

Napredni algoritmi i strukture podataka

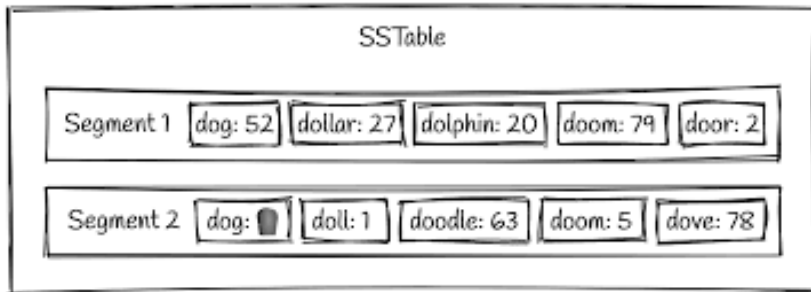
Put čiranja podataka (Read path), Keširanje sadržaja, LRU



Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

SSTable — Data

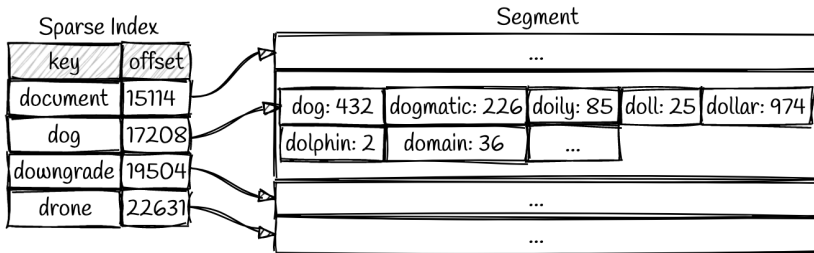
- ▶ SSTable je niz parova **ključ:vrednost** koji su sortirani i zapisani na disk
- ▶ SSTable je nepromenljiva struktura (log based struktura)
- ▶ SSTable nastaje nakon što se Memtable popuni i zapiše na disk



(LSM — Write Heavy Databases)

SSTable — Index

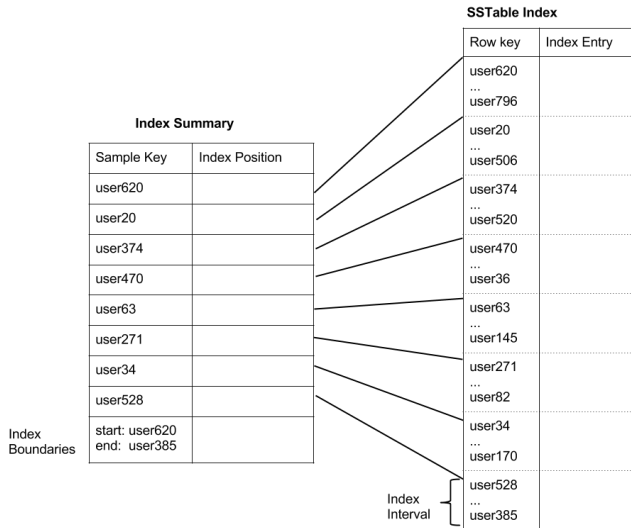
- ▶ Ideja je da što pre stignemo do sadržaja
- ▶ Struktura je relativno jednostavna:
 1. **ključ** koji se nalazi u fajlu na disku (SStable)
 2. **offset** u fajlu, tj. pozicija u fajlu na disku
- ▶ Index bi bilo lepo sortirati (ne budite lenji, učinite sebi uslugu :))



(LSM — Write Heavy Databases)

SSTable — Summary

- ▶ Neki sistemi za skladištenje podataka (npr. Cassandra) koriste dodatan element za ubrzanje čitanja — **Summary**
- ▶ Struktura je relativno jednostavna:
 1. **granice** povezanog index fajla — prvi i zadnji **ključ**
 2. **offset** ključa u index fajlu — poziciju u index fajlu



(Distributed Datastore, Summary)

SSTable — Bloom Filter

- ▶ Bloom Filter-u trebamo da kažemo koliko elemenata se očekuje, ali to nije sada problem
- ▶ Ovaj podataka nam je unapred poznat, pošto tačno znamo koliko elemenata čuva Memtable
- ▶ SSTable isto zna tačno zna koliko će elemenata biti sačuvano
- ▶ Samim tim i za Bloom Filter imamo tu informaciju unapred poznatu!
- ▶ Stoga, sve elemente možemo formirati ispravno

