

# Napredni algoritmi i strukture podataka

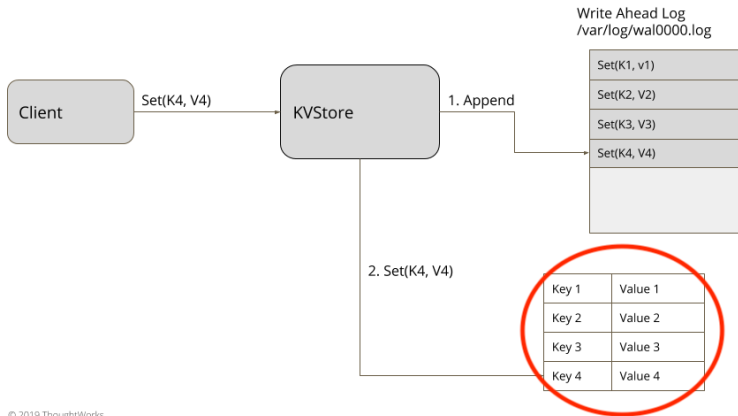
Memorijske tabele (Memtable), Eksternalizacija podešenja, Put zapisa (Write path)



**Univerzitet u Novom Sadu**  
**Fakultet Tehničkih Nauka**

## Write Ahead Log - podsećanje

- ▶ Kada se zapis upiše u neko skladište podataka, on se čuva na dva mesta i to po redosledu:
  1. Write Ahead Log (WAL)
  2. Memorijska struktura
- ▶ WAL deluje kao rezervna kopija na disku, za memorijsku strukturu tako što vodi evidenciju o svim operacijama koje su izvršene nad njom
- ▶ U slučaju ponovnog pokretanja sistema (restart), memoriska struktura se može u potpunosti oporaviti ponavljanjem operacija iz WAL-a
- ▶ Kada memorijska strukutra dostigne definisani kapacitet transformiše se u strukturu na disku, WAL se briše sa diska da bi se napravio prostor za novi WAL
- ▶ WAL se još naziv *Commit Log* u sistemima za skladištenje podataka



© 2019 ThoughtWorks

(Martin Fowler Write-Ahead Log <https://martinfowler.com/articles/patterns-of-distributed-systems/wal.html>)

# Problem 1

Zaposlili ste se u Google-u (idemoo), pravite nov sistem za skladištenje velike količine podataka, i od vas se očekuje da obezbedite sledeće osobine:

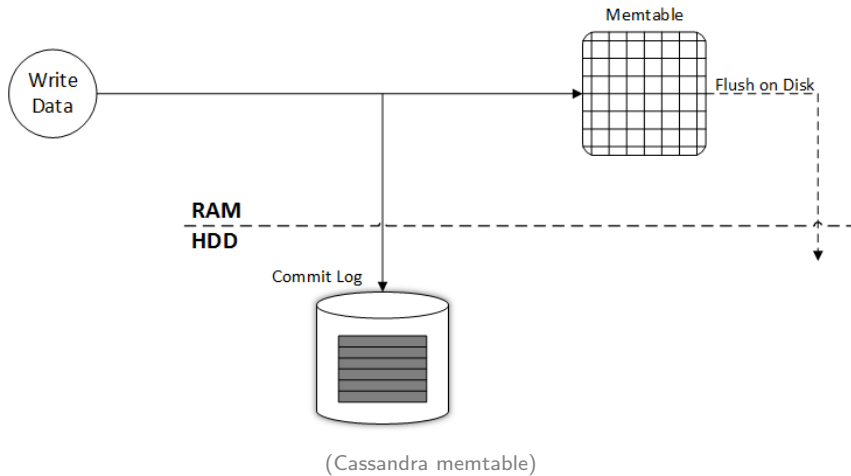
- ▶ Brz zapis podatka
- ▶ Brz odgovor klijentu da je zapis načinjen
- ▶ Obezbeiti brzo čitanje zapisa **AKO** je moguće
- ▶ Obezbediti što bržu pretragu podataka

ideje :) ?

