

Napredni algoritmi i strukture podataka

Write Ahead Log - nastavak, Memory mapped file, Transkacije



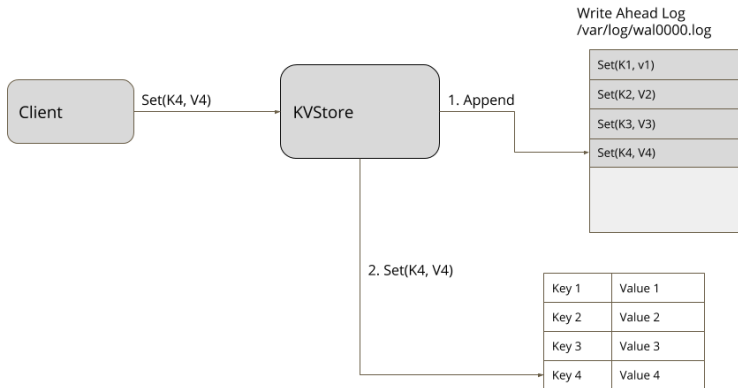
Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Kratka rekapitulacija

- ▶ WAL deluje kao rezervna kopija na disku za memorijsku strukturu tako što vodi evidenciju o svim operacijama koje su se desile
- ▶ U slučaju ponovnog pokretanja sistema, memoriska struktura se može u potpunosti oporaviti/rekonstruisati ponavljanjem operacija iz WAL-a
- ▶ WAL koristi isključivo sekvencijalne I/O operacije prilikom zapisa podataka na disk
- ▶ WAL kao struktura podatka, direktno se oslanja na strukturu zasnovanu na log-u

- ▶ Podatke u WAL možemo da dodamo
- ▶ *In place* izmena nije moguća
- ▶ Izmjena ili brisanje nekog podatka, rezultuje novim zapisom u WAL
- ▶ WAL je *append-only* struktura
- ▶ Podaci se u WAL dodaju na kraj strukture

- ▶ Kada čitamo podatke, možemo da čitamo od početka, ili da skeniramo od nekog dela
- ▶ WAL prati vremenski tok rada sa podacima
- ▶ Noviji zapisi su na kraju WAL-a
- ▶ Stariji zapisi su na početku WAL-a
- ▶ Uvek možemo da se vratimo u vremenu i da vidimo tok operacija
- ▶ Podaci se čuvaju u memorijskoj strukturi, ali se prvo sačuvaju u WAL zbog trajnosti



© 2019 ThoughtWorks

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

Format koji ćemo mi koristiti biće sličan RocksDB-u koji smo videli prošli put, ali malo uprošćen zbog jednostavnosti rada i naših potreba

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

Sadržaj WAL-a je u binarnom formatu (binarna datoteka)...

Zašto pišemo tako, zašto prosto ne serijalizujemo podatke/strukture, ideje :) ?

Write Ahead Log format - nastavak

- ▶ Podaci se čuvaju u binarnom obliku
- ▶ Za čitanje, potrebno je ispravno prolaziti kroz binarni fajl
- ▶ Čitati podatke sa njihovih pozicija shodno tipu podatka koji je na toj poziciji zapisan
- ▶ Zato unapred moramo znati strukturu WAL-a, da bi ispravno čitali podatke!
- ▶ **Za vaš projekat, ovo je format koji će biti korišćen**

- ▶ WAL zapisuje svoj sadržaj na hard disk
- ▶ Moguće je da se desi oštećenje zapisa
- ▶ Oštećenje može da nastane od strane korisnika — maliciozno ili slučajno
- ▶ Ali isto tako i usled lošeg zapisa, ili hardverskog problema
- ▶ Ove stvari moramo da uračunamo kada pravimo WAL sistem — moramo biti svesni ovog problema
- ▶ Moramo dizajnirati sistem tako da odgovori na ovaj problem

Moramo dizajnirati sistem tako da odgovori na ovaj problem...

Kako ovo rešiti, ideje :) ?

