Projektni zadatak iz predmeta

Napredni algoritmi i strukture podataka

Školska godina 2023/2024.

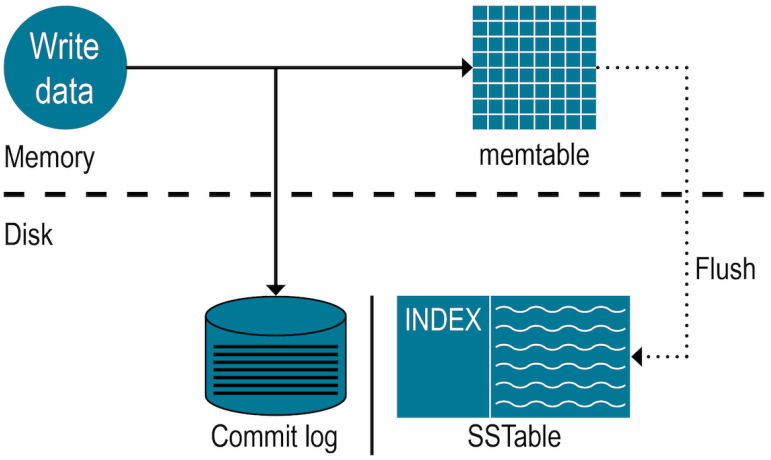
Zadatak je napraviti Key-Value engine za skladištenje podataka. Engine treba da bude implementiran kao konzolna aplikacija sa kojom korisnik može interagovati.

Osnovne operacije

- **PUT(key, value)** - Dodavanje novog zapisa u sistem. Ključ je string, a vrednost niz bajtova. Alternativno, korisnik vrednost može uneti kao string, koji će sistem naknadno konvertovati u niz bajtova.
- **GET(key) -> value** - Dobavljanje vrednosti pod zadatim ključem.
- **DELETE(key)** - Brisanje zapisa pod zadatim ključem.

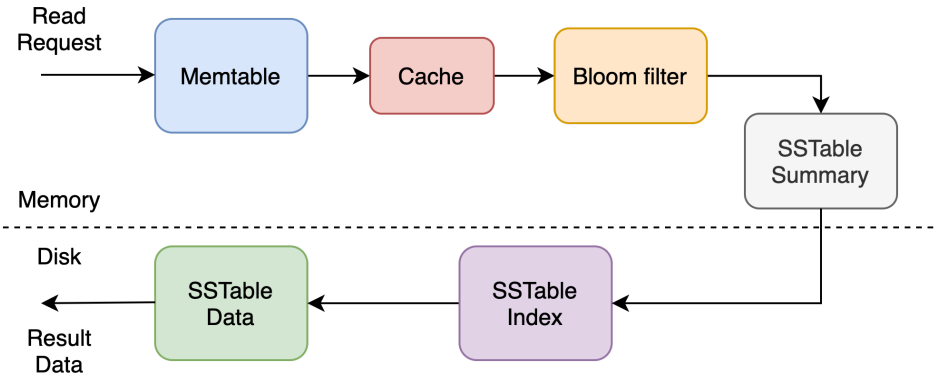
Podržane operacije treba da se oslone na ispravno implementirani read i write path. Osnovni tokovi read i write path-a opisani su ispod.

Write path



- Kada korisnik pošalje PUT ili DELETE zahtev, on se prvo zabeleži u Write-Ahead Log (Commit Log) WAL
- Kada WAL potvrdi zapis, potrebno je da se podatak doda u Memtable, koji se nalazi striktno u memoriji
- Kada se dostigne unapred definisana veličina Memtable-a, vrednosti se sortiraju po ključu i formira se novi SSTable koji se zapisuje na disk.
- Nakon toga proveravamo da li je ispunjen uslov za pokretanje kompakcija i pokrećemo ih ako jeste. Treba voditi računa o tome da kompakcije na jednom nivou mogu izazvati pokretanje kompakcija na narednom nivou.

