Student: Jovan Mladenović 1277

Profesor: Aleksandar Stanimirović

Distribuirane baze podataka - Cassandra

Sadržaj

[1. Uvod 2](#_Toc105794314)

[2. Osnove 3](#_Toc105794315)

[2.1. NoSQL 3](#_Toc105794316)

[3. Arhitektura 3](#_Toc105794317)

[3.1. Particioniranje podataka - konzistentno haširanje 3](#_Toc105794318)

[3.2. Konzistentno haširanje korišćenjem token prstena 3](#_Toc105794319)

[3.3. Višestruki tokeni na jednom fizičkom čvoru (vnodes) 4](#_Toc105794320)

# Uvod

Apache Cassandra je započela kao projekat unutar Facbook-a za brzu i laku pretragu konverzacija i drugog sadržaja koji je zanimao korisnike. Arhitektura je bila kombinovana sa modelom po kome je baziran Amazonovog Dynamo koji dozvoljava skalabilnost. Rezultat je bio visoko skalabilana baza podataka koja je mogla da izdrži najzahtevnije slučajeve korišćenja. Danas, Cassandra, je besplatna za upotrebu pod Apache 2.0 licencom. Neke od ključnih osobina Cassandre su:

* **Open** **source**
* **Fleksibilan i prepoznatljiv interfejs** – Kasandrin upitni jezik je jako sličan SQL-u, što olakšava prelazak većini programera
* **Visoke performanse** – svaki čvor može da obavlja sve operacija čitanja i upisa za razliku od tradicionalnih SQL sistema
* **Zero downtime** – ako neki čvor otkaže, korisnik se upućuje na sledeći dostupan čvor
* **Skalabilnost** – kasandra dozvoljava skaliranje prostim dodavanjem više čvorova
* **Efikasna replikacija**

# Osnove

## NoSQL

NoSQL ili Not Only SQL je baza podataka koja pruža mehanizam za čuvanje i čitanje podataka u netabelarnom obliku za razliku od relacionih baza. Pružaju fleksibilne šeme i lako skaliraju sa velikom količinom podataka i visokim brojem korisnika. Kod NoSQL baza pojedine operacije su brže nego kod SQL, izbor između njih zavisi od vrste problema kojeg je potrebno rešiti.

Neke od razlika su

|  |  |
| --- | --- |
| Relacione baze | Nosql baze |
| Moćan upitni jezik | Ogrničen upitni jezik |
| Prati ACID (Atomicity, Consistency, Isolation, and Durability) | Eventualno postaje konzistentna |
| Podržava transakcije | Ne podržava transakcije |

## Arhitektura

Kasandra se oslanja na nekoliko tehnika preuzetih od Amazonovog Dynamo distribuiranog key-value sistema. Konkretno, Kasandra se bazirano na Dynamu oslanja na:

* Particioniranje podataka korišćenjem konzistentnog haširanja.
* Multi-master replikaciju korišćenjem verzioniranih podataka
* Distribuirano članstvo klastera i detekcija otkazivanja bazirana na protokolu tračarenja (gossip)
* Ikrementalno skaliranje na pristupačnom harveru

