Neo4j – High Availability rešenja

Student: Anđela Sokolović, 944

Sadržaj

1. Uvod

Visoka dostupnost (High availability) predstavlja sposobnost sistema da bude funkcionalan bez prekida u dugom vremenskom periodu. Sa današnjim potrebama tržišta, neophodno je da sistem koji opslužuje zahteve korisnika bude pouzdan i konstantno dostupan, pa iz tog razloga High Availability postaje nezaobilazna karakteristika svakog Enterprise sistema.

Važnost visoke dostupnosti ističe se u nekoliko domena:

- Pouzdanost – Ako postoji redundantnost među komponentama tako da jedna komponenta može preuzeti ulogu u slučaju prestanka rada druge, sistem može raditi neprestano (ili gotovo neprestano).
- Skalabilnost – Uvođenjem klastera u arhitekturu pojednostavljuje se nadograđivanje sistema.
- Lakše održavanje – Sistemi koji se sastoje iz više čvorova su mnogo lakši za održavanje, jer ne postoji pritisak da se neko ažuriranje mora obaviti brzo. Dok se jedan čvor ažurira, ostali će nastaviti nesmetano da rade.
- Sigurnost – Čvorovi mogu biti podeljeni u slojeve, pri čemu svaki sloj može imati različitu ulogu, permisije i nivo zaštite.

1.1. Mere visoke dostupnosti

Visoka dostupnost se može precizno izračunati pomoću formule:

$$MTTF / (MTTF + MTTR)$$

- MTTF – Mean time to failure: Procenjeno vreme kada sistem neće biti dostupan ako popravka nije moguća
- MTTR – Mean time to repair/replace: Prosečno vreme za popravku ili zamenu komponente koja je prestala sa radom

Na osnovu ove formule, dobija se rezultat koji predstavlja meru dostupnosti sistema, koji se često utvrđuje i prebrojavanjem ’devetki’ u rezultatu (tabela 1):

Availability %	Downtime/year	Downtime/month	Downtime/week	Downtime/day
90.0% (one 9)	36.53 days	73.05 hours	16.8 hours	2.4 hours
99.0% (two 9s)	3.65 days	7.31 hours	1.68 hours	14.4 minutes
99.9% (three 9s)	8.77 hours	43.83 minutes	10.08 minutes	1.44 minutes
99.99% (four 9s)	52.6 minutes	4.38 minutes	1.01 minutes	8.64 seconds
99.999% (five 9s)	5.25 minutes	26.3 seconds	6.05 seconds	864 ms
99.9999% (six 9s)	31.56 seconds	2.63 seconds	604.8 ms	86.4 ms

Tabela 1 - Mere dostupnosti sistema

Prikazana tabela predstavlja broj sati u toku dana, nedelje, meseca ili godine kada sistem neće biti dostupan, zavisno od mere dostupnosti sistema.

2. High Availability rešenja

U bazi Neo4j, postoji dve vrste rešenja visoke dostupnosti: High Availability Clusters i Causal Clusters. High availability clusters rešenje je starija verzija sa koje je sistem migrirao na kauzalne klastere, pa će o njemu biti reči isključivo teorijski, radi poređenja sa aktuelnom verzijom. Sve komande koje budu bile navedene za HA rešenje imaju strogo ilustrativnu ulogu.

2.1. High Availability

High Availability klasteri su sačinjeni od najmanje 3 čvora: jednog mastera i dva slave čvora. Instanca master čvora izvodi write operacije i nakon toga šalje podatke slave čvorovima. Čitanje se može obaviti sa bilo kog tipa čvora. Poželjno je da svi čvorovi u klasteru imaju identičan hardver. Moguće je konfigurisati slave instancu kao arbitražnu, što znači da ona neće učestvovati u skladištenju podataka, već će imati ulogu isključivo pri odabiru novog master čvora. Arbitražni čvor ne mora imati iste hardverske mogućnosti kao ostali čvorovi, već može imati i slabije performanse.

