

Apache Cassandra

High Availability Rešenja

Đorđe Nikolić | Sistemi za upravljanje bazama podataka | jun 2022.

# Sadržaj

[Sadržaj 1](#_Toc106838568)

[Uvod 2](#_Toc106838569)

[Visoka dostupnost 2](#_Toc106838570)

[Principi 2](#_Toc106838571)

[Problemi sa tradicionalnim sistemima baza podataka 3](#_Toc106838572)

[ACID 3](#_Toc106838573)

[Monolitska arhitektura 3](#_Toc106838574)

[Master-slave arhitektura 4](#_Toc106838575)

[Sharding 5](#_Toc106838576)

[Prelaz u slučaju otkazivanja master čvora 6](#_Toc106838577)

[Apache Cassandra 7](#_Toc106838578)

[Arhitektura 7](#_Toc106838579)

[Citirana dela 9](#_Toc106838580)

# Uvod

U ovom radu će biti diskutovan koncept sistema baza podataka za visokom dostupnošću (engl. *High Availability*, u daljem tekstu HA). Biće govora o samoj definiciji datog koncepta, problema sa implementiranjem velike dostupnosti kod standardnih relacionih baza podataka, kao i načina putem kojih je moguće rešiti te probleme. Zatim, biće predstavljan sistem Apache Cassandra i njegove osnovne odlike, nakon čega će ostatak rada biti posvećen načinima i mehanizmima putem kojih ovaj sistem realizuje visoku dostupnost.

# Visoka dostupnost

Dostupnost predstavlja mogućnost korisnika da pristupi i koristi određeni servis ili resurs. [1] U slučaju da korisnik ne može koristiti servis u datom trenutku, on se smatra nedostupnim. Visoka dostupnost je sposobnost sistema da kontinualno funkcioniše, tj. bude dostupan korisnicima, bez prekida tokom određenog vremenskog perioda. Drugim rečima, prilikom dizajniranja HA sistema, fokus je na garantovanju visokog procenta dostupnosti tokom određenog vremenskog perioda, a zlatan standard u industriji je poznat kao „dostupnost pet devetki“, tj. sistem koji je dostupan 99.999% vremena. Modernizacija je dovela do toga da postoji veliko oslanjanje korisnika na pojedine servise, te je visoka dostupnost od veće važnosti nego ikada pre.

## Principi

Prilikom dizajniranja HA sistema postoje tri glavna principa koji se prate kako bi se postigla najviša moguća dostupnost [2]:

* **Eliminisanje tački propasti**: Tačka propasti je jedna komponenta koja bi u slučaju da otkaže, dovela do toga da ceo sistem propadne. Ako se servis pokreće samo sa jednog servera, onda je taj server tačka propasti. U slučaju da taj server prestane da radi, pada ceo servis.
* **Pouzdan prelaza**: Kreiranje redundantnosti u ovim sistemima je od velike važnosti. Redundantnost omogućava da rezervna komponenta preuzme u slučaju da glavna otkaže. U slučaju da dođe do toga, obavezno je obezbediti pouzdan prelaz sa komponente koja je otkazala, na rezervnu, bez gubitka podataka ili smanjenja dostupnosti.
* **Detekcija propasti**: Sistem mora biti sposoban da detektuje komponente koje otkažu, i pokrene određene procedure tako da nastavi da funkcioniše u potpunosti. Korisniku kvar ne sme biti vidljiv.

## Problemi sa tradicionalnim sistemima baza podataka

Tradicionalno, bilo je teško razviti sisteme baza podataka koje odlikuje visoka dostupnost, a to je posebno tačno za relacione sistemi baza podataka koji su proteklih nekoliko decenije činili većinu tržišta [3]. Ovi sistemi su uglavnom dizajnirani da funkcionišu na jednoj mašini.

### ACID

Jedna od najvećih prepreka ka ostvarivanju visoke dostupnosti kod tradicionalnih sistema baza podataka je to što pokušavaju da striktno garantuju ACID svojstva:

* **Atomičnost** - izvršavanje transakcije po metodi „sve ili ništa“
* **Konzistentnost** – održavanje integriteta podataka u svim kopijama
* **Izolovanost** – održavanje pravilnog redosleda izvršenja transakcija
* **Trajnost** – svi upisi se čuvaju na trajnom medijumu podataka

Arhitekte uglavnom ostvaruju ova svojstva kroz komplikovane mehanizme koji dovode do žrtvovanja dostupnosti. Kao rezultat toga, pokušaji postizanja visoke dostupnosti zahtevaju implementaciju dodatnih mehanizma preko samog sistema, sa ciljem da se originalna svojstva očuvaju.

