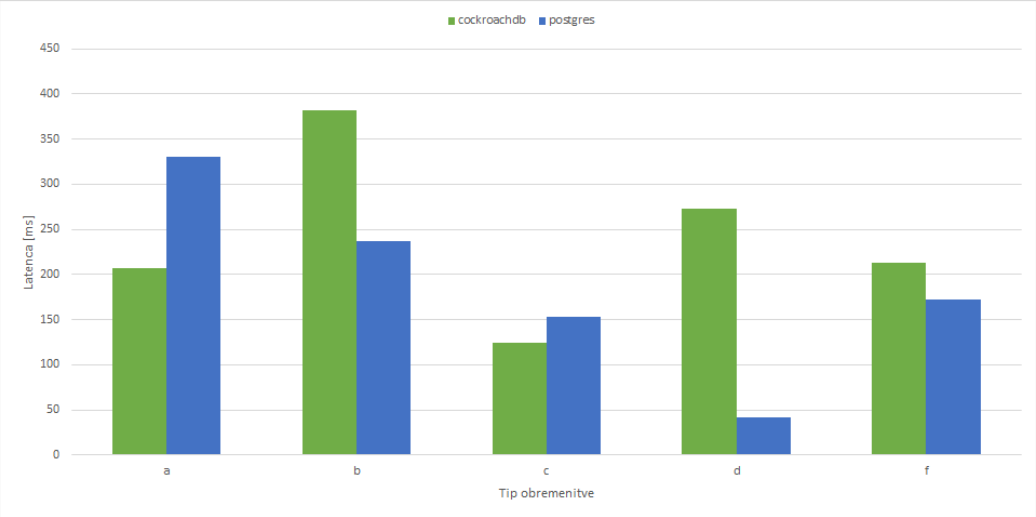
Zaradi morebitnih odstopanj sem vse teste ponovil trikrat.

4.5 Rezultati

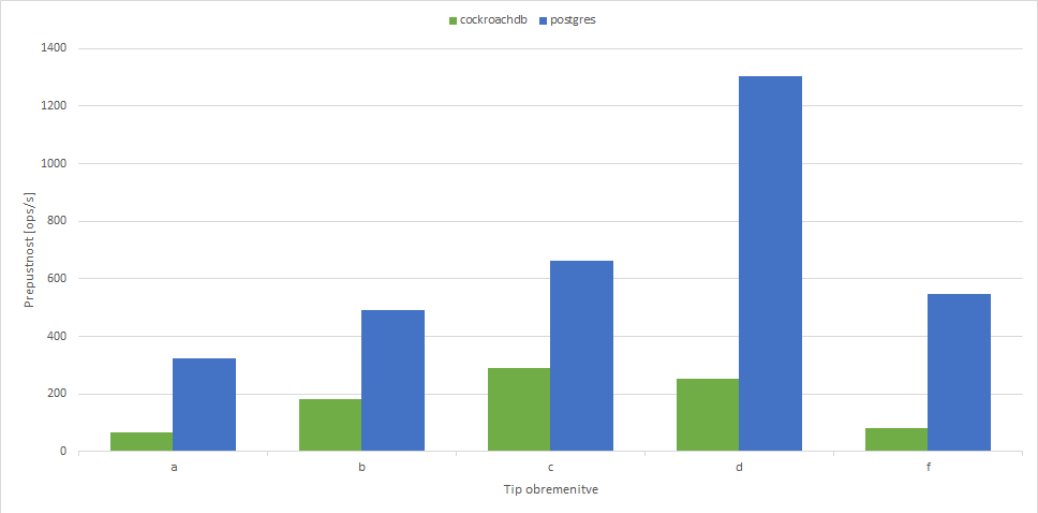
Vsi rezultati so na voljo na GitHub repozitoriju `matjazmav/diploma-ycsb` [19]. V mapi `results` se nahajajo datoteke z neobdelani podatki. Analizo rezultatov pa sem izvedel v excel datoteki `results.xlsx`. Po analizi sem prišel do spodnjih rezultatov.



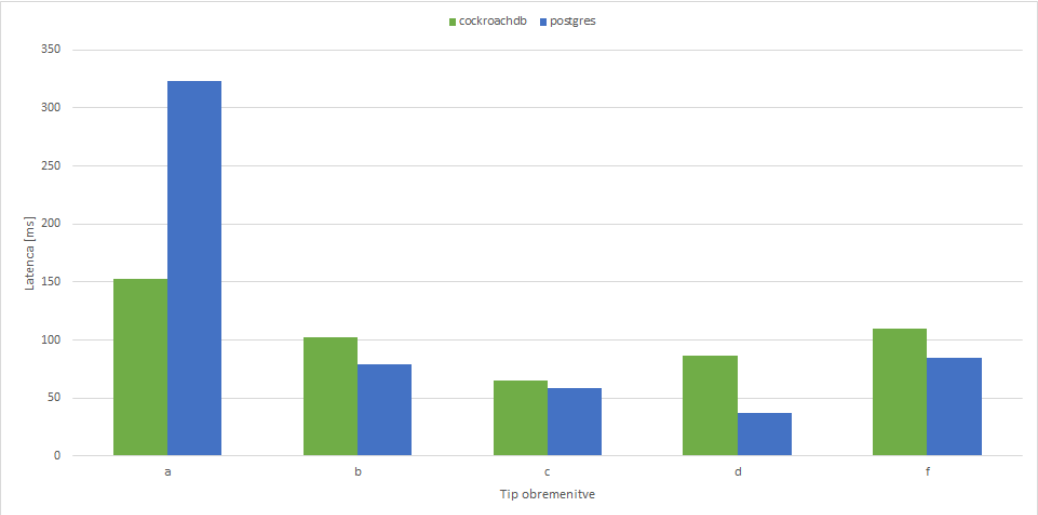
Slika 4.1: Primerjava maksimalne prepustnosti glede na tip obremenitve pri enem vozliščih.