2.2. Kauzalni klasteri

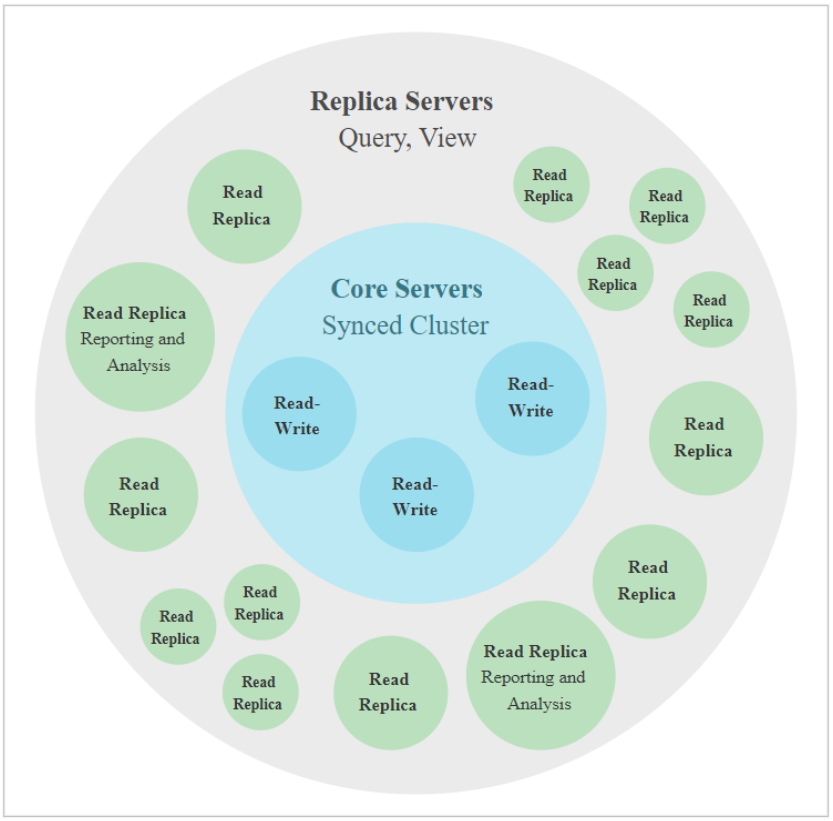
Kauzalni klasteri su nova verzija rešenja visoke dostupnosti, dostupna od 2019. godine. Glavne prednosti ovog rešenja su:

- Bezbednost: Core serveri pružaju platformu koja je otporna na otkaze za procesiranje transakcija, koja će funkcionisati dok god funkcioniše većina (više od polovine) servera.

- Skaliranje: Read replike su izuzetno lake za skaliranje, zbog čega je moguće obraditi ogromne grafove na široko distribuiranoj topologiji.
- Kauzalna konzistencija: Klijentska aplikacija će pročitati najmanje svoje upise, dok se ne uspostavi realna konzistencija.

Struktura

Kauzalni klaster se sastoji iz dva tipa čvorova: Core servera i Read replike (slika 1).



Slika 1 - Struktura kauzalnog klastera

Core serveri

Core serveri su odgovorni za čitanje i upis podataka. Njihova uloga je replikacija transakcija korišćenjem Raft protokola. Da bi se upis podataka izvršio, većina Core servera se mora složiti i prihvatiti transakciju. Broj core servera direktno utiče na brzinu izvršavanja upisa. Poželjno je imati veći broj servera zbog redundantnosti, ali je istovremeno teže uspostaviti konsenzus i ova operacija zahteva više vremena, jer se bez obzira na broj core servera u sistemu, za svaku transakciju mora složiti više od polovine ukupnog broja. Iz ovog razloga je potrebno napraviti kalkulaciju vezano za performanse i odlučiti koliko minimalno core servera je neophodno da bi sistem bio otporan na otkaze. U praksi je ovaj broj relativno

mali, i može se izračunati pomoću formule M = 2F + 1. M predstavlja broj servera koji je potreban da bi sistem bio tolerantan na F otkaza. Moguće je kreirati kauzalni klaster koji ne ispunjava datu formulu, ali takav klaster neće biti otporan na otkaze. U tom slučaju, ako broj servera koji još uvek rade ne predstavlja većinu, oni će postati read only čvorovi.

Postoje dve vrste core servera: leader i follower. Uloga leadera je da izdaje komande followerima kada je potrebno obaviti upis. Svaki core server može izvršiti upis i svi serveri imaju identične replike podataka.

Ako se core server neplanirano ugasi, u slučaju da nije bio glavni, promena se neće uočiti odmah. Veličina core klastera će se privremeno zadržati zbog dobijanja konsenzusa prilikom transakcija. Ako je u sistemu bilo 5 servera i jedan od njih se ugasi, za konsenzus će svakako biti potrebno 3 čvora, dok se konfiguracija ne obnovi.

