Napredni algoritmi i strukture podataka

Segmentirani log, Blokovi podataka, Brisanje delova loga, Izmene sa više klijenata

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Kratka rekapitulacija

- Rezervna kopija na disku za memorijsku strukturu vodeći evidenciju o operacijama koje su se desile
- U slučaju otkaza, memoriska struktura se može oporaviti/rekonstruisati ponavljanjem operacija iz WAL-a
- Koristi isključivo sekvencijalne I/O operacije
- WAL je *append-only* struktura
  - 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

Kratka rekapitulacija

- WAL koristi baferisani I/O, da bi sprečio veliki broj operacija ka disku
  - Koristiti sistemske pozive (slabija kontrola) – mmap
  - Koristiti buffer pool (veća kontrola) – rad sa blokovima podataka
- WAL čuva informacije o transkacijama koje su se desile
- WAL zapisuje svoj sadržaj na hard disk – potencijalno oštećenje
- CRC je jedan standardan način da označimo sadržaj pri zapisu — *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
- Naše odluke, i sama upotreba sistema počinu da stvaraju **nove probleme** koje **moramo** rešiti

Write Ahead Log - Problem 1

Implementirali ste WAL mehanizam, ali ste ustanovili da on kreće da raste nekontrolisano, i kao takav počinje da bude težak za ikakvu obradu – pojedinačna WAL datoteka počinje da bude usko grlo sistema.

Možemo li ovo poboljšati, ideje :) ?

Segmentirani Log

- WAL radi svoj posao prilično **lepo** – radi ono što mu je **namena i ništa više**
- To dovodi do **dodatnih** problema – jedna datoteka WAL-a može da **raste nekontrolisano** i postane **usko grlo** sistema
- Možemo **brisati starije** informacije, to je jedno rešenje
- Starije operacije nam više **nisu potrebne**
- **ALI** ta operacija čišćenja na jednoj ogromnoj datoteci nije **nužno lako** izvesti
- Može **blokirati** sistem (npr. GC u Javi)

- Možemo nekako da probamo da **podelimo** WAL datoteku
- Jedan WAL možemo podeliti u više **segmenata**
- Za svaki segment možemo da specificiramo **veličinu**
- Datoteke segmenata se skladište na disk nakon nekog **definisanog praga** - buffer pool
- U memoriji možemo da čuvamo samo (sliku/kopiju) **zadnji** segment
- Ovo su neke od standardnih ideja kako da rešimo problem — *divide and conquer* princip

Gde da čuvamo i kako da delimo segmente, ideje :) ?

Segmentacija loga - lokacija segmenata

- Segmente možemo da čuvamo na **različitim** mestima – **nije** baš praktično
- Segmente možemo da čuvamo na jednom mestu – može biti praktično
  - WAL treba da zna **samo** putanju do direktorijuma koji čuva segmente
  - Ovaj direktorijum obično nosi naziv *wal* — gle čuda :)
  - Ova ideja se može pokazati korisnom **kasnije**...
- Kada se WAL pokrene, on treba da preskenira *wal* direktrijum, i da **pokupi** segmenate
  - Segmenti **ne moraju** biti učitani u **memoriju** odmah – lazy mehanizam
  - Iz praktičnih razloga možemo da učitamo samo **poslednji** segment
  - Ideja iza ovoga jeste brži pristup — pretpostavka: možda su podaci koje tražimo **relativno skoro** zabeleženi (imamo sreće)

- Kada se WAL pokrene, on treba da preskenira *wal* direktrijum, i da **pokupi** segmenate
- Postoji nekoliko strategija
- Jedna opcija je da pokupimo **lokacije (putanje)** do segmenata – ne i same segmente
- Ovo radimo da bi mogli lakše da im pristupimo **kasnije** – nema potrebe da ponovo skeniramo *wal* direktorijum
- **ALI** uvek treba da znamo **redosled** segmenata!
- Ovo nam pomaže da znamo šta smo **pročitali** kada radimo pretragu – znamo poziciju

Kako ovo postići, ideje :) ?

Segmentacija loga - imenovanje segmenata

- Segmentacijom WAL-a moramo obezbediti **jednostavan** način za **mapiranje** offset-a WAL-a (ili rednih brojeva) u segmente
- Ovo se može uraditi na nekoliko načina
- Na primer, ime svakog segmenta se dobija spajanjem unapred poznatog prefiksa (npr wal) i offeta (ili rednog broja segmenta) i dodavanjem ekstenzije *log*
  - npr: wal_0001.log, wal_0002.log, itd.
- Ako je potrebno da pročitamo neki segment, moramo ga naći po **identifikatoru** na disku, i pretražiti sapise u njemu

Šta ako neko namerno/slučajno promeni ime datoteke, ideje :) ?

