Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jernej Sila

# Primerjava NewSQL podatkovnih baz NuoDB in VoltDB

diplomsko delo

univerzitetni študijski program računalništva in informatike

Mentor: prof. dr. Marko Bajec

Ljubljana 2015

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Današnje poslovne aplikacije pogosto zahtevajo procesiranje ogromne količine podatkov, kjer pa se tradicionalni sistemi za upravljanje s podatkovnimi bazami zaradi svoje stare arhitekture ne izkažejo za učinkovite. Za potrebe večje dostopnosti so se zato pojavile druge rešitve, t.i. NoSQL podatkovne baze. Te so bistveno učinkovitejše pri procesiranju velikih količin podatkov, vendar s to slabostjo, da ne implementirajo standardiziranega poizvedovalnega jezika SQL in ne zagotavljajo varnih transakcij (ACID). Kot ena od novih alternativ NoSQL bazam so se zato pojavile NewSQL baze, ki poleg NoSQL razširljivosti omogočajo tudi zagotavljanje skladnosti podatkov (nudijo ACID transakcije in omogočajo poizvedovanje z uporabo SQL jezika).

V diplomskem delu raziščite, zakaj tradicionalni sistemi za obvladovanje podatkovnih baz niso primerni za procesiranje velikih količin podatkov, na splošno opiše NewSQL podatkovne baze ter nato podrobneje zgradbo dveh takih baz - NuoDB in VoltDB. Omenjeni bazi primerjajte s popularno podatkovno bazo MySQL ter nato na praktičnih primerih opazuje njuni zmogljivosti.

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jernej Sila sem avtor diplomskega dela z naslovom:

*Primerjava NewSQL podatkovnih baz NuoDB in VoltDB*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Marka Bajca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 7. marca 2015 Jernej Sila

*Zahvaljujem se prof. dr. Marku Bajcu, ki me je vzel pod mentorstvo in za vse njegove napotke, komentarje in popravke pri izdelavi tega diplomskega dela. Zahvaljujem se tudi Laboratoriju za podatkovne tehnologije za pomoč pri tehnični izvedbi dela.*

Mojim domačim.

# Kazalo

# Povzetek

V diplomski nalogi sta predstavljena dva sistema za upravljanje s podatkovno bazo (NuoDB in VoltDB), ki spadata med tako imenovane podatkovne baze NewSQL. Gre za razred relacijskih in porazdeljenih podatkovnih baz, ki odpravlja režije, ki pestijo tradicionalne relacijske podatkovne baze pri doseganju razširljivosti in visoke prepustnosti podatkov. Opisane so relacijske baze na splošno, definicija razširljivost in načini za doseg le-te pri podatkovnih bazah, režije pri tradicionalnih relacijskih bazah in obstoječe rešitve za doseg razširljivosti. Poleg opisa obeh sistemov je podana tudi primerjava med njima ter popularno relacijsko bazo MySQL, bil pa je opravljen tudi preizkus zmogljivosti obeh sistemov na večjem številu računalnikov, ki strežejo podatkovno bazo z namenom opazovanja prepustnosti podatkov.

**Ključne besede:** razširljivost, NuoDB, podatkovne baze, podatkovne baze NewSQL, SUPB, VoltDB.

# Abstract

Thesis presents two database management systems (VoltDB and NuoDB) which fall under so-called NewSQL databases. They are class of relational and distributed databases which removes overheads that plague traditional relational databases in order to achieve scalable performance and higher data throughput. Here are described relational databases in general, definition of scalability and number of ways to achieve it in databases, overheads of traditional relational databases and existing market solutions. Besides describing these two NewSQL databases, they are compared with MySQL, the most popular relational database and benchmarked on more then one computer in order to observe their throughput.

**Keywords:** databases, DBMS, NuoDB, NewSQL databases, scalability, VoltDB.

# Poglavje 1

# Uvod

Razmah interneta v zadnjih letih je z razvojem spletnih aplikacij (Facebook, Twitter) povzročil, da se dnevno generirajo velike količine podatkov, saj ljudje do njih dostopajo preko računalnikov, prenosnikov, pametnih telefonov skozi ves dan. Pogosto je za daljši odzivni čas aplikacije kriv obstoječ sistem za upravljanje relacijsko podatkovno bazo (v nadaljevanju relacijski SUPB), ki ni zmožen streči velikega števila zahtev naenkrat. Današnji najbolj uveljavljen način za rešitev tega problema je uporaba SUPB-jev, ki upravljajo z nerelacijskimi bazami. Ti se arhitekturno razlikujejo od relacijskih SUPB-jev, saj za procesiranje namesto virov enega boljšega računalnika izkoriščajo vire večih računalnikov, povezanih v omrežje. Ti SUPB-ji za poizvedovanje po podatkih ne uporabljajo uveljavljenega jezika za poizvedovanje po podatkih in ne omogočajo varnih transakcij. Problem teh je tudi, da skrb za varnost in pravilnost podatkov nalagajo razvijalcem aplikacij, kar pogosto presega njihovo znanje in za podjetje predstavlja nepotreben dodaten strošek.

V zadnjih letih je na trg začel prihajati nov tip podatkovnih baz (imenovan NewSQL), ki še naprej izkorišča prednosti poizvedovalnega jezika ter relacijskih SUPB-jev.

Cilj naloge je bil opisati podatkovne baze na splošno, pojem razširljivosti SUPB-ja, predstaviti nekaj teh novih podatkovnih baz NewSQL, jih primerjati z najbolj znanim relacijskim SUPB-jem MySQL in opazovati njihovo

razširljivost pri večjem naboru strežnikov, ki strežejo podatkovno bazo.

# Poglavje 2

# Podatkovne baze

## 2.1 SUPB

SUPB je programska oprema za obvladovanje velikih količin podatkov. Med njegove naloge spada skrb za ustvarjanje, vzdrževanje in shranjevanje podatkov, nadzor nad dostopom podatkov, varnost in konsistentnost podatkov, obnova podatkovne baze ob nesrečah.

Prvi SUPB-ji so se pojavili v zgodnjih 60. letih prejšnjega stoletja zaradi omejitev shranjevanja in dostopa do podatkov v aplikacijah in datotekah. Nekaj takih omejitev [1, stran 14-17]:

1. Redundanca (podvojenost) in neskladnost podatkov: Aplikacije dostopajo do svojih podatkov in lahko uporabljajo enak podatek, ki je shranjen na večih mestih in verjetno tudi v drugačni obliki. Redundanca podatkov zahteva več prostora na disku ter spomina, več različnih kopij podatka pa ni nujno skladnih.

2. Dostop do podatkov: Datotečni sistemi ne omogočajo dostopa do podatkov na najbolj učinkovit način. Za izboljšanje učinkovitosti celotnega operacijskega sistema slednji omogočajo sočasen dostop večih uporabnikov do iste datoteke. Sočasne spremembe lahko vodijo do neskladnosti podatkov.

3. Varnost in integriteta podatkov: Nekateri podatki v podatkovni bazi niso za vsakega uporabnika. Prav tako morajo biti podatki shranjeni v točno določeni obliki ter imeti točno določeno zalogo vrednosti/domeno. Pisanje takih omejitev je težavno, še posebej ob dodajanju novih omejitev.

4. Sistemske nesreče: Ker podatkovno bazo poganja strežnik, ki je tako kot vsaka druga električna naprava podvžena izpadom energije, je zato pomembno, da se podatkovna baza po odpravi napake vrne v stanje pred nesrečo.

Z uporabo SUPB-ja se pridobi na lažjem in učinkovitejšem dostopu do podatkov, saj se ti nahajajo na enem mestu in jih je zato tudi lažje upravljati. Sočasni dostop zagotavlja tako, da izgleda, kot da bi do podatkov dostopal le en uporabnik. Z njegovo uporabo se skrajša tudi čas razvoja aplikacij, saj ni več potrebno skrbeti za pravilnost podatkov, v prihodnosti pa se ga lahko zaradi podatkovne neodvisnosti SUPB-ja zamenja na dosti lažji način.

SUPB je del sistema podatkovne baze, ki ga v celoti sestavlja pet komponent: strojna oprema, programska oprema, podatki, ljudje in postopki [1, stran 18-20]:

1. Strojna oprema: Sestavljajo jo množica med seboj povezanih fizičnih, pomnilnih, omrežnih naprav, kjer se nahaja podatkovna baza in kjer poteka interna komunikacija: en ali več računalnikov, diski, terminali, usmerjevalniki, kabli, ipd. Za želeno delovanje SUPB-ja so najbolj pomembni ustrezen procesor, spomin in prostor na disku.

2. Programska oprema: Poleg SUPB-ja jo sestavljajo še operacijski sistem, ki ga poganja in aplikacije, ki uporabljajo SUPB za dostop do podatkov.

3. Podatki: Brez podatkov ni podatkovne baze. Podatki se v bazi zbirajo glede na njeno predhodno definirano strukturo.

4. Ljudje: Vsi uporabniki, ki so v stiku s podatkovno bazo. Te se lahko razdeli v 5 glavnih skupin: sistemski administratorji (skrbijo za sistem), administratorji SUPB-ja (skrbijo za SUPB in njegovo delovanje), načrtovalci podatkovne baze (skrbijo za strukturo baze), sistemski analitiki ter programerji (skrbijo za implementacijo načrta baze) ter končni uporabniki (ti običajno dostopajo do baze preko aplikacij).

5. Postopki: So navodila in pravila, ki narekujejo načrt in namen podatkovne baze.

Njegova tipična zgradba je trinivojska: aplikacijska, logična in fizična plast. Aplikacijska plast predstavlja vmesnik v sistem za vse uporabnike sistema. Na logični plasti se izvaja procesiranje poizvedb, upravlja se s transakcijami, obnovo in shranjevanjem podatkov. Fizična plast rokuje s podatki, shranjenimi na disku v določenih podatkovnih strukturah (B-drevesa, kopice).

### 2.1.1 Porazdeljeni SUPB

Porazdeljeni SUPB je centralizirana aplikacija, ki upravlja s porazdeljeno podatkovno bazo na način, kot da bi slednja bila shranjena na enem strežniku. Porazdeljena baza je zbirka podatkov, ki so lahko shranjeni na večih strežnikih na isti fizični lokaciji ali pa razpršeni v omrežju med seboj povezanih strežnikov. Tak sistem sinhronizira vse podatke periodično in v primerih, ko več uporabnikov dostopa do istih podatkov, zagotovi da se spreminjanje in izbris podatkov na eni lokaciji samodejno poznajo pri podatkih, shranjenih nekje drugje. Porazdeljeni (razkosani ali podvojeni) niso nujno le podatki, pač pa tudi obdelava, povezana z njimi. Pojavili so se zaradi potreb po hitrejšem odgovoru poizvedb, visokih cen vzdrževanja enega/glavnega večjega strežnika, potreb po boljši zanesljivosti in razširljivosti [1, stran 481]. K razvoju porazdeljenih SUPB je v veliki meri vplivalo širjenje interneta ter napredek v tehnologiji, pri katerem se računalniške komponente ves čas cenijo.

Porazdeljeni SUPB funkcionalno nadgrajuje centraliziran SUPB tako, da nudi oziroma razširja [2, stran 703] [1, stran 486-487]:

- komunikacijske storitve za nudenje dostopa do oddaljenih vozlišč in prenosa poizvedb in podatkov med vozlišči po omrežju,
- sistemski katalog za hranjenje podrobnosti o porazdeljenosti podatkov,
- porazdeljeno obdelavo podatkov vključno z optimizatorjem poizvedb (ugotoviti najboljšo strategijo za dostop do podatkov),
- varnost za vzdrževanje uporabniških privilegijev nad porazdeljenimi podatki,
- nadzor sočasnosti za vzdrževanje konsistentnosti porazdeljenih podatkov,
- storitve obnove, kjer se v zakup vzamejo odpoved kateregakoli vozlišča v omrežju in izpad komunikacijskih povezav.
- preslikavo za ugotavljanje lokacije fragmentov podatkov na lokalnem in oddaljenem strežniku.

Arhitekturno so porazdeljeni SUPB-ji dveh vrst: homogeni in heterogeni [2, stran 697]. Homogeni na vseh strežnikih v gruči (med sabo povezana skupina strežniki, ki strežejo isto podatkovno bazo) uporabljajo enako programsko in strojno opremo, kar pa ne velja za heterogene, ki so pogosto rezultat združitve že obstoječih SUPB-jev v celoto.

Pomembna lastnost porazdeljenega SUPB-ja je uporabnikom zagotavljati naslednje transparentnosti (supb, 491):

- Transparentnost porazdeljenosti podatkov: Uporabnik ne ve, kako so podatki razdeljeni, kje se nahaja iskani podatek in ali obstaja več enakih kopij podatkov.

- Transparentnost heterogenosti: Uporabnik ne ve, da uporablja drug SUPB pri dostopu do podatkov na daljavo. Uporabnik uporablja enak jezik, kot pri običajnem dostopu do podatkov in po potrebi porazdeljeni SUPB izvede prevod poizvedbe.

- Transparentnost transakcij: Porazdeljeni SUPB zagotavlja, da se istočasne transakcije med sabo ne prepisujejo ter obnovo podatkovne baze.

- Transparentnost učinkovitosti: Narekuje, da ima porazdeljeni SUPB primerljivo hitrost v primerjavi s centraliziranim SUPB-jem ter da sistem poišče cenovno najboljši način dostopa do podatkov. Optimizatorji poizvedb se uporabljajo za zmanjšanje časa odgovora.

