NoSQL baze podataka

Predavanje 6: Dokument-orijentisae baze podataka, Consistent Hashing, SWIM

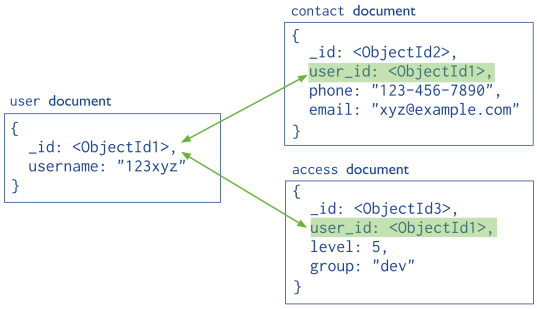
Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Modelovanje

- Najvažnija odluka pri definisaju strukture dokumanta je odrediti na koji će način dokumanti biti povezani jedan s drugim.
- Postoje dva načina povezivanja, to su:
  1. Reference
  2. Ugradjeni (embeded) dokumenti

Reference

- Kod ovog načina povezivanja dokumenata, veze izmedju podataka se ostvaruju uključivanjem referenci iz jednog dokumenta u drugi.
- Ovo je takozvani normalizovani model podataka, što je na neki način slično spoljnom ključu kod relacionih baza podataka
- Vidi se da dokumenti Contact i Access imaju referencu prema User dokumantu.



(https://www.mongodb.com/docs/manual/core/data-model-design/)

- Normalizovani model podataka bi se po pravilu trebao koristiti:
  - Kada bi ugradjivanje dokumanata rezultovalo dupliranjem podataka, a ne bi dalo znatne razlike u performansama kod čitanja podataka.
  - Kada se žele prikazati složenije veze Više-prema Više.
  - Kada se vrši modelovanje velikih hijerarhijskih skupova podataka
- Korišćenjem referenci se postiže veća fleksibilnost nego korišćenjem ugradjenih dokumanata
- ALI aplikacije moraju izvršavati više upita ka bazi da bi došle do svih podataka (nema operacije spoja)!!!

Ugradjeni (embeded) dokumenti

- Ugradjivanjem dokumanata se veze izmedju podataka smeštaju unutar jednog dokumenta.
- To izgleda kao "pod-dokument" unutar dokumanta.
- Takav model podataka se naziva denormalizovani model podataka.
- Vidljivo da su Contact i Access dokumenti ugradjeni u User dokumant, umesto da sadrže referencu prema njemu.

(https://www.mongodb.com/docs/manual/core/data-model-design/)

- Dozvoljava aplikacijama pristup i manipulaciju podacima korišćenjem samo jednog poziva prema bazi podataka.
- Rezultat ovakvog povezivanja je manji broj upita koji se moraju uputiti prema bazi podataka da bi se izvršile odredjene rednje.
- Ugradjeni dokumanti bi se trebali koristiti:
  - Kada postoji veza sadržavanja
  - Kod veza jedan-više (entiteti na strani više se ubacuju unutar entiteta na strani jedan, najčešće u obliku niza).
- Ugradjeni dokumanti imaju prednost kod čitanja podataka iz baze, jer se svi podaci dobijaju preko samo jednog upita prema bazi.

- Takodje, ažuriranje podataka koji su povezani s nekim dokumantom se može obaviti pomoću samo jedne operacije.
- Postoje i loše strane ovakvog pristupa:
  - U situacijama kad dokumenti rastu nakon kreiranja, odnosno podložni su promenama, mogu uticati na performanse kod upisivanja podataka i dovesti do fragmentacije.
  - Izbegavati ga kod velikog ažuriranja dokumenata.

Upis podataka

- Operacije upisa u bazu podataka su atomarne na nivou jednog dokumenta.
- Denormalizovani model podataka omogućava atomarnost operacija upisa u bazu podataka.
- Normalizacija podataka rasporedjivanjem u više kolekcija bi zahtevala višestruke operacije upisa čime bi se izgubila atomarnost.
- Dokument-orijentisane baze NISU realcione baze, stoga ne možemo očekivati iste osobine čak i ako nude mogućnost referenciranja

Pretraživanje baze podataka

- Upitima se specificiraju kriterijumi ili uslovi za identifikaciju dokumanata.
- Upiti mogu sadržati projekciju da bi se definisala polja čiji sadržaj se želi pročitati iz baze podataka.
- Moguće je definisati modifikatore upita za definisanje limita, sortiranja dokumenata ili preskakanja odredjenog broja dokumanata.

