UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Rok Žlender

PRIMERJAVA ZMOGLJIVOSTI NERELACIJSKIH PODATKOVNIH BAZ

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Marko Bajec

Ljubljana, 2011

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01787/2011
Datum: 02.11.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ROK ŽLENDER**

Naslov: **PRIMERJAVA ZMOGLJIVOSTI NERELACIJSKIH PODATKOVNIH BAZ**

PERFORMANCE COMPARISON OF NON-RELATIONAL DATABASE SYSTEMS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Relacijske podatkovne baze že dlje časa niso več kos vsem izzivom, s katerimi se danes srečujemo na področju hranjenja in obdelave velikih količin podatkov. Pojavljajo se alternativne rešitve, ki jih enotno imenujemo noSQL. V nalogi opišite različne noSQL rešitve. Izbrane rešitve namestite v enega izmed računalniških oblakov ter rešitve primerjajte med seboj s poudarkom na merjenju hitrosti osnovnih operacij, ki jih ponujajo.

Mentor:
prof. dr. Marko Bajec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Rok Žlender,

z vpisno številko 63020188,

sem avtor/-ica diplomskega dela z naslovom:

Primerjava zmogljivosti nerelacijskih podatkovnih baz

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom
  prof. dr. Marko Bajec
- 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 v zbirki "Dela FRI".

V Ljubljani, dne 12.12.2011 Podpis avtorja/-ice:

Zahvala

Zahvaljujem se mentorju dr. Marku Bajcu za pomoč in koristne nasvete pri izdelavi diplomskega dela.

Posebna zahvala gre moji družini, ki mi je stala ob strani skozi celoten študij in zadnja leta spodbujala k dokončanju. Tamara hvala za podporo, nasvete in spodbudne besede.

Na koncu bi se rad zahvalil še vsem razvijalcem odprtokodnih projektov. Brez njihovega dela večina programske opreme, uporabljene pri tej diplomski nalogi, ne bi bila nikoli razvita.

Kazalo

Seznam uporabljenih kratic in simbolov

ACID ACID (atomicity, consistency, isolation, durability) so pravila, ki zagotavljajo, da transakcije v podatkovni bazi vodijo iz enege veljavnega stanja v drugo veljavno stanje.

HDFS HDFS (Hadoop Distributed Filesystem) je porazdeljen datotečni sistem osnovan na Google File System datotečnem sistemu.

JSON JSON (JavaScript Object Notation) je enostaven in berljiv standard za zapis podatkov, izvira iz jezika JavaScript, zanj obstajajo knjižnjice za branje in pisanje za večino programskih jezikov.

BSON BSON (Binary JSON) Binarni zapis podatkov, ki ga uporablja MongoDB.

EC2 EC2 (Elastic Cloud Computing) Amazonova storitev strežnikov v oblaku.

EBS EBS (Elasitc Block Store) Amazonov diskovni podsistem.

JDBC JDBC (Java DataBase Connectivity) Java API za dostop do relacijskih podatkovnih baz.

YCSB YCSB (Yahoo! Cloud System Benchmark) Orodje za merjenje zmogljivosti podatkovnih baz.

CAP CAP (Consistency, Availability in Partition Tolerance) CAP teorem konsistentnost, dosegljivost in odpornost na probleme v komunikaciji.

Povzetek

Izbira sistema za shranjevanje podatkov je ena izmed bolj pomembnih odločitev pri vsakem projektu. Na izbiro niso več le relacijske podatkovne baze, ampak se vedno bolj uveljavljajo nerelacijske podatkovne baze. Nerelacijske podatkovne baze predstavljajo zanimivo alternativo, kadar imamo opravko z veliko količino podatkov ali pa želimo večjo fleksibilnost pri podatkovnem modelu.

Namen diplomskega dela je preizkusiti in izmeriti, kako se izbrane nerelacijske baze primerjajo med seboj in z relacijsko podatkovno bazo MySQL. Ker se izbrane podatkovne baze razlikujejo po namenu uporabe, sem tudi v meritvah izbral različne scenarije uporabe. Na začetku so predstavljene izbrane podatkovne baze, njihove lastnosti in podprti načini porazdelitve podatkov. Sledi predstavitev testnega okolja, ki predstavi programsko in strojno opremo, ki sem jo uporabil pri meritvah. Zaključek je namenjen predstavitvi in analizi rezultatov, pridobeljnih v meritvah.

Ključne besede:

nerelacijske podatkovne baze, NoSQL, merjenje zmogljivosti, Amazon EC2, YCSB

Abstract

Deciding on which data store to use is one of the most important aspects of every project. Besides the established relational database systems non-relational solutions are gaining in their popularity. Non-relational database systems provide an interesting alternative when we are storing large amount of data or when we are looking for greater flexibility with our data model.

Purpose of this thesis is to measure and analyze how chosen non-relational database systems compare against each other and against a relational database MySQL. Because database systems vary widely by their intended use cases I selected different test scenarios. In first chapters selected database systems, their features and their solutions to distributing data are described. Next the test environment is presented describing selected hardware and software used in performance analysis. Finally test results are presented and analyzed.

Key words:

non-relational database systems, NoSQL, performance analysis, Amazon EC2, YCSB

Poglavje 1

Uvod

Velikost podatkov, spravljenih v računalnikih, po celem svetu neprestano raste. Velika spletna podjetja kot so Amazon, Google in Facebook imajo ogromno uporabnikov in njihovi uporabniki pričukujejo ažurne in točne podatke. Poleg tega morajo biti podatki na voljo v izjemno kratkem času in tudi v primeru izpada dela sistema. Razvile so se nerelacijske podatkovne baze, ki ponujajo rešitev opisanih problemov. Seveda tudi nerelacijske podatkovne baze ne rešijo vseh problemov. Za doseganje želenih lastnosti so avtorji morali poseči po kompromisih. Brewerjev teorem ali CAP teorem pravi, da porazdeljen sistem lahko zadosti le dvema izmed treh lastnosti:

- Porazdeljen sistem mora biti konsistenten. V vsakem trenutku vrne pravo vrednost določenega podatka.
- Porazdeljen sistem mora biti vedno dosegljiv. Sistem v vsakem trenutku lahko poda odgovor na uporabnikovo zahtevo.
- Porazdeljen sistem mora biti odporen na izgubo povezave med vozlišči. V primeru prekinitive poveze med vozlišči, se sistem po ponovični vzpostavitvi, vrne v pravilno stanje.

Razvoj nerelacijskih podatkovnih baz se je v zadnjem času zelo razširil in različne podatkovne baze izbereje različni dve lastnosti CAP teorema. Izbira je prepuščena uporabniku glede na zahteve projekta.

Namen diplomskega dela je predstavitev in analiza zmogljivosti nerelacijskih podatkovnih baz. Analiza zmogljivosti se je izvajala v gruči več strežnikov in pri različnih scenarijih uporabe. Prav tako bom predstavil kako namestiti izbrane produkte v gruči ter kako izvajati meritve.

Diplomsko delo vsebuje pet poglavji. V uvodnem poglavju sta predstavljena problemsko področje in namen diplomskega dela. V drugem poglavju sledi predstavitev izbranih podatkovnih baz, njihovih posebnosti, podatkovnih struktur in načinov replikacije. V tretjem poglavju podrobno predstavim testno okolje. Opis testnega okolja vsebuje opis strojne opreme, programske opreme in izbranih scenarijev, ki sem jih izbral za testiranje. V četrtem poglavju sem predstavil in analiziral dobljene rezultate. V zaključku sem podal sklepne ugotovitve in povzamem rezultate. Dodatek vsebuje skripte razvite za zagon testnih okolji.

Poglavje 2

Nerelacijske podatkovne baze

Nerelacijske podatkovne baze niso bile razvite z namenom popolne zamenjave relacijskih. Pri nekaterih rešitvah bi nas toga shema relacijskih podatkovnih baz ovirala in so nerelacijske baze s svojo fleksibilnostjo prava izbira. Prav tako relacijske baze niso najbolj primerne za reševanje problemov z ogromno količino podatkov. Tabele ali zbirke v nerelacijskih podatkovnih bazah lahko tečejo na tisočih računalnikih in vsebujejo več tera bytov podatkov. Pri tem pa podatkovne baze avtomatično skrbijo za želeno konsistentnost in poskrbijo, da problemi s posameznimi strežniki ne ogrozijo celotne gruče.

2.1 Redis

Redis je odprtokodni podatkovni strežnik, ki zavoljo hitrosti nabor podatkov shranjuje v pomnilniku. Redis omogoča več kot enostavne ključ-vrednost shrambe. Vrednost lahko vsebuje znakovne nize, zgoščene vrednosti, sezname ter urejene sezname. Prav tako Redis omogoča nekaj osnovnih operacij nad vrednostjo, ki bi jih drugače morali opraviti v sami aplikaciji.

Redis je bil zasnovan leta 2009 kot ključ-vrednost shramba, ki bi olajšala in pospešila razvoj aplikacije za analizo spletnega obiska. Razvoj je začel Salvatore Sanfilippo, sedaj ves razvoj financira podjetje VMWare [9]. Dve leti kasneje se Redis uporablja v malih in velikih podjetjih za različne namene. Podjetje Github, ki ponuja gostovanje za sistem nadzora izvorne kode git, uporablja Redis za čakalno vrsto in shranjevanje nastavitev sistema [1]. Spletna stran Digg, ki omogoča uporabnikom deljenje in odkrivanje zanimivih vsebin, je vpeljala Redis kot števec obiskanosti deljenih spletnih strani v njihovem sistem [2].

2.1.1 Podatkovni model

Ključ v Redis bazi je binarno varen kar pomeni, da lahko vsebuje navaden niz znakov ali pa binarni zapis slike. Ker so v ključu dovoljeni vsi znaki, lahko ključ razdelimo v imenske prostore in s tem dodamo strukturo podatkov. Na primer uporabnik:1234 oziroma uporabnik:1234:geslo sta dva primera ključa, kjer z dvopičjem razdelimo ključ.

