Napredni algoritmi i strukture podataka

LSM Stabla, Kompakcije, Amplifikacije

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Read path vs Write path vs Tombstone

- Write path može stvoriti preše fragmentisanih delova — SSTable-a
- Svaki put kada se Memtable popuni, uradimo Flush na disk praveći SSTable
- Time ne samo da fragmentišemo disk, nego Read path ima problem da ispita sve te delove — čitanja postaju jako skupa vremenom
- Pored toga SSTable je nepromenljiva struktura, šta ako su podaci obrisani...
- Brisanje je zapravo dodavanja, što žnači da kada nesto obrišemo dodajemo nov element — pravimo dodatne zapise
- Ako podataka više nije aktivan i validan, trebalo bi da ga ulonimo sa diska — problem ako je podatak u više fajlova

LSM Stabla — ideja

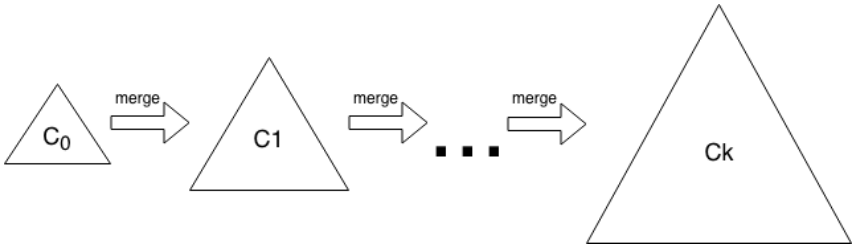
- Omogućiti da SSTable elementi ne postojie tako sami za sebe i ukloniti višak
- One su (često) deo veće strukture koja se zove **Log structured Merge Trees** ili **LSM stabla**
- SSTable je struktura tipa log-a, SSTable grade LSM stabla, LSM stabla su strukture tipa log-a
- LSM stablo je struktura podataka dizajnirana da obezbedi jeftino indeksiranje za datoteke koje imaju visoku stopu dodavanja (i brisanja) tokom dužeg perioda
- Dobar moderan primer može biti dolazni **stream** podataka koji se zapisuje u nekakvu tabelu
- Jedina stvar koja je potrebna da bi LSM imala prednost izbora, spram drugih struktura je visoka stopa ažuriranja naspram stope čitanja
- U isto vreme pretraga mora biti dovoljno česta, da bi se održavala neka vrsta indeksa

LSM Stabl — struktura

- Pošto govorimo o stablu, ono sigurno ima nekakve nivoe
- LSM stablo se sastoji od dva ili više nivao $C_i = (C_0, ..., .C_k)$
- Ove nivoe čine strukture podataka u obliku stabla
- Na primer, dvokomponentno LSM stablo ima:
  1. Manju komponentu koja je u potpunosti rezidentna u memoriji, poznata kao C_0 stablo — C_0 komponenta
  2. Veću komponentu koja je rezidentna na disku, poznatu kao C_1 stablo — C_1 komponenta

- Podaci se prvo ubacuje u C_0, a odatle migriraju/zapisuju u C_1
- Nivo C_0 služi kao bafer za zapise
- Jeftino dodati unos u C_0 stablo rezidentno u memoriji
- Broj nivoa je konfigurabilan, možemo specificirati koliko god nam je potrebno
- **Ova stvar ima smisla da bude deo konfiguracije**
- U našem slučaju nivo C_0 je Memtable, ono se nalazi u memoriji, fiksne je dužine
- Primenljivo je i služi kao bafer — da amortizuje sporost diska
- Drugi nivo C_1 čini SSTable koje nastaje kada se Memtable (nivo C_0) popuni i zapisujemo na disk

- LSM stabla su (uglavnom) sačinjena od SSTable-a
- Svaki nivo C_k LSM stabla sadrži n SSTable-a
- Spajati SSTable i obrisati nepotrebne podatke! (how cool :))
- Ovaj proces kreira veću SStable, samim tim kreira i novi C_k nivo u LSM stablu
- Briše nepotrebne podatke
- I to je sve
- Vrste:
  - Minor — Memtable u SSTable
  - Major — Spajanje više SSTable-a