Slika 4.2: Latenca pri maksimalni obremenitvi glede na tip obremenitve pri enem vozliščih.



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



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

4.6 Ugotovitve

Zmogljivostna analiza je pokazala, da se ~~Postgres z Citus razširitvijo v~~ Citus pri večini ~~primerov~~ obremenitev, katere sem testiral, odziva bolje kot CockroachDB.

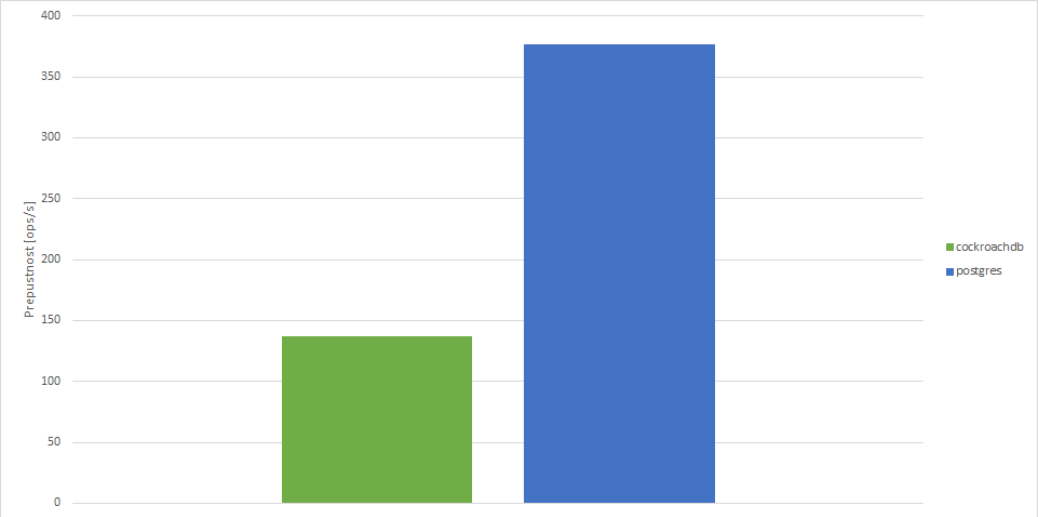
Izjema je bila latenca pri obremenitev tipa A, ker je CockroachDB dosegel približno pol manjšo latenco. Kljub temu pa je bila prepustnost bistveno nižja.

Druga izjema se je pojavila pri obremenitvi tipa C na enem vozlišču, kjer je CockroachDB dosegel približno eno četrtino večjo prepustnost kot Postgres.

4.6.1 Hipoteza 1

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

Razlog za to hipotezo je, da je podatkovna baza Postgres že dolgo na trgu [24] 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 [25], druga verzija (2.0.0) pa je bila objavljena aprila 2018. Vsaka verzija je dodala veliko novih funkcionalnosti.



Slika 4.5: Primerjava povprečne prepustnosti na enem vozlišču.

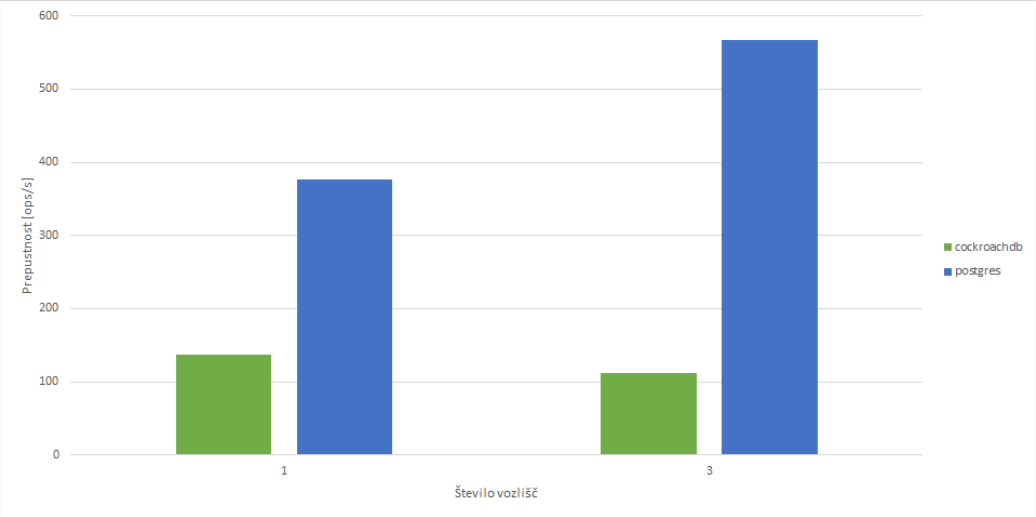
Hipotezo sem potrdil, kar kaže tudi zgornja slika 4.5. CockroachDB v primerjavi z Postgres dosega približno 2,7 krat nižjo prepustnost.