SSTable — Metadata

- ▶ Dodatno formiramo *Metadata* fajl — Merkle stablo *Data* segmenta
- ▶ Nakon što je stablo formirano, stablo zapišemo u *Metadata* fajl
- ▶ Ovde je kraj pravljenja jenog SSTable-a
- ▶ Formiranje SSTable se obično odvija u pozadini, bez narušavanja rada sistema
- ▶ **Za vaš projekat ovo nije potrebno, može sve da se dešva i sinhrono**

SSTable — Struktura

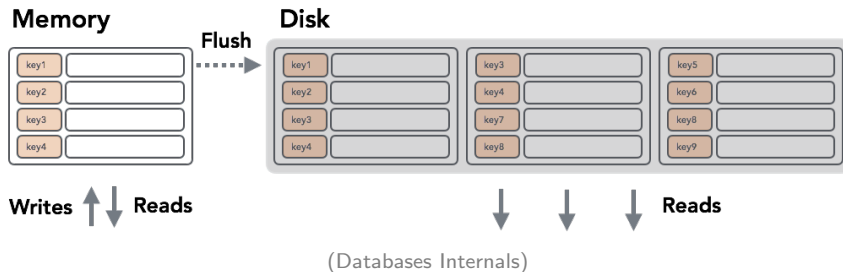
- ▶ *Data* je SSTable
- ▶ *Index* je povezani Index
- ▶ *Summary* je povezani Summary Index
- ▶ *TOC* čuva spisak formiranih elemenata za jedan SSTable
- ▶ *Filter* je Bloom filter za ključeve u *Data* delu
- ▶ *Metadat* Merkle stablo *Data dela* — dodatno smo formirali

usertable-data-ic-1-CompressionInfo.db
usertable-data-ic-1-Data.db
usertable-data-ic-1-Filter.db
usertable-data-ic-1-Index.db
usertable-data-ic-1-Statistics.db
usertable-data-ic-1-Summary.db
usertable-data-ic-1-TOC.txt

(Distributed Datastore, SSTable format)

Put čitanja podataka — Read Path

Write path smo videli na nekom od prethodnih predavanja, Read path ukratko prošlo predavanje, ali jako prosto



Pitanje 1

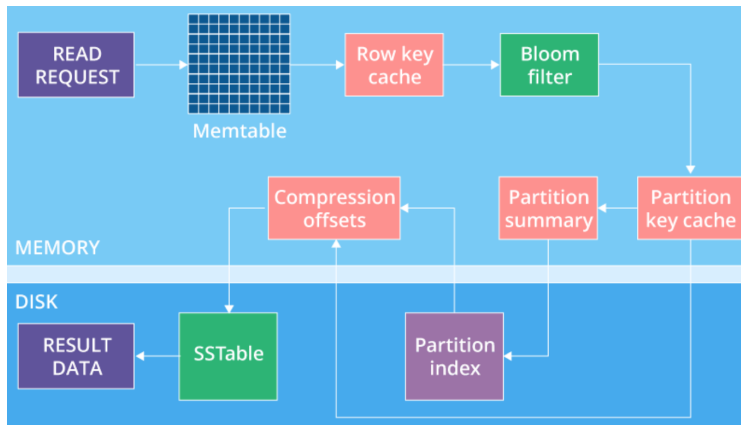
Ako pogledamo prethodnu sliku, vidimo da se svo čitanje odvija sa diska i iz SSTable-a. Zašto smo onda formirali onoliko elemenata prethodno predavanja (**HINT:** Nije da vas mučim :))...

ideje :)?

- ▶ Svi prethodno formirani elementi su nam potrebni, da bi što pre stigli do podatka koji tražimo, **AKO** je on tu
- ▶ Toliki broj fajlova nam treba zbog ovog **AKO**
- ▶ Pravimo kompenzaciju za nesigurnost Bloom Filter-a
- ▶ Što je pre moguće da dobijemo informaciju nazada
- ▶ Ili bar da nam mašinerija kaže: **ključ koji tražis nije sigurno tu**