(LSM Tree new lecel)

Kompakcije — plan

- Zbog strukture SSTable-a, ova operacija je veoma efikasna
- Koriste algoritam koji podsecća na sortiranje spajanjem — **merge sort**
- Zapisi se čitaju iz nekoliko izvora uzastopno i mogu se odmah dodati u izlaznu datoteku
- Pošto su svi ulazi sortirani i spojeni, rezultirajući fajl će imati isto svojstvo (sweeet :))
- Proces pravljenja indeksne datoteke može biti skuplja operacija u smislu složenosti
- Kada napravimo novu SSTable, moramo napraviti i sve prateće elemente za nju!

(LSM Tree compaction)

Kompakcije — proces spajanja

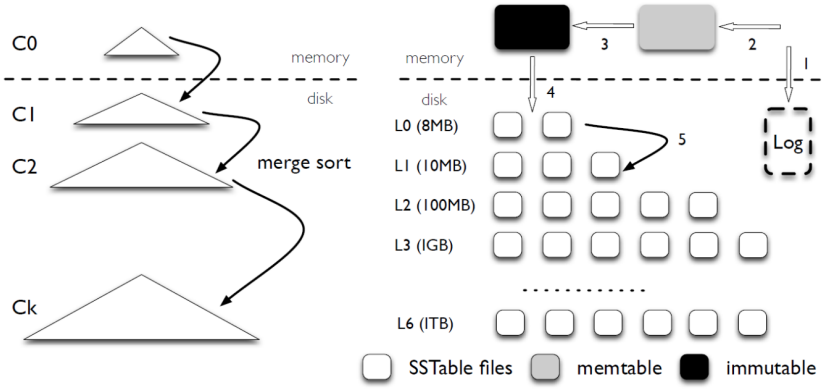
- Proces spajanja je **jako** bitna stvar kod formiranja LSM stabla
- Sa njim, čisitmo disk od bespotrebnih podataka
- Proces čitanja postaje dosta brži
- **Ali** moramo biti mudri, da ne zapadnemo u probleme!
- Postoje tri stvari koje su važne kod procesa spajanja
  1. Garancije složenosti
  2. Logika brisanja
  3. Amplifikacije

Kompakcije — garancije složenosti

- U smislu složenosti, spajanje SSTable-a je isto kao i spajanje sortiranih kolekcija
- Ima $\mathcal{O}(N)$ memoriju overhead, gde je N količina SSTtable-a koje se spajaju
- Iteratori moraju pokazivati na korespodentne elemente iz obe SSTable-e
- Na svakom koraku, stavka se uzima iz sortirane kolekcije i ponovo popunjava iz odgovarajućeg iteratora
- Tokom kompakcije, sekvencijalno čitanje i sekvencijalno pisanje pomažu u održavanju dobrih garancija performansi

Kompakcije — logika brisanja (Shadowing)

- Shadowing je neophodno da bi se osiguralo da ažuriranja i brisanja funkcionišu
- Brisanje u LSM stablu se izvodi tako što se doda specijalna oznaka (Tombstone), navodeći koji ključ je označen za brisanje
- Slično tome, ažuriranje je samo zapis sa vecćom vremenskom oznakom
- Tokom čitanja, zapisi koji su označe brisanje neće biti vracćeni klijentu
- Ista stvar se dešava i sa ažuriranjima
- Od dva zapisa sa istim ključem, vracća se onaj sa kasnijom vremenskom oznakom



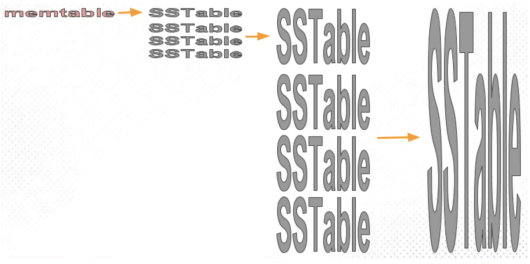
(LSM Tree compaction)