## Memorijska tabela - ideja

- ▶ Ideja iza Memorijske tabele (Memtable) je relativno jednostavna — zapisati podatke u memoriju i čitati podatke iz memorije
- ▶ **AKO** se podaci nalaze u memoriji, sve operacije su relativno brže nego da su podaci **striktno** na disku
- ▶ Memorija je brza, memorija je super, memorija je kul, svi vole memoriju
- ▶ **ALI** nemamo beskonačno memorije (recite to matematičarima :))
- ▶ **ALI** taj problem ostavljamo za kasnije :)
- ▶ Memorija je aktivna dok je sistem aktivan

- ▶ Memorija je brza, memorija je super, memorija je kul
- ▶ **ALI** memorija nije sigurna :/
- ▶ Restart sistema i naših podataka više nema (objasnite to korisnicima:))
- ▶ Iz tog razloga nam treba snažna garancija trajnosti podataka
- ▶ Zato sistem komunicira sa WAL-om prvo, koji nam daje ove garancije, pa onda zapisuje u Memtable
- ▶ Ovaj princip se pokazuje jako korisno kod **write-heavy** problema
- ▶ Zapis se dešavaa brzo, i sistem može da pošalje odgovor klijentu brzo — život ide dalje



# Memorijska tabela

- ▶ WAL nam daje trajnost podataka
- ▶ Memtable nam daje brzinu zapisa (uvek) i čitanja (**AKO** su podaci u memoriji)
- ▶ Njihova saradnja je ključ uspeha, ali to nije skroz dovoljno uvek
- ▶ Memtable treba da ima neku definisanu strukturu, da bi mogli efikasno da je pretražujemo podatke



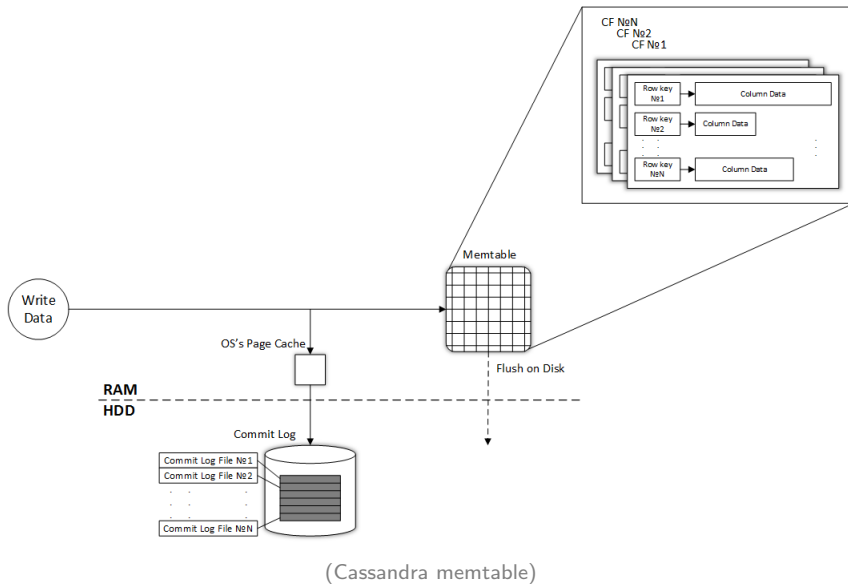
# Pitanje 1

Memtable treba da ima neku definisanu strukturu, da bi mogli efikasno da je pretražujemo podatke...

Kakvu strukturu da koristimo, ideje :) ?