Operacije ažuriranja podataka u bazi

- Ažuriranje podataka se odnosi na operacije dodavanja, izmene i brisanja podataka u bazi.
- Operacije ažuriranja menjaju podatke u jednoj kolekciji dokumanata
- Za operacije izmene i brisanja moguće je definisati kriterijume da bi se izvršila selekcija dokumanata koji se modifikuju u bazi ili brišu iz baze podataka.

Agregacija podataka

- Operacije agregiranja podataka obradjuju zapise podataka i vraćaju izračunate rezultate.
- Može se reći da da takve funkcije obradjuju podatke i prikazuju podatke u potrebnom obliku.
- Koriste se za izradu izveštaja i statističkih prikaza podataka.
- Agregirajuće operacije koriste kolekcije i dokumante kao ulaz i izlaz, poput upita.
- Po pravilu postoje tri načina agregiranja podataka:
  - Aggregation pipline
  - MapReduce
  - Agregirajuće funkcije

Prednosti

- Dokumenti su nezavisne jedinice.
- Programska logika jednostavnija za pisanje (JSON).
- Ne koriste šemu baze podataka
  - Jednostavnije se rukuje nestruktuiranim podacima, jer dokument može sadržati bilo koji par ključ-vrednost koji zahteva programska logika.
  - Jednostavna je migracija podataka jer nema potrebe da baza podataka unapred poseduje informacije o šemi baze podataka.

Nedostaci

- Kada ih ne treba koristiti
  - Kada je potrebno izvršavati kompleksne transakcije nad podacima, kada se zahteva atomarnost na nivou više dokumenata
  - Taj deo podržavaju RavenDB i RethinkDB.
  - Kada je potrebno vršiti upite, odnosno pretraživanje baze podataka složenim agregiranjem podataka, a agregaciona struktura varira zbog kontinuirane promene podataka.

Dokument baze podataka

- MongoDB
- CouchDB
- RavenDB
- RethinkDB

Pripadnost grupi

- Kada radimo sa sistemima koji operišu sa više čvorova (u režimu klastera) bitno je da znamo koji čovorovi pripadaj kom klasteru – kojoj grupi
- Ovo je jedna od fundamentalnih stvari koje moramo rešiti
- Bitno je da što je brže moguće detektujemo kada čvor više nije aktivan, i kada se on vratu u operativno stanje
- Ovo nam je bitno zato što prilikom detekcije da čvor nije aktivan moramo premestiti podatke/obradu na drugi čvor da se održi dostupnost i skalabilnost sistema
- Za ove potrebe postoje razni protokoli koji se modeluju na principu širenja virusa kroz popualciju, ili širenja trača kroz popualciju – Gossip style protokol

Osobine

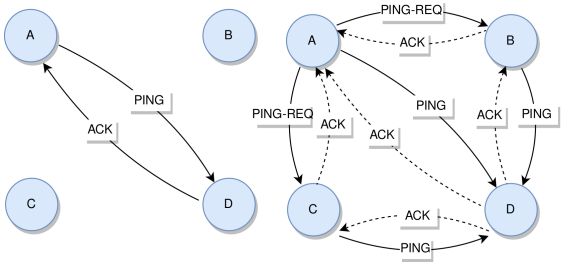
- Prilikom implementacije *Gossip styke* protokola, moramo voditi računa o svojstvima protokola koja će osigurati **efikasnost i skalabilnost**:
  1. **Completeness**, ovo svojstvo mora osigurati da se svaki kvar u sistemu otkrije;
  2. **Accuracy**, u idealnom svetu, ne bi trebalo da bude otkaza pri otkrivanju grešaka, ali u realnim uslovima to se dešava,i potrebno je da smanjimo *false positives* rezultate koliko god možemo;
  3. **Failure detection speed**, sve otkaze treba detektovati što je brže moguće, kako bi se čvor uklonio iz grupe i preraspоredili zadaci sa mrtvog čvora na žive;
  4. **Scale**, sa ovim svojstvom moramo osigurati da mrežno opterećenje koje se generiše treba da bude podjednako rasporedjeno izmedju svih procesa u grupi.
- Dobrodošli u distribuirane sisteme :)
  - *Bad things happen to good nodes, a tale of distributed systems*
- Dobrodošli u CS :)

Heartbeating

- Najjednostavniji način da ovo uradimo jeste da svaki čvor pošalje poruku svim drugim čvoroima – heartbeating
- Proces P_i šalje *heartbeat* svim članovima grupe (multicast)
- Nakon nekog vremena, ako proces P_j ne primi *heartbeat* poruku od procesa P_i, markira ga kao neaktivnog
- Ovu ideju je lako razumeti i primeniti, ali nedostaci su loša skalabilbnost
- Skalabilnost je vrlo problematična, posebno za velike grupe čvorova zato što će uvesti ogroman mrežni saobraćaj
- Takav sistem vremenom postaje neskalabilan, samim tim i neupotrebljiv
- Dakle moramo naći **pametniji** način