Kompakcije — algoritmi i amplifikacije

- Postoji nekoliko algoritama za kompakciju podataka
  - *Size-tiered* kopakcija
  - Leveled kompakcija
  - Hibridna kompakcija — kombinacija prethodna dva algoritma
- Ovi algoritmi imaju različite amplifikacione osobine koje se dešavaju tokom ovog procesa:
  - Amplifikacija čitanja označava broj operacija koje se dešavaju na disku pri zahtevu za čitanje
  - Amplifikacija pisanja n je definisano kao bajtovi podataka koji su stvarno upisani na disk kada je potrebno upisati jedan bajt podataka
  - Amplifikacija prostora se uglavnom odnosi na količinu nesakupljenih isteklih podataka, koji su ili stare verzije podataka ili izbrisani unosi

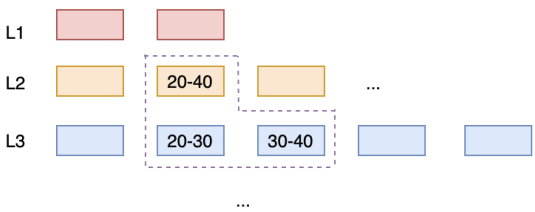
Kompakcije — Size-tiered kopakcija

- Prednost: nizak nivo amplifikacije pisanja je pogodno za sisteme koja zahtevaju intenzivno pisanje
- Nedostatak: amplifikacija čitanje i prostora je relativno visoko
- Kada na nekom nivou C_i nakupimo n SStable-a, spajamo ih da bi dobili novi nivo C_{i+1}
- Spajanje na nivou C_i može da izazove spajanje na višim nivoima lančano
- Kada uradimo spajanje, obrišemo nepotrebne SSTable-e



Kompakcije — Leveled kompakcija

- LSM-stablo se sastoji od više nivoa — naredni nivo je T puta vecći od prethodnog nivoa
- Svaki nivelisani nivo je **run** koji se sastoji od više SSTable-a — naredni **run** 10x prethodni
- Kada veličina podataka svakog nivoa dostigne gornju granicu, ovaj nivo cé se spojiti sa **run**-om sledećeg nivoa
- Pošto je sve sortirano, prilikom kompakcije gledamo gde se ključ nalazi i spajamo sa tom SSTable-om
- Problem sa kojim se suočava ovaj algoritam je amplifikacija pisanja



Zadaci

- Proširiti konfiguracioni fajl, tako da sadrži maksimalan broj nivao LSM stabla
- Implementirati algoritam za kompakciju po želji — svi potrebni parametri za algoritam su ostavljeni studentima na izbor (npr. da li se kompakuju tabele po veličini ili broju) i treba da budu konfigurabilni
- **NAPOMENE:**
  - VODITI RAČUNA O MOGUĆNOSTI LANČANIH KOMPAKCIJA
  - KOMPAKCIJE NIJE POTREBNO IMPLEMENTIRATI U POZADINSKIM PROCESIMA

Dodatni zadaci*

- Omogućiti da korisnik kroz konfiguracioni fajl bira algoritam za kompakciju
- Implementirati *Size-tiered* algoritam za kompakciju — svi potrebni parametri za algoritam su ostavljeni studentima na izbor (npr. da li se kompakuju tabele po veličini ili broju) i treba da budu konfigurabilni
- Implementirati *Leveled* algoritam za kompkaciju — svi potrebni parametri za algoritam su ostavljeni studentima na izbor (npr. da li se kompakuju tabele po veličini ili broju) i treba da budu konfigurabilni
- **NAPOMENE:**
  - VODITI RAČUNA O MOGUĆNOSTI LANČANIH KOMPAKCIJA
  - KOMPAKCIJE NIJE POTREBNO IMPLEMENTIRATI U POZADINSKIM PROCESIMA
  - *OVI ZADACI NISU OBAVEZNI, ONI SU ZA STUDENTE KOJI ŽELE VIŠE :)