Napredni algoritmi i strukture podataka

SSTable, Index, Summary, Struktura, Formiranje

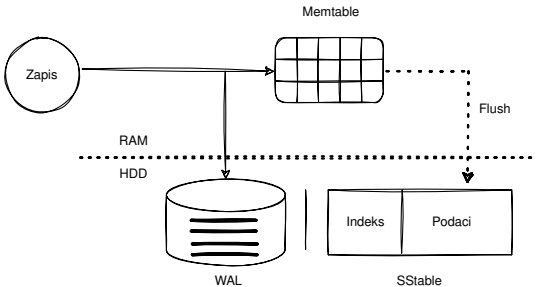
Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Rekapitulacija

- Memtable ideja je relaltivno jednostavna – zapisati podatke u memorju 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 (objansite 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

- SSTable se sastoji od nekoliko elemenata
  - **Index** deo – lakše pozicioniranje na blokove podataka
  - **Data** segmenta blokovi podataka
- Sve ove elemente je potrebno formirati kada se formira SSTable
- Korišćenje **Memtable**, **WAL** i **SStable** je poznato kao **write-path**



(Write path)

Pitanje 1

Kakvu strukturu da koristimo za sstable, ideje :) ?

SStable - ideja

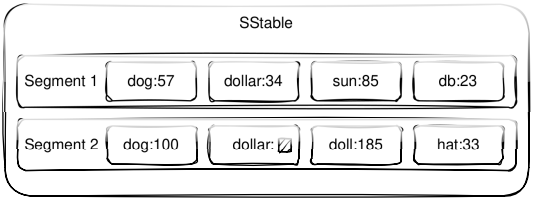
- Ideja iza tabele sortiranih stringova (**SSTable**) je relativno jednostavna
  - To je niz parova **ključ-vrednost** koji su sortirani, i zapisani na disk
  - Pod nizom sortiranih stringova ne misli se doslovno na stringove, već na **niz bajtova**
  - Ključ će biti **string** to svakako, ali vrednost može biti bilo šta
  - Zbog toga je najjednostavnije da se čuva niz bajtova
- **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!

SSTable — Tombstone

- Kada se obriše podatak, on nije momentalno uklonjen sa diska
- Sistem zapisuje specijalan podatak *Tombstone* da je neki ključ obrisan — 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

SStable – struktura

- Izabraćemo opšti oblik SSTable-a – ključ:vrednost
- Pre zapisa Memtabe-a, sve vrednosti se **sortiraju**
  - Ovo će nam trebati kasnije!!
- Ovim smo dobili **Data** segment
  - Svi podaci se nalaze tu
  - Podaci organizovani u blokove kao kod WAL-a



(SSTable)

SStable – struktura

- 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, ali i to je dobro za početak
- Dodatno znamo da je strukutra nepromenljiva i da je Log-based
- Hajde da vidimo, kako to rešavaju velikani (Niko sa teritorije Novog Sada :))
- Da vidimo konkretnu strukturu nekakvog sistema

Opšta struktura SSTable – LevelDB, RocksDB

```
<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)

Poznata struktura

- 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 – videćemo kasnije osnove kompresije podataka
- **ALI** ova strukutra nam je već donekle poznata – hajde da svedemo na nešto što već znamo

Poznata struktura

```
+---------+-----------+-----------+--- ... ---+
|CRC (4B) | Size (2B) | Type (1B) | Payload   |
+---------+-----------+-----------+--- ... ---+

CRC = 32bit hash computed over the payload using CRC
Size = Length of the payload data
Type = Type of record
       (kZeroType, kFullType, kFirstType, kLastType, kMiddleType )
       The type is used to group a bunch of records together to represent
       blocks that are larger than kBlockSize
Payload = Byte stream as long as specified by the payload size
```