Write Ahead Log - Otkrivanje grešaka

- ▶ Postoji nekoliko načina kako ovo možemo uraditi
- ▶ Jedan standardan način je da nekako označimo sadržaj prilikom zapisa u WAL — checksum
- ▶ Proveriti jedinstvenost zapisa na promene prilikom čitanja podatka
- ▶ Ako oznaka nije ista, naš zapis nije više validan — došlo je do problema
- ▶ Ako jeste, možemo nastaviti sa radom
- ▶ Postupak ponoviti za svaki zapis unutar WAL-a

Da bi se ovo rešilo, zapisi u WAL-u se obično pišu sa **cyclic redundancy check (CRC)** zapisom na početku

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

- ▶ CRC koristimo kao **error-detecting** mehanizam za otkrivanje promena u podacima
- ▶ CRC je *hash* funkcija koja detektuje promene nad podacima
- ▶ Kao tip kontrolnog zbira (checksum), CRC proizvodi skup podataka fiksne dužine na osnovu izvorne datoteke ili većeg skupa podataka
- ▶ CRC se zasniva na binarnoj podeli i naziva se i *kontrolna suma polinoma koda*
- ▶ Ovo možemo koristiti kao mehanizam potvrde da li je bilo izmena/oštećenja kada se podaci pročitaju
- ▶ Ako je došlo do promene, taj podataka više nije validan

CRC obično ide na početak zapisa svakog unosa...

Zašto :)?

Write Ahead Log - I/O

- ▶ Zapis svakog elementa na disk **odmah** daje snažnu garanciju trajnosti (što je glavna svrha posedovanja WAL-a)
- ▶ Ovo **ozbiljno** ograničava performanse i može brzo da postane usko grlo sistema
- ▶ Ako je zapis odložen ili se vrši asinhrono, to poboljšava performanse — **ALI**
- ▶ Postoji **povećan** rizik od gubitka zapisa ako se čvor sruši pre nego što se unosi zabeleže
- ▶ Većina implementacija koristi tehnike kao što je *batch*, da bi se ograničio uticaj operacije zapisivanja

Možemo li ovo poboljšati, ideje :) ?

Memory mapped file

- ▶ *Memory mapped file* nam omogućava pristup datoteci, kao da je ona učitana u memoriju u potpunosti
- ▶ To je najjednostavniji pristup datotekama i često ga koriste dizajneri baza podataka, ali i ozbiljnih aplikacija (van Novog Sada naravno :))
- ▶ *Memory mapped file* pokušava da odgovori na pitanje: **kako postupati sa podacima na disku, koji su veći od raspoložive memorije?**
- ▶ Količina memorije koja nam je dostupna, je znatno manja od količine diska koji nam je na raspolaganju

- ▶ Cena isto tako se znatno razlikuje, pogotovo u cloud-u
- ▶ Možemo da probamo da “ubrzamo” disk **AKO** radimo stvari pametnije — ipak smo inženjeri :)
- ▶ Da bi odgovorili na prethodno pitanje: **kako postupati sa podacima na disku, koji su veći od raspoložive memorije?**
- ▶ Možemo da se vratimo u rane godine računarstva (1960s), pošto ovo nije nov problem za rešavanje
- ▶ 1962 godine, grupa iz Mančestera (Atlas), razvila je ideju *virtulne memorije* koju možemo da iskoristimo (i koju drugi pametni inženjeri često koriste)

Virtuelna memorija - ukratko

- ▶ Nećemo ulaziti duboko u to kako funkcioniše virtuelna memorija — prevazilazi granice ovog kursa, i učićete na drugim predmetima :)
- ▶ Virtuelna memorija daje pokrenutom programu iluziju da ima dovoljno memorije, uprkos činjenici da je nema
- ▶ Imajte na umu, da kada program pristupa memoriji, on možda pristupa virtuelnoj memoriji
- ▶ A možda podaci kojima program pokušava da pristupi zapravo nisu u memoriji, već u sekundarnom skladištu (npr. hard disk) — *virtuelnom memoriji*