### Particioniranje podataka - konzistentno haširanje

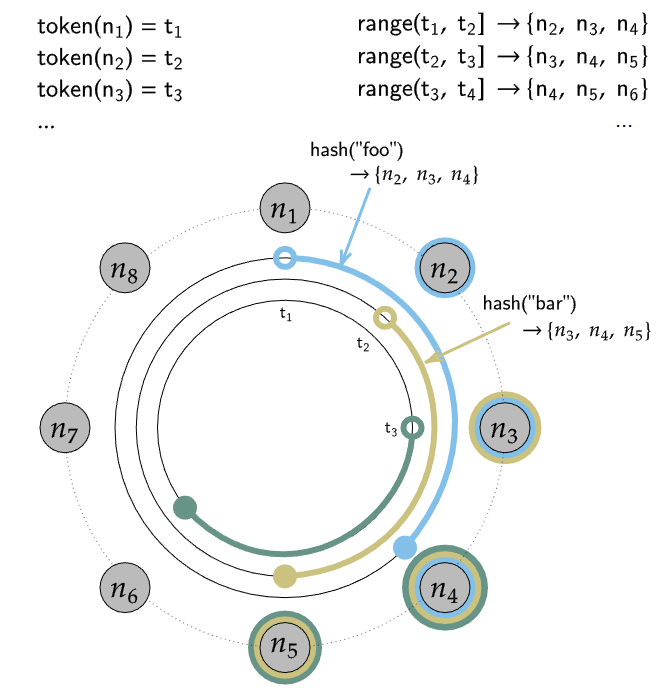
Kasandra ostvaruje horizontalnu skalabilnost particioniranjem svih podataka sačuvanim unutar sistema korišćenjem haš funkcije. Svaka particija se replicira na više fizičkih čvorova unutar klastera. Kako svaka replika prihvata izmenu podataka koju poseduje, svaki ključ mora biti verzioniran. Kasandra za razliku od Dynamo-a, koji je koristio vektorske časovnike, ima prostiji sistem – poslednji upis pobeđuje, gde svaki ključ ima svoji timestamp i poslednja verzija podatka je pobednik.

### Konzistentno haširanje korišćenjem token prstena

Kasandra particionira podatke unutar čvorova korišćenjem specijalne vrste haširanja koje se zove conzistentno haširanje. Kod obišnog data haširanja tipično se alociraju ključevi particiji uzevši haš datog ključa po modulu broja čvorova.

Kasandra umesto toga mapira svaki čvor na jedan ili više tokena na kontinualnom haš prstenu, i odlučuje o vlasništvu nad ključem haširanjem ključa na prsten i „kružeći“ po prstenu u jednoim smeru. Glavna razlika izmedju konzistentnog i naivnog haširanja je kada se broj čvorova promeni, konzistentno haširanje mora da preseli samo mali deo ključeva.

Na primer, ako imamo klaster sa osam čvorova sa podjednako distanciranim tokenima i replikacioni faktor od tri, da bi našli koji čvorovi poseduju neki ljuč mi prvo generišemo haš tog ključa i zatim idemo po prsteu u smeru kazaljki na satu dok ne dodjemo do tri različitih čvorova koji sadrže replike traženog ključa. Na slici 3.2.1 vidimo da u Dynamo sistemu, raspon ključeva (poznat kao token ranges), se mapiraju na isti skup fizičkih čvorova. U ovom primeru svi ključevi koji ne spadaju u rasponu tokena 1 i spadaju u raspon tokena 2 (grange(t1, t2]) se pamte na čvorovima 2, 3 i 4.



Slika 3.2.1 – primer klastera sa osam čvorova i replikacionim faktorom tri

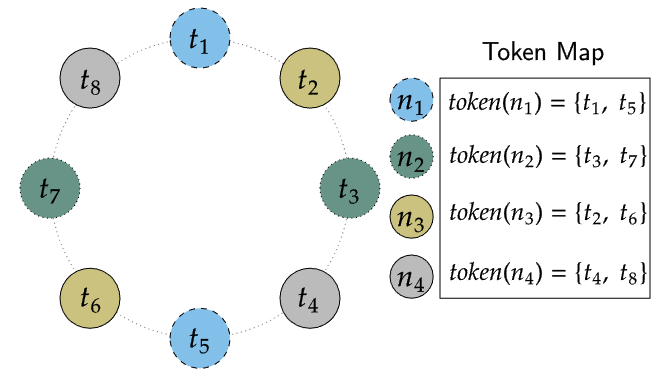
### Višestruki tokeni na jednom fizičkom čvoru (vnodes)

Konzistentno haširanje nad jednim tokenom funkcioniše dobro ako imamo mnogo fizičkih čvorova za raspodelu podataka, međutim ako imamo mali broj fizičkih čvorova sa podjednako distanciranim tokenima inkrementalno skaliranje (dodavanje samo par čvorova) je teško jer nema tokena za izbor koji bi ostavili prsten da bude balansiran. Kasandra izbegava nebalansirane tokene jer nejednak balans tokena vode do nebalansiranog opterećenja zahtevima. Na primer u primeru sa slike 3.2.1 dodavanje novog tokena bez izazivanja nebalansa.