## Memtable — struktura

- ▶ Pošto Memtable i WAL komuniciraju, možemo da koristimo strukturu WAL-a
- ▶ Naravno, ne trebaju nam svi segmenti WAL-a, neki od njih su nam bitni
- ▶ Dve najbitnije stvari su *Ključ* i *Vrednost*
- ▶ Ostale stvari WAL može i sam da doda, ne moramo čuvati te informacije u memoriji
- ▶ Nećemo ih čuvati, pre svega zato što su to *Meta* podaci koji samo troše resurse
- ▶ Podatke iz Memtable-a možemo da brišemo



## Pitanje 2

Treba (relativno) brzo da uradimo dodavanje, brisanje, pretragu itd. koju strukturu podataka da koristimo?

ideje :) ?

- ▶ Za strukturu podataka možemo da izaberemo razne opcije
- ▶ Razni sistemi za skladiženje podataka koriste razne strukture
- ▶ Ne postoji idealno rešenje
- ▶ Neki sistemi koriste stabla (Red-Black, AVL, B stablo, ...)
- ▶ Ali neki sistemi koriste nešto znatno prostije strukture za rad sa slučajnim osobinama
- ▶ Performanse su prilično dobre, a strukture se lako implementiraju

## Pitanje 3

Neki sistemi koriste nešto znatno prostije strukture za rad. Performanse su prilično dobre, a strukture se lako implementiraju

ideje :) ?

## Memorijska tabela — struktura podataka

- ▶ Jednostavna struktura koju smo radili, i koja se dosta koristi za Memtable je *SkipList*
- ▶ RocksDB i LevelDB na primer direkno koristi SkipList
- ▶ Izvod iz RocksDB dokumentacije
  - ▶ *Skiplist-based memtable provides general good performance to both read and write, random access and sequential scan.* (RocksDB Memtable Docs)
- ▶ Što se nas tiče, mi se možemo držati ove strukture podataka — učimo i ugledamo se na najbolje :)
- ▶ I **plus**, implementirali ste je na vežbama :D

## Memorijska tabela — zapis na disk

- ▶ Memtable se implementira kao struktura **fiksne** kapaciteta
- ▶ Setimo se segmenata i WAL-a
- ▶ Jedan segment može biti veličine kao i Memtable
- ▶ Memtable imaju granicu ili prag zapisa — **trashold**
- ▶ Kada se Memtable struktura popuni, prekorači se granica, ona se perzistira na disk — operacija **Flush**
- ▶ Flush operacija pravi se **SSTable** koja je neprimenljiva (više o tome naredno predavanje :))



- ▶ Trasholod ili kapacitet Memtable-a, je varijabilnog karaktera
- ▶ Na primer podrazumevana veličina Memtable-a kod LevelDB-a je oko 4MB (koristi SkipList)
- ▶ Izvod iz njihove dokumentacije:
  - ▶ *When the log file reaches a certain size (around 4 MB), its content is transferred to a new SST file and a new log file and memtable are initiated, the previous memtable is discarded.* (LevelDB docs)
- ▶ Iz ovoga možemo da zaključimo i veličinu segmenta koju LevelDB koristi

- ▶ Podrazumevane vrednosti zavise od nekoliko faktora
- ▶ Veličinu Memtable-a, možemo podešavati
- ▶ Granicu za zapis možemo podešavati
- ▶ Vrednosti ne treba **zakucavati!!** — jako loša praksa
- ▶ Omogućiti korisniku da može da menja shodno svojim potrebama i dostupnim resursima
- ▶ **ALI** omogućiti i podrazumevane vrednosti za obične korisnike ili one koji se upoznaju sa sistemom — biće vam zahvalni!

## Pitanje 4

Podrazumevane vrednosti zavise od nekoliko faktora...

ideje :) ?