### Monolitska arhitektura

Najprostiji dizajn koji može da garantuje ACID svojstva uključuje monolitsku arhitekturu kod koje sve funkcionalnosti postoje samo na jednoj mašini. Sa obzirom na to da ne postoji nikakva komunikacija između više čvorova, poštovanje ACID pravila nije komplikovano.

Diagram

Description automatically generated

Slika 1. Primer proste monolitske arhitekture servisa [3]

Povećavanje dostupnost u takvim sistemima se često postiže uvođenjem poboljšanja na hardverskom nivou, kroz RAID diskove, više mrežnih interfejsa, itd. Doduše, ništa od toga ne menja činjenicu da server ostaje tačka propasti. Ako server otkaže, servis postaje nedostupan.

Jedna od čestih metoda povećanja kapaciteta obrade zahteva kod ovakve arhitekture je prebacivanje sloja skladištenja na jednu deljenu komponentu, kao što su NAS (engl. *network attached storage*) i SAN (engl. *storage attached network*) mreže. Takve komponente su uglavnom veoma pouzdane, sa velikim brojem diskova i mrežnih interfejsa. Kao što se da primetiti, iako ovakav pristup omogućava veći kapacitet obrade zahteva, i samim tim povećava dostupnost u slučaju velikog priliva zahteva, tačka propasti i dalje postoji u formi sloja skladištenja.

Diagram

Description automatically generated

Slika 2. Primer više servera sistema baza podataka povezanih na NAS [3]

### Master-slave arhitektura

Kako su distribuirani sistemi postavi sve više prisutni, potreba za distribuiranim sistemima visokog kapaciteta je porasla. Mnoge distribuirane baze podataka i dalje pokušavaju da održe ACID svojstva (a u nekim slučajevima, samo konzistentnost, koje je i najteže svojstvo za postići u distribuiranom okruženju). Ovo je dovelo do kreiranje master-slave arhitektura.

Kod ovakvih sistema, iako mogu postojati mnogo servera koji obrađuju zahteve, samo jedan server može vršiti upise, kako bi podaci ostali konzistentni. Ovim izbegavamo scenario u kome isti podatak može biti izmenjen kroz više čvorova istovremenim zahtevima.

Ovim pristupom i dalje nije rešen problem dostupnosti, jer ako server preko koga se vrše upisi otkaže, dostupnost servisa pada. Takođe, obrada velikog broja upisa postaje potencijalno neperformantna, jer se svi upisi šalju na jedan server.

Diagram

Description automatically generated

Slika 3. Primer master-slave arhitekture [3]

### Sharding

Diagram

Description automatically generated

Slika 4. Primer master-slave arhitekture sa primenjenom *sharding* tehnikom [3]

Varijacija na master-slave arhitekturu koja omogućava bolju obradu velike količine zahteva za upis je tehnika podele podataka na delove gde svaki deo pripada jednoj grupi servera koji prate master-slave arhitekturu (engl. *sharding*). Na primer, baza korisničkih profila može biti podeljena po prezimenu korisnika, tako da svi korisnici čija prezimena počinju slovima A-M pripadaju jednoj grupi servera, tj. jednom klasteru, dok svi korisnici čija prezimena počinju slovima N-Z pripadaju drugom klasteru.

Mane ovog pristupa su brojne, a prva je to što se unose više tački propasti, gde svaki master server predstavlja jednu od njih. Takođe, odgovornost upravljanja kom klasteru treba poslati neki zahtev pada na aplikacioni sloj. Dodavanjem novih klastera, podaci se moraju izmešati tako da se ispoštuje nova podela.

Neki sistemi koji koriste ovakvu tehniku, imaju dodatan sloj apstrakcije između aplikacije i fizičkih klastera koji odgovornost upravljanja zahtevima ka odgovarajućim klasterima prebacuje sa aplikacije na sebe.

### Prelaz u slučaju otkazivanja master čvora

Jedan od čestih načina povećanja dostupnosti u slučajevima otkazivanja master čvora (koji predstavljaju tačke propasti kod ovog tipa arhitekture) je prebacivanje master odgovornosti na neki novi čvor, tj. tehnika izbora vođe. Tačna implementacija ovog algoritma se razlikuje od sistema do sistema ali je koncept isti. Uglavnom, primena ovog algoritma dovodi do povećanja dostupnosti sistema sa master-slave arhitekturom.