Ako je core server koji se isključio bio glavni čvor, čvorovi će započeti izbor novog glavnog čvora. Nakon što se glavni čvor izabere, klaster će nastaviti da funkcioniše sa istom većinom glasova kao i pre otkaza.

Read replike

Osnovna uloga read replike je skaliranje podataka iz grafa. Read replika se ponaša kao keš podataka nad kojim Core čvor može izvršiti read only operacije. Read replike se dobijaju asinhronim kopiranjem Core servera putem transakcionog loga. Replika periodično zahteva od servera novi transaction log kako bi podaci na njima ostali konzistentni. Broj read replika je obično veliki i ne treba da utiče na tolerantnost na otkaze.

Životni ciklus replike se može pratiti preko zajedničke tzv bele table. Svaka živa replika u sistemu je evidentirana na beloj tabli. Ako se replika neplanirano isključi, Core server će detektovati prekid konekcije. Stavka replike na beloj tabli će privremeno biti sakrivena. Ako se replika ne uključi u predviđenom vremenskom periodu, njena stavka će biti trajno izbrisana sa table. Jednom kada se replika ponovo uključi, moguće je da će njeno stanje podataka biti nekonzistentno. Replika će u tom slučaju zatražiti najnovije transakcije, i obrisati sve transakcije koje su izvršene delimično pre njenog isključenja.

Kauzalna konzistencija

Kauzalna konzistencija obezbeđuje da korisnik upisuje u Core server, i da zatim čita te podatke sa neke od read replika na koju su replicirani. Korisnik prilikom transakcije dobija bookmark koji će prosleđivati prilikom svake sledeće transakcije, na taj način kreirajući kauzalni lanac. Bookmark se koristi za proveru da li se na određenoj read replici nalazi traženi podatak ili ne. Ovim se osigurava da će korisnik dobiti poslednju verziju koju je on ažurirao. Sistem će se replikacijom posle nekog vremena dovesti u potpuno konzistentno stanje, ali ovaj pristup omogućava da u trenucima nekonzistencije, svaki korisnik dobije njemu poslednje poznat upis.

2.3. Oporavak od otkaza nakon dužeg vremena

Ako se desi da se čvor neplanirano isključi i ostane isključen duže vreme, neće biti moguć automatski oporavak pomoću transakcionih logova, jer se može desiti da su takvi čvovi u međuvremenu obrisani na master (core) čvoru[1]. Zbog nekonzistentnih podataka, čvor neće moći da se ponovo pridruži klasteru. U ovom slučaju, koristi se potpuni oporavak podataka na slave čvoru/read replici.

Koraci za restore su sledeći:

- Identifikovati grešku u log fajlovima

```
2017-02-12 15:33:37.334+0000 INFO
[o.n.k.h.c.SwitchToSlaveBranchThenCopy] The store is
inconsistent. Will treat it as branched and fetch a new one
from the master

2017-02-12 15:33:37.334+0000 WARN
[o.n.k.h.c.SwitchToSlaveBranchThenCopy] Current store is
unable to participate in the cluster; fetching new store from
master The master is missing the log required to complete the
consistency check
```

Prilikom pokušaja čvora da se vrati u klaster, u log fajlovima će se prikazati rezultati usaglašavanja memorijskog stanja čvora sa ostalim čvorovima u klasteru. Greška koja se pritom dobija je veoma deskriptivna i precizno daje korisniku do znanja gde je nastao problem.

[1] Naziv zavisi od verzije. Master čvor je HA terminologija, dok je Core čvor deo terminologije Kauzalnih klastera.

- Identifikovati master/core instancu

Ukoliko se koristi HA verzija, master čvor se može identifikovati korišćenjem HTTP zahteva u formatu `/db/manage/server/ha/master.`

Komanda se izdaje iz terminala, zajedno sa navođenjem parametara hosta. Ako želimo da dobijemo verbalni odgovor, dodaje se fleg –v. Odgovor je prikazan na slici 2:

```
#> curl -v localhost:7474/db/manage/server/ha/master
*   Trying 127.0.0.1
* Connected to localhost (127.0.0.1) port 7474 (#0)
> GET /db/manage/server/ha/master HTTP/1.1
> Host: localhost:7474
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 17 Feb 2017 16:38:37 GMT
< Content-Type: text/plain
< Access-Control-Allow-Origin: *
< Transfer-Encoding: chunked
< Server: Jetty(6.1.25)
<
* Connection #0 to host localhost left intact
true
```

