Napredni algoritmi i strukture podataka

SSTable, Index, Summary

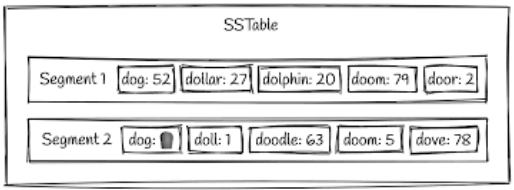
Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

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 – 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)

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
```

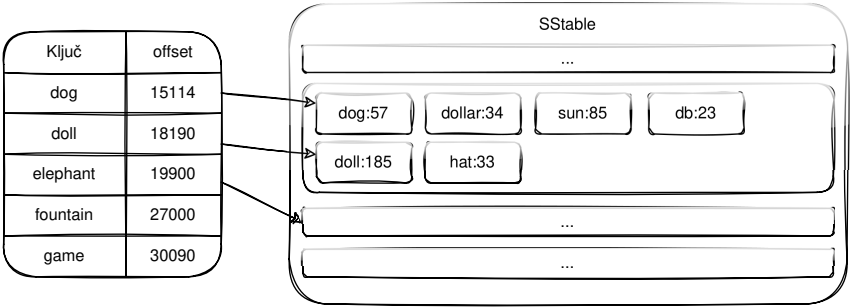
```
+--------------+---------------+--------------+--------------+----------------+-...-+--...--+
|    CRC (4B)  | Timestamp (16B) | Tombstone(1B) | Key Size (8B) | Value Size (8B) | Key | Value |
+--------------+---------------+--------------+--------------+----------------+-...-+--...--+

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
```

SSTable – Index

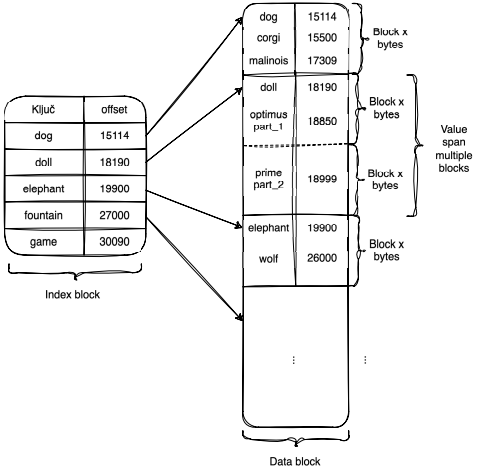
- Ideja iza svakog index-a je da što je pre moguće stignete do konkretnog sadržaja
- Pogledajte index pojmova u (svakoj) knjizi
- Ista ideja se koristi i kod SSTable-a
- Struktura je relativno jednostavna
- Sastoji se od dve vrednosti:
  1. **ključ** koji se nalazi u fajlu na disku
  2. **offset** u fajlu, tj. pozicija u fajlu na disku
- Fajl na disku u ovom slučaju je SStable – **data segment**!

SSTable – Index



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**



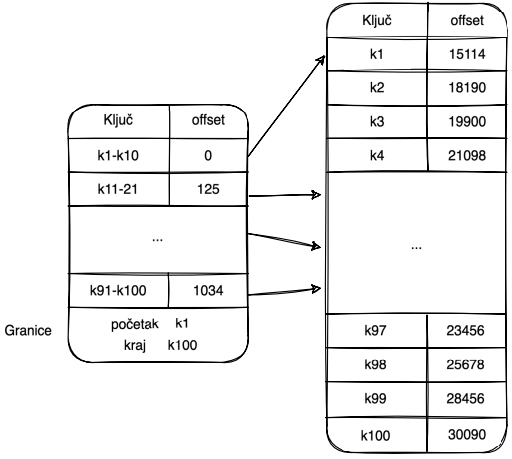
(Blok index)

Index Summary

- 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 Summary

- 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)

SSTable struktura

- Za 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)

Filter

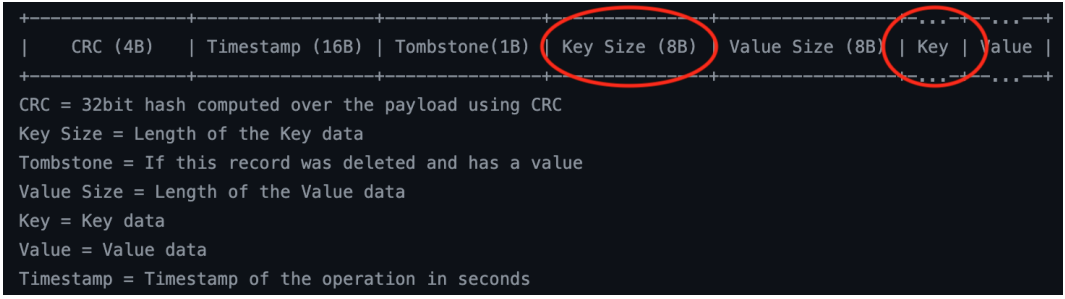
- **Filter** je zapravo zapisan **Bloom filter** na disk za nekakv skup ključeva
- Bloom filter se pita, **PRE** bilo kakvog indexa, da li se traženi ključ tu **ne** nalazi ili se **možda** nalazi
- Ako se **ne** nalazi, nema potrebe da otvaramo išta, možemo da gledamo dalje
- Ako je ključ **možda**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

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. TOC fajl u kom kažemo šta je sve od podatka dostpuno — svi prethodni elementi
- Potrebno je da formiramo **Merkle stablo** od podataka iz SSTable-a
- Nakon što je stablo formirano, stablo zapižemo u Metadata fajl

Potencijalno smanjenje prostora?

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



Zadaci

- Kada se Memtable (implementirana na prošlim vežbama) popuni do unapred definisanog praga, podatke sortirati i sačuvati na disk stvarajući SSTable
- Prilikom zapisivanja na disk, potrebno je ispravno kreirati sve definisane elemente koji čine SSTable (Data, Index, Summary, Filter, TOC, Metadata)
- Prilikom stvaranja elemenata za SSTable koristiti format imenovanja *usertable-GENERATION-ELEMENT.[db/txt]*, gde
  - *ELEMENT* može biti nešto od Data, Index, Summary, Filter, TOC, Metadata
  - *GENERATION* je indeksni broj koji se povećava svaki put kada se kreira novi SSTtable
- Za Filter koristiti Bloom Filter implementiran na nekom od prethodnih vežbi
- Za Metadata koristiti implementaciju Merkle stabla implementiranog na nekom od prethodnih termina vežbi