Niz znakov

Najenostavnejši podatkovni tip nam omogoča shranjevanje znakovnih nizov, poleg tega omogoča tudi nekatere napredne operacije. Atomarno povečanje vrednosti INCR bo pretvorilo niz v celo število, dodalo vrednost ena in vrednost shranilo nazaj. Obstajajo tudi ukazi INCRBY (poveča vrednost za podano število), DECR in DECRBY, ki odštejeta podano število.

```
>redis-cli set kljuc "Niz znakov shranjen v Redis strežniku"
OK
>redis-cli set uporabnik:1:ogledi 1
(integer) 1
>redis-cli incr uporabnik:1:ogledi
(integer) 2
```

Seznam

Seznami so implementirani kot povezani seznami. Povezan seznam omogoča dodajanje elementov na začetek in konec seznama v konstantnem času. Slabost povezanih seznamov pa je, da dostop do elementov z indeksom ni tako hiter. Seznami nam omogočajo hitro in enostavno implementacijo aplikacij kot so npr. čakalne vrste, katerim ustreza, da so podatki urejeni po vrstnem redu v katerem jih dodajamo v seznam. Ukazi za delo s seznami so zelo preprosti. RPUSH doda element na konec seznama, LPUSH doda element na zečetek seznama ter LRANGE, ki vrne elemente seznama.

```
>redic-cli RPUSH message-queue "Prvi element vrste"
OK
>redis-cli RPUSH message-queue "Drugi element vrste"
OK
>redis-cli LRANGE 0 1
1. Prvi element vrste
```

Množice

Redis množice so neurejene zbirke znakovnih nizov. Nad množicami lahko izvajamo vrsto operacij kot so unija, presek, razlika, poizvedba, če je element v množici, ter seveda dodajanje elementa v množico. Z množicami se lahko enostavno implementira razmerja med podatki.

```
>redis-cli sadd uporabnik:1:prijatelji 2
(integer) 1
>redis-cli sadd uporabniki:1:prijatelji 3
(integer) 1
>redis-cli sismember uporabniki:1:prijatelji 2
(integer) 1
>redic-cli sismember uporabniki:1:prijatelji 4
(integer) 0
```

Urejene množice

Urejene množice so, tako kot množice, zbirka znakovnih nizov s to razliko, da ima vsak element v množici tudi oceno po kateri Redis uredi množico. Redis elemente ureja ob shranjevanju, tako da je branje podatkov izredno hitro.

```
>redis-cli zadd temperature:2011 5 "Januar"
(integer) 1
>redis-cli zadd temperature:2011 -2 "Februar"
(integer) 1
>redis-cli zadd temperature:2011 25 "August"
(integer) 1
>redis-cli zrange temperature:2011 0 -1
1. Februar
2. Januar
3. August
```

2.1.2 Replikacija

Trenutna verzija ne nudi avtomatske porazdelitve podatkov na več strežnikih. Če želimo porazdeliti obremenitev, je potrebno za to poskrbeti v aplikaciji sami. Podatkovne ključe je potrebno na ponovljiv način porazdeliti med n strežniki. Največkrat se uporablja zgoščevalne funkcije. Za varnost podatkov poskrbimo s postavitvijo strežnikov v razmerje nadrejeni-podrejeni. Zelo pomembna nastavitev samega strežnika je tudi, kako pogosto shranjuje podatke

na disk. To nam omogoča, da Redis nastavimo tako, da vsako operacijo zapiše tudi na disk v dnevnik operacij. To in dejstvo, da Redis ni več niten, nam zagotavlja ACID lastnosti.

2.2 Apache Cassandra

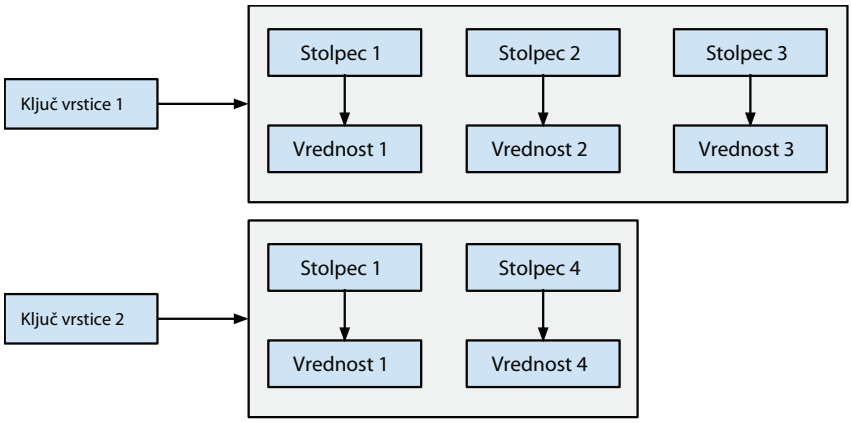
Apache Cassandra je odprtokodna porazdeljena shramba podatkov. Razvoj je začelo podjetje Facebook, ki je leta 2008 prestavilo razvoj v odprtokodno skupnost. Čeprav je bila verzija 1.0 izdana oktobra 2011, se Cassandra uporablja v največjih produkcijskih okoljih na spletu že dlje časa. Največja spletna storitev za objavljanje kratkih sporočil Twitter, ki dobi tudi do 5000 objav na sekundo, seli svojo primarno hrambo podatkov v Cassandro [3]. Cassandra je postala tako popularna zaradi naprednih rešitev in fleksibilnosti. Ne obstaja ena sama kritična točka odpovedi, saj so vsa vozlišča v gruči enakovredna. Uporabnik sam določa, kako varno hoče shraniti podatke in kakšno konsistentnost želi. Omogoča tudi dodajanje vozlišč v gručo med delovanjem in avtomatično porazdelitev podatkov in zahtev.

2.2.1 Podatkovni model

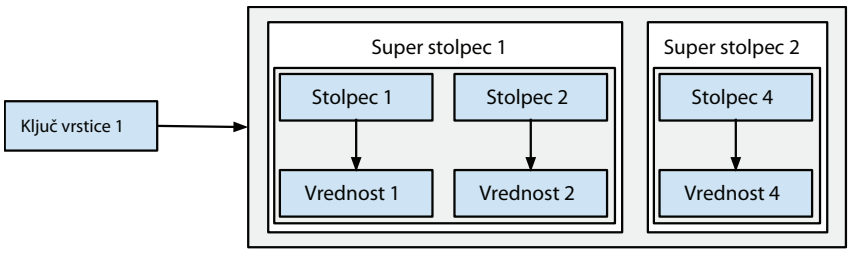
Cassandra povzema podatkovni model Googlove podatkovne baze BigTable [5]. Google definira podatkovni model podatkovne baze BigTable kot "redek, porazdeljen, trajen več dimenzijski slovar". Znanim pojmom organizacije podatkov damo nekoliko drugačen pomen. Najmanjša enota podatkov v Cassandri so stolpci (column). Stolpec obsega ime in vrednost, poleg tega lahko zraven shranimo časovni zapis kdaj je bil posamezen stolpec nazadnje spremenjen, kar nam pride prav pri reševanju konfliktov. Množica stolpcev je združena v vrstico, ki ima svoj ključ (row key). Vrstice s podobno vsebino se združujejo v družino stolpcev (column family). Na nek način je družina stolpcev podobna tabeli v relacijski bazi, vendar Cassandra ne zahteva, da v vseh vrsticah v družini obstajajo stolpci z enakim ključem (slika 2.1).

Za še večjo fleksibilnost pri strukturiranju podatkov Cassandra poskrbi s super družino stolpcev (super column family). Če so pri družini stolpcev vsebine vrstic stolpci, je pri super družini vsebina vrstic super stolpec (super column), ki nato vsebuje stolpce (slika 2.2).

Podatkovni model ni potrebno definirati v naprej. Vse kar Cassandra zahteva, je definicija družin stolpcev ali super družin. Vrstice v posameznih družinah pa so lahko popolnoma različne. V primerjavi z relacijskimi podatkovnimi bazami, ima Cassandra še eno veliko razliko. Podatki se ne urejajo ob



Slika 2.1: Družina stolpcev



Slika 2.2: Super družina stolpcev

branju, ampak se urejajo ob pisanju. V posamezni vrstici Cassandra ob pisanju uredi stolpce po njihovem imenu. Način urejanja se nastavi ob kreiranju družine stolpcev. Te omejitve, ali gledano s strani Cassandre funkcionalnosti, prisilijo v definiranje podatkovnega modela s stališča poizvedb. Samo tako se lahko doseže optimalno hitrost poizvedb.

Podatki so pri Cassandri večinoma zapisani na več vozliščih, odvisno od faktorja replikacije. Ta porazdeljnost lahko pripelje do nekonsistentnosti podatkov. Cassandra omogoča uporabniku, da pri vseh poizvedbah pošlje, poleg želene poizvedbe, tudi želeni nivo konsistentnosti. Nivo ena pomeni, da bo Cassandra vrnila rezultat, takoj ko odgovori prvo vozlišče. Pri nivoju vsi bo Cassandra poslala poizvedbo na vsa vozlišča, počakala na odgovore in vrnila vrednost z najnovejšim časovnim zapisom. Obstaja še nivo kvorum, ki počaka na odgovor večine vozlišč in vrne najnovejši zapis.

2.2.2 Replikacija

Cassandra je bila razvita specifično za porazdeljenost na množici vozlišč. Vsa vozlišča v gruči so enaka in na zunaj uporabniku dostopna kot eno samo vozlišče. Faktor replikacije (replication factor) se določi pri kreiranju podatkovne strukture in pove Cassandri, na koliko vozliščih naj bo posamezen podatek shranjen. Replikacija podatkov med vozlišči poteka s protokolom vsak z vsakim in je za uporabnika nevidna. S pravilno določitvijo faktorja replikacije lahko določimo, kako odporna bo gruča na izpade vozlišč.