- Transparentnost odpovedi: Ko odpove strežnik v gruči, bo podatkovna baza delovala naprej. Funkcije tega strežnika prevzame drug strežnik v gruči.

Poleg že omenjenih prednosti takega sistema tak sistem ponuja boljšo dostopnost, učinkovitost, deljeno in lokalno avtonomijo, integracijo, ponuja vpogled v strukturo organizacije ter ji omogoča konkurenčnost na trgu. Njegova največja slabost pa je v tem, da gre za zapleteno arhitekturo, ki poveča stroške vzdrževanja (več strežnikov, omrežna komunikacija), varnost je potrebno nuditi tako za podatke kot pri komunikaciji po omrežju, težji nadzor nad integriteto podatkov zaradi povečane komunikacije in obdelave, pomanjkanje standardov in izkušenj (to gre pripisati dejstvu, da so se taki sistemi razširili šele v zadnjih 2 desetletjih) [2, stran 694-696].

## 2.2 Lastnosti varnih transakcij

Z izrazom transakcija označujemo skupek sprememb, ki z vidika SUPB-ja predstavljajo eno enoto dela. Pomembna naloga SUPB-ja je, da zagotavlja varne transakcije - te morajo biti atomarne, konsistentne, izolirane in trajne (te lastnosti se označuje s kratico ACID). S tem je zagotavljena zanesljivost

transakcijske podatkovne baze. Lastnosti ACID predstavljajo enega najbolj pomembnih konceptov v teoriji podatkovnih baz. Pravila so [1, stran 419] [3]:

- Atomarnost: Narekuje, da se morajo vsi ukazi transakcije izvesti uspešno, da transakcija uspe. Če se neuspešno izvede samo en del transakcije, potem celotna transakcija propade (po principu vse ali nič).
- Konsistentnost: Zagotavlja, da je podatkovna baza vedno v veljavnem stanju. Tako se ob neuspešni transakciji razveljavi vse dotedanje delo te transakcije in sistem bo v stanju pred transakcijo. Ob uspešno izvedeni transakciji pa bo podatkovna baza samodejno v veljavnem stanju.
- Izolacija: Med procesiranjem dveh ali več transakcij naenkrat druge transakcije ne morejo dostopati in uporabljati vmesnih podatkov prve transakcije. V primeru sočasnih transakcij vsaka transakcija zase misli, da ima izključen dostop do uporabe podatkovne baze in se ne zaveda drugih transakcij.
- Trajnost: Potrjene spremembe transakcij so trajne. S tem se preprečuje izguba podatkov ob nedejavnosti sistema ter sistemskih napakah.

Nekaj pomembnih mehanizmov, s katerimi SUPB zagotavlja varne transakcije [3]:

- Zapisovanje v dnevnik (atomarnost, trajnost): Vse spremembe so najprej zapisane v dnevnik, preden so izvedene. To omogoča povrnitev stanja podatkovne baze v primeru sistemskega izpada.
- Zaklepanja (izolacija): Deluje tako, da transakcija označi, katere podatke uporablja ter SUPB tako prepreči drugim transakcijam uporabo teh podatkov. Zaklepanje se opravi, preden transakcija začne s procesiranjem, zaklenejo pa se tudi podatki, ki jih transakcija samo bere.
- Nadzor sočasnosti z več različicami (izolacija): Druga rešitev proti zaklepanju, ki omogoča, da pisalne transakcije ne blokirajo bralnih in

obratno. Če prva transakcija zahteva podatke, ki jih uporablja že neka druga transakcija, potem prva transakcija dobi podatke, ki so obstajali, preden se je druga transakcija začela. Nova sprememba namreč ne prepriše podatkov, temveč ustvari njihovo novo različico.

### 2.2.1 Transakcije v porazdeljenih podatkovnih bazah

Ker pri porazdeljenih podatkovnih bazah vsak strežnik ne vsebuje nujno vseh podatkov, transakcije pogosto tečejo na večih strežnikih. Ker lahko vsak strežnik preneha delovati kadarkoli, se za zagotavljanje varnih transakcij uporablja dvofazno potrjevanje, ki sestoji iz zahtev po potrditvi ter potrjevanju [3].

Pri zahtevi za potrditev eno vozlišče (koordinator) povpraša druga vozlišča (udeležence), ali naj se transakcija potrdi. Udeleženci koordinatorju sporočijo odgovor glede na to, ali se je del transakcije pri njih izvedel uspešno ali ne. Če so vsi udeleženci javili uspešnost izvedbe njihovega dela transakcij, potem koordinator transakcijo potrdi. V nasprotnem primeru jo ovrže. V obeh primerih sporoči udeležencem rezultat, s čimer omogoči, da morebitne spremembe uveljavijo tudi ostali udeleženci.

## 2.3 Relacijski podatkovni model

S podatkovnim modelom se opiše, katere podatke se hrani v podatkovni bazi in kako se ti povezujejo med sabo [1, stran 30]. Je predstavitev realnega sveta, kot ga vidi uporabnik, pri čemer se upošteva samo tiste podrobnosti, ki so pomembne. Podatki so lahko v bazi opisani na treh ravneh – s konceptualno ali logično shemo (entitete in povezave so opisane na način, ki je prilagojen SUPB-ju in ki ne vsebuje podrobnosti o uporabljenih tehnologijah za rokovanje s podatki), fizično shemo (kako in kam se podatki dejansko shranjujejo) ter z zunanjo shemo (sestavljena iz pogledov - dostop do podatkov, ki je prilagojen določenim uporabnikom, pogledi so lahko skupek večih entitet in fizično ne obstajajo v podatkovni bazi).

S tem se doseže podatkovno neodvisnost. Lahko se spremeni konceptualna shema, zunanjo shemo pa prilagodi tako, da pogledi ostanejo nespremenjeni. Na podoben način konceptualna shema skrije spremembe, ki se naredijo na fizični shemi.

Obstaja več vrst podatkovnih modelov, najbolj znani so relacijski, hierarhični, mrežni in objektni. Izmed teh je najbolj razširjen relacijski model, saj je enostaven za razumevanje. Osnovni gradniki so relacije, ki se jih da predstavljati kot tabele s stolpci in vrsticami. Razvil ga je Edgar Codd leta 1970 in zaradi uspeha je ta model zamenjal starejše modele.

Relacijski podatkovni model za predstavitev podatkov uporablja naslednje koncepte:

- Relacija: Predstavljena kot tabela s stolpci in vrsticami
- Atribut: Poimenovan stolpec v relaciji.
- Domena: Zaloga vrednosti atributa.
- N-terica: Ena vrstica v relaciji.
- Stopnja relacije: Število atributov v relaciji.
- Številnost relacije: Število n-teric v relaciji.

V relacijski podatkovni bazi velja, da so relacije in atributi v njej enolično poimenovani (ni pa narobe, če dve relaciji vsebujeta atribut z istim imenom), vsaka celica vsebuje natanko eno atomarno vrednost, vrednosti istega atributa so iz iste domene, v relaciji ni dveh n-teric, ki bi si bili enaki. Vrstni red atributov in n-teric v relacijah ni pomemben.

## 2.4 Jezik SQL

Kot že omenjeno, SUPB uporablja različne mehanizme za upravljanje s podatki: kreiranje podatkovnih struktur, vzdrževanje in pridobivanje podatkov.

Kreiranje je omogočeno z jezikom za definicijo podatkov (ustvarjanja shem), vzdrževanje se izvaja z jezikom za delo s podatki (ustvarjanje, brisanje, spreminjanje zapisov), za pridobivanje pa se uporablja poizvedovalni jezik. Pri relacijskem SUPB-ju se za upravljanje s podatki danes uporablja strukturirani poizvedovalni jezik (angl, *structured query language*, v nadaljevanju SQL).

Zgodovina jezika SQL sega v leto 1978, ko je podjetje IBM razvilo System/R, ki je temeljil na relacijskem modelu Edgarda Codda, skupaj s poizvedovalnim jezikom SEQUEL (Structured English Query Language), predhodnikom jezika SQL. Naslednje leto podjetje Relational Software (danes Oracle) izda prvi relacijski SUPB z implementacijo jezika SQL. Leta 1982 IBM izda prvi komercialni relacijski SUPB z SQL-om, imenovan SQL/DS ter nato leta 1985 še DB2. Oba sistema sta se lahko izvajala le na računalnikih IBM, a je bil DB2 kasneje na voljo še za druge sisteme (kot na primer Unix in Windows) [4].

V 80. letih so je pojavilo še več SUPB-jev s podporo jeziku SQL, a je bila vsaka implementacija različna. To se je spremenilo leta 1989, ko je ANSI (Ameriški nacionalni inštitut za standarizacijo) izdal pravila za standarde pri poizvedovalnem jeziku v SUPB-ju. Ta pravila so se še nekajkrat dopolnila, nazadnje leta 2011. Vsak proizvajalec podatkovnih baz se je po svoje prilagajal tem pravilom, njihove implementacije jezika SQL pa so si postale podobnejše. Zaradi tega danes obstaja več dialektov jezika SQL [4].

Poizvedba SQL je sestavljena iz rezerviranih in uporabniško definiranih besed. Rezervirane besede so besede, s katerimi povemo, kaj želimo narediti z uporabniško definiranimi besedami (imena relacij, atributov, n-teric, indeksov, raznih omejitev) oziroma kaj izbrati, združiti, stakniti skupaj. Čeprav so z izjemo tekstovnih podatkov vse besede neobčutljive na velikost črk, pa se za boljšo berljivost rezervirane besede pišejo z velikimi črkami, ostale pa malimi črkami. Če je stavek SQL dolg, se jih piše tako, da je vsak sklop v novi vrstici in z levo poravnanimi zamiki.

# Poglavje 3

# Razširljivost relacijskih SUPB-jev

## 3.1 Definicija razširljivosti

Ilustrirana definicija razširljivosti (angl. *scalability*) se glasi: "Kako bo rešitev problema delovala, če problem povečamo" [5]. Pove, kako se zasnova programske opreme spopade s povečano potrebo po aplikaciji. Razširljiv sistem ne le zagotavlja, da bo učinkovitost aplikacije pri povečanem prometu sprejemljiva, ampak tudi zmanjša možnosti, da bo aplikacijo potrebno preoblikovati, ko se bo pojavil problem nerazširljivosti.

Ob povečani obremenitvi ter brez dodatne strojne opreme bi število procesiranih transakcij na enoto časa (prepustnost) aplikacije morala ostati enaka, odzivni čas pa bi se povečal le linearno [6, stran 27]. Obstajata dva načina, kako razširiti sistem, pri katerem se poskuša povečati prepustnost in zmanjšati odzivni čas: navpična razširljivost (angl. *scale-up*) in vodoravna razširljivost (angl. *scale-out*) [6, stran 28] [7].

**Navpična razširljivost** Razširljivost je dosežana tako, da se poveča sistemske vire strežnika. K strojni opremi aplikacije se doda zmogljivejše in večje računalniške komponente (procesor, spomin, disk). Za navpično razširljivost se štejejo tudi razne optimizacije tako na ravni strojne kot

programske opreme.

Slabost tega pristopa je v dragih zmogljivejših komponentah, ki so težje za vzdrževanje. Aplikacija je nedosegljiva v fazi same nadgradnje. Navpična razširljivost se ustavi, ko sistem vsebuje najboljše komponente, ki jih trg lahko ponudi.

**Vodoravna razširljivost** Pri tem načinu je razširljivost dosežena tako, da se strežnik, ki poganja aplikacijo, poveže z dodatnimi strežniki, ki prevzamejo del obremenitve aplikacije. S tem ko sistem postane porazdeljen, se omogoči vporedno izvajanje večih zahtev. Na voljo ima vire, ki predstavljajo skupek vseh virov strežnikov (vozlišč) v gruči. Pri tem je pomembno, da so vsa vozlišča v gruči enako obremenjene, saj investicija v nasprotnem primeru predstavlja nepotreben strošek.

Pri priklopu novih vozlišč je aplikacija dosegljiva. V nasprotju z navpično razširljivostjo z izpadom enega vozlišča aplikacija ne postane nedosegljiva, saj delo tega vozlišča prevzamejo druga vozlišča v gruči. Potreben je poseg v kodo aplikacije, saj je potrebno omogočiti porazdeljevanje zahtev med vozlišči.

## 3.2 Vodoravna razširljivost SUPB-jev

Vodoravna razširljivost poteka preko replikacije (angl. *replication*) podatkovne baze oziroma njene fragmentacije (angl. *partitioning*). Na kateri strežnik se naslovi zahtevo pa ponavadi skrbi uravnalec obremenitve.

### 3.2.1 Replikacija

Replikacija je proces kopiranja in vzdrževanja podatkovne baze na redundantnih strežnikih [8]. Obstajata dva modela replikacije: gospodar-suženj in gospodar-gospodar. Njuna glavna razlika je v omejitvah pisalnih operacij posameznega strežnika.

**Gospodar-suženj** Gre za najbolj preprost model replikacije, kjer en strežnik služi kot gospodar, drugi pa so sužnji. Na gospodarja lahko aplikacija piše in bere, na sužnju pa le bere. Po pisanju gospodar nato sam posodobi še svoje sužnje, da odražajo stanje podatkovne baze gospodarja in je podatkovna baza na vseh strežnikih konsistentna.