Slika 2 - Odgovor na HTTP master zahtev

Ako čvor koji smo kontaktirali jeste master, dobijeni odgovor će biti 200 (OK). U slučaju da je čvor slave ili je njegova uloga nepoznata, odgovor će biti 401 (Not found). Ukoliko je omogućena sigurnosna provera, prilikom slanja zahteva moraju se poslati i kredencijali korisnika, u formatu `--user <username>:<password>.`

U kauzalnim klasterima, Core čvor se može pronaći izvršavanjem komande `CALL dbms.cluster.role(database-name).` Ova komanda se može izvršiti nad svakom instancom u sistemu, i kao odgovor na nju dobija se uloga čvora u string formatu. Uloga čvora može biti LEADER, FOLLOWER ili READ_REPLICA.

Generalnija komanda je `CALL dbms.cluster.overview()` , koja kao odgovor vraća čitavu topologiju klastera, uključujući detalje o svakoj instanci. Svaka instanca poseduje ID, adresu, grupu servera kojoj pripada i ulogu u sistemu (tabela 2). Uloga je prikazana u formatu mape, gde je ključ baza podataka kojoj čvor pripada, a vrednost uloga čvora u toj bazi. Ova komanda se može pozvati isključivo sa Core instance, jer jedino takav čvor ima uvid u topologiju sistema.

id	addresses	groups	databases
08eb9305-53b9-4394-9237-0f0d63bb05d5	[bolt://neo20:7687, http://neo20:7474, https://neo20:7473]	[]	{system: LEADER, neo4j: FOLLOWER}
cb0c729d-233c-452f-8f06-f2553e08f149	[bolt://neo21:7687, http://neo21:7474, https://neo21:7473]	[]	{system: FOLLOWER, neo4j: FOLLOWER}
ded9eed2-dd3a-4574-bc08-6a569f91ec5c	[bolt://neo22:7687, http://neo22:7474, https://neo22:7473]	[]	{system: FOLLOWER, neo4j: LEADER}
00000000-0000-0000-0000-000000000000	[bolt://neo34:7687, http://neo34:7474, https://neo34:7473]	[]	{system: READ_REPLICA, neo4j: READ_REPLICA}
00000000-0000-0000-0000-000000000000	[bolt://neo28:7687, http://neo28:7474, https://neo28:7473]	[]	{system: READ_REPLICA, neo4j: READ_REPLICA}
00000000-0000-0000-0000-000000000000	[bolt://neo31:7687, http://neo31:7474, https://neo31:7473]	[]	{system: READ_REPLICA, neo4j: READ_REPLICA}

Tabela 2 - Odgovor overview komande

Takođe, na sličan način kao u HA implementaciji, core čvor se može dobiti i putem HTTP zahteva, izvršavanjem komande `/db/manage/server/core/writable` iz terminala. Odgovor je veoma sličan odgovoru iz HA primera (slika 3). Čvor će vratiti kod 200 ako jeste leader, i 401 ako je follower ili mu je uloga nepoznata.

```
#> curl -v localhost:7474/db/manage/server/core/writable
*   Trying ::127.0.0.1
* Connected to localhost (127.0.0.1) port 7474 (#0)
> GET /db/manage/server/core/writable HTTP/1.1
> Host: localhost:7474
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 17 Feb 2017 16:38:37 GMT
< Content-Type: text/plain
< Access-Control-Allow-Origin: *
< Transfer-Encoding: chunked
< Server: Jetty(9.2.9 v20150224)
<
* Connection #0 to host localhost left intact
true
```

Slika 3 - Odgovor na master HTTP master zahtev u kauzalnom klasteru

Što se sigurnosti tiče, isto pravilo važi za kauzalne klastere kao i za HA. Ukoliko su sigurnosna podešavanja omogućena, uz zahtev se moraju poslati i korisničko ime i lozinka.