2.3 Apache HBase

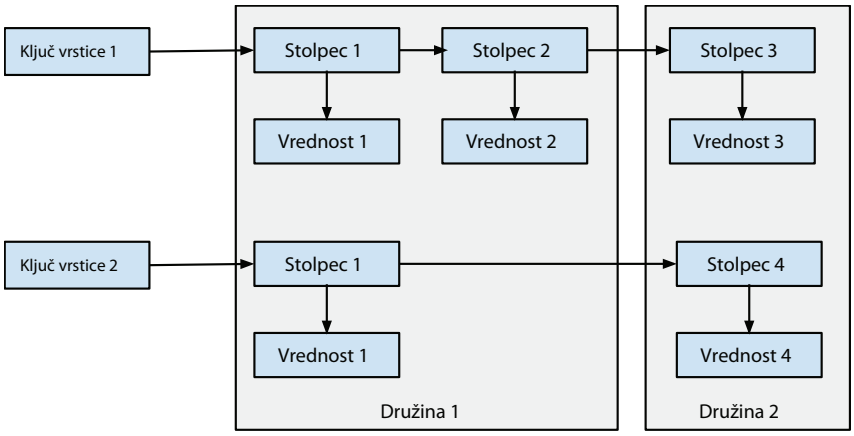
Apache HBase je odprtokodna, nerelecijska, porazdeljana podatkovna baza osnovana na Google BigTable podatkovni bazi. HBase za delo potrebuje HDFS datotečni sistem. HDFS porazdeljen datotečni sistem priskrbi Hadoop ogrodje. Hadoop je prav tako Apache projekt narejen za porazdeljeno obdelavo velike količine podatkov. Kot HBase je tudi Hadoop osnovan na Googlovih idejah: specifično MapReduce in Google File System. Hadoop zagotavlja varnost podatkov in njihovo porazdaljenost na množici vozlišč. Podjetje Facebook, ki trenutno velja za največje socialno omrežje, je prispevalo k razvoju in uporablja HBase za shranjevanje uporabniških sporočil [4].

2.3.1 Podatkovni model

Podobno kot Cassandra tudi HBase povzema model BigTable. Vrstice s podatki so shranjene v označenih tabelah. Vrstica ima ključ, ki se ga da razvrstiti, in poljubno število stolpcev. Tabela je shranjena tako, da imajo lahko vrstice v isti tabeli zelo različno število stolpcev. Ob kreiranju tabele določimo tudi družine stolpcev, ki jih bo tabela vsebovala. Ko določamo ključe se je potrebno zavedati, da so ključi urejeni binarno. Slika 2.3 prikazuje strukturo podatkov v HBase tabeli.

2.3.2 Replikacija

Hbase se pri replikaciji in porazdelitvi podatkov na več strežnikov zanaša na ogrodje Hadoop. Hadoop poskrbi, da so podatki zapisani na več kot enem strežniku in v primeru izpada vozlišča, prenos teh podatkov na novo vozlišče, tako da je zadoščeno nastavitvam replikacije. Hadoop je bil napisan za postavitev na tisočih povprečnih računalnikih, kjer je napaka prej pravilo kot izjema,



Slika 2.3: HBase tabela

zato so vanj vgradili mehanizem za odkrivanje napak ter hitro in avtomatsko popravilo.

2.4 MongoDB

MongoDB je odprtokodna nerelacijska podatkovna baza. Razvoj je začelo podjetje 10gen v letu 2007, prvo javno izdajo pa so predstavili leta 2009. Od vseh prej opisanih rešitev je najbližje relacijskim podatkovnim bazam. Podatki so shranjeni v dokumentih, nad podatki lahko ustvarimo kompleksne indekse in ima bogat poizvedbeni jezik. MongoDB smo uporabili za prenovo spletne strani Examiner.com, ki je najhitreje rastoča spletna stran z lokalnimi vsebinami v ZDA. Vse podatke smo iz relacijskega modela v MSSQL migrirali v MongoDB in pridobili na fleksibilnosti in hitrosti.

2.4.1 Podatkovni model

MongoDB strežnik lahko vsebuje več podatkovnih baz v katerih so shranjene zbirke (collection). V zbirke se zapisujejo dokumenti in v dokumentih množica polj. Polje je definirano kot par ključ vrednost, kjer je ključ niz znakov, vrednost je lahko:

- Enostavni tip npr. celo število, niz znakov, časovni zapis, število s plavajoči vejico, itd.
- Dokument

- Množica vrednosti

MongoDB ne zahteva definiranja sheme v naprej, prav tako ni potrebno ustvariti zbirk pred shranjevanjem. MongoDB bo ustvaril zbirke ob prvem zapisu. Indeksi so lahko sestavljeni in vsebujejo katerikoli tip podatkov in vrednosti. MongoDB tako kot relacijske podatkovne baze omogoča urejanje rezultatov ob poizvedbi in ne ob zapisu. Če ob zapisu dokumenta ne določimo primarnega ključa, ga bo MongoDB zapisal sam. Primer podatkov v MongoDB podatkovni bazi in enostavna poizvedba.

```
> db.uporabniki.save({ime:'Janez', priimek:'Novak', starost:32})
"ok"
> db.uporabniki.save({ime:'Peter', priimek:'Golob', starost:45})
"ok"
> db.uporabniki.find({ime:"Janez"})
  {   "_id" : {   "$oid" : "4e9b3614cc93741aef020d2d"   },
    "starost" : 32,   "ime" : "Janez",   "priimek" : "Novak"   }
  > db.uporabniki.find({}, {ime : 1, starost:1}).sort({starost:1})
  { "_id" : ObjectId("4e9b3788f7973c542f745320"), "ime" : "Janez",
    "starost" : 32 }
  { "_id" : ObjectId("4e9b378df7973c542f745321"), "ime" : "Peter",
    "starost" : 45 }
```

2.4.2 Replikacija

MongoDB podpira dve tehnologiji za replikacijo podatkov in horizontalno skaliranje. MongoDB podpira avtomatsko porazdelitev ali sharding podatkov na več strežnikov. Če podatki prerastejo en sam strežnik, lahko podatke porazdelimo na dodatne strežnike, ne da bi pri tem spreminjali samo aplikacijo. Za dodatno varnost podatkov, MongoDB vpelje koncept replikacijskih množic (replica set). Ena particija ali shard je shranjena v replikacijski množici, v kateri je več strežnikov. Strežniki med sabo določijo trenutno primarni strežnik, na katerega bo aplikacija zapisovala podatke. V gruči imamo tako lahko več particij podatkov, ki so shranjene v replikacijskih množicah, sestavljenih iz več strežnikov. S tem dobimo večjo zmogljivost gruče in varnost podatkov.

Poglavje 3

Testno okolje

3.1 Strojna oprema

Nereleacijske podatkovne baze so bile razvite primarno za shranjevanje in branje ogromne količine podatkov, ki jih ne moremo shraniti na samo en strežnik. Ker nimam dostopa do strojne opreme na kateri bi lahko izvajal teste, sem se odločil za postavitev testnega okolja v Amazonovem EC2 sistemu. Infrastrukturo v oblaku ponuja tudi podjetje Rackspace s svojo storitvijo Rackspace Cloud Servers. Odločil sem se za Amazon EC2 predvsem zaradi boljšega poznavanja. Amazon je tudi rahlo cenejši in ima boljša orodja za delo v ukazni vrstici. EC2 mi je omogočil zagon potrebnih zmogljivosti, ko sem jih potreboval in zaustavitev le teh po koncu testiranja. Amazon ponuja več tipov virtualnih strežnikov. Od povsem osnovnih z 600MB pomnilnika in eno računsko enoto, do tistih z 68GB pomnilnika in 26 računskimi enotami. Sam sem izbral tip, ki ga Amazon imenuje Large. Amazon zagotavlja sledeče karakteristike:

- 7.5GB pomnilnika
- 4 EC2 računske enote
- 850GB lokalne hrambe podatkov
- Visoka hitrost dostopa do trajne podatkovne shrambe
- 64 bitna platforma

Amazon je uvedel EC2 računsko enoto zato, da bi uporabnikom zagotovil približno enake karakteristike na zelo različni fizični opremi. Ena EC2 računaska enota naj bi bila primerljiva z 1.0-1.2GHz 2007 Opteron ali 2007

Xeon procesorjem [6]. V času pisanja tega dela, je operacijski sistem v Large primerku prepoznal dvo jedrni 2.66Hz Intel Xeon procesor, kar ustreza približno 4 EC2 računskim enotam, kot jih obljublja Amazon. Teste zmogljivosti sem izvajal na štirih Large virtualnih strežnikih. Amazon omogoča tudi izbiro podatkovnega centra, v katerem želimo zagnati te strežnike. Ker je cena na uro najmanjša v njihovem centru na vzhodni obali ZDA, sem se odločil za to regijo. Uporaba takega strežnika stane $0,34 na uro. Preračunano na mesec, ki ima 30 dni, nas bi strežnik stal 244,80 ameriških dolarjev.

Arhitektura Amazon EC2 virtualnih strežnikov omogoča, da so nekateri viri dodeljeni posameznemu strežniku npr. procesorska moč in pomnilnik, nekateri pa se delijo med več strežnikov npr. mrežna povezava in diskovni podsistem. Na svojih testnih strežnikih sem uporabil EBS diskovni podsistem. Do EBS naprav dostopa strežnik preko mrežnih povezav. Tip strežnika large ima hitrost dostopa omejeno na 1Gbit/s. Predvidevam, da bodo izbrane podatkovne baze, ki pogosto pišejo ali berejo iz trdega diska, zaradi te lastnosti imele slabše izhodišče. Amazon omogoča, da povežemo več EBS enot v RAID polje in s tem povečamo zmogljivost vhodno izhodnega sistema. Pri izvajanju meritev se nisem spustil v to optimizacijo.