```
+--------------+-----------------+---------------+---------------+-----------------+-...-+--...--+
|    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
```

Poznata struktura

- 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**

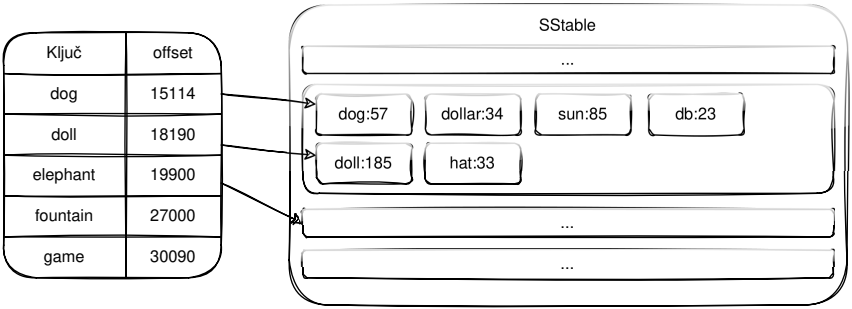
Pitanje 2

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 – **data segment**!
- Index bi bilo lepo sortirati (ne budite lenji, učinite sebi uslugu :))

SSTable – Index



Pitanje 3

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

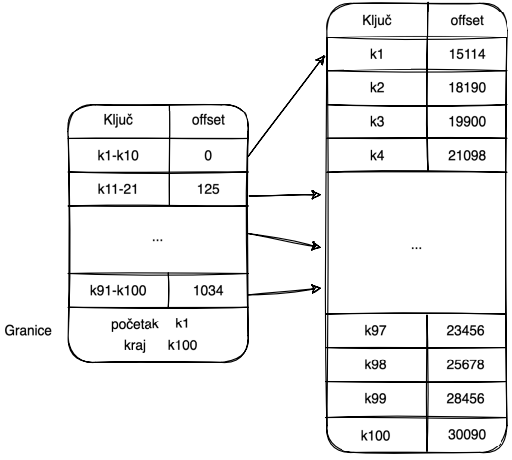
Index Summry

- 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
- Obično čuva uzorak ključeva po principu uzorkovanja
  - npr svaki deseti ključ
  - Ne preterivati, ova struktura bi trebala da bude u memoriji – što manja

Index Summry

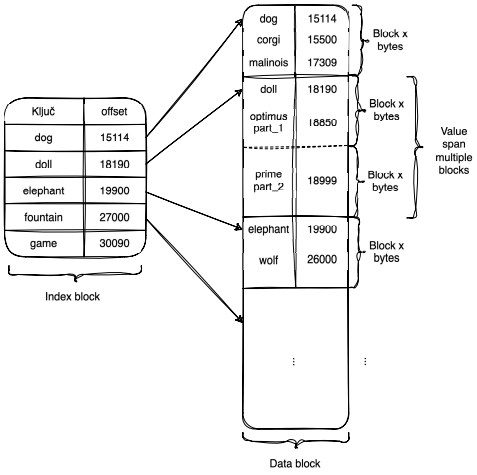
- Index Summry nam pomaže da se brže pozicioniramo do podataka
- Ako na primer imamo 100 ključeva (k0-k99)
  - Urorkujemo svaki deseti ključ: k0, k10, k20,...
  - Tražimo ključ k27
  - Pošto je sve sortirano znamo da je k27 posle k20
  - Pozicioniramo se u Index na ključ k20
  - Uradimo **scan** do k27, ovo nije zahtevno :)
  - Imamo ključ k27, direkt pristupimo podacima u **Data** segmentu



(Index Summry)

Blok index

- Index može biti relativno velika datoteka i nalazi se na disku
- Blokovski pristup je mnogo bolja opcija
- Organizovati ključeve u blokove nekako, pošto je i **Data** deo blokovski organizovan
- Blok index sadrži **jedan ključ** po **bloku podataka** – početak bloka
- Održavamo **selective** ili **sparse index**, pretraga po istom principu kao i *Summry*
- Može i dosta kampresije, ali o tome drugi put :)



(Blok index)

Pitanje 4

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

Novi problem

- 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 SStable sastavljena **samo** od index i data dela **pogrešna** :(
- Pa hajde da vidimo šta konkretni sistemi čuvaju sve od podataka
- Razni sistemi mogu da imaju neke specifične dodatke, ali ograničićemo se na neki opšti oblik

SSTable struktura

- Kao primer, možemo da vidmo postojeći sistem za skladištenje podataka — Cassanda
- 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)

SSTable struktura

- Veći deo sa ovog spiska smo već prošli
- Sve nam neće trebati, zato što nećemo praviti sistem za produkciju
- Stoga *Statistics* delo možemo da zanemarimo generalno, a *CompressionInfo* **za sada**
- *TOC*, *Data*, *Index*, *Summry* 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 :)?

SSTable filter

- Pa kao što mu ime kaže treba da filtriramo **nešto**
- **ALI** podataka može biti jako puno
- Sistemi kao što su Cassandra, Dynamo, RocskDB itd. čuvaju enormen količine podataka i to na vše 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
- Izbegnemo nepotrebne pretrage

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 zahteva da mu se sepcificira koliko zapisa se očekuje da on skaldišti...

Znamo li ovo, ideje :)?

Filter

- 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 koliko će elemenata će biti sačuvano – kreiranje SSTable-e
- Samim tim i za Bloom Filter imamo tu informaciju unapred poznatu – sweet :)
- Stoga, sve elemente možemo formirati ispravno
- I sa ovim jednostavnim dodatkom, ubrzati enormno proces pretrage

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 zapižemo 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**

Napomena

- Voditi računa da imamo (minimalno) dve moguće opcije za čuvanje informacija za SSTable – bitno za vaš projekat!

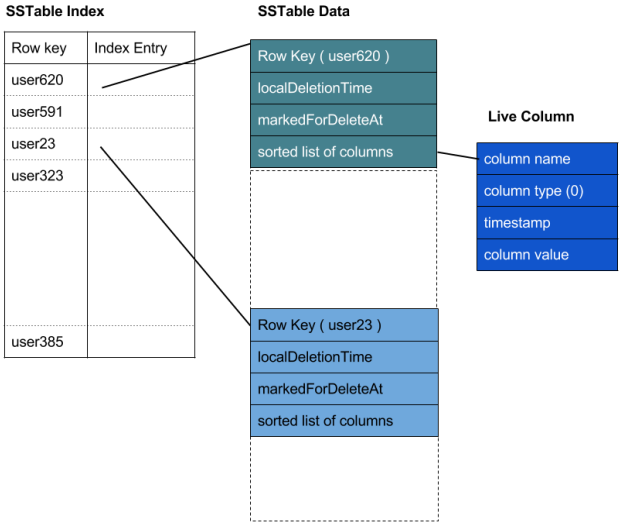
```
<beginning_of_file>
[data block 1]
[data block 2]
...
[data block N]
[meta block 1]
...
[meta block K]
[metaindex block]
[index block]
[Footer]        (fixed size; starts at file_size - sizeof(Footer))
<end_of_file>
```

(LevelDB Format)

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

(Cassandra format)

Informativno, struktura Cassandra SSTable jb verzija



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

Pročitati za narednu sedmicu

- Dynamo: Amazon's Highly Available Key-value Store
- Cassandra - A Decentralized Structured Storage System
- Google BigTable paper
- Fast and exact analysis for LRU caches