Read path



- Kada korisnik pošalje GET zahtev, prvo proverimo da li se zapis nalazi u Memtable strukturi (ako je tu, vratimo odgovor)
- Nakon toga proveravamo da li je se zapis nalazi u Cache strukturi (ako je tu, vratimo odgovor)
- Nakon toga proveravamo jednu po jednu SSTable strukturu tako što učitamo njen Bloom Filter i pitamo da li je ključ prisutan. Ako nije, prelazimo na naredni SSTable, a ako možda jeste, moramo proveriti ostale strukture trenutne tabele.
  Napomena: Naredni SSTable se određuje u odnosu na odabrani algoritam za kompakciju. Kada neuspešno pretražimo sve SSTable kandidate na jednom nivou LSM stabla, tada prelazimo na naredni nivo. Postupak se ponavlja sve dok ne pronađemo ključ ili ne dođemo do poslednjeg nivoa.
- Prvo proverimo da li se ključ nalazi u Summary opsegu. Ako se nalazi, pronađemo poziciju u Index strukturi na koju treba da odemo
- U Index-u pronađemo poziciju u Data strukturi sa koje treba da pročitamo zapis. Kada pročitamo vrednost, vratimo odgovor korisniku.

1. Komponente

1.1 Write-Ahead Log (WAL)

- Struktura jednog WAL zapisa treba da izgleda kao na slici ispod.

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

- WAL treba implementirati kao segmentirani log. Svaki segment ima fiksan broj zapisa koje korisnik specificira.
- WAL segment ne može biti obrisan dok se podaci ne perzistiraju trajno u SSTable
- Prilikom čitanja zapisa iz WAL-a, treba proveriti integritet podataka oslanjajući se na CRC polje.

- WAL zapisi se čitaju sa diska jedan po jedan, a ne učitavanjem celih segmenata u memoriju.

Dodatni zahtevi:

- **1.1[DZ1]** - Svaki segment treba bude fiksne dužine koja je specificirana u bajtovima od strane korisnika.

1.2 Memtable

- Apstraktna struktura podataka koja podržava operacije dodavanja i (logičkog) brisanja elemenata.
- Implementirana je tipom hash map.
- Maksimalnu veličinu Memtable-a specificira korisnik tako što navodi broj elemenata.
- Kada se sistem pokrene, Memtable treba popuniti zapisima iz WAL-a

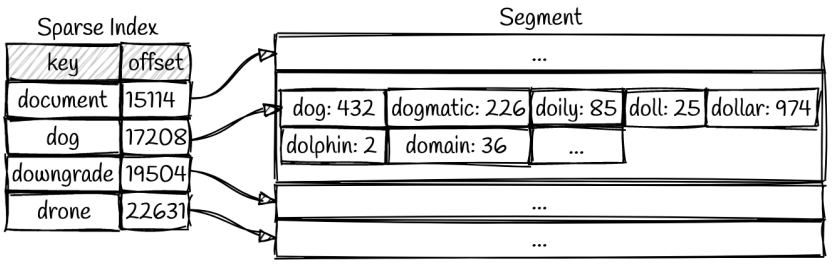
Dodatni zahtevi:

- **1.2[DZ1]** - Konkretne strukture na koje se Memtable može osloniti su hash mapa, skip lista ili B stablo. Korisnik sistema bira koju implementaciju će sistem koristiti.
- **1.2[DZ2]** - Memtable može imati N instanci u memoriji (gde N određuje korisnik), N-1 read-only i 1 read-write tabelu. Flush se radi kada popunimo svih N tabela.

