NoSQL baze podataka

Predavanje 8: Kolonski orijentisane baze podataka, Protokoli verifikacije i pronalaska podataka

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Merkle stablo - uvod

- Merkle stablo ili **hash stablo** je stablo u kome je svaki čvor označen **hash** vrednošću
- Ova stabla omogućavaju efikasnu i bezbednu verifikaciju sadržaja velikih struktura podataka
- Merkle stablo je razvijeno 1979 godine, a 2002 je patent istekao (blago nama :))
- Ovo stablo je generalno **binarno** po prirodi
- Za razliku od uobičajenog postupka kod većine stabla, ovo stablo se formira od lista (dna) ka korenu (vrhu)

- Formalno gledamo, Merkle stabla uzimaju skup podataka $(x_1, \ldots, x_n)$ na ulaz
- Povratnu vrednost je **Merkel root hash** $h = \mathrm{MHT}(x_1, \ldots, x_n)$
- MHT **collision-resistant hash funkcija**
- *Hash* funkcija je **collision-resistant hash funkcija** ako je teško pronaći dva ulaza koja *hash*-iraju isti izlaz
- Formalno, za ulaze a i b, $a \neq b$ ali $H(a) = H(b)$

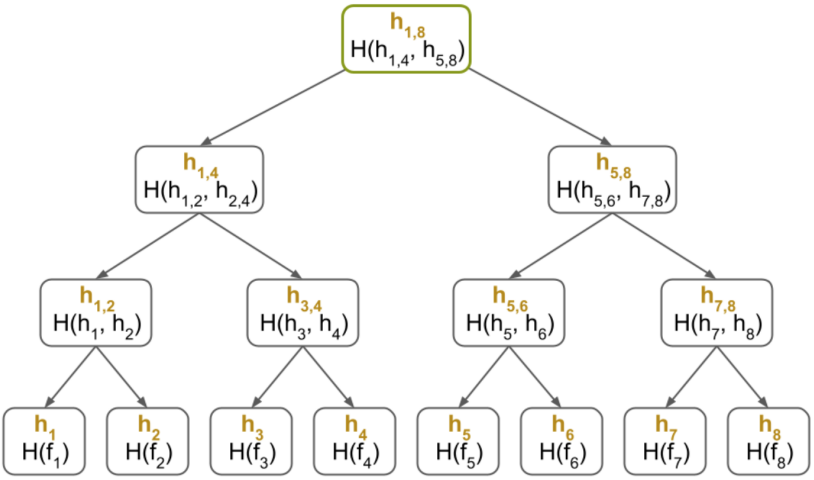
Merkle stablo - formiranje

- Algoritam za formiranje Merkle stabla je relativno jednostavan
- Merkle stablo ima **bottom-top** pristup, zbog svoje specifičnosti
- Formiranje stabla počinje od dna tj. konkretizovanih podataka — **data block**
- Polako idemo do vrha, gradeći **Merkle root** element
- Prvi element koji gradimo je **list**
- Svaki podatak propustimo kroz *hash* funkciju, i tako formiramo prvi nivo — **list**

- Svaki **list** propustimo kroz *hash* funkciju, a svaka **dva susedna** elementa grade naredni nivo propuštajući njihove zajedničke hash vrednosti kroz hash funkciju
- Pošto radimo sa binarnim stablima, ako na nekom nivou nemamo odgovarajući čvor, možemo da dodamo *empty* elment da bi algoritam mogao da se nastavi
- Kada propustimo poslednja dva čvora kroz hash funkciju dobijamo **Merkle root** element
- Time se algoritam za formiranje završava i formirali smo Merkle stablo

Merkle stablo – formiranje, primer

- Pretpostavimo da imamo 8 blokova podataka (fajlova) $f = (f_1, ..., f_8)$
- Svaki podataka f_i propustimo kroz hash funckiju H i dobijamo njegov *hash*
- Dobijamo hash vrednost za prvi nivo $h_i = H(fi), h_i = (h_1, ..., h_8)$
- H reprezentuje **collision-resistant hash** funkciju



(Decentralized Thoughts, Merkle trees)

Merkle stablo - napomena

- Ono što smo deobili na kraju $h_{1,8}$ je **Merkle root hash**
- Obratiti pažnju da svaki čvor u stablu čuva **hash** vrednost
- Listovi čuvaju hash vrednost (blokova) podataka $h_i = (h_1, ..., h_8)$
- Čvorovi koji nisu listovi, i nisu **Merkle root hash**, čuvaju *hash* vrednost svoje dece — **internal node**