- Izvršiti backup master/core čvora

U ovom slučaju, zahteva se pravljenje kompletne kopije podataka, kako bi oni mogli da se iskopiraju na slave/follower čvor.

```
$neo4j-home> mkdir /mnt/backup
```

```
$neo4j-home> bin/neo4j-admin backup --from=192.168.1.34 --
backup-dir=/mnt/backup --name=graph.db-backup
Doing full backup...
2017-02-01 14:09:09.510+0000 INFO [o.n.c.s.StoreCopyClient]
Copying neostore.nodestore.db.labels
2017-02-01 14:09:09.537+0000 INFO [o.n.c.s.StoreCopyClient]
Copied neostore.nodestore.db.labels 8.00 kB
2017-02-01 14:09:09.538+0000 INFO [o.n.c.s.StoreCopyClient]
Copying neostore.nodestore.db
2017-02-01 14:09:09.540+0000 INFO [o.n.c.s.StoreCopyClient]
Copied neostore.nodestore.db 16.00 kB
```

- Iskopirati backup na čvor na kom je došlo do otkaza

Kopiranje se vrši komandom:

```
scp -r /path/to/neo4j/backup
username@<SLAVE_ADDRESS>:/path/to/destination
```

- Stopirati instancu čvora na kom je došlo do otkaza

Za stopiranje se koristi sledeća komanda:

```
$NEO4J_HOME/bin/neo4j stop
```

- Napraviti backup podataka čvora na kom je došlo do otkaza (Opciono)

Prilikom kopiranja podataka sa mastera na slave čvor (sa leadera na followera), poželjno je odraditi i backup slave/follower čvora, kako bi sistem mogao da se vrati u to stanje za slučaj da kopiranje ne uspe.

- Restore backup (oporavak)

U ovom koraku, potrebno je učiniti backup kopiju sa master čvora validnom verzijom podataka na slave čvoru. U starijim verzijama, ova operacija se izvodi prostim kopiranjem backup-a u relevantni folder, dok u novim postoji komanda restore:

```
$NEO4J_HOME/bin/neo4j-admin restore --from=/mnt/backup --
database=graph.db-backup –force
```

Navođenjem parametra –force na kraju komande, sistemu se nalaže da prepiše sve podatke koji se već nalaze u destinacionom folderu.

- Startovati instancu

Nakon uspešnog kopiranja podataka, potrebno je startovati instancu slave/follower čvora:

```
$NEO4J_HOME/bin/neo4j start
```

Čvor će automatski započeti discovery proces kako bi se priključio klasteru, i zatražiti transakcione logove kako bi ispravio eventualnu nekonzistentnost nastalu nakon kreiranja backupa na masteru.

- Obrisati stare fajlove (Opciono)

Ukoliko se pravio backup slave čvora, i čvor je nakon kopiranja uspešno startovan i priključen klasteru, njegov backup postaje beskoristan i može se obrisati. To se može učiniti sledećom komandom:

```
rm -rf $NEO4J_HOME/data/databases/graph.db-old
```

2.4. Poređenje HA i Kauzalnih klastera

U nastavku će biti obrađene osnovne razlike između prevaziđenog HA modela i nove verzije, kauzalnih klastera. Najveća razlika ova dva modela je naprednija arhitektura novije verzije, kojoj je dodat još jedan sloj. Ovaj sloj garantuje da neće doći do grananja ili kompromitovanja podataka, što u HA verziji nije bilo moguće. Još jedna bitna funkcionalnost koja je dodata je automatski load balancing aplikacija na klasteru, koji se postiže zahvaljući bolt+routing feature-u. Nova verzija time eliminiše potrebu za dodatnim slojem za load balancing koji je često bio neophodan i naknadno dodavan u HA verziji. Uklanjanje ovog sloja pojednostavljuje arhitekturu sistema i čini je robusnijom. U tabeli 3 je data kompletna komparacija ovih verzija.

HIGH AVAILABILITY	**KAUZALNI KLASTERI**
Paxos protokol	Raft protocol
Minimum jedna instanca + arbitražne instance	Minimum 3 instance
Jedan master čvor i 0 ili više slave čvorova	Nekoliko core servera (3,5,7) + Read replike
Transakcije se usvajaju na masteru i šalju slave čvorovima, bez garancije o uspešnosti transakcije	Transakcija se usvaja tek kada se većina Core servera složi glasanjem
Postoji rizik od grananja	Nema rizika od grananja