Dynamo pristup zahteva koršćenje „virtuelnih čvorova“ kako bi se rešio ovaj problem. Virtuelni čvorovi rešavaju ovaj problem dodeljivanjem više tokena po jednom fizičkom čvoru. Dodavanje višestrukih tokena jednom fizičkom čvoru unutar prstena male klastere činimo većima i dodavanjem samo jednog fizičkog čvora deluje kao da smo dodali mnogo. Kasandra definiče nekoliko stavki kako bi se shvatio ovaj koncepti:

* **Token** – jedna pozicija na dynamo haš prstenu
* **Endpoint** – jedan fizički IP i port na mreži
* **Host ID** – jedinstveni identifikator za jedan fizički čvor , nalazi se na jednom endpointu i sadrži nekoliko Tokena
* **Virtuelni čvor** – token na haš prstenu u vlasništvu jednog fizičkog čvora

Mapiranje tokena se nalazi unutar **Token Map** mape gde Kasandra prati koja se pozicija na prstenu mapira na koji fizički čvor. Primer sa slike 3.3.1 predstavlja klaster sačinjen od četiri fizička čvora dodeljivanje svakom po dva tokena.



Slika 3.3.1 – klaster sačinjen od četiri fizička čvora dodeljivanje svakom po dva tokena

Višestruki tokeni po fizičkom čvoru imaju sledeće benefite:

* Kada se doda novi čvor on prihvata približno istu količinu podataka od drugih čvorova što dovodi do podjednake distribucije podataka unutar klastera.
* Kada se isključi čvor gubi količinu podataka približno jednakoj količini na drugim čvorovima, ponovo održavajući podjednaku raspodelu
* Ako švor postane nedostupan, opterećenje upita se distribura po više drugih čvorova

Međutim, ima i svojih nedostataka

* Svaki token dodaje i do *2 \* (Replikacioni faktor - 1)* dodatnih „komšija“ na prsten, što dovodi do većeg broja kombinacija otkaza čvorova gde gubimo pristup delu token prstena.
* Održavanje na nivou klastera je usporeno. Na primer ako se poveća broj tokena po čvoru, broj diskretnih operacija popravke koje klaster mora da obavlja se povećavaju
* Operacije koje zahvataju veliki opseg tokena mogu biti usporene

# Verzioniranje podataka i podesiva kozistentnost

Kasandra vrši replikaciju svake particije na više čvorova unutar klastera kako bi održala dostupnost i izdržljivost. Kada dođe do promene podataka, vrži se kaširanje particionog ključa kako bi se odredio raspon tokena kome podatak pripada, zatim vrši replikaciju promene na osnovu strategije repliciranja.

Sve strategije replikacije imaju faktor replikacije (RF), koji govori Kasandri koliko kopija particije treba da postoje. Na primer ako je RF = 3, podaci se upisuju u tri različite replike. Replike su uvek takve da se nalaze na različitim fizičkim čvorovima. Strategije replikacije mogu takođe da odluče da preskoče čvorove koji se nalaze u istom domenu otkaza kao što su na primer isti rafovi ilidata centri.

## Strategija replikacije

Svaki keyspace ima svoju strategiju replikacije. Sve produkcije bi trebale da koriste *NetworkTopologyStrategy* dok za testiranje klastera dok raspored data centara nije poznat bi trebala da se koristi strategija *SimpleStrategy.*

* **NetworkTopologyStrategy** - zahteva da se specificira faktor replikacije za svaki data centar unutar klastera, čak i ako klaster ima samo jedan data centar. NetworkTopologyStrategy je preporučen jerolakšava dodavanje novih fizičkih ili virtuelnih data centara u klaster. Takođe ako je broj rafova veći ili jednak faktoru replikacije NetworkTopologyStrategy garantuje kreiranje replika untar data centra na različitim rafovima specificiranom od strane izdajice (Snitch-a).
* **SimpleStrategy** - se odnosi prema svim čvorovima identično, ne uzimajući u obzir konfigurisane data centre ili rafove.