Za zagon strežnika nam Amazon nudi dve orodji. Spletni vmesnik preko katerega lahko s spletnim brskalnikom zaženemo želen strežnik, in pa orodja za ukazno vrstico, katere lahko enostavno uporabimo v lastnih skriptah.

```
ec2-run-instances ami-zda0cf8b3 --region us-east-1 \
  --instance-type m1.large --key rok-east
```

Zgornji ukaz bo zagnal en primerek tipa Large. Ostali parametri določajo:

- Ami-21f53948 je identifikacijska koda obstoječe pred konfigurirane slike operacijskega sistema. V tem primeru sem uporabil 10.04 Long Term Support verzijo Ubuntu Linux operacisjkega sistema.

- Region parameter definira v kateri regiji se bo primerek zagnal. Ostale možnosti so še us-west-1 zahodna obala ZDA, eu-west-1 Irska, ap-southeast-1 Singapur in ap-northeast-1 Tokyo.

- Instance-type parameter definira tip primerka in s tem njegovo zmogljivost.

- Key ali ključ definira kateri par ključev, ki smo jih pred tem kreirali, bomo uporabili za dostop do primerka.

Vse teste sem izvajal ob približno enakem času. Viri so pri Amazonovih strežnikih deljeni, kar bi lahko vplivalo na rezultate. Še vedno lahko na rezultate vplivajo tretje osebe. Upam, da bom z omejitvijo časa testiranja te vplive omejil in bodo na vse teste vplivali približno v enaki meri.

Amazon za spremljanje uporabe virov na virtualnih strežnikih ponuja svojo storitev CloudWatch. CloudWatch v svoji osnovni verziji poroča vsakih 5 minut, napredna verzija pa vsako minuto. Viri, ki jih bom spremljal na strežnikih, so uporaba CPUja, število pisanj in branj na disk ter količina podatkov, ki se bere ali piše.

Kot klienta za izvajanje testov bom uporabil en EC2 large virtualni strežnik na isti lokaciji, kot bodo tekle podatkovne baze. En sam klient ima dovolj zmogljivosti, da bo generiral zahteve in ne bo predstavljal omejitve.

3.2 Programska oprema

Strežnike v Amazonovem EC2 sistemu sem želel imeti prižgane samo toliko časa, kolikor je bilo potrebno, zato sem se odločil, da spišem inštalacijske skripte, ki bodo zagnale želene gruče na zahtevo. Za osnovo sem izbral operacijski sistem Ubuntu 10.04 LTS za katerega obstajajo tudi zagonske slike za Amazon EC2. V času pisanja so bile stabilne verzije izbranih podatkovnih baz in njihovi vmesniki API:

- MongoDB 2.0.1, MongoDB Java klient 2.6.5
- Redis 2.4.3, Jedis JAVA klient 2.0.0
- Cassandra 0.8.7, zadnja verzija podprta s strani YCSB orodja
- HBase 0.90.4
- MySQL 5.1.41-3ubuntu12.10, MySQL JDBC klient Connector/J 5.1.18

Za zagon gruče s HBase in Cassandro sem uporabil odprtokodni projekt Whirr. Apache Whirr je skupek knjižnic, ki nudijo enostaven zagon podprtih podatkovnih baz v Rackspace ali Amazon strežnikih. Trenutno podprta programska oprema je Cassandra, Hadoop, ZooKeeper, HBase, elasticsearch, Voldamort in Hama.

```
whirr launch-cluster --config=hbase.config
whirr destroy-cluster --config=hbase.config
```

Zgornja Whirr ukaza omogočata, da lahko enostavno zaženemo in ugasnemo želeno gručo. Konfiguracijske datoteke so dostopne v dodatku A.

Za zagon Mongo, Redis in MySQL gruč sem spisal lastne skripte. Skripte so na voljo v dodatku A.

```
# Ukazi za zagon in ustavitev Mongo, Redis in MySQL gruč.
mongo.sh start
redis.sh start
mysql.sh start
```

3.2.1 Testno orodje

Za izvajanje testov sem uporabil odprtokodno orodje imenovano YCSB (Yahoo! Cloud System Benchmark). Pri Yahooju so orodje razvili prav z namenom iskanja najboljših rešitev za shranjevanje podatkov v porazdeljenih podatkovnih bazah. YCSB je napisan v Javi in je izredno modularen. Že napisani klienti obstajajo za MongoDB, Cassandro, HBase, Voldemort, Infinispan in podatkovne baze z JDBC podporo [7]. Podpora za Redis prav tako obstaja, vendar sem moral dodati podporo za porazdelitev obremenitve na več Redis strežnikov. Prav tako lahko sami definiramo lastne delovne obremenitve, ki bolje simulirajo našo aplikacijo. Delovne obremenitve definirajo podatke, katere bomo vnašali v podatkovne baze in transakcije, katere bomo izvajali v naših meritvah. YCSB podpira veliko število podatkovnih baz, zaradi tega so transakcije zelo enostavne.

- Pisanje – zapiše en vnos v podatkovno bazo, vnos je sestavljen iz primarnega ključa in podatkovnih polj.
- Posodobitev – popravi en vnos v podatkovni bazi določen s primarnim ključem.
- Branje – prebere en zapis iz podatkovne baze določen s primarnim ključem.
- Iskanje – išče zapise katerih ključ je večji od podanega.
- Brisanje – izbriše en zapis iz podatkovne baze določen s primarnim ključem.

```
# Ukaza za vnos podatkov in izvajanje ene delovne obremenitve
java -cp build/ycsb.jar:db/mongodb/lib/* com.yahoo.ycsb.Client -load \
  -db com.yahoo.ycsb.db.MongoDbClient -p mongodb.database=ycsb \
```

```
  -p mongodb.url=mongodb://10.32.73.242:27017 -P workloadRokA \
  -p recordcount=10000000
java -cp build/ycsb.jar:db/mongodb/lib/* com.yahoo.ycsb.Client -t \
  -db com.yahoo.ycsb.db.MongoDbClient -p mongodb.database=ycsb \
  -p mongodb.url=mongodb://10.32.73.242:27017 -P workloadA
```

Prvi ukaz bo v podani MongoDB streznik vnesel 10.000.000 zapisov. V mojih testih je bil podan naslov MongoDB router strežnika, ki zahteve naslovi na pravo vozlišče v gruči. Drugi ukaz pa izvede delovno obremenitev definirano v dokumentu workloadA.

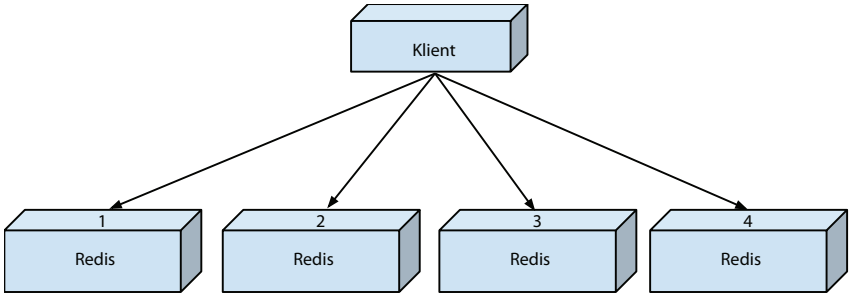
3.3 Nastavitve podatkovnih baz

Podatkovne baze sem namestil na 4 strežnike. Moje poznavanje optimizacije nastavitev za izbrane podatkovne baze ni za vse baze enako. Z večjo optimizacijo posameznih baz, bi lahko nepravično vplival na rezultate meritev. Odločil sem se, da podatkovne baze nastavim le toliko, kolikor je potrebno, da delujejo v porazdeljenem načinu na štirih strežnikih. Poizkušal sem izbrati arhitekture gruč tako, da so med seboj čim bolj primerljive. Še posebaj Cassandra in HBase omogočata zelo natančno nastavitev želenih lastnosti. Določimo lahko na koliko strežnikov se določeni podatki replicirajo in s tem nastavljamo razmerje med hitrostjo zapisovanja, hitrostjo branja in varnostjo podatkov.

3.3.1 Redis

Redis strežniki bodo tekli v master načinu brez replikacije. Vsak bo shranjeval svoj nabor podatkov. Klient bo odgovoren za funkcijo, ki določi na kateri strežnik se določeni podatki zapišejo. Taka arhitektura ni primerna za produkcijsko okolje, saj v primeru izpada enega strežnika ostanemo brez podatkov shranjenih na le tem strežniku. Privzeto Redis zapiše podatke na disk po 15 minutah, če se je spremnil vsaj en zapis, po 5 minutah, če se je spremenilo vsaj 10 zapisov in po 1 minuti, če se je spremenilo vsaj 10000 zapisov. Redis omogoča beleženje vseh operacij v trenutku, ko se zgodijo, s tem bi lahko ob izpadu strežnik vrnili v zadnje veljavno stanje. Ta možnost privzeto ni omogočena.

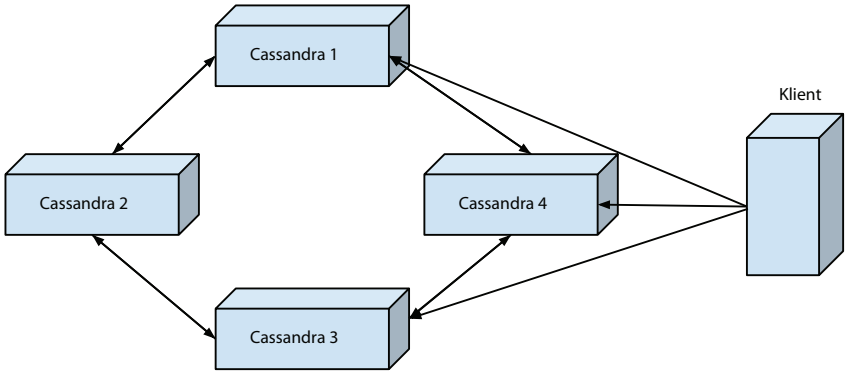
Na sliki 3.1 je prikazana arhitektura gruče Redis strežnikov, ki sem jo uporabil za meritve.