1.3 SSTable

Jedan SSTable se sastoji iz sledećih elemenata

- **1.3.1 Data** - Sadrži konkretne podatke
  - Struktura jednog zapisa unutar može biti identična strukturi WAL zapisa, a možemo i izostaviti neka polja.
  - Ne možemo učitati ceo sadržaj u memoriju, već blok po blok.
- **1.3.2 Filter** - Bloom Filter svih ključeva iz Data strukture
- **1.3.3 Index** - Indeks za Data strukturu



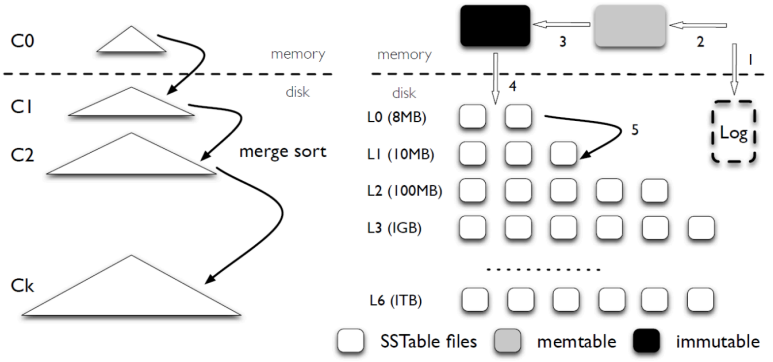
  - Ne možemo učitati ceo sadržaj u memoriju, već blok po blok.
  - Svaki zapis sadrži ključ iz Data strukture i njegov offset
- **1.3.4 Summary** - Indeks za Index strukturu
  - Pored toga što sadrži ključeve i offset-e u Indeks strukturi, treba da sadrži i granice Index fajla, što su minimalna i maksimalna vrednost ključa.
  - Ne možemo učitati ceo sadržaj u memoriju, već blok po blok.
- **1.3.5 Metadata** - Merkle stablo svih vrednosti iz Data strukture

- Sistem treba da podrži operaciju validacije Merkle stabla koju inicira korisnik. On bira SSTable, a sistem treba da detektuje da li je i gde došlo do izmena u data strukturi te tabele.

Dodatni zahtevi:

- **1.3[DZ1]** - Korisnik može da bira stepen proređenosti Summary i Index strukture (na primer stepen 5 za Summary znači da će pozicija svakog 5. zapisa iz Index strukture biti zabeležena u Summary strukturi).
- **1.3[DZ2]** - SSTable strukture mogu biti zapisane u zasebnim fajlovima ili u istom fajlu. Korisnik bira koju opciju želi. Promena konfiguracije ne treba da utiče na mogućnost čitanja tabela kreiranih u prošlosti.
- **1.3[DZ3]** - Format SSTable zapisa treba da odstupa od WAL formata na sledeći način:
  - Ukoliko je *tombstone* polje postavljeno na true, *value* i *value size* polja treba izostaviti
  - Na sva polja čija vrednost je numerička treba primeniti varijabilni enkoding
  - U polju *key* ne čuva se konkretna vrednost ključa, već numerička vrednost koja mu je dodeljena u globalnom rečniku kompresije koji postoji na nivou svih SSTabela
  - Korisnik kroz konfiguraciju bira da li se kompresija radi ili ne

1.4 LSM stabla



- SSTable strukture organizujemo u LSM stablo i za svaki SSTable moramo znati kom nivou pripada.
- Korisnik definiše maksimalan broj nivoa LSM stabla.
- Kada se ispuni uslov za kompakciju tabela, potrebno je pokrenuti odgovarajući algoritam.
- Sistem treba da podrži size-tiered algoritam za kompakciju, a korisnik specificira sve potrebne parametre za algoritam.

Dodatni zahtevi:

- **1.4[DZ1]** - Sistem treba da podrži i size-tiered i leveled algoritme za kompakciju, a korisnik specificira koji algoritam će biti korišćen, kao i sve potrebne parametre za odabrani algoritam.