- Ako nam na nekom nivou fali par za neki element, prosto dodamo **prazan hash** da bi formirali par
- Može se lako generalizovati i izražunati Merkle stablo za bilo koji broj n podataka
- Formalno zapisano, prethodni primer se može zapisati kao $h_{1,8} = \mathrm{MHT}(f_1, ..., f_8)$
- Merkel stabla se formiraju rekurzivno, od dna ka vrhu
- Ovaj proces može biti procesno zahtevan!
- To nikada nemojte izgubiti iz vida

Merkle stabla - upotreba

- Merkle stabla se dosta koriste kod validacije podataka na različitim mestima
- Šta je potrebno da se sinhrnozuje, Merkle stablo može da nam kaže bez konkretnih podataka
- Danas se dosta koristi, i uglavnom se vezuje za *Blokchain* tehnologije
- **Merkle stabla** $\neq$ **Blokchain**
- Merkel stabo nije jedino upotrebljeno za Blokchain
- Merkel stablo nije smišljeno za *Blokchain*
- Ali jeste osnova *Blokchain* tehnologije
- Pogledajte koliko Blochain troši struje, pa će vam biti jasna konstatacija — *Ovaj proces može biti procesno zahtevan!*

Pitanje 1

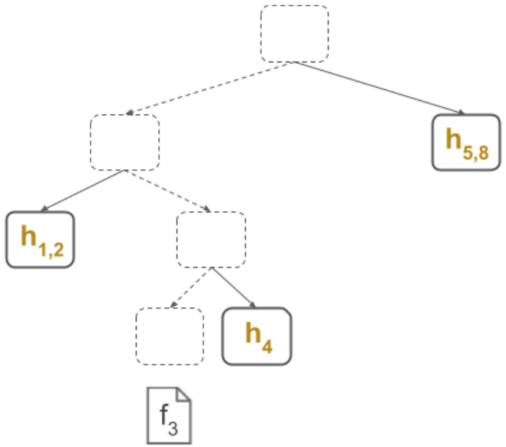
Zašto formiramo stablo, zar ne bi bilo jednostavnije da formiramo lanac, možda možemo izbeći rekurzije...

Ideje :)?

Merkel dokaz

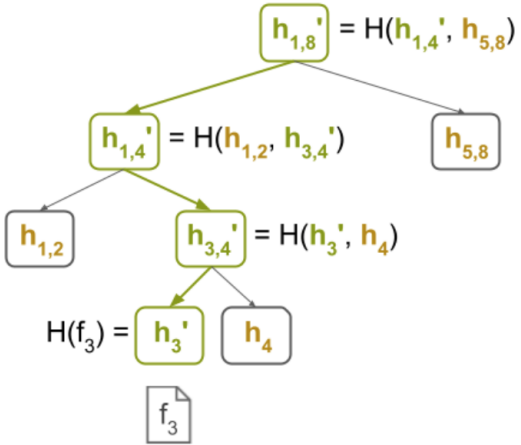
- Ako bi formirali *hash-eve* kao lanac, vrlo je moguće da bi utrošili manje resursa
- Proces bi verovatno bio znatno jednostavniji, rekurzije mogu biti nezgodne
- **ALI** taj proces ima jednu nezgodnu osobinu — manu
- Ako bi trebali da proverimo integritet podataka, ili da li podataka pripada nekom skupu morali bi da sačuvamo/poredimo **kompletan** skup
- Ako su nam podaci na više čvorova, to znači da bi morali da prebacujemo ceo skup podataka kroz mrežu
- U najmanju ruku, to nije lepo i nije kulturno :)
- Zato se (binarno) stablo pokazalo boljim izborom

- Ako se vratimo na problem integriteta podataka koji su prebav ceni negde na skladištenje (npr. Dropbox)
- Ključna ideja je da, nakon što preuzmemo podatak f_i, tražimo mali deo Merkle stabla — **Merkle dokaz**
- **Merkle dokaz** nam omogućava da proverimo da li preuzeti podatak f_i nije slučajno ili zlonamerno izmenjen
- Ako imamo podatak f_3, zanima nas da li je on deo većeg skupa podataka



(Decentralized Thoughts, Merkle trees)

- Da bi to odredili, treba nam samo mali deo Merkle stabla, da bi formirali Merkle root
- h_3 možemo izračunati propuštajući podatak kroz *hash* funkciju
- Delovi koji nam trebaju su $h4, h_{1,2}, h_{5,8}$
- Sa ovim delovima možemo stići do Merkle root-a
- Ova ideja se dosta koristi kod Torrent sistema



(Decentralized Thoughts, Merkle trees)

