

Napredni algoritmi i strukture podataka

SSTable, Index, Summary, Struktura, Formiranje



Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

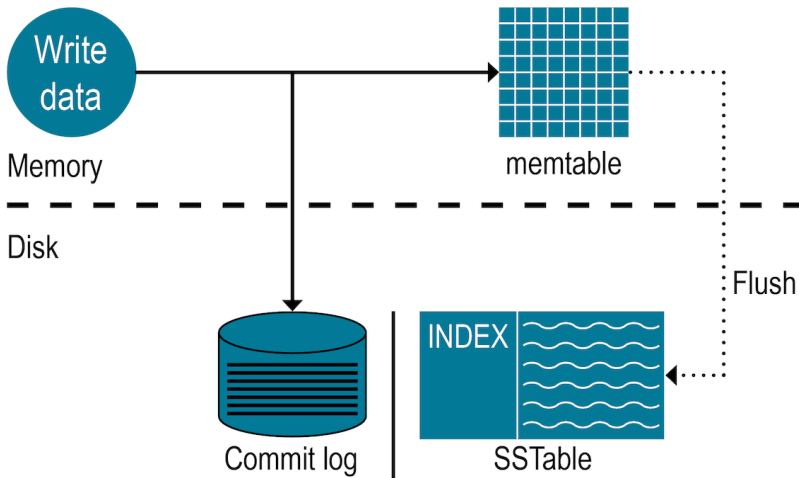
Memtable - rekapitulacija

- ▶ Memtable ideja je relativno jednostavna — zapisati podatke u memoriju i čitati podatke iz memorije
- ▶ Memorija je brza, memorija je super, memorija je kul
- ▶ **ALI** nemamo beskonačno memorije (recite to matematičarima :))
- ▶ **AKO** se podaci nalaze u memoriji, sve operacije su relativno brže nego da su podaci **striktno** na disku
- ▶ **ALI** memorija nije sigurna :/

- ▶ Restart sistema i naših podataka više nema (objasnite to korisnicima:))
- ▶ Iz tog razloga nam treba snažna garancija trajnosti podataka
- ▶ Memtable komunicira sa WAL-om, koji nam daje ove garancije
- ▶ Ovaj princip se pokazuje jako korisno kod **write-heavy** problema
- ▶ Kada se Memtable struktura popuni, ona se perzistira na disk (Flush) i pravi se **SSTable** koja je neprimenljiva
- ▶ Veličinu Memtable-a, možemo podešavati

Prošli put jako malo o SSTable

1. SSTable se sastoji od nekoliko elemenata
2. Sve ove elemente je potrebno formirati kada se formira SSTable
3. Uopšteno gledano, SSTable ima **index** deo – lakše pozicioniranje na potrebne delove **data** segmenta, koji čini drugi deo
4. Korišćenje **Memtable**, **WAL** i **SStable** je poznato kao **write-path**



(Cassandra write path)

SStable - ideja

- ▶ Ideja iza tabele sortiranih stringova (**SSTable**) je relativno jednostavna
- ▶ To je niz parova **ključ-vrednost** koji su sortirani, i zapisani na disk
- ▶ **SStable** je nepromenljiva struktura (log based struktura) — nema brisanja ni izmene sadržaja
- ▶ Memtable zapisati na disk, sa relativno sličnom strukturom (ako ne i istom)

- ▶ Zavisi od toga šta čuvate, i kakav vam je **model podataka**
- ▶ Zavisi od upotrebe!
- ▶ Pod nizom sortiranih stringova ne misli se doslovno na stringove, već na **niz bajtova**
- ▶ Ključ će biti **string** to svkako, ali vrednost može biti bilo šta
- ▶ Zbog toga je najjednostavnije da se čuva niz bajtova

SSTable — Tombstone

- ▶ Kada se obriše podatak, on nije momentalno uklonjen sa diska
- ▶ Sistem zapisuje specijalan podatak *Tombstone* da je neki ključ obrisani — markira ga za brisanje
- ▶ Brisanje elemenata je zapravo **nov zapis** u SSTable — SSTable je nepromenljiva struktura
- ▶ Fizičko brisanje sa diska se odvija kada se desi specifičan proces — kompakcije

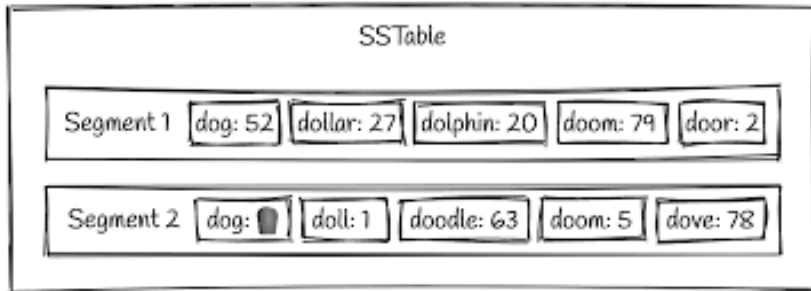
Pitanje 1

Kakvu strukturu da koristimo, ideje :) ?