1.5 Cache

- Cache implementirati koristeći LRU algoritam
- Korisnik može da specificira maksimalnu veličinu keša
- Potrebno je voditi računa o tome da keš ne sadrži zastarele verzije podataka i da se ispravno ažurira prilikom svake operacije čitanja

1.6 Block manager i block cache

- Disk je segmentiran na stranice fiksne veličine. Najmanji deo diska nad kojim se obavljaju IO operacije je jedna stranica. Blok je logički segment čija veličina je umnožak veličine stranice.
- Sadržaju fajlova na disku mora se pristupati preko Block manager sloja. Block manager podržava operacije čitanja i pisanja blokova. Prilikom čitanja, navodi se putanja do fajla i redni broj bloka. Prilikom pisanja, navodi se putanja do fajla, redni broj i sadržaj bloka. **Direktno čitanje i pisanje u fajl van block manager strukture nije dozvoljeno.**
- Blok može biti veličine 4KB, 8KB ili 16KB i specificira se kroz eksternu konfiguraciju.
- Block manager oslanja se na Block cache na sledeći način:
    - Prilikom operacije čitanja, block manager prvo proverava da li se blok nalazi u kešu. Tek ako utvrdi da se blok ne nalazi u kešu, čita sadržaj bloka sa diska, upisuje ga u keš i prosleđuje sadržaj bloka kao povratnu vrednost.
    - Prilikom operacije pisanja, menja se sadržaj bloka na disku i ažurira se sadržaj bloka u kešu, ako je on tu bio prisutan.
- Block cache koristi LRU algoriram i njegova veličina podešava se kroz konfiguraciju.

2. Proširenja sistema

2.1 Eksterna konfiguracija

- Sva podešavanja koja korisnik može definisati, a koja su navedena kroz specifikaciju projekta, treba da se postave kroz eksterni konfiguracioni fajl čiji sadržaj se proverava pri pokretanju sistema.
- Za sva podešavanja koja nedostaju u konfiguracionom fajlu sistem treba da dodeli default vrednosti koje se navode u kodu.
- Format datoteke birate sami (JSON, YAML itd)

2.2 Ograničenje stope pristupa

- Implementirati pomoću Token Bucket algoritma.
- Korisnik može da podesi parametre algoritma (broj tokena, interval resetovanja).
- Stanje token bucket-a treba čuvati serijalizovano u sistemu kao i svaki drugi zapis, uz to da krajnjem korisniku treba zabraniti da vidi taj zapis ili poziva bilo kakve operacije nad njim.

2.3 Operacije sa probabilističkim tipovima

Potrebno je proširiti skup operacija sistema sledećim operacijama:

- **Rad sa tipom BloomFilter**
    1. Kreiranje nove instance
    2. Brisanje postojeće instance
    3. Dodavanje novog elementa u neku instancu
    4. Provera da li je element prisutan u nekoj instanci
- **Rad sa tipom CountMinSketch**
    1. Kreiranje nove instance
    2. Brisanje postojeće instance
    3. Dodavanje novog događaja u neku instancu
    4. Provera učestalosti događaja u nekoj instanci
- **Rad sa tipom HyperLogLog**
    1. Kreiranje nove instance
    2. Brisanje postojeće instance
    3. Dodavanje novog elementa u neku instancu
    4. Provera kardinaliteta
- **Rad sa tipom SlmHash**
    1. Čuvanje fingerprint-a prosleđenog teksta
    2. Računanje Hemingove udaljenosti za dva fingerprint-a

Napomena: Probabilističe tipove treba serijalizovati i čuvati kao obične zapise u sistemu. Međutim, oni ne treba da budu vidljivi pozivom drugih operacija niti njihove vrednosti možemo menjati običnom PUT operacijom.

2.4 Operacije skeniranja

Potrebno je proširiti skup operacija sistema sledećim operacijama:

- **PREFIX_SCAN(prefix, pageNumber, pageSize) -> [page](key, value)**
  Dobavljanje liste zapisa čiji ključ počinje zadatim prefiksom. Prefiks je tipa string. Rezultati treba da budu sortirani rastuće po vrednosti ključa. Pored toga potrebno je omogućiti paginaciju. Parametri pageNumber i pageSize specificiraju redni broj stranice koji treba vratiti, kao i veličinu stranice.
- **RANGE_SCAN(range, pageNumber, pageSize) -> [page](key, value)**
  Dobavljanje liste zapisa čiji ključ se nalazi u zadatom opsegu. Parametrom range definišu se minimalna i maksimalna vrednost ključa. Rezultati treba da budu sortirani rastuće po vrednosti ključa. Pored toga potrebno je omogućiti paginaciju. Parametri pageNumber i pageSize specificiraju redni broj stranice koji treba vratiti, kao i veličinu stranice.