Slika 3.1: Redis gruča

3.3.2 Apache Cassandra

Vozlišča v gruči Cassandra strežnikov so enakovredna. Različno kot pri Redis gruči, se lahko klient poveže na katerokoli vozlišče za branje podatkov. Vozlišča med seboj komunicirajo in klientu vrnejo pravi podatek. Nivo replikacije sem v testni gruči, na kateri sem izvajal meritve, nastavil na 1, kar pomeni, da bo določen podatek shranjen samo v enem vozlišču. Na sliki 3.2 je prikazana arhitektura gruče Cassandra strežnikov, ki sem jo uporabil za meritve.

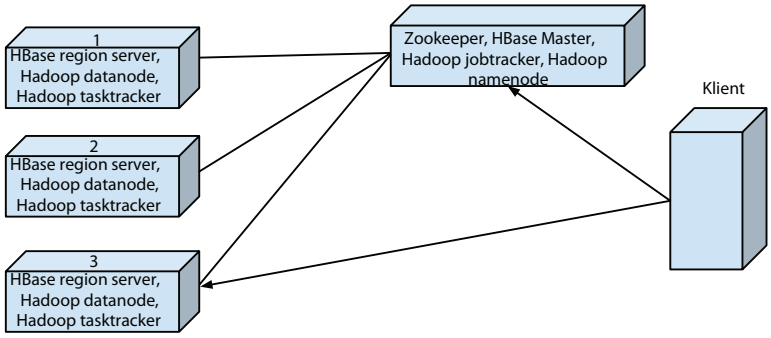


Slika 3.2: Cassandra gruča

3.3.3 Apache HBase

HBase za delovanje potrebuje glavni stražnik, na katerem je naložen Zookeeper, HBase master in Hadoop namenode in jobtracker. Za vrednost Hadoop replikacije sem uporabil 1. HBase ima v majhni gruči to pomanjkljivost, da je en strežnik namenjen nadrejenim procesom in ne izpolnjevanju zahtev kli-

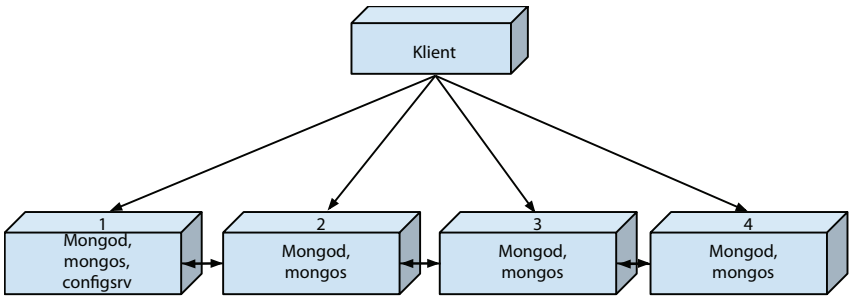
entov. Na sliki 3.3 je prikazana arhitektura gruče HBase strežnikov, ki sem jo uporabil za meritve.



Slika 3.3: HBase gruča

3.3.4 MongoDB

Gruča vsebuje štiri mongo podatkovne procese mongod, štiri mongo povezovalne procese mongos in en konfiguracijski proces configsrv. Klient se poveže s katerimkoli mongos procesom, ki nato opravi operacijo na pravilnih mongod procesih. Podatki niso podvojeni. Na sliki 3.4 je prikazana arhitektura gruče MongoDB strežnikov, ki sem jo uporabil za meritve.

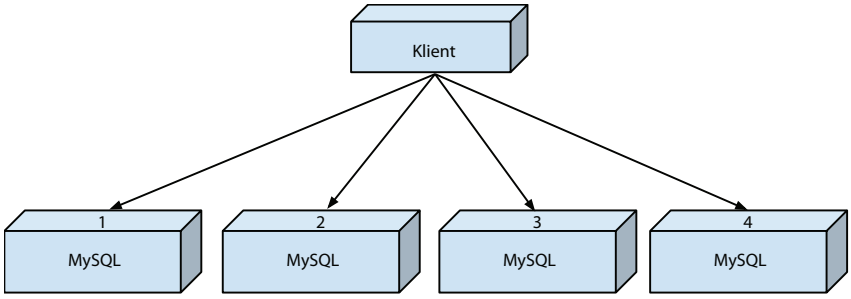


Slika 3.4: MongoDB gruča

3.3.5 MySQL

MySQL, podobno kot Redis, preloži vse naloge porazdeljevanja podatkov na klienta. Strežniki med seboj ne komunicirajo. Na sliki 3.5 je prikazana arhi-

tektura gruče MySQL strežnikov, ki sem jo uporabil za meritve.



Slika 3.5: MySQL gruča

3.4 Testni scenariji

Izbrane podatkovne baze se zelo razlikujejo po svoji arhitekturi in zaradi tega ni vsaka primerna za vse tipe uporabe. Da bi ugotovil katera podatkovna baza se bolje odreže v določenih okoliščinah, sem izbral štiri scenarije uporabe.

3.4.1 Scenarij A

Scenarij A simulira uporabo podatkovne baze za shranjevanje velike količine zapisov v dnevnik. Velika večina operacij so pisanja.

- Delež branj 0%
- Delež posodobitev 0%
- Delež pisanj 95%
- Delež iskanj 5%
- Iskanje uporablja novejše podatke

3.4.2 Scenarij B

Scenarij B simulira uporabo, kjer je največ branj. Zahtevani podatki so novejši. Ta scenarij predstavlja spletno stran, na kateri je zanimiva samo novejša vsebina.

- Delež branj 98%

- Delež posodobitev 1%
- Delež pisanj 1%
- Delež iskanj 0%
- Branje uporablja novejše podatke

3.4.3 Scenarij C

Scenarij C predstavlja sporočilni sistem, kjer se podatki berejo in pišejo v približno enakem razmerju. Starejši podatki nas ne zanimajo.

- Delež branj 50%
- Delež posodobitev 0%
- Delež pisanj 50%
- Delež iskanj 0%
- Branje uporablja novejše podatke

3.4.4 Scenarij D

Scenarij D uporablja za porazdelitev branj Zipfov zakon. Prevladujejo branja, pisanj je le en odstotek.

- Delež branj 99%
- Delež posodobitev 0%
- Delež pisanj 1%
- Delež iskanj 0%
- Berejo se podatki porazdeljeni po Zipfovem zakonu

3.5 Struktura testnih podatkov

Zapisi v podatkovne baze so sestavljeni iz ključa in 15-ih tekstovnih polj. Vsako tekstovno poljo bo vsebovalo natančno 100 znakov. Približna velikost enega zapisa bo tako 1.5kB. Tukaj so mogoča odstopanja, saj vsaka podatkovna baza shranjuje podatke na svoj način. Cassandra na primer z vsakim poljem shranjuje tudi časovni zapis zadnjega popravka. V podatkovne baze bom pred izvajanjem testov shranil 10.000.000 zapisov.

Poglavje 4

Analiza rezultatov

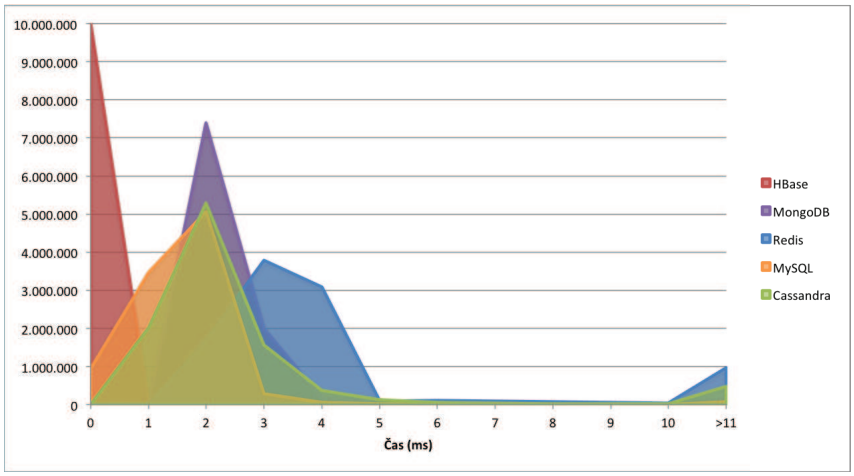
4.1 Začetni vnos podatkov

Začetni vnos podatkov nam prikazuje kako hitro lahko izbrane podatkovne baze shranjujejo podatke. V tabeli 4.1 so zbrani podatki o hitrosti vnosa podatkov v podatkovne baze. Najhitreje je 10 milijonov zapisov shranila MySQL gruča, s precej višjim povprečnim številom operacij na sekundo, kot vse ostale podatkovne baze. Prav tako je najnižji povprečen čas zapisa. HBase je druga na seznamu kljub temu, da je HBase gruča nekoliko drugačna kot vse ostale in ima samo 3 vozlišča, ki so shranjevala podatke. Zanimivo je tudi, da je HBase zapisal 99 odstotkov zahtev v manj kot 0ms. To prikazuje tudi slika 4.1. Pri HBase bazi se lepo vidi, da je več kot 99 odstotkov vnosov bilo shranjenih v manj kot 0ms.

Tabela 4.1: Statistika vnosov

	Čas izvajanja	Št. operacij /s	Povprečen čas	99% zahtev	95% zahtev
Cassandra	23,8 min	7053	5,477 ms	71	9
Hbase	19,4 min	8908	4,3945 ms	0	0
MongoDB	61,5 min	2771	7,1165 ms	24	4
MySQL	14,4 min	11963	3,275 ms	7	3
Redis	66,6 min	2734	7,025 ms	61	16

Redis podatkovna baza je bila edina, ki ni shranila vseh zapisov. Približno 800.000 zapisov ni bilo uspešno shranjenih. Časovna omejitev povezave je pri Redisu nastavljena na precej majhno vrednost in je bila pri tako velikem številu zahtev pogosto prekoračena. Podobno zakasnitev sem opazil tudi pri HBase, kjer je bil najdaljši čas izvajanja ene zahteve kar 58 sekund.