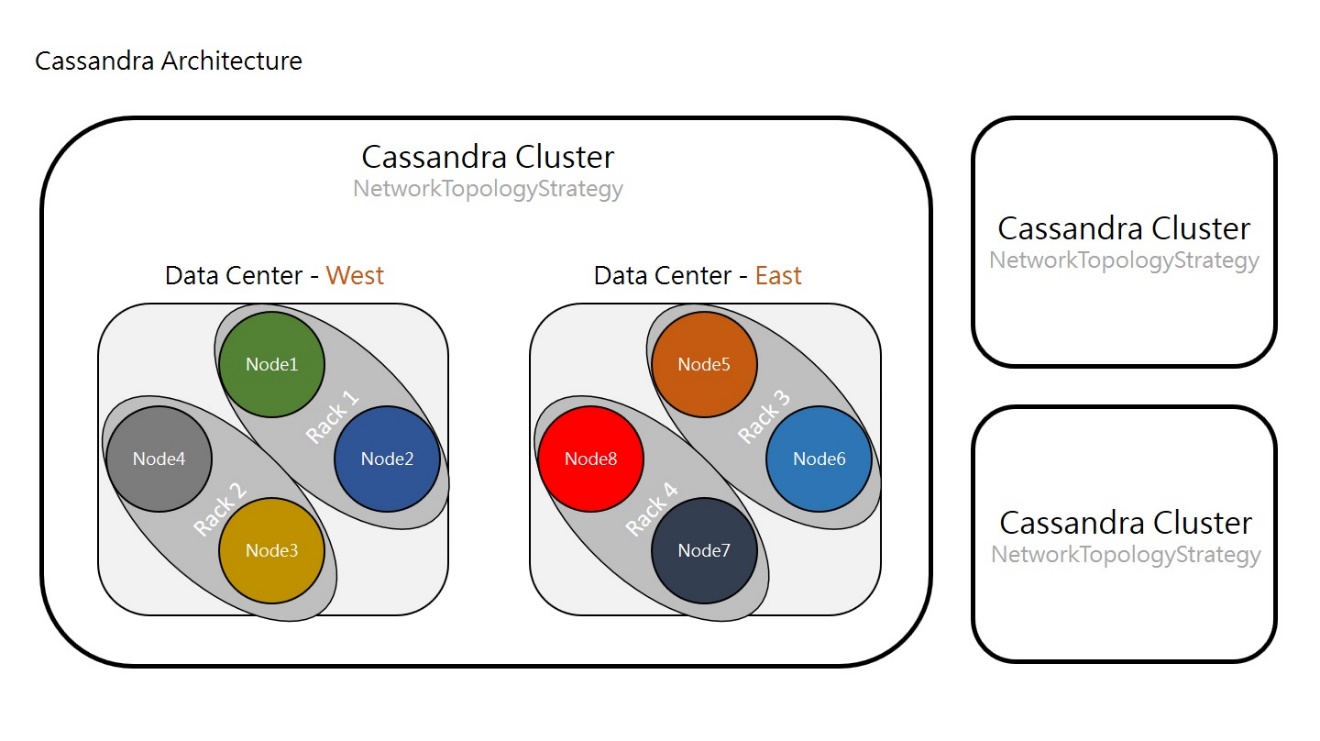
Čak i sistemi koji implementiraju gorepomenute tehnike, imaju veći broj mana:

* Aplikacije moraju da poznaju topologiju baze (ova mana se otklanja unosom novog sloja apstrakcije)
* Particije podataka se moraju pažljivo planirati
* Skaliranje obrade upisa je loše
* Prebacivanje master odgovornosti unosi dodatnu kompleksnost u sistem, posebno kod klastera koji se nalaze na više lokacija
* Dodavanje novih klastera zahteva novo razmeštanje podataka

# Apache Cassandra

Apache Cassandra je NoSQL (nerelaciona) distribuirana baza podataka koda otvorenog tipa čiji je glavni fokus na odličnoj skalabilnosti, otpornosti na greške, i dostupnosti. Bazirana je na Amazon Dynamo, i Google Bigtable sistemima, i originalno je razvijana od strane Facebook-a, iako je od jula 2008. godine celokupni kod sistema potpuno dostupan javnosti.