SStable — struktura

- ▶ Izabraćemo opšti oblik SSTable-a
- ▶ Niz parova **ključ-vrednost**, gde su obe vrednosti zapravo niz bajtova
- ▶ Pre zapisa Memtable-a, sve vrednosti se **sortiraju** — ovo nam trebati kasnije!!
- ▶ Ovim smo dobili **data** segment
- ▶ Svi podaci se nalaze tu

Opšta struktura SSTable



(LSM — Write Heavy Databases)

- ▶ Prethodna slika ne kazuje baš puno :(
- ▶ Vidimo markiranje ključeve za brisanje — Tombstone
- ▶ Vidimo nekakve segmente i nekakve parove **ključ-vrednost**, ali šta sa njima...
- ▶ Ali bar vidimo neku opštu ideju
- ▶ Hajde da vidimo, kako to rešavaju velikani (Niko sa teritorije Novog Sada :))
- ▶ Da vidimo konkretnu strukturu nekakvog sistema

Opšta struktura SSTable

```
<beginning_of_file>
[data block 1]
[data block 2]
...
[data block N]
[meta block 1]
...
[meta block K]
[metaindex block]
[index block]
[Footer]          (fixed size; starts at file_size - sizeof(Footer))
<end_of_file>
```

(LevelDB SSTable structure, Documentation)

- ▶ Vidimo da je SSTable podeljen na nekakve blokove podataka
- ▶ Ono što nas za sada najviše zanima je struktura **data block** dela
- ▶ Ostali podaci nam nisu toliko zabavni **za sada** — videćemo ih kasnije **Data block** sadrži konkretne podatke u parovima **ključ-vrednost**, CRC proverom, vremenskom odrednicom, ...
- ▶ **ALI** podaci su kompresovani, mi se time neće baviti — ako nekoga zanima može da pogleda kako se podaci mogu kompresovati zarad uštede prostora
- ▶ **ALI** ova strukutra nam je već donekle poznata — hajde da svedemo na nešto što već znamo

```

+-----+-----+-----+-----+-----+...+...--+
|  CRC (4B)  | Timestamp (16B) | Tombstone(1B) | Key Size (8B) | Value Size (8B) | Key | Value |
+-----+-----+-----+-----+-----+...+...--+

```

CRC = 32bit hash computed over the payload using CRC

Key Size = Length of the Key data

Tombstone = If this record was deleted and has a value

Value Size = Length of the Value data

Key = Key data

Value = Value data

Timestamp = Timestamp of the operation in seconds

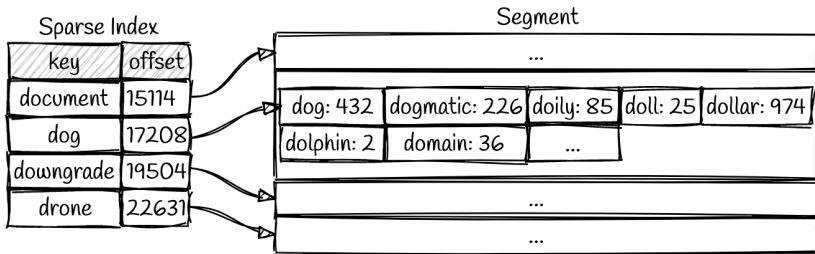
- ▶ Ova ideja zapravo čini čitanje ovakve strukture **prilično sporo**
- ▶ Količina podataka koje jedna SSTable može da čuva može biti nekada izražena u Gigabajtima (pa i više)
- ▶ Veličina zavisi od konfiguracije i upotrebe
- ▶ Setite se kompleksnosti za **scan** operaciju kod struktura tipa log-a
- ▶ Da bi malo ubrzali ceo sistem, treba da napravimo dodatan deo — deo koji ubrzava **pretragu**
- ▶ Treba nam deo koji će se tačno pozicionirati na deo u fajlu gde je ključ — **index**

Pitanje 2

Kakvu strukturu da koristimo za index i šta on da čuva, ideje :) ?