Slika 4.1: Začetni vnos podatkov

4.2 Scenarij A

Prve meritve scenarija A sem izvajal tako, da sem YCSB orodju dopustil, da je maksimalno obremenil strežnike. To se je hitro izkazalo za problematično. Strežniki so po nekaj minutah postali preobremenjeni in število operacij na sekundo je padlo na 0. Odločil sem se, da bom omejil število operacij na sekundo, ki jih YCSB sproži na 1500 in 2.700.000 operacij skupno. Če bodo podatkovne baze zmogle to število operacij, bo test končan po pol ure. MySQL JDBC gonilnik, ki sem ga uporabil, ni dokončal nobene operacije iskanja, saj so vrnila preveč podatkov in klientu je zmanjkalo spomina. Scenarija A z MySQL bazo nisem dokončal. Pri Redis podatkovni bazi prav tako nisem mogel dokončati testa. Redis strežniški procesi so se konstantno ugašali in po nekaj minutah sem test prekinil. Rezultati HBase, Cassandra in MongoDB baz so v tabeli 4.2.

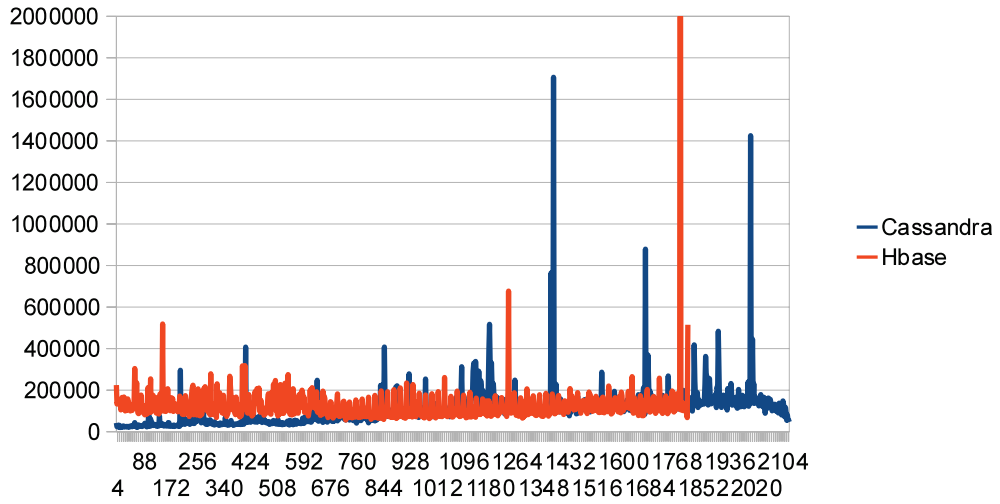
Tabela 4.2: Statistika scenarija A

	Čas izvajanja	Št. operacij /s	Povprečen čas iskanja	Povprečen čas pisanj
Cassandra	36,5 min	1232	91,578 ms	3,041 ms
Hbase	30,2 min	1493	109,832 ms	0,162 ms
MongoDB	97 min	461	136,545 ms	15,5 ms
MySQL	/	/	/	/
Redis	/	/	/	/

Pri HBase bazi, kljub dodatnim iskanjem, ni bilo opaziti nobenih zakasni-

tev pri hitrosti zapisov. Pri Mongu na drugi strani se je čas zapisov visoko povečal, čeprav sem izvajal precej manj pisanj kot v začetnem vnosu podatkov. Cassandra je bila najhitrejša pri iskanjih in nekaj počasnejša pri pisanjih od HBase base.

Slika 4.2 prikazuje povprečno hitrost iskanja v določenem trenutku izvajanja testa. Vidi se, da Cassandra na začetku vrača rezultate hitreje kot HBase potem pa se približno izenačita. Čeprav ima Cassandra več trenutkov, ko se čas iskanja poveča, je skupni povprečni čas še vedno manjši.



Slika 4.2: Povprečna hitrost iskanja

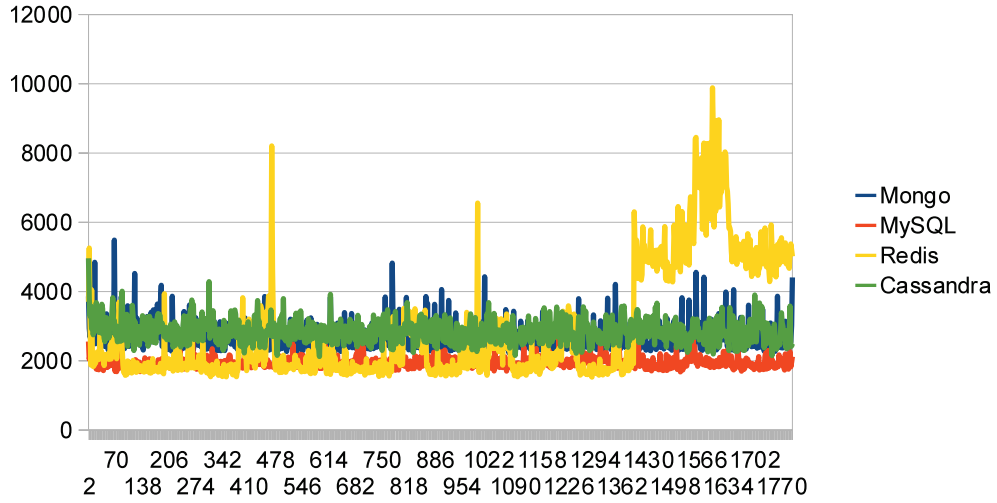
4.3 Scenarij B

Kot pri scenariju A, sem tudi pri scenariju B omejil skupno število operacij na 2.700.000 in 1500 operacij na sekundo. Vse podatkovne baze, razen HBase, so s scenarijem opravile v optimalnem času. Pri HBase me je zelo presentil slab čas branj, ki jih je bilo v scenariju B kar 98 odstotkov. Kot je razvidno iz tabele 4.3, je v povprečju HBase potreboval kar 19ms za operacijo branja. Interno HBase operacije branja razume kot iskanje. HBase nima indeksov, ki bi omogočali dostop do posamezne vrstice ali stolpca. Najmanjša enota je blok v HFile datoteki, katero moramo nato preiskati, da dobimo želen podatek [8]. Kljub temu, da branja trajajo nekaj dlje, HBase še vedno zapisuje najhitreje.

Tabela 4.3: Statistika scenarija B

	Čas izvajanja	Št. operacij na sekundo	Povprečen čas posodobitve (ms)	Povprečen čas pisanj (ms)	Povprečen čas branj (ms)
Cassandra	30 min	1500	2,3	2,488	2,846
Hbase	85 min	529	0,024	0,047	19,225
MongoDB	30 min	1500	4,153	4.291	2,625
MySQL	30 min	1500	2,016	2.165	1,945
Redis	30 min	1500	2,768	6,281	2,872

Na sliki 4.3 je razvidno razmerje med podatkovnimi bazami pri branju. MySQL in Redis sta na začetku testa še skupaj, proti koncu pa je Redis počasnejši od vseh. Redis tudi pri tem testu ni uspešno dokončal vseh operacij. Neuspešno je bilo izvedenih 3000 pisanj in posodobitev in približno 460.000 branj.



Slika 4.3: Povprečna hitrost branja

4.4 Scenarij C

Pri scenariju C sem skupno število operacij zmanjšal na 1.500.000, število operacij na sekundo pa obdržal na 1500. Ponovno sem imel veliko problemov

z Redis podatkovno bazo. Nikakor mi ni uspelo uspežno zaključiti izvajanje meritev. Pri velikem številu pisanj Redis ne zmore dovolj hitro shranjevati podatkov na disk in procesi se ugasnejo. HBase trpi za enako hibo kot v scenariju B, branja podatkov niso dovolj hitra in tudi v tem scenariju jih je bilo veliko. Pri Cassandri se lepo vidi, kako večje število pisanj vpliva na čas branj. Branja vzamejo skoraj desetkrat več časa kot pri scenariju B.

Tabela 4.4: Statistika scenarija C

	Čas izvajanja	Št. operacij na sekundo	Povprečen čas pisanj (ms)	Povprečen čas branj (ms)
Cassandra	30 min	839	2.183	21.157
Hbase	45 min	551	0.106	36.011
MongoDB	16.6 min	1500	2.981	2.596
MySQL	16.6 min	1500	4.768	3.788
Redis	/	/	/	/

V tabeli 4.4 so časi branj posameznih podatkovnih baz precej razlikujejo. Eno izmed možnih razlag ponuja tudi vpogled koliko podatkov se prebere iz diska. Podatke sem pridobil iz Amazon CloudWatch storitve za posamezna vozlišča v gruči in jih seštel po podatkovnih bazah in časih, ko se je izvajal scenarij C.

- MySQL 68 kiloBytov
- MongoDB 99 megaBytov
- Cassandra 26 gigaBytov
- HBase 62.5 gigaBytov

Razlike so skoraj neverjetne. MySQL se v malo več kot 16 minutah in skoraj 800.000 branjih skoraj ne dotakne diska, vse zahteve se postrežejo iz pomnilnika. Medtem ko HBase prebere 62 gigaBytov.

4.5 Scenarij D

Pri scenariju D so se vse podatkovne baze, razen HBase, zelo dobro izkazale, zato sem povečal število operacij na sekundo v testu na 2500. Pri testiranju HBase se test ni izvedel v celoti niti pri 1500 operacijah na sekundo. Redis

je kljub dobrim rezultatom in časi primerljivimi z ostalimi, neuspešno opravil približno 10 odstotokov operacij. Z MySQL, Redis in MongoDB sem test izvedel še pri večjih obremenitvah. Največje število operacij scenarija D je zmogel MySQL kar 9166, sledi Redis 4650 in MongoDB 3577.