- Merkle dokaz pokšava da nam kaže da li podatak pripada Merkle stablu ili ne
- Da se nedvosmisleno dokaže valjanost podataka koji su deo skupa podataka, bez skladištenja celog skupa podataka i dobacivanja kroz mrežu
- Da bi se osiguralo da je skup podataka deo vecćeg skup podataka bez otkrivanja kompletnog skupa podataka ili njegovog podskupa
- S obzirom na to da su jednosmerne *hash* funkcije namenjene da budu algoritmi bez kolizija, dva *hash-a* ne mogu da budu ista
- Ove osobine mogu biti primamljive za razne tipove aplikacija (ne samo *Blokchain*)

Problem 2

Zaposlili ste se u Amazonu (lepo), razvijate nov sistem za skladištenje podataka i od vas se očekuje da razvijete sistem za efikasnu verifikaciju sadržaja velkih skupova podataka. Pred vama su sledeća ograničenja:

- Ne potrošimo previše resursa ako je moguće
- Kroz mrežu ne saljemo same podatke — nikako nije isplativo

Ideje :)?

Anty-entropy - ideja

- Merkle stabla se mogu koristiti za sinhronizaciju podataka i proveru ispravnosti kopija u sistemima sa više čvorova (peers) u distribuiranom sistemu
- Ne moramo da poredimo čitave podatke da bismo shvatili šta se promenilo
- Možemo samo da uporediti *hash* stabala
- Kada shvatimo koji listovi su promenjeni, odgovarajući komad podataka može da se pošalje preko mreže i sinhronizuje na svim čvorovima — znatno jednostavnije
- Ova ideja se dosta koristi kod velikih sistema za skladištenje podataka (Amazon DynamoDB, Cassandra, ScylaDB, ...)

Anty-entropy - algoritam

- Algritam je relativno jednostavan i odvija se u tri koraka
  1. Napraviti Merkle stablo za svaku repliku (čvor) koja čuva kopiju podataka
  2. Uporedite Merkle stablo da bi otkrili razlike
  3. Razmenite šta je potrebno od selova podataka da svi imaju isti skup kopija
- Izgradnja Merkle stabla je resursno **intenzivna operacija**, opterećuje disk I/O i koristi dosta memorije :(
- Time plaćamo manje slanje podataka krož mrežu — **Nema besplatnog ručka**
- Ovaj proces se često odbija u pozadini, da ne bi blokirali ostatak sistema

- Provera kreće od vrha stabla, ako je *root hash* identičan, nema potrebe za popravkama
- Ako to nije slučaj, prelazimo na **levo dete**, zatim na **desno dete** — obilazimo stablo
- Postupak se nastavlja dok ne stignemo do bilo kakve razlike u podacima
- Kada ustanovimo šta je različito, samo taj deo skupa podataka treba da se popravi — pošalje kroz mrežu
- Kreće proces razmene podataka

Prošli put

- Analizirali smo baze podataka tipa Ključ-Vrednost koje su jednostavne **hash tabele**, gde se:
  - Svi pristupi podacima vrše preko primarnog ključa i objekat se vraća kao rezultat
- Kolonski-Orijentisanie baza podataka su motivisane potrebom za modelom koji je nešto više od objekat-vrednost.
- Kolonski orijantisane baze podataka su zapravo podskup ključ-vrednost skladišta podataka
- Za operacije pronalaženja podataka koriste skup ključeva.

Nastavak

- Popšto podaci više nisu na jednom mestu, postavlja se pitanje kako podatke pronaći?
- Sa druge strane, pogotovo je specifično za baze koje nemaju *master* čvor, postavlja se pitanje kako će *koordinator* pronaći ključ koji tražimo?
- Podaci se nalaze negde u klasteru, a za odredjivanje na koji čvor da ih smestimo koristmo *Consistent hashing*
- Sistemi za skladištenje podataka liče na *hash* tabele

Distribuirna hash tabela

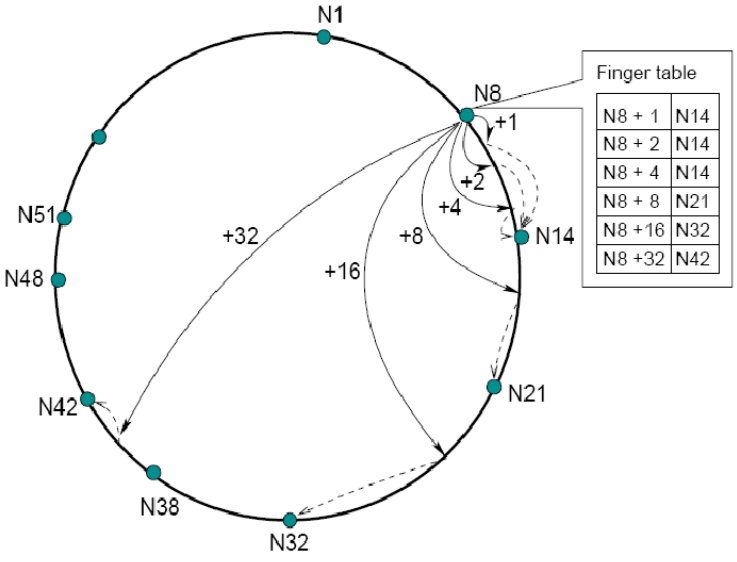
- Distribuirna hash tabela (DHT) imaju nekolko osobina:
  - Autonomija i decentralizacija: čvorovi zajedno čine sistem bez ikakve centralne koordinacije.
  - Tolerancija grešaka: sistem treba da bude pouzdan (u nekom smislu) čak i sa čvorovima koji se neprekidno pridružuju, napuštaju i otkazuju.
  - Skalabilnost: sistem treba da funkcioniše efikasno čak i sa hiljadama ili milionima čvorova.
- Ako pogledamo recimo sisteme kao što je Apache Cassandra ili Amazon Dynamo, oni se jako oslanjaju na ove osobine