Podobno zmogljivostno primerjavo so decembra 2017 izvedli na zasebni raziskovalni univerzi v Bruslju [25]. Z orodjem YCSB so primerjali obremenitve tipa A, B in C. Ugotovili so, da se CockroachDB odziva približno 10 krat slabše kakor Postgres.

4.6.2 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. [5].



Slika 4.6: Primerjava povprečne prepustnosti ob skaliranju baze na tri vozlišča.

To hipotezo sem ovrgel. Meritve so pokazale, da CockroachDB doseže celo slabšo prepustnost na treh vozliščih. To je prikazano na zgornji sliki 4.6.

Razlog za tak rezultat naj bi bila primerjava med konfiguracijo na enem in treh vozliščih [35]. 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.

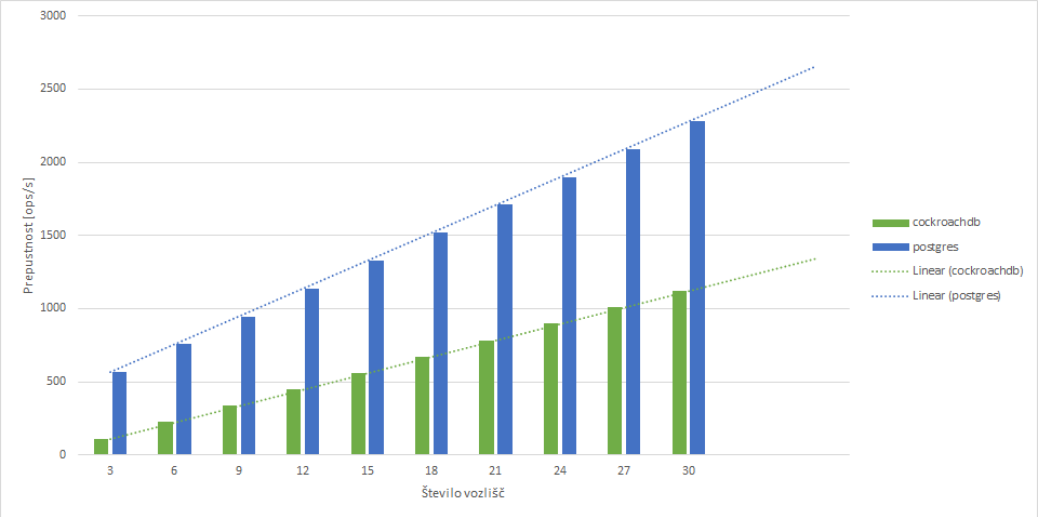
Rezultati neuradne TPC-C zmogljivostne analize, katero so izvedli v CockroachLabs, kažejo, da je CockroachDB linearno skalabilen [6]. Ob skaliranju z 3 na 30 vozlišč ter ob sorazmernem povečanju obremenitve so dosegli tudi sorazmerno večjo prepustnost.

Prišel sem do ugotovitve, da je CockroachDB linearno skalabilna podatkovna baza, če velja naslednje:

1. Obremenitev mora biti enakomerno porazdeljena med vsa vozlišča.

2. Faktor replikacije mora ostati 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 Postgres konfiguraciji vsi odjemalci direktno povezujejo na eno vozlišče (koordinatorja), to vozlišče je ozko grlo, saj bo ob večanjem v neki točki prišlo do zasičenosti resursev [3].



Slika 4.7: Optimistična ocena rasti povprečne prepustnosti.

4.7 ~~Pregled primerjalnih analiz drugih avtorjev~~

Poglavje 5

Sklepne ugotovitve

Literatura

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

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

[3] 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].

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

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

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

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

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

[9] 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.

[10] 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].

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

[12] 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.

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

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

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

[16] 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.

[17] High-Availability Database (HA DB). Dosegljivo: https://raima.com/rdme-high-availability-database, Jun 2018. [Dostopano: 30. 6. 2018].

[18] 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].

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

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

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

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

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

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

[25] 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].

[26] Serializable, Lockless, Distributed: Isolation in CockroachDB. Dosegljivo: https://www.cockroachlabs.com/blog/serializable-

lockless-distributed-isolation-cockroachdb. [Dostopano: 24. 7. 2018].

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

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

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

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

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

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

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

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

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

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