Ta model omogoča, da lahko aplikacija bere z večih strežnikov, a obstoj enega strežnika gospodarja predstavlja kritično točko odpovedi - v primeru njegove odpovedi aplikacija svojih podatkov ne more spreminjati.

**Gospodar-gospodar** Je nadgradnja modela gospodar-suženj, saj obstoj večih strežnikov gospodarjev ne predstavlja kritične točke odpovedi in kjer se posodabljanje ne izvršuje samo v eno smer (od gospodarja proti sužnju). Obstaja pa možnost konflikta oziroma prepisovanje podatkov (na primer oba gospodarja v istem času vnašata zapis z isto vrednostjo ID-ja). Rešitev je več in je odvisna od potreb aplikacije: vsak gospodar je zadolžen za spreminjanje točno določene tabele ali pa se vsakemu gospodarju dodeli enoličen razpon ID vrednosti za primarni ključ, ki jih lahko uporabi. Druga slabost je težavno zagotavljanje varnih transakcij in taki sistemi po večini izgubijo to lastnost.

Oba modela sama po sebi ne zagotavljata konsistentnosti podatkovne baze na vseh strežnikih v vsakem trenutku, saj posodobitev ni trenutna, ampak je zaradi različnih hitrosti strežnikov različna. Če na primer neka transakcija bere podatke delček časa po tem, ko druga transakcija spreminja podatke, tako ni nujno, da prva transakcija prebere najbolj sveže podatke. Ta problem se lahko reši s sinhonsko replikacijo (vsa pisanja so na vseh strežnikih izvedena pred branjem).

### 3.2.2 Fragmentacija

Pri fragmentaciji se objekti podatkovne baze (na primer tabele, indeksi) razdelijo na več manjših delov, kjer vsak strežnik dobi en del. Fragmentacija v SUPB-ju lahko poteka na tri načina [9]:

1. Navpična fragmentacija: Vsak strežnik vsebuje del stolpcev celotne tabele. Fragmentacija je podobna normalizaciji podatkovne baze, le da se stolpci (atributi) poleg tega, da se nahajajo v drugi tabeli, nahajajo še na drugem strežniku.

2. Vodoravna fragmentacija: Vsak strežnik vsebuje del vrstic tabele. Tako na primer en fragment vsebuje podatke o strankah s priimkom od A do K, drug fragment pa podatke o ostalih strankah.

3. Kombinacija obeh zgornjih načinov fragmentacije.

Fragmentacija mora biti izveden tako, da transakcije v prihodnosti čim manj poizvedujejo po podatkih, ki se nahajajo na večih fragmentih, saj v nasprotnem primeru ni nobenih pridobitev. Stiki in izbira podatkov z večih fragmentov zavirajo učinkovitost in razširljivost SUPB-ja.

### 3.2.3 Uravnavanje obremenitve

Za porazdeljevanje obremenitve ponavadi skrbi del programske ali strojne opreme, postavljen pred gručo strežnikov. Cilj opreme je zagotoviti, da se izvaja zastavljen načrt razširljivosti (npr. bralne zahteve se preusmeri na bralne strežnike), da se viri porabljajo učinkovito (boljši strežniki dobijo več dela), da se dostopa do dosegljivih strežnikov, da je transparentna (uporabnik ne ve zanjo) in da je konsistentna (v primeru če neka aplikacija vzdržuje neko stanje, se povezane zahteve preumerijo na isti strežnik) [10, stran 555-556].

Obstaja več algoritmov, na podlagi katerih se izbere naslednji strežnik za zahtevo. Nekaj najlažjih za implementacijo in ki ne zadržujejo zahtev v vrsti [10, stran 562-563]:

1. po naključju,

2. po principu *round-robin* (naslednji strežnik se izbere kot naslednji v nekem vnaprej definiranem zaporedju),

3. glede na najmanj aktivnih povezav,

4. glede na najboljši odzivni čas,

5. glede na razpršenost (zahteve z nekega območja vedno streže isti strežnik),

6. glede na utež (npr. zmogljivejši strežniki imajo večjo utež in bodo zato dobili več dela);

## 3.3 Razlogi za težavno razširljivost tradicionalnih SUPB

Arhitektura tradicionalnih SUPB-jev je stara 40 let in še iz časov, ko je bila tovrstna programska oprema namenjena večinoma le procesiranju poslovnih podatkov, danes pa se je uporaba SUPB-jev z razmahom interneta med drugimi razširila še na trge kot so podatkovna skladišča, spletna družbena omrežja, spletne igre. Te aplikacije dnevno generirajo, shranjujejo, procesirajo velike količine podatkov [14]. Pionir na področju podatkovnih baz Mike Stonebraker je s sodelovci v poročilu *OLTP pod drobnogledom in kaj smo odkrili* prišel do zaključka, da tradicionalni SUPB-ji poleg uporabnega dela veliko časa namenjajo delu z mehanizmi, ki skrbijo za integriteto podatkov in predstavljajo 88 % časa vsega izvajanja transakcije [15]. Ti mehanizmi vključujejo:

- upravljanje indeksov (B-drevesa): zahteva procesor in vhodno/izhodne enote, saj so ti shranjeni na disku,

- pisanje v dnevnik: SUPB na zahtevo transakcije piše dvakrat - poleg v bazo še v dnevnik, ki se nahaja na disku,

- zaklepanje vseh uporabljenih podatkov transakcije,

- skrb za deljene strukture pri večnitnosti (B-drevesa, tabela zaklepov, tabele virov),

- upravljanje medpomnilnika: podatki so shranjeni na straneh diska s fiksno dolžino, medpomnilnik upravlja, katere strani so shranjene v njem.

## 3.4 Boljša razširljivost - druge rešitve

### 3.4.1 Podatkovne baze NoSQL

Dandanes je zelo razširjen način doseganja razširljivosti spletnih aplikacij z vidika podatkovnih baz uporaba podatkovnih baz NoSQL. Z izrazom NoSQL opišemo vrsto SUPB-jev, ki v splošnem omogočajo shranjevanje podatkov po principu ključ-vrednost, za poizvedovanje po podatkih ne uporabljajo jezika SQL, podatki se ne hranijo v strukturah s predhodno določeno shemo (slednje so dinamične - ni potrebno, da imajo vsi zapisi v zbirki iste atribute) in z lahkoto razširjajo vodoravno [11]. Podatkovne baze NoSQL po večini tečejo na arhitekturi, kjer si strežniki v gruči ne delijo ničesar in kjer poteka fragmentacija samodejno.

Glavne slabosti podatkovnih baz NoSQL so:

1. Opustitev uporabe jezika SQL: Jezik SQL se razvija in dopolnjuje že od leta 1986 in je na svoji dolgi razvojni poti premagal že marsikatero oviro. Opustitev SQL jezika pomeni zavreči stran njegove prednosti: izrazen jezik, varnost podatkov, podatkovno integriteto in dobre prakse.

2. Ne omogoča varnih transakcij: Za NoSQL je značilno, da istočasno ne morejo zagotavljati konsistentnosti (vsa vozlišča vidijo enake podatke v istem času), dostopnosti (odgovor na vsako zahtevo, ne glede na uspešnost) in tolerance do odpovedi vozlišč (sistem nadaljuje z delom kljub izgubi dela sistema), ampak samo katerikoli dve od njih (CAP teorem). Za NoSQL zato namesto ACID velja teorem BASE, ki se osredotoča na toleranco do odpovedi vozlišč ter dostopnost, opusti pa

konsistentnost. NoSQL posledično ne omogočajo transakcijske konsistentnosti, temveč le eventualno konsistentnost.

3. Skrb za skladnost podatkov na aplikacijski ravni: Razvijalci aplikacij morajo sami sestaviti podatkovni model in določiti, v kakšni povezavi so si posamezne entitete. Ker baze NoSQL ne omogočajo transakcijske konsistentnosti, morajo razvijalci ustrezno zaščititi občutljive podatke, kar pa po večini presega znanje razvijalcev aplikacij. Njihovo znanje in čas je bolje uporabljen pri razvijanju privlačnih in funkcionalnih uporabniških vmesnikov, ne pa za razvijanja nečesa, kar je v baze vgrajeno že 30 let.

### 3.4.2 Podatkovne baze NewSQL

V zadnjih letih se je pojavil nov razred relacijskih podatkovnih baz (imenovane pod skupnim imenom NewSQL), ki se poleg podpore jeziku SQL ter zagotavljanju ACID osredotoča še na razširljivost, ki jo nudijo baze NoSQL ter visoko zmogljivost [12]. Za te podatkovne baze je značilno, da lahko tečejo na večih strežnikih in ne vsebujejo vseh mehanizmov, ki jih uporabljajo tradicionalni relacijski SUPB-ji in ki predstavljajo ozko grlo pri doseganju večje zmogljivosti (ali pa so le-ti prilagojeni).

Podatkovne baze NewSQL se lahko razvrsti v naslednje tri kategorije [13]:

1. Novi MySQL pogoni za shranjevanje podatkov: Omogočajo enak programski vmesnik kot MySQL, a za shranjevanje uporabljajo tehnologijo, ki je za potrebe razširljivosti bolje optimizirana. S tem poskušajo zaobiti ozko grlo, ki ga predstavlja hitrost pisanja/branja z diska. Predstavniki teh so TokuTek, Xeround, GenieDB.

2. Nova arhitektura: Predstavljajo popolnoma novo rešitev, zgrajene iz nič s konceptom porazdeljenosti v mislih. Sem sodijo VoltDB, NuoDB, Clustrix.

3. Transparentna fragmentacija: Pri teh se lahko uporabi obstoječo relacijsko podatkovno bazo, podatkovna baza NewSQL pa omogoča vmesno opremo za fragmentacijo podatkovne baze na večih strežnikih. Predstavniki te kategorije so dbShards, ScaleDB, MySQL Cluster.

# Poglavje 4

# Opis podatkovnih baz NewSQL

## 4.1 NuoDB

Gre za SUPB, zgrajen iz nič ter z mislimi o potrebah današnjega trga, hkrati pa izkorišča prednosti današnjih tehnologij. Tako na primer uporablja disk le za shranjevanje in pridobivanje podatkov, za poizvedovanje po podatkih pa spomin enega ali več strežnikov, saj se uporabljeni podatki večinoma nahajajo tam. Razlog je v tem, da se hitrost diska v zadnjem času ni spremenila, medtem ko se je povečal glavni spomin in njegova hitrost kot tudi hitrost in zanesljivost omrežja. NuoDB lahko teče na fizičnih ali virtualnih računalnikih, v oblaku in med večimi oblaki, omogoča pa tudi geoporazdelitev [16].

Sestavljen je iz treh plasti: administrativne (aplikacijske), transakcijske in pomnilne [16]. Administrativna plast predstavlja vhod v bazo za uporabnike preko vmesnikov različnih programskih jezikov, agente in posrednike. Na transakciji plasti se nahajajo transakcijski pogoni, na pomnilni pa pomnilni pogoni.

**Administrativna plast** Pri posrednikih in agentih gre za proces iste vrste, le da imajo posredniki dodatno zadolžitev. Poleg nalog agentov, ki zbirajo statistiko posameznih pogonov na obeh plasteh, skrbijo še za dodeljevanje povezav novo priključenim odjemalcev najbolj dostopnemu

transakcijskemu pogonu (izravnavajo obremenitev). Vsak SUPB mora imeti najmanj enega posrednika, vsak strežnik pa vsaj enega agenta. Agenti so v nasprotju s posredniki omejeni samo na strežnike, na katerih tečejo. Strežniki, ki poganjajo podatkovno bazo, se povezani v domeno.

**Transakcijska plast** Transkacijsko plast sestavljajo transkacijski pogoni, kjer se poizvedba SQL razčleni, pretvori, optimizira in izvede v spominu računalnika. Transakcijski pogoni skrbijo za atomarnost, konsistetnost in izolacijo transakcij.

Več transakcijskih pogonov povečuje prepustnost SUPB-ja. Noben transakcijski pogon ne vsebuje celotne podatkovne baze. Če transakcijski pogon ne vsebuje iskanega podatka, ga poskuša pridobi preko omrežja od drugih transakcijskih pogonov. Če ga tudi drugi nimajo, se na koncu povpraša še pomnilni pogon, ki pridobi podatek z diska ali svojega spomina.

**Pomnilna plast** Pomnilni pogoni na tej plasti ne procesirajo stavkov SQL, ne poznajo tabel, vrstic ali indeksov. Skrbijo za shranjevanje podatkov, ki jih transakcije na višji plasti potrdijo ter pridobivanje informacij z diska. Vsebujejo popolno kopijo podatkovne baze. Večje število pogonov na tej plasti naredi podatkovno bazo bolj obstojno. Na enem strežniku je lahko več pogonov. Pri transakcijskih je odvisno od količina spomina strežnika, pri pomnilnih pa več kot 1 nima smisla.

Zaradi interne zgradbe SUPB-ja, ki se razlikuje od relacijskega SUPB-ja, je shranjevanje podatkov na pomnilni medij pri NuoDB implementirano po principu ključ-vrednost, saj pomnilni pogon rokuje z majhnimi ključi (identifikator struktur, ki se imenujejo atomi), ki imajo vrednosti večje dolžine (predstavlja vsebino atoma). To ne pomeni, da je podatkovna baza v NuoDB brez shem, saj navzven deluje kot vsak drugi relacijski SUPB.