## Arhitektura



Slika 1. Arhitektura sistema Apache Cassandra na visokom nivou [4]

Cassandra je organizovana po distribuiranim klasterima koji sadrže više homogenih čvorova (*nodes*), i po njima su podaci raspoređeni i organizovani tako da nikad ne postoji takozvana „jedna tačka propasti“ [5]. Čvorovi su u stalnoj međusobnoj komunikaciji i razmenjuju informacije o trenutnom stanju koristeći *peer-to-peer* komunikacioni protokol. Na svakom čvoru se održava sekvencijalni *commit log* koji beleži svaki upis podataka, koji se nakon toga indeksiraju i upisuju u strukturu u memoriji, nazvanu *memtable* (jedna za svaku porodicu kolona). Svaki put kada se ova struktura popuni, podaci se upisuju na disk u formi *SSTable* fajlova. Svaki proces upisa u bazu je praćen automatskim particionisanjem i replikacijom na čvorovima u klasteru. Sistem periodično konsolidira podatke u SSTable fajlovima kroz specijalan proces kompakcije (engl. compaction) putem kog takođe briše zastarele podatke označene za brisanjem *tombstone* objektom. Kako bi se osiguralo stanje konzistentnosti svih podataka, sistem koristi različite *repair* mehanizme.

Cassandra je particionisana baza podataka koji svoje podatke čuva u redovima. Redovi su organizovani u tabele (porodice kolona) sa obaveznim primarnim ključem. Pristup bazi se vrši kroz CQL jezik, koji koristi sličnu sintaksu kao i SQL, jezik popularan među relacionim bazama podataka. Jedna od najvećih razlika između ova dva jezika, kao i odlika Apache Cassandra sistema koji ga najviše razlikuje od relacionih sistema baza podataka, je to što koncept spajanja tabela ne postoji. Takođe, nemoguće je pisati pod-upite. Podaci se čuvaju u denormalizovanom formatu, i prilikom dizajniranja baze podataka za neku aplikaciju, mora se pristupiti drugačije nego prilikom dizajniranja relacione baze podataka za istu namenu.

Zahtevi za čitanje ili upis podataka mogu biti poslati na bilo koji čvor u klasteru. Kada se klijent poveže na neki od čvorova, taj čvor postaje koordinator za interakciju sa tim klijentom. Preko koordinatora se vrši komunikacija između klijenta i čvorova koji zapravo sadrže potraživane podatke. On takođe određuje, na osnovu konfiguracije klastera, kom čvoru treba da prosledi zahtev na procesiranje.

# Citirana dela

|  |  |
| --- | --- |
| [1] | [Na mreži]. Available: https://en.wikipedia.org/wiki/High\_availability. |
| [2] | [Na mreži]. Available: https://www.techtarget.com/searchdatacenter/definition/high-availability. |
| [3] | R. Strickland, Cassandra High Availability, Packt, 2014. |
| [4] | [Na mreži]. Available: https://jiankaiwang.gitbooks.io/itsys/content/cassandra/architecture.html. |
| [5] | [Na mreži]. Available: https://docs.datastax.com/en/cassandra-oss/3.x/cassandra/architecture/archIntro.html. |
| [6] | [Na mreži]. Available: https://www.sciencedirect.com/science/article/pii/B9780128191545000187. |
| [7] | [Na mreži]. Available: https://www.javatpoint.com/query-processing-in-dbms. |
| [8] | [Na mreži]. Available: https://logicalread.com/sql-server-query-processing-w01/. |
| [9] | [Na mreži]. Available: https://www.sqlshack.com/sql-server-execution-plans-overview/. |
| [10] | [Na mreži]. Available: https://www.sqlshack.com/adaptive-query-processing-in-sql-server-2017/. |
| [11] | [Na mreži]. Available: https://subscription.packtpub.com/book/data/9781838826215/1/ch01lvl1sec09/intelligent-query-processing. |
| [12] | [Na mreži]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-ver15. |
| [13] | [Na mreži]. Available: https://docs.microsoft.com/en-us/sql/t-sql/functions/approx-count-distinct-transact-sql?view=sql-server-ver15. |
| [14] | [Na mreži]. Available: https://www.youtube.com/watch?v=bbXM3Pk9Ejw. |
| [15] | [Na mreži]. Available: https://www.sqlshack.com/sql-table-variable-deferred-compilation-in-sql-server-2019/. |
| [16] | [Na mreži]. Available: https://www.red-gate.com/simple-talk/databases/nosql/apache-cassandra-data-modeling-and-query-best-practices/. |
| [17] | [Na mreži]. Available: https://cassandra.apache.org/doc/latest/cassandra/architecture/overview.html. |
| [18] | [Na mreži]. Available: https://cassandra.apache.org/doc/latest/cassandra/operating/backups.html. |