Napomena: Potrebno je dokumentovati i obrazložiti algoritme koji implementiraju ove operacije.

2.5 Operacije sa iteratorima

Potrebno je proširiti skup operacija sistema sledećim operacijama:

- **PREFIX_ITERATE(prefix) -> iterator**
  Kreiranje iteratora nad listom zapisa čiji ključ počinje zadatim prefiksom. Rad sa iteratorm je interaktivan: pozivi operacije **next** vraćaju naredni zapis, dok operacijom **stop** prekidamo rad sa iteratorom. Rezultati treba da budu sortirani rastuće po vrednosti ključa.
- **RANGE_ITERATE(range) -> iterator**
  Kreiranje iteratora nad listom zapisa čiji ključ se nalazi u zadatom opsegu. Rad sa iteratorm je interaktivan: pozivi operacije **next** vraćaju naredni zapis, dok operacijom **stop** prekidamo rad sa iteratorom. Rezultati treba da budu sortirani rastuće po vrednosti ključa.

Napomena: Potrebno je dokumentovati i obrazložiti algoritme koji implementiraju ove operacije.

Ocenjivanje

Za ocenu **6** treba implementirati:

- Osnovne operacije koje se oslanjaju na opisani read i write path. Komponente koje treba da budu uključene u read i write path su: 1.1, 1.2, 1.6 i 1.3 bez 1.3.5
- Zahteve: 2.1

Za ocenu **7** treba implementirati

- **Sve navedeno za ocenu 6**
- Komponente: 1.3.5 i 1.5
- Dodatne zahteve 1.2[DZ1] i 1.3[DZ1]

Za ocenu **8** treba implementirati

- **Sve navedeno za ocenu 7**
- Zahteve: 2.2
- Dodatne zahteve: 1.1[DZ1], 1.2[DZ2]

Za ocenu **9** treba implementirati

- **Sve navedeno za ocenu 8**
- Komponente: 1.4
- Zahteve: 2.3

Za ocenu **10** treba implementirati

- **Sve navedeno za ocenu 9**
- Zahteve: 2.4, 2.5
- Dodatne zahteve: 1.3[DZ2], 1.3[DZ3], 1.4[DZ1]

Pravila polaganja

- Projekat se radi u timovima koji imaju 3-5 članova. Članovi tima ne moraju slušati vežbe u istom terminu.
- Primarni programski jezik za implementaciju je Golang. Druge opcije su C, C++ i Rust, ali u tom slučaju nemate pomoć ako naiđete na probleme koji su vezani za odabrani jezik.
- Potrebno je koristiti Git i GitHub/GitLab za rad na projektu. Repozitorijum može biti javan ili privatan, s tim da ako je privatan morate omogućiti pristup asistentu. Na repozitorijumu se mora videti kontinualan rad tima na projektu.
- Članovi tima treba da podele posao ravnomerno. Predmet nije moguće položiti ako niste učestvovali ili ste minimalno učestvovali u izradi projekta.
- Projekat se polaže u dva roka: februar i septembar.
- Ukoliko uradite zahteve za 10 i projektni zadatak za 10+, dobijate ocenu 10 bez izlaska na usmeni deo ispita. Ovo pravilo važi u februarskom roku, dok u septembru ne postoji opcija da dobijete ocenu 10+.
- Upotreba biblioteka za serijalizaciju, osim encoding/binary paketa ili sličnih biblioteka koje prevode numeričke vrednosti u niz bajtova i obrnuto, strogo je **zabranjena**. Neophodno je poštovati formate koji su definisani specifikacijom. Izuzetak je (de)serijalizacija konfiguracionog fajla za koji možete koristiti JSON, YAML ili neki treći format.