Scalable Weakly Consistent Infection-style Process Group Membership – SWIM

- Ovaj protokol funkcioniše na principu direktnog i indirektnog ping-a da bi detektovao koji čvorovi nisu aktivni
- Omogućava:
  - Detekciju otkaza čvorova u sistemu – koristi direktni i indirektni ping
  - Disiminaciju informacija kroz klaster – *piggybacking* na svaki ping (direktan i indirektan) možemo poslati i dodatne podatke ka drugom čvoru
    - Podataka može biti bilo šta



Disiminacija informacija

- Zgodno, zato što ne moramo da koristimo novi zahtev preko mreže samo za npr. replikaciju
- Koristi mehanizam *pretpostavki* – čvor nije odmah proglašen za mrtav, već nakon nekog vremena Δ_t
- Ako čvor ne dobije povratnu poruku nazad od pingovanog čvora, direktno ili indirektno, onda on propagira poruku *pretpostavke* da je čvor nedostupan – nijr živ
- **Čvor koji je označen na ovaj način i dalje nije izbačen iz grupe**
- Ako bilo ko u grupi, uspe da prokomunicira sa markiranim čvorom, čvor se ponovo označava živim
- Ako nakon nekog predefinisnaog vremena Δ_t označeni čvor ne promeni stanje, onda biva eliminisan iz grupe
- U nekom momentu (Eventualna stvar) će svaki čvor dobiti potrebnu informaciju

- U inicijalnoj verziji koristi se UDP protokol za slanje poruka
- Manje opterećujemo mrežu
- Postoje razne dodatne implementacije, neke korite i TCP zbog većih garancija – Lifeguard
- Danas je možda i nejčešće korišćeni protokol za ove potrebe
- Popularan zbog svojih dobrih osobina i jednostavnosti razumevanja i implementacije
- Matematički dokazan i modelovan – ova osobina je izuzetno bitna u distribuirnaim sistemima
- Kada je nešto **formalno** dokazano, imamo stroge garancije da će protokol/sistem raditi
- Optimizacija, ili brzina izvršavanja nisu definisani dokazom!!

Pitanje 1

Da li nam je ovo dovoljno, da li uvidjate potencijalan problem :) ?

Parcijalnost u distribuirnaim sistemima

- Parcijalni otkaz je aspekt distribuiranih sistema; asinhrona priroda procesa i mrežne infrastrukture čini otkrivanje kvarova složenom temom
- Detektori otkaza obično obezbedjuju način za identifikaciju i rukovanje greškama korišćenjem konstantnog vremenskog ograničenja
- Nakon odredjenog praga Δ_t, detektor proglašava čvor van mreže – mrtav
- Ako se setimo *hearthbit* mehanizma i ping-a procesa P_i i P_j, da bi znali kada da proces pogralismo za mrtav treba nam i neko vreme Δ_t
- **ALI**, pošto imamo binarni signal, teško je razlikovati offline proces od sporog
- Drugim rečina kako da ustanovimo Δ_t :)

Pitanje 2

Kako da ustanovimo Δ_t :) ?

The Φ Accrual Failure Detector

- Ideja iza ovog detektora je relativno jednostavna – ne oslanjati se na binarnu vrednost, već procenjivati spram istorijskih vrednosti
- Vrednost Φ se kontinuirano **podešava spram trenutnog stanja mreže**
- Ping signali stižu sa mreže, a svaki interval se čuva u *uzorku* kolekcije koji ima fiksnu veličinu
- Kolekcija uzorka se koristi za procenu distribucije signala ϕ prema formuli:
  - $\phi(t_{now}) \stackrel{\text{def}}{=} -\log_{10}(P_{later}(t_{now} - T_{last}))$
    - T_{last} – prijem poslednjeg ping-a
    - t_{now} – trenutni timestamp
    - P_{later} – verovatnoća da će ping stići u vremenskom intervalu $t_{now} - T_{last}$
- Pošto sve dolazne intervale $t_{now} - T_{last}$ čuvamo u kolekciji, onda se P_{later} izračunava pomoću funkcije kumulativne distribucije

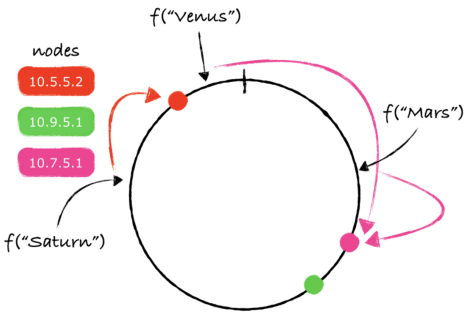
Uvod

- Consistent Hashing je distribuirana šema heširanja koja funkcioniše nezavisno od broja servera ili objekata u distribuiranoj heš tabeli
- Funkcioniše tako što čvorovima dodeljuje poziciju u apstraktnom krugu ili heš prstenu
- Ovo omogućava serverima i objektima da se skaliraju bez uticaja na ceo sistem.
- U distribuiranim sistemima se koristi za rešavanje balansiranja opterećenja
- Dosledan je jer dodavanje ili uklanjanje servera ne izazva ponovno izračunavanje heš tabele