### 4.1.1 Atomi

Gradniki SUPB-ja so atomi, ki so implementacija porazdeljenih objektov [16, stran 11]. Atomi imajo podobno kot objetki v objektno usmerjenem programiranju tip in primerke. Česar ti objekti nimajo, atomi pa imajo, so kopije. Za primer, NuoDB ima atom, ki je tipa Tabela. Vsaka tabela v pogonu je predstavljena s primerkom tega atoma. Vsak transakcijski/pomnilni pogon, ki vsebuje to tabelo, ima kopijo tega primerka atoma.

Atomi so objekti, ki predstavljajo točno določeno vrsto informacije (podatek, indeks, shema itd.) in ki komunicirajo preko omrežja med svojimi primerki ter drugimi atomi, ki sodelujejo pri izvedbi transakcij [17]. Transakcijski pogon, na katerem se izvaja transakcija, lahko naredi vrsto sprememb na neki skupini atomov. Na primer, vnos novega zapisa spremeni najmanj atoma Zapis (vsebuje metapodatke o zapisu) in Podatki (dejanski podatki, ki jih vsebuje zapis). Vsak spremenjeni atom pošlje sporočilo o replikaciji svojim kopijam na drugih pogonih, da se novi zapis pojavi povsod.

Atomi lahko pošiljajo več vrst sporočil, a najpogostejša so sporočila o replikaciji [16, stran 11]. Označujejo, da je transakcijski pogon naredil spremembo, ki naj se pokaže tudi pri njegovih kopijah na drugih transakcijskih pogonih. Ta sporočila so majhna (povprečno 18 bajtov) in nesočasna. Sistem sporočanja vsebuje izravnalnik, kjer se sporočila kopičijo in ko doseže določeno velikost ali neko posebno sporočilo zahteva takojšnje ukrepanje, se sporočila pošljejo v skupini. V nekem času se tako lahko različne kopije atomov med sabo razlikujejo, saj še niso prejeli sporočila o spremebah. Na koncu, ko so prejeta vsa sporočila o spremembah in spremembe narejene, so vse kopije atomov konsistentne.

Vrste atomov [16, stran 12-13]:

- Atom Katalog: Sledi vsem atomom, ki se nahajajo v SUPB-ju. Za vsak atom ve njegov identifikator.
- Atom Glavni katalog: Vsak transakcijski in pomnilni pogon vsebuje kopijo tega atoma, saj ima vpogled nad vsemi ostalimi atomi (in njihove

primerke) ter sledi vsem drugim strežnikom, ki so priključeni SUPB-ju. Ko se SUPB-ju priključi nov pogon, je ta atom prva stvar, ki jo zahteva. Ko transakcijski pogon ustvari novo vrstico ali sekvenco števil, njen Glavni katalog obvesti Glavne kataloge na drugih pogonih o spremembi.

- Atom Podatkovna baza: Prisoten v vsakem transakcijskem in pomnilnem pogonu. Nosi informacijo o podatkovni bazi kot celoti (informacije o avtentikaciji, privilegijih uporabnikov, itd.). Upravlja z atomi Shema.
- Atom Transakcijski pogon: Tudi ta je prisoten na vseh pogonih (tako transakcijskih kot pomnilnih). Sledi vsem transakcijam v podatkovni bazi, jim dodeljuje identifikatorje ter pošilja sporočila o replikaciji.
- Atom Shema: Ustvarja atome Tabela, Katalog tabele in Zaporedje. Skrbi za vloge in njene privilegije.
- Atom Zaporedje: Ustvarja vrednosti za samodejno ustvarjanje ključev ter druge operacije za zaporedja v SQL.
- Atom Katalog tabel: Ima podobno funkcijo kot Glavni katalog, le da so ti atomi Zapisi, Podatki in Indeks, ki so povezani s tabelo podatkovne baze.
- Atom Tabela: Ustvarja atome Zapisi in Indeks. Sledi definiciji tabele. Ne vsebuje podatkov, pač pa le informacije, ki so potrebne za najdbo in interpetiranje podatkov. Če je format tabele spremenjen, atom Tabela njen stari format obdrži, da se podatki tabele lahko prenesejo.
- Atom Zapisi: Vsak atom tega tipa sledi 2.000 vrsticam in njenim različicami. Za vsako vrstico shranjuje
  - identifikator transakcije, ki je naredila spremembo,
  - identifikator formata definicije tabele, ko je bila vrstica ustvarjena,
  - številka, ki nakazuje vrstni red različice,

  - posreden kazalec na podatke (podatki se nahaja v atomu Podatki).

- Atom Blob: Preprostejša različica atoma Zapisi, saj nimajo različic.
- Atom Podatki: Vsebujejo podatke, ki se ne delijo. Ti atomi imajo končno dolžino 50.000 bajtov in če jo presežejo, transakcijski pogon ustvari nov atom Podatki z novimi različicami vrstic.
- Atom Indeks: Hranijo indekse, ki so sestavljeni iz ključa in številke vrstice. Indeksni dostop je sestavljen iz vpogleda v indeks za določenim ključem ali razponom, pridobitvijo številk vrstic ter nato dostop do vrstic s temi številkami. Številka vrstica se razgradi v številko določenega atoma Zapisi za to tabelo in vrstico v tem atomu. Tudi atomi Indeks imajo tako kot atomi Podatki končno dolžino.

### 4.1.2 Konsistentnost in izolacija transakcij

#### Nadzor sočasnosti z več različicami

NuoDB za vzdrževanje konsistetnosti bralnih in pisalnih transakcij uporablja nadzor sočasnosti z več različicami (angl. *multiversion concurrency control*, v nadaljevanju MVCC) [16, stran 11]. Transakcije se ne čakajo, temveč preberejo zadnjo različico podatka, ki ga potrebujejo.

Spreminjanje in izrbis vrstice podatkov v NuoDB se pozna kot dodajanja novih različic vrstice, ki čakajo na odobritev, dokler se transakcija ne potrdi. Odobritev različic se dovoli le v primeru, če transakcijski pogon ugotovi, da je ta transakcija lastnik zadnje razliciče vrstic podatkov, ki jih spreminja in da je sprememba vidna na vseh drugih pogonih.

Zaradi tega je razveljavitev transakcij enostavna - vrstice, ki čakajo na odobritev, se nikoli ne potrdijo in se jih zavrže. Kar pomeni, da so sporočila o posodobitvah optimistična (predpostavljanje, da ne bo prišlo do napak). Transakcija pošlje sporočilo transakcijskemu pogonu o narejeni spremembi takoj in nadaljuje z izvajanjem (zato so sporočila tudi nesočasna). Če transakcijski pogon dovoli spremembo, ni nobenega problema. V nasprotnem pri-

meru transakcija blokira, ampak le do takrat, ko se mora dokončno odločiti ali potrditi spremembo. Nesočasnost in optimistično obnašanje skupaj s pošiljanjem sporočil v paketih omogoča NuoDB, da se izogne morebitnim zakasnitvam in nepredvidljivim situacijam omrežja v oblaku.

Pisanje sprememb podatkovne baze v dnevnik je zaradi MVCC lažje, saj se novi zapisi samo dodajajo. Za dnevnik je zadolžen pomnilni pogon, ki med drugimi tudi arhivira podatke. Te spremembe so majhne in zato ne zmanjšujejo hitrosti izvajanja transakcij. Tudi posodobitve v dnevnik so lahko zapisane, še preden se transakcija potrdi. Če transakcija ni potrjena, se to zabeleži in ob naslednjem branjem dnevnika se zato povezane posodobitve preskočijo, ko se v delih spreminja arhiv. MVCC nudi tako optimistično izvajanje tudi na pomnilni ravni.

### Koordinacija

Vsak atom ima svojega predsednika, ki je potreben pri pisalnih operacijah, ki zahtevajo koordinacijo pri razreševanju konfliktov [16, stran 14]. Predsednik je prva kopija atoma, katerega najbolj pogosta naloga je dodeljevanje enoličnih identifikatorjev atomom, ki jih nek atom ustvari. Informacija, ki jo nosi vsak atom, je identiteta njegovega predsednika in seznam naslednikov, če predsednik iz različnih razlogov postane nedosegljiv.

Na primer, predsednik atoma Transakcijski pogon pošlje enolične identifikatorje za transakcije svojim kopijam, ki le-te ustvarjajo. Predsednik atoma Shema dodeljuje identifikatorje za atome Tabela in Zaporedje. Predsednik atoma Zaporedje pošlje identifikatorje, ki se uporabijo kot ključi zaporedij.

Predsednik atoma Zapisi je zadolžen še za koordinacijo posodobitev vrstic. Ko transakcija posodobi vrstico, spremembo naredi na lokalni ravni in pred pošiljanjem sporočil o replikaciji najprej za dovoljenje vpraša svojega predsednika, v katerem so podatki o identifikatorju te transakcije, številka vrstice, ki jo spreminja in identifikator transakcije, za katero vidi, da je naredila zadnjo različico te vrstice. Če predsednik na globalni ravni vidi, da je ta transakcija res naredila zadnjo različico te vrstice, dovoli spremembo ter

na globalni ravni posodobi različico te vrstice ter identifikator transakcije, ki je naredila spremembo vrstice. V nasprotnem primeru sporoči transakciji, da počaka, da se tekoča transakcija zaključi. Transakcija, ki zahteva spremembo, nato povpraša atom Transakcijskega pogona na lokalni ravni, kdaj se bo tekoča transakcija zaključila. Če se transakcija, na katero se čaka, uspešno izvede, bo posodobitev čakajoče transakcije neuspešna, v nasprotnem primeru pa bo posodobitev uspešna.

### 4.1.3 Orodja

#### Nastavitvena datoteka

Nastavitvena datoteka med drugimi definira domeno v NuoDB (ime in geslo za dostop), ali proces baze teče kot posrednik ali agent ter ime/številko IP drugega strežnika, kateremu priključujemo ta strežnik.

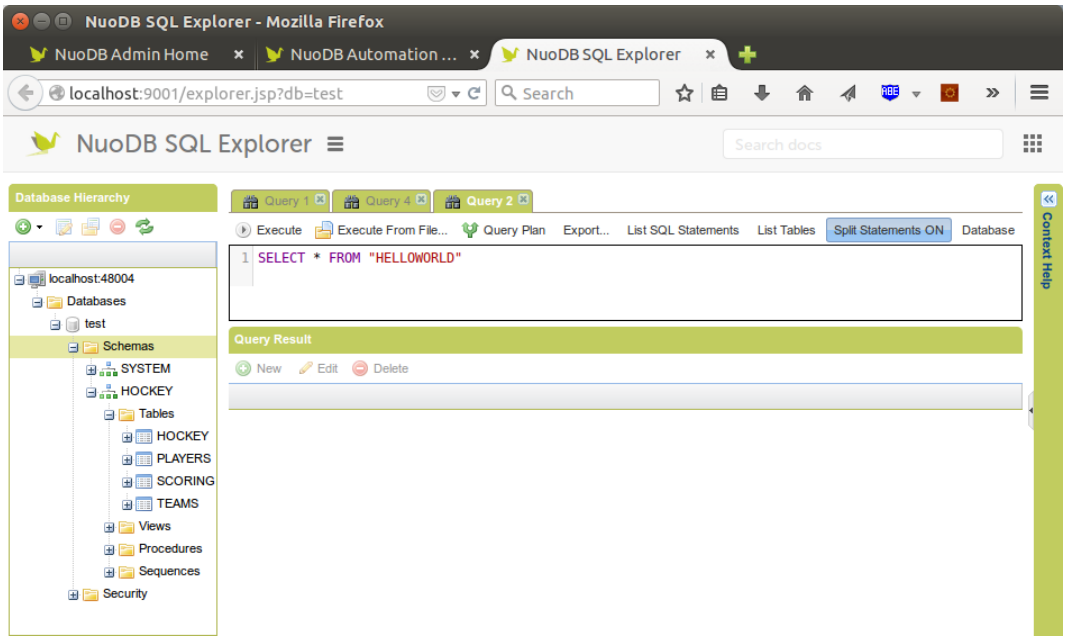
#### Spletni vmesnik za upravljanje z bazami

NuoDB za lažjo administracijo domene podatkovne baze poleg konzole v ukazni vrstici nudi tudi spletni vmesnik, preko katerega lahko uporabnik ustvarja/zavrže podatkovno bazo. V obliki tabelaričnih podatkov in grafov spremlja aktivnosti celotne podatkovne baze ali njenih posameznih pogonov in za novo dodane strežnike k domeni določi vrsto pogona, ki se bo tam izvajal (nič ne preprečuje, da bi se na nekem strežniku izvajalo več pogonov).

Spletna konzola ponuja tudi vmesnik ( 4.1) za vpis poizvedb SQL in ogled njenih rezultatov ter vpogled v strukturo podatkovne baze - definicije tabel, ključev, sprožilcev, pogledov.