Tabela 4.5: Statistika scenarija D

	Čas izvajanja	Št. operacij na sekundo	Povprečen čas pisanj (ms)	Povprečen čas branj (ms)
Cassandra	18	2500	2,229	3,265
Hbase	/	/	/	/
MongoDB	18	2500	4,424	2,651
MySQL	18	2500	2,252	1,901
Redis	18	2500	5,082	2,017

4.6 Možne izboljšave testiranja

Med pregledovanjem rezultatov in ostalih podatkov, ki sem jih zbral med izvajanjem testov, sem našel več stvari, katere bi v naslednjih testih spremenil.

Spremljanje virov na strežnikih mi je nudilo delni vpogled v dogajanje na njih med izvajanjem scenarijev, vendar bi si želel dodatno spremljati nekatere karakteristike. Predvsem so mi manjkali podatki o tem, koliko časa so strežniki zapravili pri čakanju na disk. Zanimivo bi bilo tudi spremljati, koliko zahtev so podatkovne baze postregle iz pomnilnika in za koliko zahtev je bilo izvedeno branje iz trdega diska.

Scenariji, ki sem jih izbral, predstavljajo le delček operacij, ki se vsakodnevno izvajajo v produkcijskih okoljih. Poizvedbe so bolj kompleksne in pisanja se izvajajo v več tabelah naenkrat. Na nek način so podatkovne baze delovale kot ključ vrednost shrambe. Vse poizvedbe so iskale po primarnem ključu in to polje je tudi bilo edino z indeksom. V našem podjetju Examiner.com imamo v nekaterih MongoDB zbirkah tudi po 40 kompleksnih indeksov, kar drastično spremeni čas pisanja.

V produkcijskih okoljih se nikoli ne zgodi, da bi bila določena vrednost zapisana samo na enem strežniku. Vse podatkovne baze, ki sem jih testiral, ponujajo neke vrste replikacijo, da v primeru izgube strežnika, ne izgubimo tudi podatkov. Kompleksnost takšne gruče se zelo poveča in poveča se tudi število pisanj na strežnikih. Želel bi preveriti, kakšna je primerjava med podatkovnimi bazami, kjer se podvajanje podatkov na več strežnikov enostavno vključi v

nastavitvah in ostalimi, pri katerih je potrebno strežnike dodati in postaviti v razmerje nadrejeni-podrejeni.

Poglavje 5

Sklepne ugotovitve

Primerjanje nerelacijskih podatkovnih baz je zahtevno opravilo. Različne podatkovne baze so namenjene različnim namenom uporabe. Pri meritvah so mi bile v izjemno pomoč skripte, ki so omogočale zagon gruč v parih minutah. Z orodjem YCSB pa sem, na ponovljiv način in s ponovljivimi obremenitvami, izvajal meritve hitrosti gruč. Veliko vlogo pri meritvah je odigralo tudi okolje Amazon EC2. Povsem možno je, da bi bili rezultati pri lastni strojni opremi povsem drugačni.

Rezultati testiranj kažejo, da pri lepo strukturiranih podatkih, enostavnih poizvedbah in indeksu samo na primarnem ključu, nerelacijske baze ne uspejo slediti hitrosti MySQL podatkovne baze. Tukaj je potrebno poudariti, da sta bila Redis in MySQL v priviligiranem položaju, saj je za porazdelitev podatkov na pravo vozlišče skrbel klient in ne podatkovna baza. MongoDB, Cassandra in HBase imajo vse vgrajene algoritme, ki podatke avtomatsko razdelijo po vozliščih. Pri 100 ali več vozliščih bi bil ročni nadzor porazdeljevanja podatkov praktično nemogoč. Pri tako velikem številu vozlišč je izpad vozlišča zelo verjeten dogodek. Zanimivo bi bilo primerjati kako se hitrosti poizvedb in zapisov odzovejo na izpad vozlišča in kako hitro podatkovne baze podatke prenesejo na druga vozlišča.

Pomembno se je tudi zavedati, da so vse testirane nerelacijske podatkovne baze v zgodnjih fazah razvoja. MySQL na drugi strani je prisoten že več kot 15 let. Trenutno nobena od njih ni univerzalna rešitev za vse probleme. So pa s fleksibilnostjo podatkovnega modela, rešitvami porazdelitve podatkov in s skalabilnostjo zanimiva alternativa bolj uveljaveljenim relacijskim podatkovnim bazam.

Dodatek A

Konfiguracijske datoteke Whirr in zagonske skripte

A.1 Konfiguracijska datoteka za HBase

hbase.config

```
# Ime gruce.
whirr.cluster-name=hbase-0.90

# Stevilo in tip vozlisc v HBase gruci.
whirr.instance-templates=1 zookeeper+hadoop-namenode+
   hadoop-jobtracker+hbase-master,3 hadoop-datanode+
   hadoop-tasktracker+hbase-regionserver

# Nivo replikacije podatkov.
hbase-site.dfs.replication=1

# EC2 nastavitve in identifikacija
whirr.provider=aws-ec2
whirr.identity=${env:AWS_ACCESS_KEY_ID}
whirr.credential=${env:AWS_SECRET_ACCESS_KEY}

# Tip amazon instance moznosti opisane na http://aws.
   amazon.com/ec2/instance-types/
whirr.hardware-id=m1.large
# Slika operacijskega sistema Ubuntu 10.04
```

```
whirr.image-id=us-east-1/ami-da0cf8b3
# Lokacije gruce.
whirr.location-id=us-east-1

# Verzija HBase.
whirr.hbase.tarball.url=http://apache.cu.be/hbase/hbase
    -0.90.4/hbase-0.90.4.tar.gz

# Verzija Hadoop.
whirr.hadoop.tarball.url=http://archive.cloudera.com/cdh
    /3/hadoop-0.20.2-cdh3u1.tar.gz
```

A.2 Konfiguracijska datoteka za Cassandra podatkovno bazo

cassandra.config

```
# Ime gruce.
whirr.cluster-name=cassandra

# Stevilo Cassandra vozlisc
whirr.instance-templates=4 cassandra

# EC2 nastavitve in identifikacija
whirr.provider=aws-ec2
whirr.identity=${env:AWS_ACCESS_KEY_ID}
whirr.credential=${env:AWS_SECRET_ACCESS_KEY}

# Tip amazon instance moznosti opisane na http://aws.
    amazon.com/ec2/instance-types/
whirr.hardware-id=m1.large
# Slika operacijskega sistema Ubuntu 10.04
whirr.image-id=us-east-1/ami-da0cf8b3
# Lokacije gruce.
whirr.location-id=us-east-1
# Kljuci za dostop do gruce
whirr.login-user=ubuntu
# Cassandra Verzija
whirr.cassandra.version.major=0.8
```

```
whirr.cassandra.tarball.url=http://apache.osuosl.org//
    cassandra/0.8.7/apache-cassandra-0.8.7-bin.tar.gz
```

A.3 Redis zagonska skripta

redis.sh

```
#!/bin/bash
source ec2.conf

start () {
  ec2-add-group redis-master --region $ZONE -d Redis-
     master
  ec2-describe-group --region $ZONE redis-master | head
     -n 1 | awk '{ print $2 }' > redis-master.group
  MASTER_GROUP=`cat redis-master.group`
  for i in $(seq 1 $INSTANCE_COUNT)
  do
    INSTANCE_INFO=$(ec2-run-instances $AMI_ID --region
       $ZONE -t $INSTANCE_TYPE -k $EC2_KEYPAIR -g
       $MASTER_GROUP --user-data-file install_redis.sh)
    INSTANCE_ID=$(echo "$INSTANCE_INFO" | grep INSTANCE
       | awk '{print $2}')
    done="false"
    while [ $done == "false" ]
    do
      INSTANCE_INFO=$(ec2-describe-instances --region
         $ZONE $INSTANCE_ID | grep $INSTANCE_ID)
      INSTANCE_STATUS=$(echo "$INSTANCE_INFO" | grep -c
         running)
      if [ "$INSTANCE_STATUS" -eq "1" ]; then
        done="true"
        INSTANCE_IP=$(echo "$INSTANCE_INFO" | awk '{
           print $15}')
        CLUSTER_IPS[$i]=$INSTANCE_IP
      else
        sleep 5
      fi
    done
  done
```

```
  OUTPUT=$(printf ",%s" "${CLUSTER_IPS[@]}")
  OUTPUT=${OUTPUT:1}
  echo $OUTPUT
  exit
}

usage() {
  cat << EOF
  usage: $0 start/stop

  Start or stop redis cluster.

  OPTIONS:
    -h      Show this message
EOF
}

if [ -z $1 ]
then
  usage
  exit 1
fi

case $1 in
  "start") start ;;
  * ) usage;;
esac
```

install_redis.sh

```
#!/bin/bash -ex
wget https://raw.github.com/Zlender/NoSQL-config-files/
   master/redis.conf
sudo mkdir /etc/redis
sudo mv redis.conf /etc/redis/redis.conf;
sudo sysctl vm.overcommit_memory=1

sudo apt-get update
sudo apt-get install -y make gcc
wget http://redis.googlecode.com/files/redis-2.4.3.tar.
```

```
    gz
sudo tar zxf redis-2.4.3.tar.gz -C /usr/local/
cd /usr/local/redis-2.4.3
sudo make
sudo mkdir /mnt/redis
sudo mkdir /var/log/redis
sudo /usr/local/redis-2.4.3/src/redis-server /etc/redis/
    redis.conf
```