Chord protokol

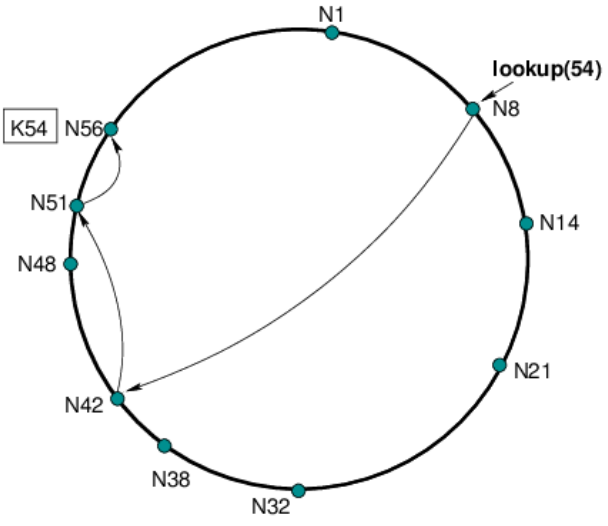
- Jedan od protokola za rutiranje zahteva u peer-to-peer sistemima
- Oslanja se na osnovnu ideju podele prostora u prsten koju donosi *Consistent hashing*
- Omogućava efikasnu pretragu u prstenu složenosti $\mathrm{O}(\log n)$
- Sam po sebi ne skladišti podatke, ali onogućava da se zahtev izrutira do čvora koji čiva podatke
- Danas se dosta koristi (ili varijacija) u raznim peer-to-peer sistemima

Pretraga

- Osnovni pristup je prosledjivanje upita nasledniku (successor) čvora, ako ne može da pronadje ključ lokalno.
- Ovo će dovesti do $O(N)$ vremena upita gde je N broj mašina u prstenu.
- Da bi izbegao linearnu pretragu, Chord primenjuje brži metod pretraživanja zahtevajucći da svaki čvor zadrži **tabelu pokazivača (finger table)** koja sadrži do m unosa – m je broj bitova u *hash* ključu
- I_{ti} zapis u tabeli čvora n sadrži identitet prvog čvora s koji sledi n za najmanje 2^{i-1} na krugu identifikatora.
- Stoga $s = \text{successor}(n + 2^{i-1} \mod 2^m)$.



(Grid and Peer-to-Peer Resource Discovery Systems)



(Grid and Peer-to-Peer Resource Discovery Systems)

Dolazak novog čvora u sistem

- Prilkom dodavanja novog čvora u sistem prvo moramo da odredimo gde čvor leži na prstenu
- Kada se doda novi čvor u sistem, potrebno je da ispoštujemo par koraka:
  - Inicijalizujemo čvor n.
  - Obavestimo druge čvorove da ažuriraju svoje prethodnike i tabele pokazivača.
  - Novi čvor preuzima svoje odgovorne ključeve od svog naslednika.

Stabilizacija

- Da bi se osigurala ispravna pretraživanja, svi pokazivači naslednika moraju biti ažurirani
- Stoga, protokol stabilizacije radi periodično u pozadini, i ažurira tabele pokazivača naslednika
  - čvor n pita svog naslednika za svog prethodnika p i odlučuje da li bi p trebalo da bude n-ov naslednik (ovo je slučaj ako se p nedavno pridružio sistemu).
  - Obaveštava n-ovog naslednika o njegovom postojanju, tako da može da promeni svog prethodnika u n
  - Ažuriramo tabele pokazivača
  - Periodično proverava da li je prethodnik živ

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
- Merkle tree original paper
- Merkle tree easier paper
- Amazon Dynamo db paper Chord protocol

Pitanja

Pitanja :) ?