#### Vnos podatkov

Podatke se lahko vstavlja preko konzole s pomočjo stavkov SQL (INSERT), v primeru večjih količin pa se lahko uporabi program `nuoloader`, ki naloži podatke iz datoteke. Za osnoven zagon programa se poleg avtentikacijskih podatkov za dostop do baze (ime in strežnik baze ter ime in geslo uporabnika)




Slika 4.1: Orodje Raziskovalec SQL, ki ga ponuja NuoDB.

navede še ciljno shemo baze, samo datoteko s podatki ter INSERT stavek, ki bi se sicer uporabil za vnos ene vrstice. Primer:

```
nuoloader test@localhost --user dba --password goalie --schema test --import
     data.csv --to "INSERT INTO players VALUES (?,?,?,?,?)"
```

Pri `import` možnosti se lahko poleg imena datoteke navede še znak za ločilo med podatki v vrstici (v primeru vejice to ni potrebno), številko vrstice, od katere naj se začnejo nalagati podatki (v primeru metapodatkov v datoteki). Primer:

```
nuoloader test@localhost --user dba --password goalie --schema test --import
     data.tsv,seperator=tab,skip=2 --to "INSERT INTO players VALUES
    (?,?,?,?,?)"
```

**Orodje za prenos podatkov iz baz drugih proizvajalcev v bazo NuoDB**

Je program, ki ga poganjamo preko ukazne vrstice in s pomočjo katerega lahko administrator preko gonilnika JDBC prenese sheme in obstoječe podatke iz vseh popularnejših SUPB-jev v NuoDB. Ker omogoča tudi prenašanje baz iz NuoDB, se ga lahko uporabi za ustvarjanje kopij pri NuoDB.

## 4.2 VoltDB

Čeprav gre tudi pri VoltDB za SUPB, zgrajen na novi tehnologiji, pa se VoltDB bolj osredotoča na delovanje brez mehanizmov, ki pri tradicionalnih SUPB-jih povzročajo režijo in ovirajo boljšo razširljivost ter njihovo zamenjavo. Transakcije se tako izvajajo v spominu, s čimer ni več potrebe po upravljanju medpomnilnika. Podatkovna baza je samodejno fragmentirana med vozlišči v gruči, samodejno replicirana v strežniku in med strežniki. Vsako vozlišče se izvaja avtonomno, s čimer je odpravljena večnitnost oziroma zaklepanje [18]. Temelji na odprtokodni podatkovni bazi H-Store, ki je plod sodelovanja med raziskovalci večih univerz v ZDA (MIT, Yale in Brown). Vendar H-Store za razliko od VoltDB na primer ne ponuja možnosti za upravljanje in vzdrževanje gruče, a zaradi odprtokodnosti omogoča nadaljnje optimizacije. [19]

Transakcije se v VoltDB izvajajo preko postopkov, ki predstavljajo način pisanja transakcij in ki prinašajo želeno visoko učinkovitost, saj zmanjšujejo količino komunikacije med SUPB-jem in aplikacijo za izvršitev poizvedbe [20].

VoltDB najboljše rezultate prikaže pri aplikacijah, ki zahtevajo veliko pisalnih operacij in visoko prepustnost. Te aplikacije so tiste, ki sprotno obdelujejo podatke za analitiko in odločanje v realnem času in tiste, ki obdelujejo podatke, ki se generirajo z veliko hitrostjo [21].

### 4.2.1 Fragmentacija

Fragmenti so osrednji del SUPB-ja, saj je pri VoltDB samo prek teh možno doseči vzporednost izvajanja [21]. Vozlišča, ki predstavljajo en fragment, so vezana na jedro procesorja (v smislu obdelave, ne hranjenja podatkov) in ne na strežniki, saj je fragmentiranje na ravni procesorja arhitekturno bolj preprosto in s čimer se pride tudi do boljših rezultatov učinkovitosti. Ker so fragmenti avtonomni, ni zaklepanja ter večnitnosti in je zato koda bolj zanesljiva, razhroščevanje pa manj problematično. Fragmentira se tako podatke po uporabniško definiranem ključu kot procesiranje v povezavi s podatki. Poleg fragmentiranih tabel VoltDB pozna tudi tabele, ki se v celoti nahajajo na posameznih fragmentih. To so predvsem tiste, ki se ne spreminjajo veliko in naredijo operacije stikov manj potratne.

Vsa vozlišča so v VoltDB obravnavana enako, v nasprotju z veliko večino drugih porazdeljenih sistemov, ki različnim delom gruče razdelijo različne vloge (soglasje, upravljanje z metapodatki, dejansko procesiranje) [21]. Kot ostali porazdeljeni sistemi ima tudi VoltDB več podsistemov, a gruča interno upravlja, kakšne odgovornosti so dodeljene različnim vozliščam. Vsaka vozlišče lahko preneha delovati in gruča bo hitro dodelila vloge tega vozlišča sovozliščam.

Večanje števila jeder strežnika pomeni več fragmentov, vendar dodajanje jeder na neki točki ne vpliva več na boljšo prepustnost, saj bodo transakcije v nekem trenutku zapolnile razpoložljivi prostor vrat strežnika, skozi katere kominicirajo s sovozlišči/odjemalcem. Podobno velja za večanje števila strežnikov v gruči. Za dobro učinkovitost (čim manjši zamiki in manjše motnje med vozlišči) je bolje, da so vsi strežniki na istem omrežnem stikalu. Očitno je, da je priporočeno največje število strežnikov tolikšno, kot jih podpira stikalo. VoltDB najbolje deluje s strežniki, ki imajo od 4 do 16 jeder in v gruči z 2 do 32 strežnikov, priključenimi na isto omrežno stikalo [24].

### 4.2.2 Trajnost

Podpora trajnosti transakcij v VoltDB se začne s transakcijskim modelom (v obliki postopkov). Ko transakcija uspe in so spremembe potrjene v podatkovni bazi, VoltDB ponuja več načinov, kako te spremembe narediti trajne. Če strežnik, kjer teče podatkovna baza, neha delovati zaradi nekega razloga, se vsebina ohrani in obnovi na zadnje konsistentno stanje. Način je odvisen od zahtevane stopnje trajnosti [22]:

1. Posnetki podatkovne baze omogočajo osnovno trajnost. Gre za kopije vsebine podatkov v času, ko je narejen posnetek in ki se shrani na disk. Posnetki podatkovne baze so lahko narejeni ročno ali pa samodejno med uporabniško določenimi časovnimi intervali (ponavadi v minutah). Če strežnik neha delovati, se ob naslednjem zagonu prebere zadnji posnetek podatkovne baze.

2. Asinhrono pisanje ukazov v dnevnik omogoča napredno trajnost s tem da ustvari tako posnetek podatkovne baze kot dnevnik o vseh transakcijah med posnetki. Če strežnik neha delovati, se pri obnovi podatkovne baze poleg zadnjega posnetka baze ponovno izvedejo tudi transakcije, ki so nastale po tem zadnjem posnetku. Zaradi tega je izguba le nekajsekundna (z uporabo le posnetkov baze je ta nekajminutna).

3. Sinhrono pisanje ukazov v dnevnik nudi najboljšo trajnost. Je nadgradnja asinhronskega pisanja ukazov v dnevnik. Razlika med načinoma je v tem, da se dnevnik ustvari, ko je transakcija zaključena, a še nepotrjena. Z drugimi besedami, nobena transakcija ni potrjena, če ni zapisana v dnevniku in tako ni nobena transakcija izgubljena. Edina slabost tega je, da je potrebna boljša tehnologija diska, saj se v nasprotnem primeru pozna na hitrosti izvajanja transakcij.

### 4.2.3 Dosegljivost

VoltDB nudi funkciji *K-varnost* in *Zaznavanje fragmentov v omrežju* za varovanje pred lokalnimi izpadi strojne opreme ter funkcijo *Replikacijo podatkovne baze* kot varovanje pred bolj razsežnimi izpadi med večimi vozlišči v gruči [23].

1. K-varnost deluje tako, da podvaja fragmente znotraj gruče podatkovne baze. Vse kopije fragmenta delujejo istočasno, da je konsistentnost zagotovoljena ves čas. Če je vozlišče deležno izpada, njeno delo prevzamejo njene kopije, dokler se to vozlišče ne popravi in vključi nazaj v gručo. Uporabnik lahko poda vrednost K pred zagonom podatkovne baze, ki pove, za kolikšen faktor reduntantnih fragmentov naj se ustvari v bazi. Večja vrednost po drugi strani zmanjša sistemske vire enega fragmenta.

2. Zaznavanje fragmentov v omrežju deluje v sodelovanju s K-varnostjo. V K-varni gruči zaznavanje fragmentov varuje pred izpadi omrežja, kjer bi sicer gruča mislila, da je prišlo do izpada enega ali več vozlišč. Če izpade omrežje med dvema vozliščema, gruča to pomanjkanje povezave vidi kot izpad vozlišča. Možno je, da bi nato dva ali več segmentov v gruči začela delovati ločeno. Zaznavanje omrežnih fragmentov zagotovi, da preživi samo en segment v gruči. Funkcija je kritična za situacije, kjer bi nihanja v omrežju lahko vplivala na povezljivost gruče.

3. Replikacija podatkovne baze zagotavlja podobno funkcijo kot K-varnost, le da namesto replikacije fragmentov lokalno replicira podatke celotne gruče na nek oddaljen kraj.

V kombinaciji s funkcijami trajnosti, kot je pisanje ukazov v dnevnik, VoltDB omogoča dobro zaščito proti potencialnimi nezaželenimi situacijami, ki se pojavljajo v današnjem okolju poslovanja.

### 4.2.4 Orodja

#### Poizvedovanje in vnos podatkov

Čeprav se VoltDB najbolj obnese, ko se za dostop do podatkov uporabljajo postopki, pa VoltDB ponuja tudi `sqlcmd`, interaktivno konzolo za poizvedovanje po bazi na `ad-hoc` način, ki pride prav pri procesu razhroščevanja in polnjenja baze s podatki. V primeru vnosa velikih količin podatkov je lahko podobno kot pri NuoDB uporabljen program za nalaganje podatkov iz datoteke, `csvloader`.

Program `csvloader` predpostavlja, da se stolpci podatkov zapisa nahajajo v istem vrstnem redu, kot so definirani v shemi tabele, kamor se nalagajo. Primer klica programa, ki naloži podatke iz datoteke towns.txt v tabelo towns, izpusti prvo vrstico datoteke, podatki pa so med seboj ločeni z navpičnico (in ne kot privzeto z vejico):

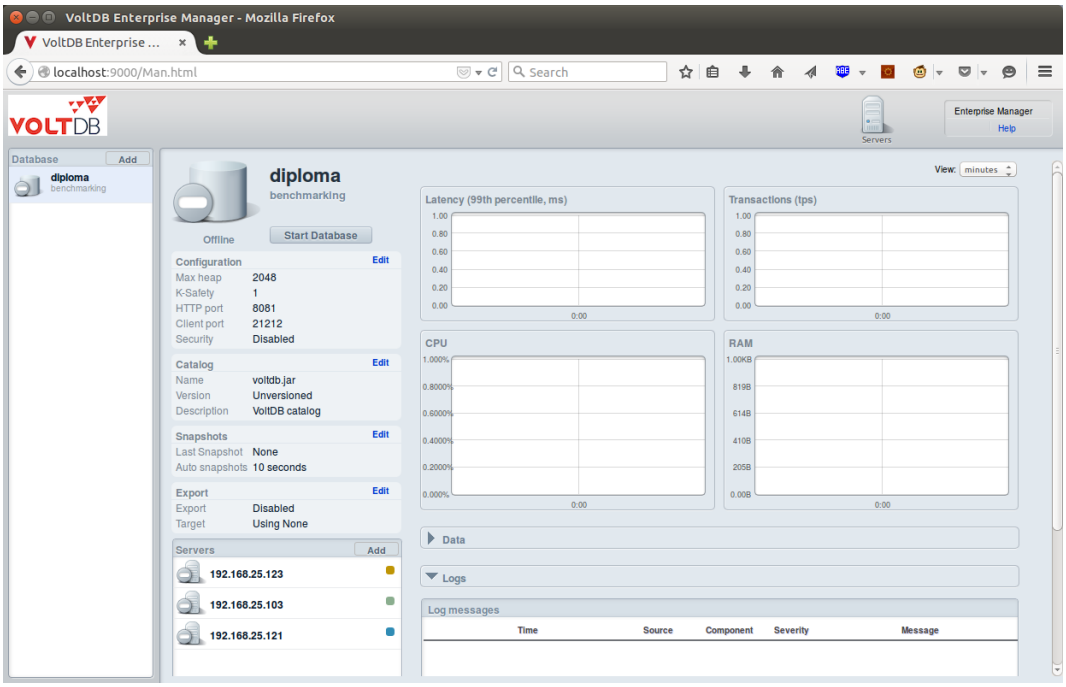
```
csvloader --separator "|" --skip 1 --file towns.txt towns
```