- Na primer RocksDB koristi **dodatan deo broj loga**

```
+---------+-----------+-----------+--- ... ---+
|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) | Size (2B) | Type (1B) | Log number (4B)| Payload   |
+---------+-----------+-----------+----------------+--- ... ---+
Same as above, with the addition of
Log number = 32bit log file number, so that we can distinguish between
records written by the most recent log writer vs a previous one.
```

Učitavanje segmenata

- WAL obično ima **specijalizovan** folder koji čuva **sve** segmente
- Kada se WAL pokreće, potrebno je da **preskeniramo** taj folder i da vidimo **koliko** segmenata ima
- Nakon toga, možemo da **formiramo** strukturu unutar WAL-a sa **putanjama** i offset-ima za svaki **segmet**, radi lakšeg kasnijeg rada i pronalaska
- U memoriju WAL-a, možemo učitati **samo poslednji** segment
- Prvi/Poslednji segment možemo i **dodatno** markirati ako želimo – *_START, _END*
- Dobijati segmente kao
  - Koristeći samo naziv datoteke
  - Iskoristiti zapise unutar blokova da zaključimo koji blok je **pre kog** – sekvencijalni zapisi

Koji pristup, ideje :) ?

Write Ahead Log - Problem 2

Kako, ili po čemu podeliti WAL na segmente, ideje :) ?

Podela na segmente

- Našu WAL datoteku možemo **podeliti** po **velikom** broju kriterijuma:
  - Možemo koristiti broj zapisa
  - Istu veličinu segmenta
  - Različitu veličinu segmenta
  - Grupisati po transakcijama
  - Vremenski
  - ...
- Kada radimo podelu, treba da zaključimo šta nam **pomaže** – razmišljati i planirati
- Treba da vidimo šta nam omogućava **lepu saradnju** sa ostakom sistema
- Šta nam donosi **očekivane performanse** u sistemu – **prediktabilnost, jako bitna stvar**

Onda, kako to uraditi, ideje :) ?

Blokovska organizacija podataka

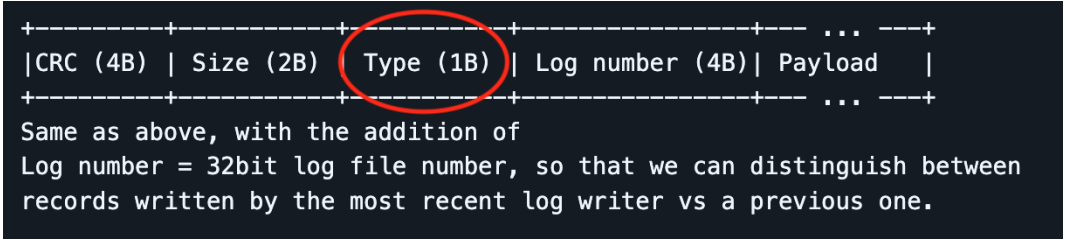
- Naš pojednostavljeni format, **ne može** da nam odgovori na sve probleme...
- Prilično **trošimo vremena i resursa**, stvarmo veliku defragmentaciju
- **Nećemo** pomoći buffer pool-u a onda i OS-u da lepo rade svoj posao
- Zato **BLOKOVI PODATAKA** – **ALI**, moramo biti obazrivi
- Prednosti blokova podataka:
  - Svi su jednaki, svi su uniformni
  - Lepo se mapira na buffer pool
  - OS može lepo da mapira na blokove diska
  - Očekivane performanse
- Desi se i **neka** mana koju **moramo** rešiti :) – **ništa nije savršeno!**

Struktura

- Blok je veličine $\mathrm{kBlockSize} = 32\mathrm{kb}$
  - Ako zapis ne može da stane u preostali prostor, prostor popuniti **praznim (null) podacima** – padding
  - Ako je zapisa veći nego veličina bloka, **moramo voditi evidenciju koji deo (chunk) smo zapisali**
  - Veličina bloka može biti i veća, **ALI** umnožak $\mathrm{kBlockSize}$, $\mathrm{n} * \mathrm{kBlockSize}, \mathrm{n} > 0$

Oznake delova bloka

- Zapis može da popuni ceo blok, ali ne mora – **Označiti nekako**
- Imamo **četiri specifične** oznake: *FULL*, *FIRST*, *MIDDLE*, *LAST*
  - *FULL* zapis popunjavava sadržaj **celog korisničkog zapisa**
  - *FIRST*, *MIDDLE*, *LAST* su tipovi koji se koriste za korisničke zapise koji su **podeljeni na više fragmenata** – zbog granica bloka
    - *FIRST* je tip **prvog** fragmenta korisničkog zapisa
    - *LAST* je tip **poslednjeg** fragmenta korisničkog zapisa
    - *MIDDLE* je tip **svih medju** fragmenata korisničkog zapisa



Napomena

U ostatku gradiva (do kraja semestra), ako nije drugačije rečeno, podrazumevamo da sve operacije nad **zapisima** izvodimo koristeći **blokovski pristup**

Brisanje delova loga

- Prethodnom idejom **rešili** smo problem performasni našeg WAL-a
- **ALI**, nismo rešili **sve** probleme, ima ih još :)
- Velika datoteka **neće** više biti uskor grlo sistema
- Ali sada možemo dobiti **puno** malih datoteka na disku
- Ako to ne rešimo, imaćemo **problem** da će male datoteke zauzeti **sav** disk!
- Dodatno, male datotke mogu **dodatno** da defragmentišu disk – dodatan problem
- Rešimo jedan problem, pojavi se drugi :(

Kako ovo rešiti, ideje :) ?