- Koristi se i kao dinamički mehanizam za *sharding* podataka
- Koristi se dosta kada imamo situaciju da se očekuje da čvorovi dolaze i odlaze a mreže
- Koristi se dosta u P2P sistemima, i poboljšava skalabilnost
- Omogućava replikaciju samo dela sadržaja ukoliko čvor padne

Algoritam

- Postavlja prostor čvorova u prsten
- Kretanje prstena u smeru kazaljke na satu odgovara rastućem redosledu lokacija
- Da bi postavili čvorove u prsten, potrebno je da heširamo neko obeležje npr. adresu
- Svaki zahtev može da opsluži serverski čvor koji se prvi pojavi u ovom obilasku u smeru kazaljke na satu
- Zahtv ili ključ heširamo pre poredjenja, moramo ih postaviti u isti prostor



Problem

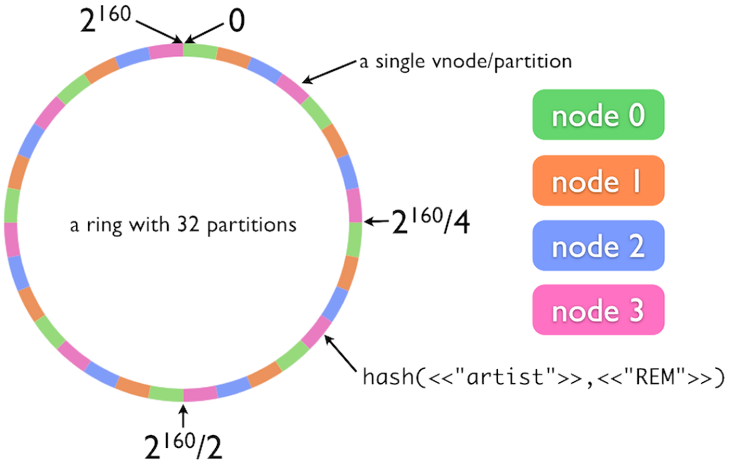
- Više nismo ograničeni brojem čvorova koje možemo imati u sistemu
- Imamo jednostavan mehanizam za pronalazak čvora koji treba da prihvati zahtev ili ključ
- ALI ovo je loš pristup jer dovodi do neujednačene distribucije podataka izmedju čvorova ako se čvorovi postavljaju nasumično
- U slučaju kvarova, ovo zahteva da se svi podaci kojima upravlja taj čvor moraju u potpunosti premestiti na sledeći pojedinačni čvor.

Rešenje

- Srećom rešenje problema je relativno jednostavno
- Sve što treba da uradimo jeste da postignemo bolju raspodelu
- Ovo možemo postići deljenjem heš opsega na manje podopsege
- Ovo ubrzava proces ponovnog balansiranja nakon dodavanja ili uklanjanja čvorova.

Virtuelni čvorovi

- Prethodni problem je rešen upotrebom *virtuelnih čvorova*
- Kada se doda novi čvor, on donosi i nekoliko virtuelnih čvorova da bi održao uravnotežen klaster
- Broj virtuenih čvorova se naziva *težinama* i može biti isti za sve čvorove ili različit u zavisnosti od perfirmansi čvora
- Ova jednostavna tehnika obezbedjuje da su ključevi/zahtevi ravnomerno rasporedjeni medju serverima



(https://docs.riak.com/riak/kv/latest/learn/concepts/clusters/index.html)

Prednost

- Prednost ovoga leži u tome da ako se jedan čvor ukloni iz sistema (i njegovi virtuelni čvorivi), samo taj deo ključeva će biti nasumično raspodeljen po preostalim čovorovima
- Ostali ključevi ostaju nepromenjeni
- Jedan od najčešće korišćenih mehanizaama za balansiranje opterećenja i distribuiranim sitemima
- Dosta se koristi u velikim sistemima za skladištenje podataka

Dodatni materijali

- Making Sense of NoSQL A guide for managers and the rest of us
- Database Internals
- NoSQL Distilled A Brief Guide to the Emerging World of Polyglot Persistence
- Seven Databases in Seven Weeks
- SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol
- The Φ Accrual Failure Detector

Pitanja

Pitanja :) ?