#### Administracija

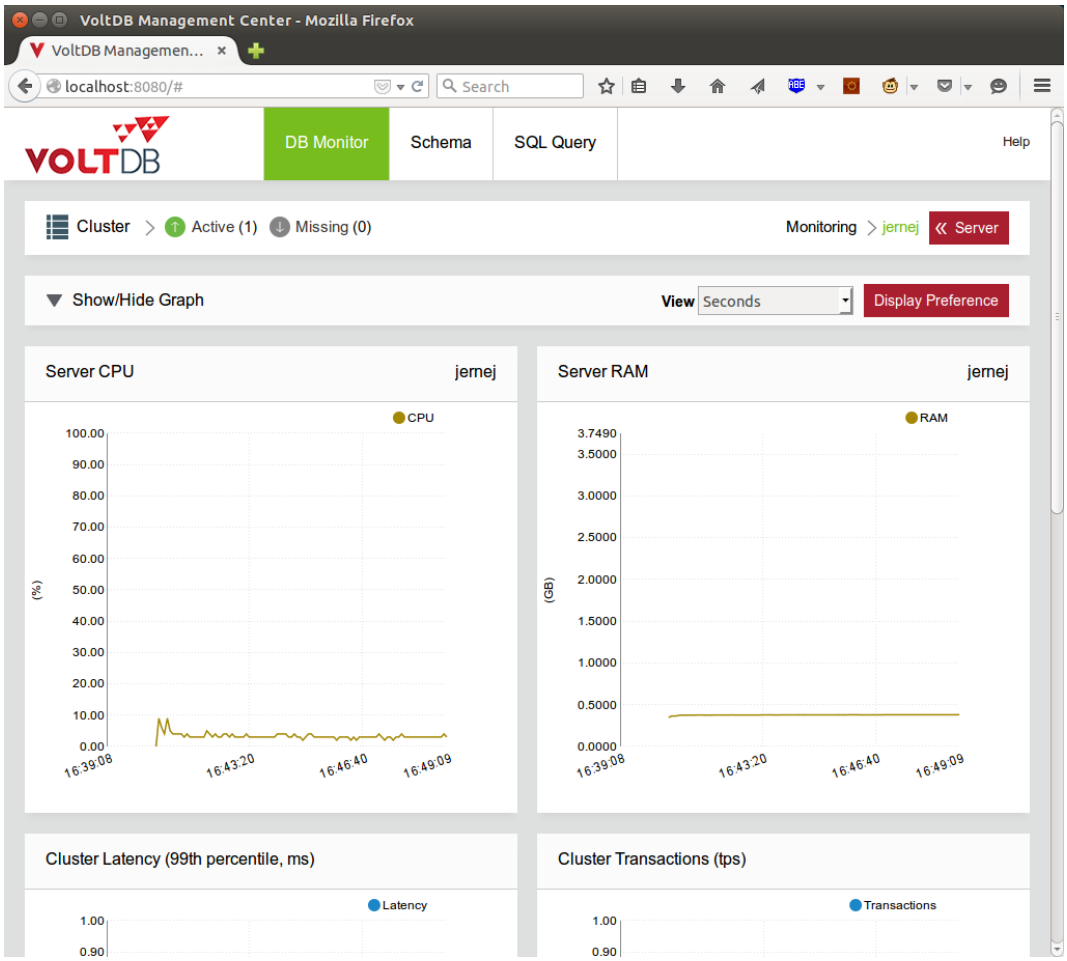
VoltDB nudi naslednja administracijska orodja, ki delujejo le ob aktivni podatkovni bazi. Prva dva sta implementirana v obliki spletnega vmesnika.

**Upravitelj podatkovnih baz v VoltDB** Služi za poenostavitev administracije gruče baze. Omogoča porazdelitev programskega paketa VoltDB in kataloga med ostala vozlišča gruče, kar administratorjem prihrani veliko ponavljajočega dela. Poleg administracije nudi tudi realno-časovno statistiko baze (zamik, hitrost izvajanja transakcij, poraba procesorja in spomina, vidno na sliki slika 4.2). Poleg vmesnika nudi še aplikacijski programski vmesnik (API), ki administratorjem omogoča še nadaljnjo prilagajanje opravil, povezanih z upravljanjem.

**Spletni studio** Podobno kot prej opisano orodje omogoča spremljanje ključnih statistik dogajanja v SUPB-ju (slika 4.3). Poleg tega omogoča tudi vpogled v vsebino baze (sheme, postopki,...), vmesnik za vnašanje




Slika 4.2: Orodje Upravitelj podatkovnih baz v VoltDB.




Slika 4.3: Spletni studio VoltDB.

poizvedb SQL ter poročilo o katalogu. Slednje vsebuje podrobne informacije o katalogu - sheme, postopke, stavke SQL, količino spomina, ki ga porabi. Vsebuje tudi informacije o sistemu in poročilo o dogajanju v spominu.

**Program voltadmin** Izvajanje administrativnih ukazov nad bazo preko ukazne vrstice.

## 4.3 Primerjava VoltDB in NuoDB z MySQL

V tem razdelku je predstavljenih nekaj ključnih razlik med SUPB-jema NuoDB in VoltDB ter najbolj popularnim relacijskim SUPB-jem MySQL.

**Porazdeljen sistem** Obe bazi spadata pod porazdeljene podatkovne baze, MySQL je bil prvotno zgrajen za poganjanje na enem strežnik.

**Fragmentacija** Najbolj razširjen način vodoravne razširljivosti z MySQL je fragmentacija. Razvijalec mora pri tem sam poskrbeti za rokovanje s podatki in njihovo varnost. Pri VoltDB fragmentacija poteka samodejno, potrebno je samo navesti ključ, po katerem se fragmentira podatke. NuoDB ne pozna fragmentiranja, saj se na strežniku, določenim za hranjenje podatkov, hrani popolna kopija podatkovne baze.

**Replikacija** Več strežnikov je pri MySQL ponavadi povezanih kot gospodar-suženj, pri NuoDB in VoltDB pa kot gospodar-gospodar.

**Poizvedovanje** Pri MySQL se do podatkov dostopa preko strani, ki se z diska naložijo v medpomnilnik. Pri VoltDB se vsi podatki nahajajo v spominu strežnika. Pri NuoDB pa se podatki nahajajo tako v spominu kot pomnilniku (do slednjega se dostopa le, če podatka ne vsebuje noben spomin strežnika v gruči).

**Varne transakcije** Vsi trije SUPB-ji omogočajo varne transakcije (lastnosti ACID). V nasprotju z MySQL, VoltDB in NuoDB pri dostopu do podatkov teh ne zaklenejo in s tem preprečijo, da bi bralne transakcije, ki ne spreminjajo vsebine baze, čakale.

**Obnova podatkovne baze** Vse trije SUPB-ji omogočajo pisanje v dnevnik izvedenih transakcij, iz katerega se bere pri obnovi baze. VoltDB in NuoDB nudita več in učinkovitejše načine shranjevanja sprememb baze, ki ne predstavljajo večje režije pri procesiranju uporabnega dela.

**Glavne ovira** Pri MySQL sta glavni oviri pri doseganju večje odzivnosti disk in mehanizmi, ki zagotavljajo integriteto podatkov. Pri obeh sistemih NewSQL je to spomin.

**Sheme** Pri vseh treh SUPB-jih je potrebno definirati sheme v podatkovne baze, le da so te pri MySQL nefleksibilne (ne da se jih čez čas spremeniti), pri VoltDB in NuoDB pa se jih lahko tekom izvajanja sistema spreminja.

**Podpora** MySQL je najbolj popularen SUPB in posledično ima ogromno podpore tako s strani profesionalcev kot skupnosti. Ker sta VoltDB in NuoDB relativno novi tehnologiji, je podpora manjša in omejena na razvijalce produkta, a pridobivata na prepoznavnosti in posledično tudi na podpori skupnosti.

**Različice** MySQL teče na vseh operacijskih sistemih, ne glede na arhitekturo, NuoDB in VoltDB pa le na 64-bitnih operacijskih sistemih. VoltDB je le za Linux, NuoDB pa neodvisen od operacijskega sistema.

MySQL je prosto dostopen. NuoDB ponuja razvijalsko (neomejena uporaba v razvijalskem okolju), zastonjsko (omejena na dva gostitelja v produkcijskem okolju brez podpore geo-porazdeljenosti), profesionalno (neomejeno število gostiteljev v produkcijskem okolju brez podpore geo-porazdeljenosti) in podjetniško različico (neomejeno število gostiteljev v produkcijskem okolju s podporo geo-porazdeljevanju). VoltDB ima podjetniško (s poskusno dobo 1 meseca) in odprtokodno različico (ne vsebuje funkcij trajnosti in dostopnosti).

# Poglavje 5

# Preizkus zmogljivosti podatkovnih baz NewSQL

Pri diplomski nalogi je bila uporabljena tehnika preizkusa zmogljivosti sistema (angl. *benchmarking*), pri katerem se preizkuša odziv sistema pri različnih in večinoma ekstremnih pogojih. Pri SUPB-jih se lahko pri tem nek SUPB primerja z drugim SUPB-jem, opazuje določeno mersko količino (na primer število izvedenih transakcij na časovno enoto, zamik med zahtevo in odgovorom) ali pa se isto aplikacijo požene na različnih nastavitvah strojne opreme. V diplomski nalogi je bil preizkus narajen z vidika opazovanja števila izvedenih transakcij na sekundo.

## 5.1 Odjemalec

### 5.1.1 Platforma Node.js

Obe bazi ponujata gonilnike za pisanje odjemalcev v vseh popularnejših programskih jezikih (gonilnik podatkovne baze je knjižnica, ki ponuja metode za dostop do SUPB-ja ter izvajanje transakcij). Nekaj jih je uradno podprtih s strani razvijalcev baze, veliko pa jih je tudi takih, ki jih vzdržuje skupnost.

Za razvoj odjemalca je bil pri testu obeh SUPB-jev uporabljen gonilnik za Node.js. Izbiran je bil na podlagi zmožnosti ustvarjanja asinhronih klicev,

preprostega simuliranja več sočasnih uporabnikov ter aktualnosti pri razvoju modernih spletnih aplikacij. Z uporabo asinhronih klicev se namreč ciljni sistem lahko obremeni najbolj.

Node.js je platforma, ki razvijalcem omogoča pisanje strežniških aplikacij v jeziku JavaScript na vseh večjih operacijskih sistemih. Kodo v javascriptu v realnem času prevaja Googlov pogon V8, ki je vgrajen v brskalnik Google Chrome in je odprtokoden.

V nasprotju s podobnimi tradicionalnimi tehnologijami, kjer ena zahteva ustvari novo nit in s tem zasede nekaj sistemskih virov, v Node.js obstaja samo ena nit. Večuporabniško delovanje je dosežena tako, da operacije ne blokirajo izvajanja programa. Za to se uporablja asinhrone klice (operacija ne čaka na odgovor), vhodno-izhodne operacije, ki so glavni vir režij, pa se izvajajo v ozadju. Ko je rezultat na voljo, se s pomočjo dogodkov izvede povratni klic, ki je bil prvotno podan operaciji, ki je izvedla asinhorni klic. To omogoča pisanje visoko prepustnih aplikacij, kjer najbolj izstopajo realnočasovne.

Zaradi asinhronih klicev je bil odjemalec programiran tako, da se je po določenem času izvajanja zahtev zaključil ter shranil rezultat.

### 5.1.2 Implementacija

Za preizkušanje so bile uporabljene tri vrste poizvedb: vnos, iskanje in spreminjanje podatkov. Pri vsaki poizvedbi se dostopa do ene tabele.

Poglavitna razlika med odjemalcem v NuoDB in odjemalcem v VoltDB je poleg vmesnika tudi v implementaciji za povezavo do vseh strežnikov v gruči podatkovne baze:

- Pri VoltDB se odjemalca pri ustvarjanju povezave poveže z vsemi strežniki v gruči, ki poganjajo podatkovne baze in kar omogoča že sam gonilnik. To je razvidno iz priloženega izseka kode odjemalca A.2).

- Pri NuoDB je potrebno ustvariti več povezav do same baze in jih združiti v bazen (angl. `pooling`), saj je narava NuoDB taka, da ko

odjemalec naveže stik z bazo, mu posrednik dodeli transakcijski pogon, s katerim nato odjemalec komunicira. Ker je odjemalec v Node.js iz ene niti, se zato uporablja samo en transakcijski pogon.

Ko aplikacija zahteva podatek iz baze, najprej pridobi prosto povezavo iz bazena povezav. Ko pa je poizvedovanje končano pa se povezavo spusti nazaj v bazen. Upravljanje s povezavami je implementirano s pomočjo zunanje odprtokodne knjižnice. To je razvidno tudi iz izsek kode A.1, ki prikazuje pomembne dele odjemalca.

Izseki kode, ki prikazujejo povezovanja do strežnikov in dostop do baze, so tako za NuoDB ( A.1) kot VoltDB ( A.2) priloženi diplomskemu delu.

## 5.2 Namestitev in nastavitev NuoDB

### 5.2.1 Zagon procesa baze

Programski paket se namesti na strežnike, pripravljene za tvorjenje gruče podatkovne baze, na vsakem se posodobi nastavitveno datoteko (vrsto procesa, ime in geslo domene, ip ali naslov strežnika, ki se mu priključuje) in nato zažene proces podatkovne baze. Domeno se nato upravlja (kreiranje baz, dodajanje/izključevanje pogonov) preko spletne konzole oziroma programa, ki se izvaja preko ukazne vrstice.

Pri privzetih nastavitvah se je na enem strežniku gruča ustvarila preko podloge 'En strežnik', pri večih strežnikih pa je bila uporabljena podloga 'Več strežnikov'. Nastavitev gruče pri tej bazi se je spreminjala z dodajanjem dodatnih pomnilnih pogonov na strežnikih, kjer jih podloga 'Več strežikov' ni ustvarila.

### 5.2.2 Kreiranje podatkovne baze

Po ustvarjeni domeni se lahko v njej ustvari eno ali več podatkovnih baz. Slednjo se ustvari preko programa, ki se izvaja preko ukazne vrstice in pri

katerem se poda pot do direktorijev, kjer se bosta nahajali arhiv ter dnevnik baze. Preden se začnejo definirati tabele, indeksi in omejitve, je potrebno ustvariti še shemo (nekakšen imenski prostor znotraj ene baze). Pri definiranju shem, tabel, indeksov in raznih omejitev se uporablja jezik SQL, ki se lahko vnaša tudi preko spletne konzole. Za potrebe diplomske naloge se je v domeni ustvarila ena podatkovna baza z eno shemo. V njej pa tabela `users` na način, ki se uporablja tudi pri drugih bazah SQL.