SSTable — Index

- ▶ Ideja iza svakog index-a je da što je pre moguće stignete do konkretnog sadržaja
- ▶ Pogledajte index pojmova u (svakoj) knjizi
- ▶ Ista ideja se koristi i kod SSTable-a
- ▶ Struktura je relativno jednostavna
- ▶ Sastoji se od dve vrednosti:
 1. **ključ** koji se nalazi u fajlu na disku
 2. **offset** u fajlu, tj. pozicija u fajlu na disku
- ▶ Fajl na disku u ovom slučaju je SSTable!
- ▶ Index bi bilo lepo sortirati (ne budite lenji, učinite sebi uslugu :))



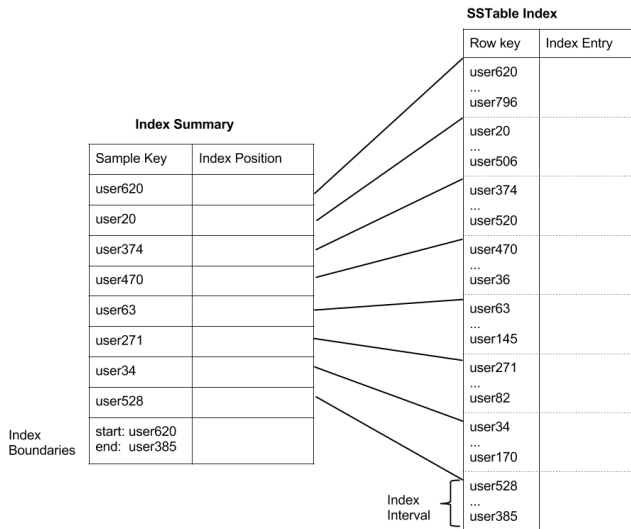
(LSM — Write Heavy Databases)

Pitanje 3

Da li nam ovo dovoljno? Vidite li probleme :)?

Index Summry

- ▶ Problem je vrlo jednostavan
- ▶ Vrlo lako može da se desi da imamo dosta SSTable struktura na disku
- ▶ Samim tim imamo i dosta Index struktura na disku
- ▶ To bi značilo da moramo otvoriti dosta fajlova da bi proverili da li je ključ tu ili ne
- ▶ Neki sistemi za skladištenje podataka (npr. Cassandra) koriste dodatan element za proveru
- ▶ Ovaj element se zove **Summary** koji može da čuva dva podataka
 1. **granice** index fajla — prvi i zadnji **ključ** u tom index-u
 2. **offset** ključa u index fajlu — poziciju u index fajlu



(Distributed Datastore, Summary)

Pitanje 4

Da li nam ovo dovoljno? Vidite li probleme :)?

- ▶ Malo smo smanjili broj čitanja
- ▶ Nećemo otvoriti svaki index fajl
- ▶ Otvorićemo svaki n – ti fajl
- ▶ Situacija je bolja, ali i dalje imamo manje više isti problem
- ▶ To nas dovodi do toga da je ideja da je SSTable sastavljena **samo** od index i data dela **pogrešna** :(
- ▶ Pa hajde da vidimo šta konkretni sistemi čuvaju sve od podataka

SSTable struktura

- ▶ Za primer, možemo da vidmo postojeći sistem za skladištenje podataka — Cassandra
- ▶ Ovaj sistem čuva svoje podatke na jednom mestu (kao i WAL)
- ▶ Na slici pored, vidimo da imamo dosta više delova od samo index-a i data dela :O
- ▶ **usertable-data-ic-1-TOC.txt** fajl sadrži spisak svih fajlova za konkretan SSTable

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)

- ▶ Veći deo sa ovog spisak smo već prošli
- ▶ Sve nam neće trebati, zato što nećemo praviti sistem za produkciju
- ▶ Kompresijom podatka se isto nećemo baviti
- ▶ Stoga, *CompressionInfo* i *Statistics* delove možemo da zanemarimo
- ▶ *TOC*, *Data*, *Index*, *Summary* znamo šta je
- ▶ Poslednja stvar koja nam ostaje, a koja nam najviše pomaže da **znatno** ubrzamo proces čitanja je deo **Filter**

Pitanje 5

Poslednja stvar koja nam ostaje, a koja nam najviše pomaže da **znatno** ubrzamo proces čitanja je deo **Filter**

Čemu služi filter, ideje :)?

- ▶ Pa kao što mu ime kaže treba da filtriramo **nešto**
- ▶ **ALI** podataka može biti jako puno
- ▶ Sistemi kao što su Cassandra, Dynamo, RocksDB itd. čuvaju enormen količine podataka i to na vse čvorova!
- ▶ Ako moramo da negde čuvamo sve ključeve, opet ništa nećemo postići :(
- ▶ Treba nekako da potrošimo što manje resursa da nam kaže da ključ nije **sigurno** prisutan, da možemo da pitamo dalje

Pitanje 6

Treba nekako da potrošimo što manje resursa da nam kaže da ključ nije **sigurno** prisutan, da možemo da pitamo dalje...