- ▶ Svi formirani elementi čine put čitanja podataka – **Read Path**
- ▶ I svi oni moraju da se povežu u sinhronu celinu, **AKO** želimo da dobijemo informaciju u razumenom vremenu
- ▶ Pa hajde da vidimo kako to rade neki poznati sistemi —velikani (npr. Apache Cassandra)
- ▶ Videćemo kako taj sistem pravi koreografiju elemenata i šta nam još fali
- ▶ A možda i nešto možemo da izbacimo :)
- ▶ Pa da probamo da napravimo naš **Read Path**

Apache Cassandra — Read Path



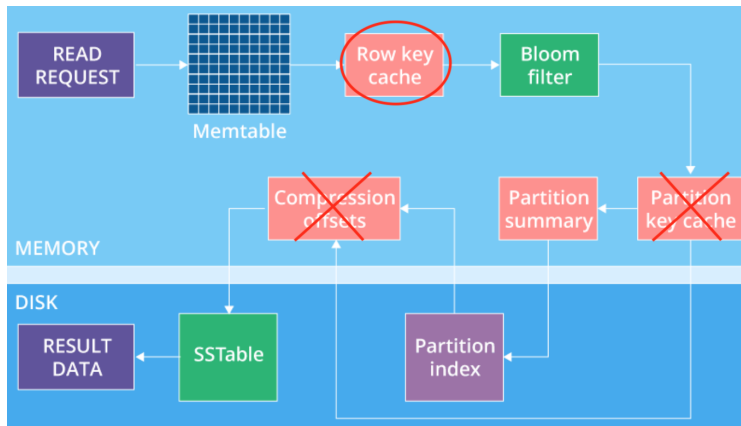
(Cassandra Performance: The Most Comprehensive Overview You'll Ever See)

Razlike

- ▶ Ako pogledamo prethodni dijagram
- ▶ Možda vam deluje komplikovano :), ali nije
- ▶ Deo oko kompresije smo rekli prošli put da ćemo ignorisati
- ▶ On služi da zauzmemo još manje resursa na disku
- ▶ Razlika je što ovde imamo dva ključa :O — Cassandra ima drugačiji model
- ▶ Naš model ima samo jedan :'(— mi koristimo opšti model podataka

- ▶ Pošto naš model koristi jedan ključ, drugi ćemo prosto ignorisati :D
- ▶ Ostaje nam samo jedan — *Row key*
- ▶ **ALI**, na dijagramu postoje elementi koji se zovu **cache**
- ▶ I još se nalazi se pre Bloom filter-a
- ▶ Pa ovo je vrlo zanimljivo....
- ▶ Znači da možda može dodatno da ubrza posao :D

Cache element



Pitanje 2

Slušali ste arhitekturu računara, šta je tamo radila **keš memorija**

ideje :)?

Keširanje podataka

- ▶ Keširanje podataka je proces koji skladišti više kopija podataka ili datoteka na privremenoj lokaciji za skladištenje — **cache**
- ▶ Cache čuva podatke za softverske aplikacije, servere i web pretraživače, ali i za sistemski software
- ▶ Ovaj proces osigurava da korisnici ne moraju da preuzimaju informacije svaki put kada pristupe web lokaciji ili aplikaciji
- ▶ Čitava ideja je nastala oko ideje: kako bi ubrzali učitavanje sajta, ili što pre pristupili podacima

- ▶ Čitanje podataka sa diska je (relativno) sporo, pogotov kada imate dosta podataka
- ▶ Razmišljajte na nivou Facebook, Instagram, Amazon, ...
- ▶ Sa druge strane disk je relativno jeftin, i ima ga prilično dosta
- ▶ Čitanje podataka iz memorije je (relativno) brzo
- ▶ Ali opet, memorije nemamo toliko puno — ma šta neko rekao
- ▶ U memoriji su obično najsvežiji podaci

- ▶ Cena nije ista kao cena diska, a ni kapacitet
- ▶ Plus ako se sistem restartuje nema više podataka
- ▶ U memoriji čuvati samo podake kojima se (relativno) skoro pristupalo
- ▶ Ne čavi sve podatke!
- ▶ Ovo će omogu'čiti brži pristup podacima
- ▶ Ali moramo naći način da čuvamo najnovije informacije

- ▶ Postoje razni načini i strategije kako čuvati podatke u cache-u
- ▶ Možemo da rotiramo sadržaj, ako je struktura fiksne dužine
- ▶ Imamo nekakav fiksni skup, koliko elemenata može da uvek bude u cache-u
- ▶ Noviji elementi izbacuju one starije
- ▶ Možemo definisati život podataka u cache-u (time to live — TTL)
- ▶ ...