Tabela 3 - Razlike izmedju HA i kauzalnih klastera

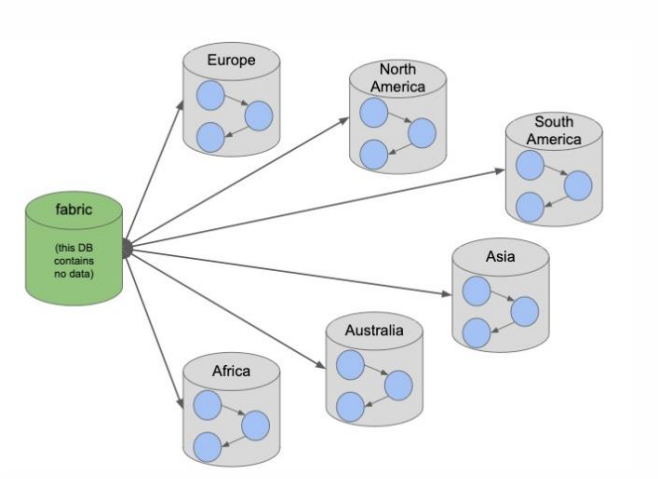
Pojašnjenje pojmova

Paxos, Raft – Protokoli za postizanje konsenzusa u distribuiranim sistemima u kojima može doći do otkaza. Raft protokol se bazira na Paxosu, sa dodatkom da deli problem na manje celine koje se mogu relativno nezavisno rešavati, radi boljeg razumevanja i određivanja prioriteta problema.

Grananje – Za razliku od implementacije kauzalnih klastera, gde se core čvorovi moraju složiti oko transakcije pre nego što se ona distribuira ostalim čvorovima, u HA verziji nema konsenzusa oko transakcije. Master čvor šalje ostalim čvorovima transakciju bez garancije o uspešnosti, a sistem se može podesiti tako da uspešnost transakcije ne zavisi od uspešnosti njenog kopiranja na ostale čvorove. Ovim pristupom može se izazvati nekonzistentnost sistema i korupcija podataka, jer se može desiti da transakcija koja se odigrala na masteru nije uspela da se izvrši ni na jednom drugom čvoru.

3. Dodatak – Fabric

U cilju skalabilnosti i poboljšanja performansi, Neo4j baza podataka se može podeliti na više manjih celina koje su nezavisne jedna od druge. U verziji 4.0, Neo4j je predstavio koncept kojim je postalo moguće obići više različitih baza podataka istovremeno koristeći isti query (slika 4). Ovaj koncept se naziva Fabric i predstavlja logičku reprezentaciju jedinstvene baze, koja je sačinjena od nekoliko različitih distribuiranih baza. Najosnovnija verzija bazira se upravo na ideji da postoji više baza i jedinstvena tačka pristupa, ali su za opus ovog rada daleko interesantnije verzije koje omogućavaju visoku dostupnost sistema.