## Verzioniranje podataka

Kasandra koristi mutaciono timestamp verzioniranje za garantovanje eventualnu konzistentnost podataka. Kada god izmena udje u sistem ima sa sobom timestamp klijentskog časovnika, ako ne onda od dobija timestamp od šasovnika koordinatorskog čvora. Ažuriranja se razrešavaju po principu poslednji upis pobeđuje.

Kako repike unutar Kasandre mogu nezavisno da prihvataju izmene, moguće je da neke replike imaju novije verzije od drugih. Kako bi se garantovala eventualna kozistentnost Kasandra implementira **anti-entropy repair <repair>** gde replike kalkulišu hijerarhijska haš stabla na osnovu svojih data setova (Markle stabla) koja se mogu upoređivati između replika kako bi identifikovali podatke koji se ne poklapaju, kao što je u Dynamo-u kod Kasandre replike generišu periodično Markle stabla i razmenjuju ih međusobno i sinhronizuju podatke koji se ne poklapaju. Za razliku od Dynamo-a, Kasandra, implementira takođe **sub-range repair** i **incremental repair.** Sub range repair dozvoljava Kasandri da poveća rezoluciju kaš stabla (čak do nivoa particije) kreirajući viče stabla koja obuhvataju porcije celog skupa podataka. Incremental repair dozvoljava Kasandri da popravli particije koje su se promenile od poslednje popravke.

## Podesiva kozistentnost

Kasandra podržava „po poeraciji“ razmenu između konzistentnosti i pristupašnosti pomoći **Nivoa konzistentnosti.** Nivoi konzistentnosti su verzija Dynamo-ovog *R + W > N* mehanizma konzistentnosti koji određuje da broj čvorovia koji izvršavaju čitanje (R) i broj švorova koji izvrčavaju upis (W) mora biti veći od faktora replikacije (N). U kasandri se mežutim bira iz niza nivoa konzistentnosti koji dozvoljavaju operatoru da bira ponašanje R i W bez poznavanja faktora replikacije. Postoje:

* ONE – samo jedna replika mora da odgovori
* TWO – dve replike moraju da odgovore
* THREE – tri replike moraju da odgovore
* QUORUM – većina (n/2 + 1) replika moraju da odgovore
* ALL – sve replike moraju da odgovore
* LOCAL\_QUORUM – većina replika u lokalnom datacentru (u kome god datacentru se nalazi koordinator) moraju odgovoriti.
* EACH\_QUORUM – većina replika u svakom data centru moraju da odgovore
* LOCAL\_ONE – samo jedna replika mora da odgovori, unutar višestrukih data centar klastera ovo garantuje da se zahtev ne šalje na remote datacentre

Operacije upisa se uvek šalju svim replikama, bez obzira na nivo konzistentnosti.

# Člansto distribuiranog klastera i detekcija otkaza

Protokoli replikacije i particioniranje podaataka se oslanja na informacije koji čvorovi su živi a koji mrtvi unutar klastera kako bi se operacije čitanja i upisa optimalno rutirale. U Kasandri informacije od ovome se dele distribuirano kroz mehanizam detekcije otkaza baziranom na protokolu tračarenja (gossip).

## Tračarenje (gossip)

Tračarenje je kako Kasandra propagira osnovne informacije o klasteru. U kasandri čvorovi razmenjuju informacije ne samo o svom stanju već i o čvorovima za koje oni imaju informacije. Ove informacije se verzioniraju časovnim vektorom *[generation, version],* gde je generacija je timestamp i verzija je logički sat koji se uvečava svake sekunde. Ovaj par časovnika dozvoljava Kasandrinom tračarenju da ignoriše stare verzije stanja klastera. Svaki čvor unutar klasterra izvršava zadatak trtačarenja nezavisno i periodično. Svake sekunde svaki čvor unutar klastera:

* Ažurira lokalno heartbeat stanje (verziju) i konstruiše čvorov lokalni pogled na klastera.
* Bira nasumično drugi čvor sa kojim razmenjuje trač.
* Pokušava da komunicira se nedostupnim čvorem, ako takav postoji

Trač takođe propagira token metadata i verzije podataka. Ako