## Memorijska tabela — rotacije

- ▶ Naš sistem može da ima nekoliko aktivnih Memtable instanci
- ▶ Ovo nije tako redak slučaj
- ▶ Ako imamo više instanci, onda možemo da radimo rotiranje instanci
- ▶ Kada se jedna instanca Memtable-a popuni podacima, nju serijalizujemo na disk u pozadini
- ▶ **AKO** hoćemo da sistem i dalje bude sposoban da prihvata zapise, možemo da aktiviramo drugu instancu

- ▶ Druga instanca postaje aktivna da prihvata zapise, dok se prva zapisuje na disk
- ▶ I tako rotiramo tabele, da bi sistem bio *uvek* dostuapn za upise
- ▶ U početku trebaće nam više resursa da instanciramo dve tabele **ALI** zato kasnije, biće sposobniji da prihvatimo nove zapise brže!!
- ▶ U modernom software-u, **to je ono što se traži**
- ▶ Zato ne optimizujte stvari pre vremena, zato što:
  - ▶ *Premature optimization is the root of all evil.* (Sir Tony Hoare)

## Pitanje 5

Vrednosti ne treba **zakucavati**, već omogućiti korisniku da može da menja shodno svojim potrebama i dostupnim resursima

Kako ovo da postignemo, ideje :) ?

## Eksternalizacija podešenja

- ▶ Moderan software je prilično komplikovan (ne govorimo o korisničkim sajtovima :))
- ▶ Ako ga napravimo tako da korisnici mogu da ga koriste samo na jedan način zapašćemo brzo u probleme
- ▶ Nemaju svi korisnici i organizacije mogućnost (ili znanja) da koriste software sa dosta npr. resursa
- ▶ Možda se njihove mogućnosti menjaju vremenom
- ▶ Možda hoće da optimizuju neke delove, zarad boljeg rada sistema

- ▶ Jednostavno rešenje za ove probleme je prosta — omogućiti konfiguraciju sistema van samog sistema
- ▶ To možemo da uradimo na (bar) dva načina:
  1. Kroz programski kod — interno
  2. Kroz spoljne elemente — eksterno
- ▶ Interna konfiguracija zahteva (često) izmene koda, što često nismo u mogućnosti (zatvoren kod, rekompajliranje biblioteke, ....) — dobro za podrazumevane vrednosti
- ▶ Eksterna konfiguracija zahteva (često) restart sistema da bi izmene bile izvršene — dobro za lakše podešavanje



## Pitanje 6

Ako probamo da eksterno konfiguriramo sistem (npr. Memtable), gde bi pisali konfiguraciju, i u kom formatu?

ideje :) ?

- ▶ Konfiguracioni elementi se obično nalaze na istom mestu kao i kompajliran element koji pokrećete
- ▶ Možemo da čuvamo u specifičnim mestima (etc folder, ENV varijable operativnog sistema)
- ▶ Često se konfiguracija zapisuje u *konfiguracionim fajlovima*
- ▶ Kao format možemo da koristimo bilo koji format koji vam je poznat (JSON, YAML, TOML, ...)
- ▶ Kroz jedan konfiguracioni fajl možemo da konfigurišemo više delova sistema (npr. WAL i Memtable)
- ▶ Neki konfiguracioni elementi mogu i da se preklapaju (npr. veličina Memtable i veličina segmenta WAL-a)

## Eksternalizacija podešenja — podrazumevane vrednosti

- ▶ Kada pravimo sistme koji se konfiguriše kroz eksterne fajlove, trebamo obezbediti podrazumevane vrednosti — **default**
- ▶ Ovo možemo da uradimo na dva mesta, da se osiguramo i zaštitimo od potencijalnih problema
  1. Obezbediti fajl sa default vrednostima — isti fajl za konfiguraciju samo već popunjen vrednostima
  2. **AKO** takav fajl ne postoji, obezbediti da kroz kod postoje default opcije koje program može da iskoristi
- ▶ Na ovaj način imamo redudanciju, i sistem nam je stabilniji
- ▶ Ovo nije obaveza, ali je generlano lepa praksa

## Pitanje 7

Ako probamo da eksterno konfiguriramo sistem (npr. Memtable), gde čuvati konfiguraciju?

ideje :) ?