Pitanje 3

Sada znamo šta je *Row cache*, ali kako ćemo čvati podatke u njemu...Koliko dugo, kada ih izbacivati...

ideje :)?

- ▶ Tačan odgovor ne postoji, ili bar meni nije poznat :)
- ▶ Jedna strategija se ne može primeniti na sve slučajeve
- ▶ Zavisi od aplikacije
- ▶ Zavisi od slučaja korišćenja
- ▶ Zavisi od resursa
- ▶ ...

Cache — Zahtevi

- ▶ Cache sistemi obično imaju zahteve koje treba da isputene, i neki od njih su:
 - ▶ Fiksna veličina: Cache memorija treba da ima neke granice, tj. da ograniči upotrebu memorije
 - ▶ Brz pristup: Operacija dodavanja elementa u cache i traženja treba da bude brza, poželjno $\mathcal{O}(1)$ vreme
 - ▶ Zamena unosa: U slučaju da je dostignuto ograničenje memorije, cache treba da ima efikasan algoritam za izbacivanje unosa kada je memorija puna
- ▶ Dodavanje ograničenja je obično eksternizovano u nekakvu konfiguraciju — konfiguracioni fajl, ili neki drugi oblik

Pitanje 4

Kako da ispunimo ove zahteve...struktura, algoritam,...

ideje :)?

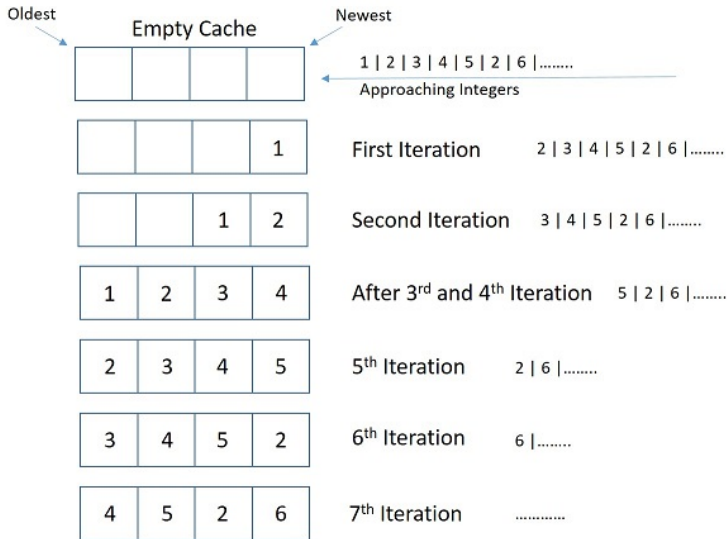
LRU

- ▶ Najmanje nedavno korišćen (LRU) organizuje elemente po redosledu korišćenja
- ▶ Omogućavajući da brzo identifikujemo koja stavka nije korišćena najduže vreme
- ▶ Zamislite stalak za odeću, gde je odeća uvek okačena na jednoj strani
- ▶ Da bi pronašli predmet koji smo najmanje koristili, pogledate predmet na drugom kraju staka
- ▶ Ukratko, to je politika izbacivanja iz keša
- ▶ Navodimo kada se naš keš popuni i doda se novi element, uklanjamo najmanje nedavno korišćenu stavku iz keša

Pitanje 5

A šta ako uzmemo neki element iz LRU...šta radimo sa pozicijama...

ideje :)?



(Opendenus, Implement Least Recently Used (LRU) Cache)

► Prednosti:

- Super brzi pristupi: LRU keš čuva stavke po redosledu od nedavno korišćenih do najmanje korišćenih — oba mogu pristupiti u $\mathcal{O}(1)$ vremenu
- Super brza ažuriranja: Svaki put kada se pristupi stavci, ažuriranje keša traje $\mathcal{O}(1)$ vremena

► Mane:

- LRU keš koji čuva n stavki zahteva spregnutu listu dužine n , i hash mapu koja sadrži n stavki — to je $\mathcal{O}(n)$ prostor, ali to su i dalje dve strukture podataka (za razliku od jedne)

Pitanje 6

Sada imamo cache, super, ali šta sa njim...šta on da čuva...

ideje :)?