Low-Water Mark

- Treba nam mehanizam koji će reći WAL mašineriji koji deo segmenata je **bezbedno obrisati**
- WAL zna putanju do *wal* direktorijuma, odakle da briše segmente
  - putanju do *wal* direktorijuma, odakle da briše segmente
  - putanju do **svakog** segmenta, brzo OS može da obriše stvari
- Prethodne **odluke** se pokazaju kao korisne :)
- *Low-Water Mark* ideja daje najniži indeks ili **low water mark**, pre koga se segmeti **mogu obrisati**
- Kratko rečeno, to je indeks u WAL-u **unapred definisan** (moguće je i adaptivno), koji pokazuje koji deo WAL-a može da se obriše
- Uglavom možemo obrisati **sve osim poslednjeg** segmenta

- Ako ovaj mehanizam radi u isto vreme kada i WAL, može doći do **dodatnog** problema – gle čuda :)
- Zaustavićemo WAL mašineriju da bi **obrisali segmetne** (Java GC)
- Ovaj mehanizam se uglavom pokreće kao **nezavisan proces** (ili nit više o tome naredni semestar, valjda :)))
- Dakle, potreban je mehanizam koji **neće zaustaviti** rad WAL-a, ali već koji će u pozadini **obrisati nepotrebne segmente**
- U velikim sistemima baza podatka, ovo može da se radi kada se dešava *snapshot sistema*, zarad dodatne sigurnosti od greške i otkaza

Izmene sa više klijenata

- Do sada smo WAL posmatrali samo kao sistem koji ima **jednog** klijenta
- U sistemima baza podataka to je (uglavom) slučaj — baza je klijent za WAL
- Ali WAL se može koristiti i za **druge** aplikacije
- Sistemi zasnovani na WAL-u i log-u su **jako popularni** u zadnje vreme u velikim sistemima (Big Data, Cloud)
- Tada možemo imati **više** korisnika koji nešto pišu/čitaju
  - Ako ne vodimo računa, podaci mogu biti nekonzistentni
  - Jedan klijent može da obriše ili promeni informacije drugog
  - WAL može da se sačuva informacije u pravom redosledu, ali ne mora
  - ...

Write Ahead Log - Problem 3

Kako da dodajemo ili menjamo informacije u WAL-u, a da inforamcije i dalje budu konzistentne?

Kako ovo rešiti, ideje :) ?

Serijalizacija posla

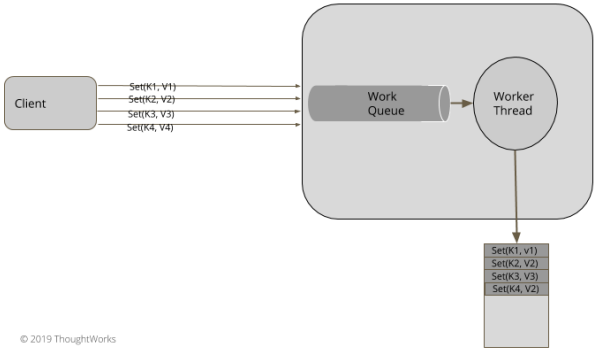
- Rešenje je **relativno** jednostavno
- Možemo napraviti **red čekanja**, i u njega dodavati poslove
- WAL će povremeno kupiti inforamcije iz ovog reda i upisivati ispravno kako su se oni dešavali
- Ovaj proces se još naziva i *serijalizacija* posla ili red čekanja
- Prednost ove ideje je što klijent brže dobija odgovor nazad, i može da nastavi sa poslom
- **ALI** uvek treba razmisliti o trajnosti podataka, da li i taj red treba da se perzistira na disk
- Ovo naravno nije **jedini** način, postoje razni drugi ali je za sada jednostavan :)

Write Ahead Log - Problem 4

Uvek treba razmisliti o trajnosti podataka, da li i taj red treba da se perzistira na disk?

Ideje :) ?

Kratak odgovor bi bio, zavisi od tipa aplikacije i okruženja u kom se izvršava :)



(Martin Fowler Singular Update Queue

https://martinfowler.com/articles/patterns-of-distributed-systems/singular-update-queue.html)

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

Pročitati za narednu sedmicu

- Merkle tree original paper
- Merkle tree easier paper
- Amazon Dynamo db paper