## 5.3 Namestitev in nastavitev VoltDB

### 5.3.1 Zagon procesa baze

Pri zaganjanju procesa podatkovne baze v VoltDB je le-temu potrebno podati katalog (pri VoltDB ne obstaja 'prazna' podatkovna baza) ter nastavitveno datoteko gruče (opcijsko). Katalog je datoteka `jar`, ki se s pomočjo programa `voltdb` prevede iz `sql` datoteke. Vsebuje definicije shem, postopkov ter navodila o fragmentiranju tabel in postopkov. Bolj kompleksne postopke je možno pisati s pomočjo programskega jezika Java. V nastavitveni datoteki pa so navedene lastnosti gruče baze, kot so: koliko strežnikov se pričakuje v gruči, koliko fragmentov naj ima en strežnik, kolikšna je vrednost `k`, ali naj omogoča sprejem zahtev preko prenosnega protokola HTTP, na koliko časa se ustvarjajo posnetki baze, kje se shranjujejo, koliko je zadnjih različic.

Za priključitev več strežnikov v gruči je potrebno za zagon baze na vsakem strežniku zagnati enak ukaz. Ta vsebuje pot do kataloga, pot do nastavitvene datoteke in ime gostitelja, ki deluje kot koordinator na novo priključenih strežnikov. Šele ko je priključenih toliko strežnikov, kot je navedeno v nastavitveni datoteki gruče, SUPB začne delovati. Kot druga rešitev proti temu ponavljajočemu postopku pa se lahko uporabi Upravitelja, ki opravi nameščanje in zaganjanje na želenih strežnikih namesto uporabnika. Za delovanje Upravitelja je potrebno pred tem strežnike pripraviti, da omogočajo oddaljeni dostop preko varne lupine (angl. *secure shell*, v nadaljevanju SSH) s pomočjo ključev (da ni potrebe po vpisovanju gesel) ter zagotoviti, da ima

oddaljen uporabnik pravice za pisanje v direktorij.

Pri diplomski nalogi so bili parametri število strežnikov v gruči, število fragmentov na strežniku ter faktor k-varnosti tisti, s katerimi se je največ preizušalo. Večji ko je faktor za k-varnost, manj enoličnih fragmentov ima baza na voljo ter več procesorske moči se porabi zaradi vzdrževanja konsistentnosti kopij.

### 5.3.2 Katalog

Katalog vsebuje definicijo relacije `helloworld` z atributi o jeziku, prevodu za besedo hello in prevodu za besedo world. Relacija je fragmentirana po vrednosti atributa o jeziku.

Vsebuje še tri definicije postopkov iz prevedenih Javinih razredov: vnos, spreminjanje in iskanje pozdrava v nekem jeziku. Pri pisanju postopka je na voljo bogat API iz paketa `org.voltdb`, v osnovi pa sestoji iz definiranja stavka SQL z razredom `SQLStmt`, postavitvijo poizvedbe SQL v vrsto z razredom `voltQueueSQL` ter vrnitvijo rezultata poizvedbe z `voltExecuteSQL` (razvidno iz priloge B.2). Take osnovne postopke sicer generira že VoltDB sam iz informacij, ki jih pridobi iz kataloga ob prevajanju.

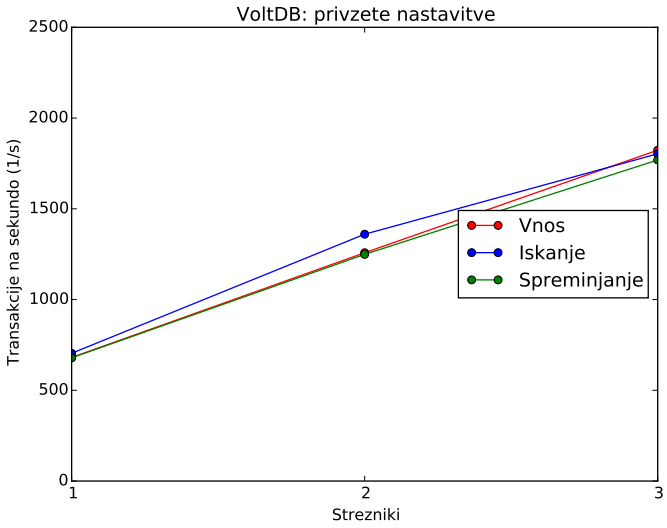
Ključ za fragmentiranje tako relacij kot postopkov je v veliko primerih atribut, po katerem se filtrira vrstice poizvedbe (v sklopu WHERE poizvedbe SQL), kar je razvidno tudi iz priloge B.1

# Poglavje 6

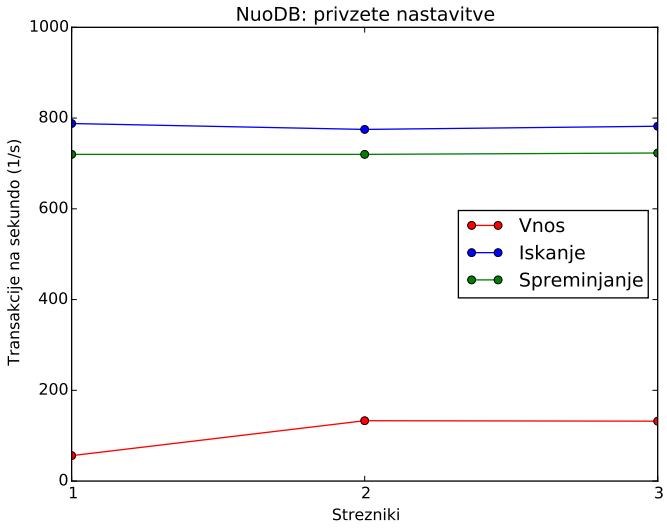
# Rezultati

Grafa 6.1 (VoltDB) in 6.2 (NuoDB) prikazujeta, kako se vrednost količine transkacij na sekundo spreminja pri različnemu številu strežnikov, ki strežejo bazo. Za VoltDB se da razbrati, da se razširja in da hitrost operacije vnosa in spreminjanja (pisalnih operacij) nič ne zaostajajo za hitrosti operacije iskanja (bralne operacije). Pri NuoDB pa se razbere, da se baza ne razširja in tudi razlike med bralnimi in pisalnimi operacijami so očitne. Primerjava obeh grafov med sabo pa pokaže, da pri enem strežniku NuoDB kljub siceršnji nerazširljivosti v primerjavi z VoltDB doseže boljše rezultate pri operacijah iskanja in spreminjanja.

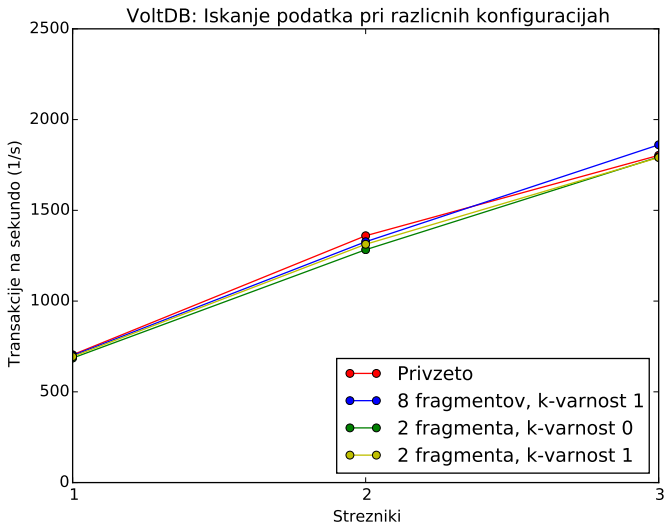
Primerjave rezultatov pri spreminjanju nastavitev gruče pri VoltDB za vnos, iskanje in spreminjanje zapisov prikazujejo grafi 6.3, 6.4 in 6.5. Po trditvah proizvajalca baze bi morala nastavitev gruče z dvema fragmentoma (črta z zeleno in rumeno barvo) prikazati boljše rezultate od ostalih dveh nastavitev, saj so se za preizkus uporabljeni procesorji z dvema jedroma. Na grafu 6.3 izredno odstopa operacija vnosa pri nastavitvah gruče z 8 fragmenti in k-varnostjo 1, kar bi lahko bila posledica motenj v gruči oziroma nepravilnega razmerja med številom fragmentov ter številom jeder enega strežnika. Na splošno pa rezultati nihajo, kar namiguje na to, da se bo SUPB v različnih primerih uporabe različno obnesel in za dosego najboljšega delovanja bo kupec moral izvesti preizkus zmogljivosti SUPB-ja na svojih podatkih.



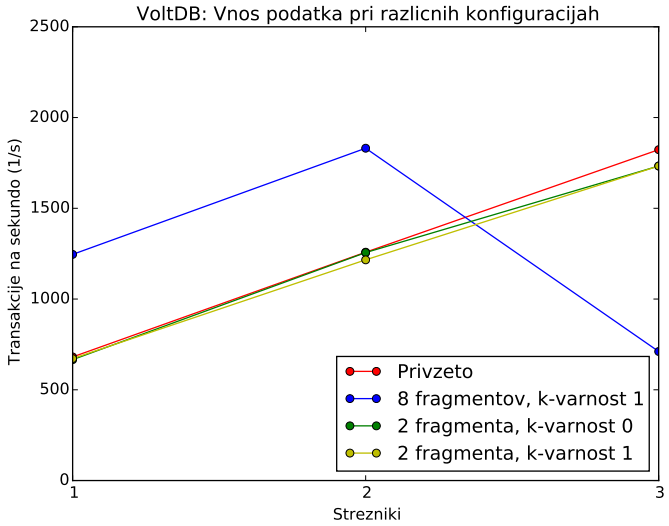

Slika 6.1: Vnos, iskanje in spreminjanje zapisov nad bazo pri VoltDB pri privzetih nastavitvah gruče.



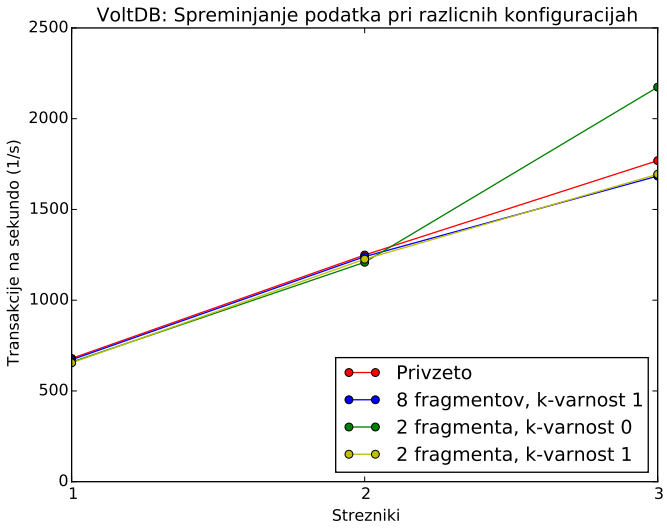

Slika 6.2: Vnos, iskanje in spreminjanje zapisov nad bazo pri NuoDB pri privzetih nastavitvah gruče.




Slika 6.3: Vnos zapisov nad bazo pri VoltDB pri različnih nastavitvah gruče.



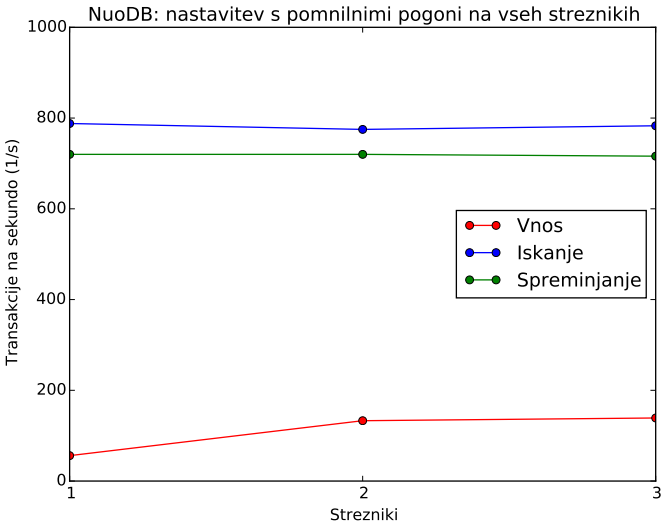

Slika 6.4: Iskanje zapisov nad bazo pri VoltDB pri različnih nastavitvah gruče.