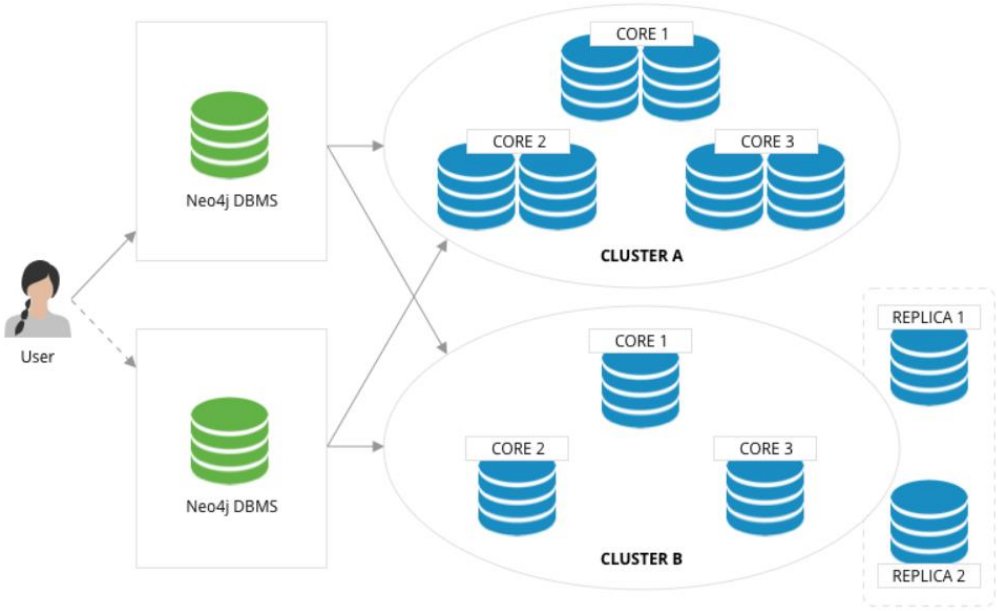
Slika 4 - Ilustracija Fabric koncepta

3.1. Fabric deployment bez jedinstvene tačke otkaza

Slika 5 - Fabric sa jedinstvenom klasterom

Rešenje koje se nameće sa upotrebom Fabric funkcionalnosti je sledeće: Konfigurisati minimalan set čvorova koji će činiti kauzalni klaster, i zatim napraviti dve identične Fabric instance koje će služiti za pristup klasteru. Ovim se postiže to da podaci mogu biti replicirani i grupisani u klastere radi povećanja sigurnosti i dostupnosti, a korisnik ne mora biti svestan postojanja klastera.

3.2. Razvoj Sistema sa više klastera



Slika 6 - Fabric sa više klastera

Ovo rešenje se bazira na prethodnom, uz poboljšanje performansi. Nepovezani grafovi se mogu nalaziti na istom klasteru, kao što je u prethodnom primeru bilo ilustrovano, ili se mogu odvojiti u posebne klastere radi poboljšanja propusnosti. Klasteri mogu imati svoje replike, a svi klasteri su, kao i u prethodnom primeru, povezani na identično konfigurisane Fabric instance, i korisniku se prikazuju kao jedinstvena celina. Ovaj pristup je pogodan za enterprise korisnike kojima su visoke performanse od velikog značaja, kao i za baze podataka koje se particionišu.

3.3. Konfiguracija

Prilikom konfiguracije više instanci Fabric-a, neophodno je navesti adrese routing servera koji se koriste za pristup klasteru. Ovaj parametar je neophodan kako bi ispunio osnovnu funkcionalnost klastera – otpornost na otkaze. U slučaju otkaza jednog Fabric servera, klijent će automatski biti preusmeren na drugi. Sledeći primer (slika 7) se bazira na ideji da u klasteru postoji 3 baze podataka.

```
dbms.mode=SINGLE

fabric.database.name=example
fabric.routing.servers=server1:7687,server2:7687

fabric.graph.0.name=graphA
fabric.graph.0.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.0.database=db1

fabric.graph.1.name=graphB
fabric.graph.1.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.1.database=db2

fabric.graph.2.name=graphC
fabric.graph.2.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.2.database=db3
```

Slika 7 - Fabric konfiguracija

Konfiguracioni parametri sa slike imaju sledeće značenje:

- Routing servers – Fabric server koji se koriste za pristup klasteru
- Uri – Lista adresa Core servera. U slučaju da Fabric server kontaktira core server koji je neaktivan, automatski će se kontaktirati drugi server.

U primeni konfiguracije na prvi primer, postojaće samo prvi graf sa navedenim adresama servera u klasteru. Konfiguracija za drugi primer biće nešto drugačija, jer se sada graf može podeliti na nezavisne klastere i odvojene baze podataka, pa će shodno tome u konfiguraciji postojati 3 različite baze, i uri svake baze će biti drugačiji.

4. Zaključak

Visoka dostupnost je veoma bitan faktor u svakom sistemu koji opslužuje veliki broj zahteva. Svaki sistem ima različitu potrebu za dostupnošću, ali svima je zajedničko da se prestanak rada sistema direktno odražava na profit koji sistem postiže, zbog čega je svima u interesu da sistem bude nedostupan što je kraće moguće na godišnjem nivou.

U ovom radu fokus je bio na povećanju mera dostupnosti u bazi Neo4j. Obrađeni su osnovni koncepti kojima se visoka dostupnost postiže, i napravljena je paralela između zastarelih i prevaziđenih metoda, i unapređenih metoda koje se i danas koriste.

Zaključak koji je studentkinja izvela prilikom istraživanja za potrebe pisanja ovog rada je da je visoka dostupnost u bazi Neo4j veoma dobro razrađena i odlično dokumentovana, zbog čega bi bilo veoma lako primeniti je na realne potrebe.

Literatura

1. Dokumentacija baze Neo4j - Clustering
2. Dokumentacija baze Neo4j - Fabric
3. Monitoring kauzalnog klastera
4. Poredjenje HA i kauzalnog klastera