Da li smo radili nešto ovako, ideje :)?

Filter

- ▶ **BLOOM FILTER :D!!!**
- ▶ **Filter** je zapravo zapisan **Bloom filter** na disk za nekakv skup ključeva
- ▶ Bloom filter se pita, **PRE** bilo kakvog indexa, da li se traženi ključ tu **ne** nalazi ili se **možda** nalazi
- ▶ Ako se **ne** nalazi, nema potrebe da otvaramo išta, možemo da gledamo dalje
- ▶ Ako je ključ **možda** tu, onda moramo proveriti, i za provere koristimo index strukture!!
- ▶ Njih koristimo da brže stignemo do podatka, **AKO** je ključ tu

Pitanje 7

Ali Bloom filter ožahteva da mu se sepcificira koliko zapisa se očekuje da on skaldišti...

ideje :)?

- ▶ Bloom Filter-u trebamo da kažemo koliko elemenata se očekuje, ali to nije sada problem
- ▶ Ovaj podatak 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

Formiranje SSTabele

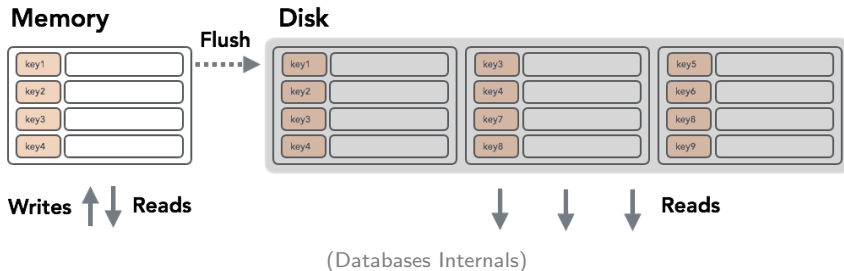
- ▶ Kada se Memtable napuni, ona se zapisuje na disk i formira SSTable
- ▶ Koristeći dostupne podatke dodatno formiramo;
 1. Bloom Filter za dostupnim kjučevima
 2. SSTable Index pošto znamo na kojoj poziciji u SSTable fajlu je koji ključ — sortirati
 3. SSTable Summary, pošto je SSTable Index već formiran i znamo pozicije ključeva — na početak staviti opseg, a ostatak sortirati
 4. TOC fajl u kom kažemo šta je sve od podatka dostpuno — svi prethodni elementi
- ▶ **ALI** ovde nije kraj :(

Finale

- ▶ Ostaje nam još **jedna stvar** koju dodatno treba da formiramo
- ▶ Dodatno formiramo *Metadata.txt* fajl o.O
- ▶ Potrebno je da formiramo **Merkle stablo** od podataka iz SSTable-a
- ▶ Nakon što je stablo formirano, stablo zapišemo u Metadata fajl
- ▶ Ovde je kraj pravljenja jednog SSTable-a
- ▶ **write path** je kompletan!
- ▶ Formiranje SSTable se obično odvija u pozadini, bez narušavanja rada sistema
- ▶ **Za vaš projekat ovo nije potrebno, može sve da se dešva i sinhrono**

Čitanje i pisanje sadržaja Memtable i SSTable

Write path smo videli prošli put, Read path radimo naredni put ali samo informativno...



Potencijalno smanjenje prostora?

Šta sa ovim podacima...Da li treba da ih čuvamo kada već imamo Index...?

CRC (4B)	Timestamp (16B)	Tombstone(1B)	Key Size (8B)	Value Size (8B)	Key	Value
----------	-----------------	---------------	---------------	-----------------	-----	-------

CRC = 32bit hash computed over the payload using CRC

Key Size = Length of the Key data

Tombstone = If this record was deleted and has a value

Value Size = Length of the Value data

Key = Key data

Value = Value data

Timestamp = Timestamp of the operation in seconds

Informativno, struktura Cassandra SSTable jb verzija

SSTable Index

Row key	Index Entry
user620	
user591	
user23	
user323	
user385	

SSTable Data

Row Key (user620)
localDeletionTime
markedForDeleteAt
sorted list of columns

Row Key (user23)
localDeletionTime
markedForDeleteAt
sorted list of columns

Live Column

column name
column type (0)
timestamp
column value

Dodatni materijali

- ▶ Database Internals: A Deep Dive into How Distributed Data Systems Work
- ▶ Dynamo: Amazon's Highly Available Key-value Store
- ▶ Cassandra - A Decentralized Structured Storage System
- ▶ Structured storage LevelDB
- ▶ Google BigTable paper
- ▶ System Overview: LevelDB

Pitanja

Pitanja :) ?