A.4 MongoDB zagonska skripta

mongo.sh

```
#!/bin/bash
source ec2.conf

start () {
  ec2-add-group mongo-cluster --region $ZONE -d Mongo-
      cluster
  ec2-describe-group --region $ZONE mongo-cluster | head
      -n 1 | awk '{ print $2 }' > mongo-cluster.group
  SECURITY_GROUP=`cat mongo-cluster.group`
  for i in $(seq 1 $INSTANCE_COUNT)
  do
    if [ "$i" -ne $INSTANCE_COUNT ]; then
      DATA_FILE="install_mongo.sh"
    else
      DATA_FILE="install_mongo_router.sh"
      CONFIG_SERVERS=$(printf ",%s:27019" "${CONFIG_IPS
          [@]}")
      CONFIG_SERVERS=${CONFIG_SERVERS:1}
      sed -i -e "s|CONFIG_SERVERS=.*|CONFIG_SERVERS=
          $CONFIG_SERVERS|" $DATA_FILE
    fi
    INSTANCE_INFO=$(ec2-run-instances $AMI_ID --region
        $ZONE -t $INSTANCE_TYPE -k $EC2_KEYPAIR -g
        $SECURITY_GROUP --user-data-file $DATA_FILE)
    INSTANCE_ID=$(echo "$INSTANCE_INFO" | grep INSTANCE
        | awk '{print $2}')
    done="false"
```

```
    while [ $done == "false" ]
    do
      INSTANCE_INFO=$(ec2-describe-instances --region
          $ZONE $INSTANCE_ID | grep $INSTANCE_ID)
      INSTANCE_STATUS=$(echo "$INSTANCE_INFO" | grep -c
          running)
      if [ "$INSTANCE_STATUS" -eq "1" ]; then
        done="true"
        INSTANCE_IP=$(echo "$INSTANCE_INFO" | awk '{
            print $15}')
        if [ "$i" -ne $INSTANCE_COUNT ]; then
          SHARD_IPS[$i]=$INSTANCE_IP
          CONFIG_IPS[$i]=$INSTANCE_IP
        else
          SHARD_IPS[$i]=$INSTANCE_IP
          ROUTER_IPS[$i]=$INSTANCE_IP
        fi
      else
        sleep 5
      fi
    done
  done
  SHARD_SERVERS=$(printf ",%s:27018" "${SHARD_IPS[@]}")
  SHARD_SERVERS=${SHARD_SERVERS:1}
  echo "Shards: " $SHARD_SERVERS
  CONFIG_SERVERS=$(printf ",%s:27019" "${CONFIG_IPS[@]}"
      )
  CONFIG_SERVERS=${CONFIG_SERVERS:1}
  echo "Config servers: " $CONFIG_SERVERS
  ROUTER_SERVER=$(printf ",%s:27017" "${ROUTER_IPS[@]}")
  ROUTER_SERVER=${ROUTER_SERVER:1}
  echo "Router server: " $ROUTER_SERVER
  MONGO_SETUP=$(printf "db.runCommand( { addshard : \"%s
      :27018\" } );\n" "${SHARD_IPS[@]}")
  MONGO_SETUP="$MONGO_SETUP\ndb.runCommand( {
      enablesharding : \"ycsb\" } );"
  MONGO_SETUP="$MONGO_SETUP\ndb.runCommand( {
      shardcollection : \"ycsb.usertable\", key : { _id :
      1 } } );"
```

```
  echo -e "$MONGO_SETUP"
  echo -e "$MONGO_SETUP" | $MONGO_HOME/bin/mongo
    $ROUTER_SERVER/admin
  exit
}

usage() {
  cat << EOF
  usage: $0 start

  Start mongo cluster.

  OPTIONS:
    -h      Show this message
EOF
}

if [ -z $1 ]
then
  usage
  exit 1
fi

case $1 in
  "start") start ;;
  * ) usage ;;
esac
```

install_mongo.sh

```
#!/bin/bash -ex

wget http://fastdl.mongodb.org/linux/mongodb-linux-
   x86_64-2.0.1.tgz
tar zxf mongodb-linux-x86_64-2.0.1.tgz
mv mongodb-linux-x86_64-2.0.1 /usr/local/
mkdir /mnt/log
mkdir /mnt/data
mkdir /mnt/data_configsrv
/usr/local/mongodb-linux-x86_64-2.0.1/bin/mongod --fork
```

```
    --logpath /mnt/log/mongodb.log --logappend --dbpath /
    mnt/data --shardsvr
/usr/local/mongodb-linux-x86_64-2.0.1/bin/mongod --
    configsvr --fork --logpath /mnt/log/mongdb_configsrv.
    log --logappend --dbpath /mnt/data_configsrv
```

install_mongo_router.sh

```
#!/bin/bash -ex

CONFIG_SERVERS
    =10.40.215.78:27019,10.40.207.38:27019,10.99.7.251:27019

wget http://fastdl.mongodb.org/linux/mongodb-linux-
    x86_64-2.0.1.tgz
tar zxf mongodb-linux-x86_64-2.0.1.tgz
mv mongodb-linux-x86_64-2.0.1 /usr/local/
mkdir /mnt/log
mkdir /mnt/data
mkdir /mnt/data_configsrv
/usr/local/mongodb-linux-x86_64-2.0.1/bin/mongod --fork
    --logpath /mnt/log/mongdb.log --logappend --dbpath /
    mnt/data --shardsvr
/usr/local/mongodb-linux-x86_64-2.0.1/bin/mongos --fork
    --logpath /mnt/log/mongdb_router.log --logappend --
    configdb=$CONFIG_SERVERS
```

A.5 MySQL zagonska skripta

mysql.sh

```
#!/bin/bash
source ec2.conf

start () {
  ec2-add-group mysql --region $ZONE -d MySQL
  ec2-describe-group --region $ZONE mysql | head -n 1 |
     awk '{ print $2 }' > mysql.group
  SECURITY_GROUP=`cat mysql.group`
  for i in $(seq 1 $INSTANCE_COUNT)
  do
```

```
    INSTANCE_INFO=$(ec2-run-instances $AMI_ID --region
        $ZONE -t $INSTANCE_TYPE -k $EC2_KEYPAIR -g
        $SECURITY_GROUP --user-data-file install_mysql.sh
        )
    INSTANCE_ID=$(echo "$INSTANCE_INFO" | grep INSTANCE
        | awk '{print $2}')
    done="false"
    while [ $done == "false" ]
    do
      INSTANCE_INFO=$(ec2-describe-instances --region
          $ZONE $INSTANCE_ID | grep $INSTANCE_ID)
      INSTANCE_STATUS=$(echo "$INSTANCE_INFO" | grep -c
          running)
      if [ "$INSTANCE_STATUS" -eq "1" ]; then
        done="true"
        INSTANCE_IP=$(echo "$INSTANCE_INFO" | awk '{
            print $15}')
        CLUSTER_IPS[$i]=$INSTANCE_IP
      else
        sleep 5
      fi
    done
  done
  OUTPUT=$(printf ",%s" "${CLUSTER_IPS[@]}")
  OUTPUT=${OUTPUT:1}
  echo $OUTPUT
  exit
}

usage() {
  cat << EOF
  usage: $0 start

  Start MySQL cluster.

  OPTIONS:
    -h      Show this message
EOF
}
```

```
if [ -z $1 ]
then
  usage
  exit 1
fi

case $1 in
  "start") start ;;
  * ) usage;;
esac
```

install_mysql.sh

```
#!/bin/bash -ex

sudo apt-get update
sudo bash -c "DEBIAN_FRONTEND=noninteractive aptitude -q
   -y install mysql-server"
sleep 5
sudo stop mysql
sudo mv /var/lib/mysql /mnt/
sudo ln -s /mnt/mysql /var/lib/mysql
sudo sed -i '/^bind-address.*/ s/^/#/' /etc/mysql/my.cnf
sudo sed -i '/\/var\/lib\/mysql\/\* rwk,/a\
  \/mnt\/mysql\/ r,\
  \/mnt\/mysql\/\* rwk,' /etc/apparmor.d/usr.sbin.
     mysqld
sudo start mysql
wget https://raw.github.com/Zlender/NoSQL-config-files/
   master/create_table.sql
mysql -uroot -e <<EOSQL "UPDATE mysql.user SET Password=
   PASSWORD('ycsb_root') WHERE User='root'; FLUSH
   PRIVILEGES;"
EOSQL
mysqladmin -uroot -pycsb_root create ycsb
mysql -uroot -pycsb_root ycsb < create_table.sql
```

A.6 Skupne nastavitve

ec2.conf

```
export MONGO_HOME=$HOME/mongodb-linux-i686-2.0.1
export EC2_KEY=~/.ssh/$EC2_KEYPAIR.pem
export EC2_URL=https://ec2.us-east-1.amazonaws.com
export EC2_PRIVATE_KEY=$HOME/.ssh/pk-*.pem
export EC2_CERT=$HOME/.ssh/cert-*.pem
export AWS_ACCESS_KEY_ID=[ACCESS_KEY]
export AWS_SECRET_ACCESS_KEY=[SECRET_ACCESS_KEY]
export INSTANCE_TYPE=m1.large
export INSTANCE_COUNT=4
export AMI_ID=ami-da0cf8b3
export EC2_KEYPAIR=diploma
export ZONE=us-east-1
```

Literatura

[1] (2011) Interview with Scott Chacon on Git and GitHub. Dostopno na: http://www.infoq.com/interviews/chacon-github

[2] (2011) Explaining the Story View Counts. Dostopno na : http://about.digg.com/blog/story-view-counts

[3] (2010) Cassandra @ Twitter: An Interview with Ryan King. Dostopno na : http://nosql.mypopescu.com/post/407159447/cassandra-twitter-an-interview-with-ryan-king

[4] (2010) The Underlying Technology of Messages. Dostopno na : https://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919

[5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat idr., Bigtable: A Distributed Storage System for Structured Data v zborniku *OSDI'06: Seventh Symposium on Operating System Design and Implementation*, Seattle, WA, november 2006.

[6] (2011) Amazon EC2 FAQs. Dostopno na : http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it

[7] (2011) Yahoo! Cloud Serving Benchmark (YCSB). Dostopno na: https://github.com/brianfrankcooper/YCSB/wiki

[8] L. George *HBase The Definitive Guide*, O'Reilly Media, Inc., 2011, pogl. 8.

[9] T. Macedo, F. Oliveira *Redis Cookbook*, O'Reilly Media, Inc., 2011, stran ix.