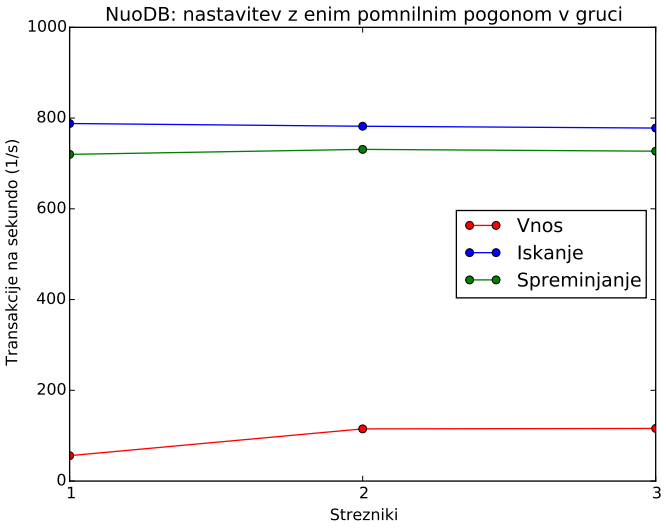

Slika 6.5: Spreminjanje zapisov nad bazo pri VoltDB pri različnih nastavitvah gruče.

Grafa 6.6 in 6.7 prikazujeta vnos, iskanje in spreminjanje zapisov nad bazo NuoDB pri različnih nastavitvah gruče. Rezultati so identični, kar pomeni, da dodajanje ali ukinjanje pomnilnih pogojev ne vpliva v veliki meri na število izvedenih transakcij na sekundo. Problem z nerazširljivostjo pa je še vedno prisoten.

Vsi rezultati, strnjeni v tabeli 6.1 (celico sestavljajo tri številke, ki predstavljajo količino število transakcij na sekundo pri 1, 2 in 3 strežnikih v gruči). Pri NuoDB je `pp` okrajšava za pomnilni pogon, pri VoltDB `f` za fragment, `k` pa za faktor k-varnosti.




Slika 6.6: Vnos, iskanje in spreminjanje zapisov nad bazo NuoDB pri nastavitvah gruče s pomnilnim pogonom na vsakem strežniku.




Slika 6.7: Vnos, iskanje in spreminjanje zapisov nad bazo NuoDB pri nastavitvah gruče z enim pomnilnim pogonom.

| nastavitev/poizvedba | vnos | iskanje | spreminjanje |
|---|---|---|---|
| VoltDB, privzeto | 681, 1258, 1823 | 704, 1360, 1803 | 679, 1249, 1769 |
| VoltDB, 8 f, 1 k | 1246, 1831, 711 | 700, 1327, 1861 | 672, 1240, 1684 |
| VoltDB, 2 f, 0 k | 666, 1256, 1733 | 686, 1283, 1794 | 659, 1209, 2174 |
| VoltDB, 2 f, 1 k | 672, 1216, 1733 | 694, 1313, 1791 | 655, 1226, 1694 |
| NuoDB, privzeto | 56, 133, 132 | 788, 775, 782 | 720, 720, 723 |
| NuoDB, 1 pp | 56, 115, 116 | 788, 782, 778 | 720, 731, 727 |
| NuoDB, 3 pp | 56, 133, 139 | 788, 775, 783 | 720, 720, 716 |

Tabela 6.1: Vsi rezultati meritev

# Poglavje 7

# Sklepne ugotovitve

Sodeč po raziskovanju na internetu bi morala biti oba SUPB-ja skoraj linearno razširljiva z vsakim dodanim strežnikom. Rezultati v diplomski nalogi to potrjujejo le delno. Medtem ko je pri VoltDB razširljivost očitna, pa pri NuoDB temu ni tako (malo se kaže le pri poizvedbi vnosa). Razlogov za to je lahko več: strojna oprema, komunikacija med strežniki. Pri poskusu razreševanja težave je bil vzpostavljen kontakt z uradno podporo pri NuoDB, vendar do rešitve v zastavljenem roku ni prišlo.

Sledi še nekaj ugotovitev med primerjanjem obeh SUPB-jev:

- Vzpostavitev porazdeljenega sistema je pri obeh bazah relativno enostavno. Razlikujeta se v namestitvi programskega paketa na strežnike. Pri NuoDB je potrebno programski paket namestiti na vsak strežnik, pri VoltDB pa le na enem, saj lahko na ostale strežnike programski paket skupaj s katalogom porazdeli Upravitelj VoltDB. Ker slednje poteka preko SSH, je na vsakem strežniku zato potrebno namestiti SSH programsko opremo.

- Pri obeh bazah je potrebno spremeniti nastavitvene podatke pred zagonom. VoltDB preko spletnega Upravitelja VoltDB omogoča centralno urejanje nastavitev gruče, pri NuoDB pa je potrebno podatke za nastavitev gruče spreminjati na vsakem strežniku.

- Za preizkus je bila opravljena tudi vzpostavitev gruče podatkovne baze na treh fizičnih strežnikih ter 3 navideznih strežnikih, kjer slednji sprva niso zadostovali priporočenim strojnim zahtevam obeh baz (vrsta procesorja ter manjša količina spomina). Bilo je ugotovoljeno, da se je pri NuoDB baza vzpostavila, pri VoltDB pa ne, saj baza pri vzpostavljenju zasede določeno količino spomina, pomembno pa je, da ima procesor vsaj dve jedri. Na koncu navidezni računalniki niso bili vključeni v preizus zaradi prevelike razlike v strojni opremi.

- Delo z NuoDB bolj spominja na delo z običajno relacijsko podatkovno bazo (delo z bazo, uporabo gonilnikov za dostop do baze, ki uporabljajo standarden vmesnik), kot pa delo z VoltDB. Glavna razlika VoltDB z NuoDB je, da je potrebno pisati shranjene postopke in načrtovati transakcije, ki se bodo večinoma izvajale le na enem fragmentu. A po drugi strani VoltDB omogoča več nastavitev in boljšo prilagodljivost dani strojni opremi.

- Preizkušanje baze pri različnih nastavitvah je bilo lažje pri VoltDB, saj ni bilo potrebe po spreminjanju procesa baze pri vsaki drugačni nastavitvi gruče. Število strežnikov, ki strežejo bazo, se je dalo regulirati kar preko odjemalca.

VoltDB in NuoDB sodita med trenutno najbolj znane predstavnike podatkovnih baz NewSQL. Slednje imajo potencial prodreti na področje, kamor podatkovnim bazam NoSQL verjetno ne bo uspelo - na poslovno področjo, kjer se posluje z občutljivimi podatki in je konsistentnost podatkov izrednega pomena. A ker ta tehnologija še ni dovolj preizkušena in mlada, bo verjetno še nekaj časa ostala v domeni podjetij, ki poslujejo preko spleta in uporabljajo najnovejše tehnologije kot konkurenčno prednost.

# Dodatek A

# Odjemalca

## A.1 NuoDB

```
var nuodb = require('db-nuodb'),
    genericPool = require('generic-pool'),
    config = require('./config.json');

var pool = genericPool.Pool({
    max: 20,
    create: function(callback) {
        new nuodb.Database(config)
            .connect(function(error) {
                callback(error, this);
            });
        },
    destroy: function(db) {
        db.disconnect();
    }
});

global.select = function () {
    pool.acquire(function (error, client) {
        client.query().select('*').from('users').where('email = ?', ['
            jernej@example.com'])
            .execute(function (error, result) {
                pool.release(client);
            });
    });
}
```

## A.2 VoltDB

```
var voltjs = require('voltjs'),
    servers = ['192.168.25.123', '192.168.25.103', '192.168.25.121'];

var querySelect = (new voltjs.VoltProcedure('Select', ['string'])).getQuery
    ();

var configs = servers.map(function(server) {
    var cfg = new voltjs.VoltConfiguration();
    cfg.host = server;
    return cfg;
});

var client = new voltjs.VoltClient(configs);

global.select = function () {
    querySelect.setParameters(['slovenian']);
    client.callProcedure(querySelect, function(errorCode, eventCode, results
        ) {

    });
}
```

# Dodatek B

# VoltDB

## B.1 Katalog

```
CREATE TABLE HELLOWORLD (
   HELLO VARCHAR(15),
   WORLD VARCHAR(15),
   LANGUAGE VARCHAR(15) NOT NULL,
   PRIMARY KEY (LANGUAGE)
);

PARTITION TABLE HELLOWORLD ON COLUMN LANGUAGE;

CREATE PROCEDURE FROM CLASS Insert;
CREATE PROCEDURE FROM CLASS Select;
CREATE PROCEDURE FROM CLASS Update;

PARTITION PROCEDURE Select ON TABLE HELLOWORLD COLUMN LANGUAGE;
PARTITION PROCEDURE Insert ON TABLE HELLOWORLD COLUMN LANGUAGE;
PARTITION PROCEDURE Update ON TABLE HELLOWORLD COLUMN LANGUAGE;
```

## B.2 Postopek za vnos

```
import org.voltdb.*;

public class Insert extends VoltProcedure {

    public final SQLStmt insert = new SQLStmt("INSERT INTO HELLOWORLD VALUES
         (?,?,?);");
```

```
    public VoltTable[] run( String language, String hello, String world)
        throws VoltAbortException {
        voltQueueSQL(insert, language, hello, world);
        return voltExecuteSQL();
    }
}
```

# Literatura


[1] C. Coronel, S. Morris, P. Rob. Database Systems: Design, Implementation, and Management, 9th edition. Cengage Learning. 2012.

[2] T. M. Connolly, C. E. Begg. Database Systems, A Practical Approach to Design, Implementation and Management, Fourth Edition, Addison-Wesley, 2005.

[3] ACID, Wikipedia: The Free Encyclopedia. Zadnjič posodobljeno 15. 10. 2014. Dosegljivo na: http://en.wikipedia.org/wiki/ACID [Dostopano septembra 2014].

[4] A Brief History of SQL. Dosegljivo na: http://www.w3computing.com/sqlserver/brief-history-sql [Dostopano septembra 2014]

[5] Meaning of Scalability. Dosegljivo na: `http://www.hyperdictionary.com/search.aspx?define=scalability` [Dostopano septembra 2014].

[6] I. Gorton. Essential Software Architecture, 2nd edition. Springer-Verlang Berlin Heidelberg, 2011.

[7] Scalability, Wikipedia: The Free Encyclopedia. Zadnjič posodobljeno 19. 9. 2014. Dosegljivo na: `http://en.wikipedia.org/wiki/Scalability` [Dostopano septembra 2014].

[8] Replication (computing), Wikipedia: The Free Encyclopedia. Zadnjič posodobljeno 24. 9. 2014. Dosegljivo na: http://en.wikipedia.org/wiki/Replication_(computing) [Dostopano septembra 2014]

[9] Partition (database), Wikipedia: The Free Encyclopedia. Zadnjič posodobljeno 9. 7. 2014. Dosegljivo na: http://en.wikipedia.org/wiki/Partition_(database) [Dostopano septembra 2014]

[10] B. Schwarz, P. Zaitsev, Vladim Tkachenko. High Performance MySQL: Optimization, Backups, and Replication , 3rd edition. O'Reilly Media, Inc. 2012

[11] NoSQL, Wikipedia: The Free Encyclopedia. Zadnjič posodobljeno 15. 10. 2014. Dosegljivo na: http://en.wikipedia.org/wiki/NoSQL [Dostopano oktobra 2014].

[12] NewSQL, Wikipedia: The Free Enyclopedia. Zadnjič posodobljeno 22. 8. 2014. Dosegljivo na: http://en.wikipedia.org/wiki/NewSQL [Dostopano septembra 2014].

[13] P. Venkatesh, Nirmala S. NewSQL - The New Way to Handle Big Data. Dosegljivo na: http://www.opensourceforu.com/2012/01/newsql-handle-big-data [Dostopano septembra 2014].

[14] M. Stonbraker, R. Cattell. 10 Rules for Scalable Performance in 'Simple operation' Datastores. Dosegljivo na: http://delivery.acm.org/10.1145/1960000/1953144/p72-stonebraker.pdf [Dostopano oktobra 2014].

[15] S. Harizopoulos, D. J. Abadi, S. Madden, M. Stonebraker. OLTP through the looking glass, and what we found there. Dosegljivo na:

http://dl.acm.org/citation.cfm?id=1376713 [Dostopano septembra 2014].

[16] NuoDB. NuoDB Emergent Architecture. Dosegljivo na: http://go.nuodb.com/rs/nuodb/images/Greenbook_Final.pdf [Dostopano septembra 2014].

[17] Seth Proctor. Exploring the Architecture of the NuoDB Database, Part 1. Dosegljivo na: http://www.infoq.com/articles/nuodb-architecture-1 [Dostopano septembra 2014].

[18] VoltDB. Techical overview. Dosegljivo na: http://downloads.voltdb.com/datasheets_collateral/technical_overview.pdf [Dostopano septembra 2014].

[19] H-Store Documentation: Frequently Asked Questions. Dosegljivo na: http://hstore.cs.brown.edu/documentation/faq [Dostopano decembra 2014].

[20] VoltDB. Application Brief: Transactions. Dosegljivo na: http://downloads.voltdb.com/app_briefs/voltdb_transactions.pdf [Dostopano septembra 2014].

[21] VoltDB. Application Brief: Partitioning. Dosegljivo na: http://downloads.voltdb.com/app_briefs/voltdb_partitioning.pdf [Dostopano septembra 2014].

[22] VoltDB. Application Brief: Durability. Dosegljivo na: http://downloads.voltdb.com/app_briefs/voltdb_durability.pdf [Dostopano septembra 2014].

[23] VoltDB. Application Brief: Availability. Dosegljivo na: http://downloads.voltdb.com/app_briefs/voltdb_availability.pdf [Dostopano septembra 2014].

[24] VoltDB. Planning Guide. Dosegljivo na:
`http://downloads.voltdb.com/documentation/PlanningGuide.pdf`
[Dostopano septembra 2014].