- ▶ Operativni sistem (OS) će odvojiti poseban deo sekundarnog skladišta **kao virtuelnu memoriju**
- ▶ OS će prebacivati blokove sa diska, u memoriju na blokove koji nisu dugo korišćeni, i vraćati nazad na disk po potrebi — *Swap*
- ▶ OS će to raditi (relativno) brzo i efikasno — **iskoristimo to!**
- ▶ Kao primer pogledajte kako OS upravlja aplikacijama koje su pokrenute, ali dugo nisu korišćene
- ▶ Ovo je jedna od raznih tehnika razvijenih zarad boljeg iskorišćenja i podele resursa u situacijama gde postoji više aplikacija ili korisnika

UNIX sistemski poziv mmap

- ▶ mmap je veoma koristan alat za rad sa I/O
- ▶ mmap je sistemski poziv, što znači da brigu oko sinhronizacije i Swap-a prepuštamo onome ko to radi dobro i efikasno — OS
- ▶ Izbegava stvaranje dodatne kopije bafera u memoriji (za razliku od Standardnog IO-a)
- ▶ Iz perspektive programera, čitanje pomoću mmap-irane datoteke izgleda kao normalna operacija pokazivača i ne uključuje dodatne pozive
- ▶ Dosta se koristi u dizajnu baza podataka

*Pretpostavka je da radi isto i na Windows-u, ili Windows ima svoj neki ekvivalent (kao npr. ceo svet UUID, Windows GUID)

mmap nedostaci

- ▶ Nedostaci mmap-a koji se danas pominju manje su relevantni uz savremeni hardver
- ▶ mmap stvara dodatani *overhead* na strukture podataka *kernel-a* potrebnih za upravljanje memorijskim mapama
- ▶ U današnje vreme i veličinama memorije, ovaj argument ne igra bitnu ulogu
- ▶ Ograničenje veličine mmap datoteke u memoriji
- ▶ Većinu vremena, *kernel* je *memory friendly*, a 64-bitne arhitekture omogućavaju mapiranje većih datoteka

Transakcije

- ▶ O transakcijama samo ukratko, prevazilazie granice ovog kursa, i radićete na drugim predmetima :)
- ▶ Ali su nam važne da bi razumeli granice zapisa
- ▶ Transakcija simbolizuje jedinicu rada koja se obavlja u okviru sistema za upravljanje bazom podataka (ili sličnog sistema)
- ▶ Transakcije su nezavisne
- ▶ Transakcija ne sme da se izvrši polovično
- ▶ **Transakcija mora da se završi u celosti, ili se ona odbacuje**

Transakcije i WAL

- ▶ Kada zapisujemo skup informacija u WAL, bitno je da znamo da li su povezane ili ne
- ▶ Ako jesu moramo ih tretirati zajedno
- ▶ Ovo je jako bitno, pogotovo ako moramo da popunimo memorijske strukture informacijama iz WAL-a
- ▶ Ako ne bi popunili informacije ispravno, imali bi problem — nekonzistenciju
- ▶ To znači da ono što smo zapisali, i ono što smo pročitali nije isto — to ne sme da se desi
- ▶ Zamislite da ubacite novce u bankomat, i pogledate stanje a novca ima manje nego što ste ubacili :)

Kada zapisujemo skup informacija u WAL, bitno je da znamo da li su povezane ili ne...

Kako ovo postići, ideje :) ?

- ▶ Jednostavna ideja, ali funkcionalna je da označimo operacije koje idu zajedno
- ▶ Zapisi koji čine jednu transakciju možemo da počnemo sa specijalnim simbolom/blokom < START >
- ▶ Zapisi koji čine jednu transakciju možemo da završimo sa specijalnim simbolom/blokom < COMMIT >
- ▶ Svakoj transakciji se dodeljuje jedinstveni identifikator — razne strategije su moguće
- ▶ Kad rekonstruišemo zapis iz WAL-a, rekonstruišemo sve informacije zajedno od < START > do < COMMIT > za svaku transakciju

Dodatni materijali

- ▶ Write-Ahead Log for Dummies (nije uvreda :))
- ▶ Write Ahead Log Martin Fowler
- ▶ Database Internals: A Deep Dive into How Distributed Data Systems Work
- ▶ Read, write and space amplification
- ▶ ARIES/NT: A Recovery Method Based on Write-Ahead Logging for Nested Transactions
- ▶ Linux mmap OS call
- ▶ Exploring mmap using go

Write Ahead Log - Pitanja

Pitanja :) ?