Read Path — keširanje

- ▶ Cache je poslednji element koji nam fali da kompletiramo read path
- ▶ Cache čuva ključeve i vrednosti kako im je korisnik **zadnje** pristupao
- ▶ Cache se nalazi u memoriji u potpunosti
- ▶ **AKO** je podatak u cache-u, vratite ga korisniku — jako brzo :D
- ▶ **AKO** podatak nije u cache-u, onda moramo da vidimo da li je prisutan uopšte — taj put traje :(
- ▶ **ALI**, zato imamo sve one silne strukture da nam pomognu i ubrzaju koliko jeto moguće

- ▶ To znači da moramo učitati **Bloom filter** sa diska i videti da li je ključ **možda** tu
- ▶ Ako nije, odmah javimo korisniku, da ključ nije prisutan — podatak nije sačuvan
- ▶ Ako je **možda** tu, učitati **summry** i videti da li je ključ u tom opsegu
- ▶ Ako nije, odmah javimo korisniku, da ključ nije prisutan — podatak nije sačuvan
- ▶ Ako **jeste**, pronaći ga u **index** strukturi, i uzeto **offset**
- ▶ Kada imamo **offset**, možemo da se pozicioniramo na **Data** deo i da pročitamo podatak i vratimo korisniku

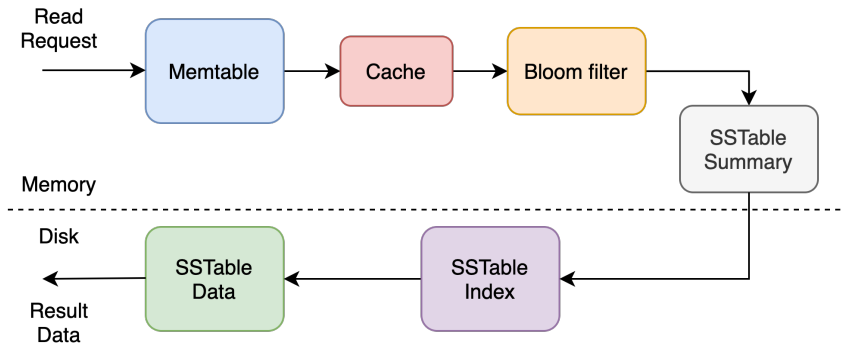
Pitanje 7

Ok, ali kako nam tu cache pomaže...

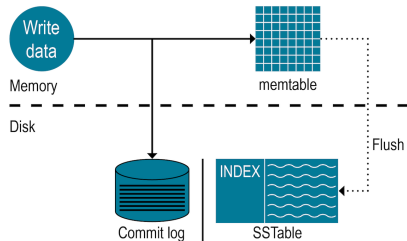
ideje :)?

- ▶ Podatke u cache-u moramo da ažuriramo svaki put kada se korisnički podatak locira!
- ▶ Kada dobiemo podataka, pre nego što ga vratimo korisniku prvo ga zapišemo u cache
- ▶ Ova prosta strategija nam omogućava da kod sledeće pretrage **MOŽDA** ne moramo da idemo po disku
- ▶ Promašaj keša je ishod u kojem sistem ili aplikacija postavlja zahtev za preuzimanje podataka iz keš memorije, ali ti specifični podaci trenutno nisu u keš memoriji
- ▶ Ovo se nekada beleži zarad nekakvih analitika i promene politike šta i kada keširati
- ▶ Kada se uputi zahtev za brisanje nekog podatka, **AKO** je on u cache-u, možemo ga obrisati

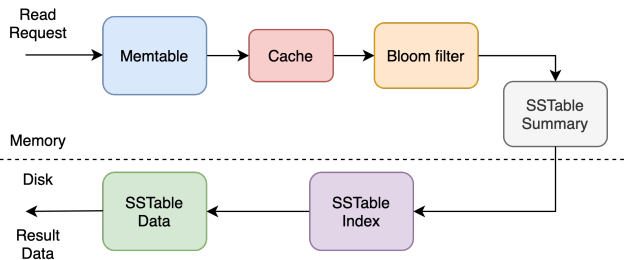
Read path



Read/Write path



(Distributed Datastore, SSTable format)



Dodatni materijali

- ▶ A Low Overhead High Performance Buffer Management Replacement Algorithm
- ▶ Policy with Applications to Video Streaming
- ▶ <https://hazelcast.com/glossary/cache-miss/>
- ▶ Data Caching in Cassandra

Pitanja

Pitanja :) ?