## Eksternalizacija podešenja — mesto čuvanja

- ▶ Gde čuvati konfiguracione fajlove je nezgodno pitanje
- ▶ Ono nekada može da zavisi od operativnog sistema koji koristimo
- ▶ UNIX-like operativni sistemi imaju specifičan folder za ove namene koji možemo da iskoristimo (možda i windows)
- ▶ Aplikacija porazumevano može da traži konfiguraciju tamo
- ▶ Možemo da ih čuvamo na istom mestu odakle se aplikacija pokreće — tj. gde je binary koji pokrećemo
- ▶ Možemo na nekim egzotičnim mestima — nije baš preporuka

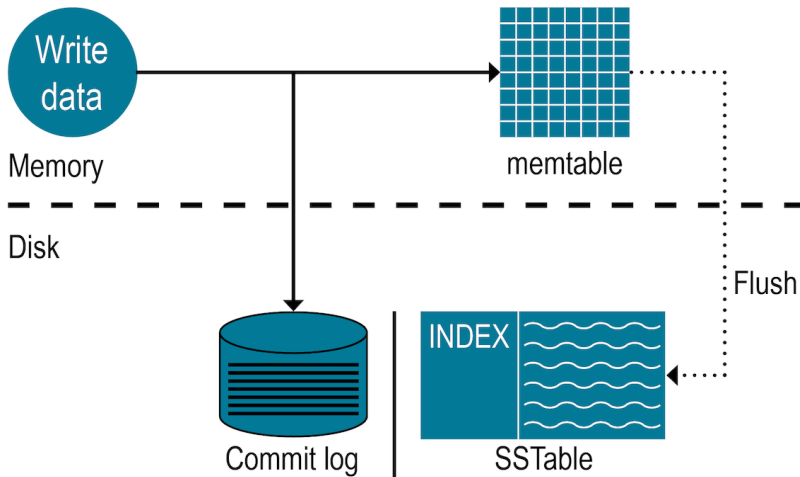
## Putanja zapisa - write path

- ▶ Videli smo da je Memtable fensi naziv za strukturu u memoriji koja se popunjava podacima
- ▶ Kada se popuni definisani kapacitet, podaci se zapisuju na disk i formira se SSTable
- ▶ Memtable je promenljiva struktura, možete raditi brisanje i izmene
- ▶ SSTable je nepromenljiva struktura, nema izmena i brisanja
- ▶ Da bi obezbedili trajnost Memtable-a, imamo WAL
- ▶ WAL je isto tako neprimenljiv i štiti Memtable
- ▶ SSTable su podaci perzistirani na disk

## Pitanje 8

U kakvom su odnosu WAL, Memtable i SSTable...?

ideje :) ?



(Cassandra write path)



- ▶ Svi prethodno definisani elementi čine put zapisa — **write path**
- ▶ Svaki element ima jednu i jedinstvenu ulogu
  - ▶ *Do One Thing And Do It Well.* (Unix philosophy)
- ▶ Kompoziciom tih elemenata dobijamo jednostavan sistem
- ▶ Koji često rešava kompleksne probleme
- ▶ Oni nam omogućavaju da imamo brz zapis podataka
- ▶ Ali isto tako i trajnost podataka

# Algoritam

1. Korisnik je poslao zahtev — nekakvu operaciju (dodavanje, čitanje, izmena, brisanje — CRUD)
2. Podatak se prvo zapisuje u **WAL**
3. Kada WAL potvrdi zapis, podatak se zapisuje u **Memtable**
4. Koraci **(2)** i **(3)** se ponavljaju dokle god ima mesta u Memtable-u
5. Ako je kapacitet Memtable-a popunjen, Memtable sortira parove ključ-vrednost
6. Sortiraten vrednosti se zapisuje na disk formirajući **SSTable**
7. Možemo isprazniti **Memtable** ili napravitu nov, a prethodni uništiti ili rotirati

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
4. O ovome više sledeći put
5. **Za vaš projekat, ovo je algoritam koji trebate da pratite :)**
6. **Ovim ste uradili polovina posla celokupnog projekta**

## Dodatni materijali

- ▶ RocksDB Memtable Docs
- ▶ 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 :) ?