Ključ - vrijednost baze podataka

Hristina Adamović, Nada Zarić
Softversko Inženjerstvo i Informacione Tehnologije, Univerzitet u Novom Sadu, Fakultet tehničkih nauka
adamovic.sv32.2020@uns.ac.rs, zaric.sv63.2020@uns.ac.rs

I. PROBLEM

Tradicionalne relacione baze podataka dizajnirane su da rade isključivo sa strukturiranim podacima prema određenoj šemi ili modelu. Dugo vremena su se održale kao vodeći način skladištenja podataka zbog očuvanja integriteta podataka. Početkom 21. vijeka došlo je do naglog porasta količine podataka generisanih putem interneta i digitalnih uređaja [1]. Ovim su prevaziđene sposobnosti tradicionalnih tehnologija za upravljanje podacima. Dodatno, primjena tabelarnih struktura može dovesti do redundantnosti, dupliciranja podataka i problema sa performansama pri skaliranju [2].

NoSQL baze podataka (*Non-SQL*) mogu riješiti spomenute probleme. Motivacije za ovaj pristup uključuju jednostavnost dizajna, lakše horizontalno skaliranje [1], precizniju kontrolu dostupnosti i smanjenje nesklada između objektno-relacionog modela podataka [3]. Tokom vremena, razvile su se četiri glavne vrste NoSQL baza podataka [4]: dokument baze podataka (*document database*), ključ-vrijednost baze podataka (*key-value database*), grafovske baze podataka (*graph database*) i skladište širokih kolona (*wide-column stores*).

Fokus ovoga rada je na implementaciji ključ-vrijednost baze podataka. Ključ-vrijednost baze podataka koriste se u situacijama kada je potrebno brzo pronalaženje entiteta po ključu [5]. Svaki ključ ima pridruženu vrijednost koja može imati različite strukture.

Rješenje mogu koristiti programeri koji rade na projektima kao što su:

- IoT (*Internet of Things*) sistemi [6] - u ovakvim sistemima postoji veliki broj uređaja i senzora koji generišu velike količine podataka.
- Sistemi za keširanje podataka - podaci kojima se često pristupa mogu se keširati u ključ-vrijednost bazu kako bi se smanjilo vrijeme pristupa tim podacima.
- Veb aplikacije (društvene mreže, e-trgovina, blogovi) - za pohranu sesijskih podataka.

II. TEORIJSKE OSNOVE

Potpoglavlje II.A opisuje domen problema relacionalnih baza podataka. Potpoglavlje II.B definiše zahtjeve koje rješenje treba da ispuni. Potpoglavlje II.C opisuje ključne pojmove neophodne za razumijevanje rješenja ovog problema.

A. Domen problema relacione baze podataka

Relacione baze podataka se suočavaju sa dva glavna problema:

- Podaci su strogo strukturirani. Struktura podataka se mora unaprijed definisati, što može biti ograničavajuće u situacijama gdje se zahtjevi često mijenjaju [2].
- Neefikasno rukovanje velikim količinama podataka. Sa porastom količine podataka, upiti bazi podataka postaju sporiji i memorijski zahtjevniji [2].

B. Zahtjevi koje ključ-vrijednost baza podataka treba da ispuni

Ključ-vrijednost baza podataka skladišti podatke u obliku ključ-vrijednost, čime se postiže fleksibilnost i skalabilnost [5]. Problem rukovanja velikim količinama podataka, rješava se upotrebom probabilističkih struktura podataka koje zahtevaju manje resursa u poređenju sa deterministčkim strukturama koje se koriste u relacionim bazama podataka. Dodatno, ključ-vrijednost baza podataka koristi strukture smještene u radnoj memoriji, što omogućava brži pristup podacima nego kada se pristupa samo disku.

C. Algoritmi i strukture podataka potrebni za rješenje problema

Da bi se realizovalo navedeno rješenje potrebno je implementirati iduće algoritme i strukture podataka:

- *Token Bucket* [9] – algoritam se sastoji od spremnika koji sadrži određeni broj tokena koji se periodično obnavljaju. Svaki token predstavlja jedinicu dozvoljenog protoka podataka. Kada paket podataka stigne, provjerava se ima li dovoljno tokena u spremniku. Ako ima dovoljno tokena, paket se šalje i oduzima odgovarajući broj tokena. Ako nema dovoljno tokena, paket se odgodi ili odbaci.
- *Least Recently Used* [7] – algoritam za keširanje. Uklanja ili zamjenjuje podatke koji najduže nisu korišteni u keš memoriji.
- *Write ahead log* (WAL) [10] – ova struktura podataka se čuva na disku. Služi kao sigurnosna kopija stanja sistema. Osigurava da sve akcije i komande budu sačuvane, kako bi se moglo obnoviti stanje sistema u slučaju otkaza.
- *SSTable* [13] – ova struktura podataka se čuva na disku i sastoji se iz tri dijela:
    - *Summary File* – sadrži informacije o rasponima ključeva i upućuje na *Index File*

[1] Horizontalno skaliranje je proces dodavanja više računarskih resursa kako bi se povećala sposobnost sistema da rukuje većim opterećenjem ili količinom podataka.

koji omogućava dalje pretraživanje podataka.
    - *Index File* – sadrži indekse koji pokazuju na lokacije ključeva u *Data File*-u. Indeksi su obično organizovani kao stablo.
    - *Data File* – sadrži sortirane podatke, obično u binarnom formatu. Svaki red podataka je zapisan kao par ključ–vrijednost.
- *Memtable* [11] – struktura podataka koja se koristi za privremeno pohranjivanje nedavno upisanih podataka prije trajnog zapisivanja na disk.
- *Skip Lista* [12] – probabilistička struktura podataka. Omogućava brzo pretraživanje pomoću višestrukih nivoa povezanih listi. Broj elemenata se povećava od gornjih ka donjim nivoima. Na ovaj način je omogućeno „preskakanje“ elemenata koji nisu relevantni za pretragu.
- *Bloom Filter* [8] – probabilistička struktura podataka, dizajnirana da brzo i efeikasno odredi da ovog problemali je neki element prisutan u skupu. Čuva se u radnoj memoriji. Upit vraća odgovor koji sugeriše mogućnost prisutnosti elementa na disku ili sigurno odsustvo istog.

III. RJEŠENJE

Implementacija ključ-vrijednost baze podataka sastoji se iz dvije glavne komponente:

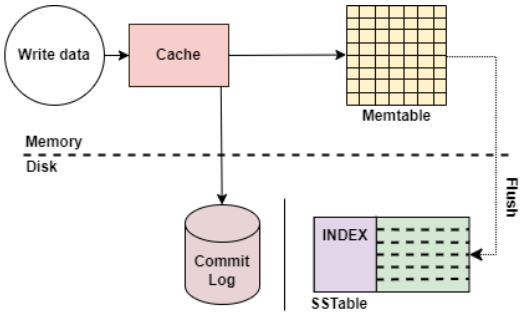
1. operacija upisa i promjene podataka (*write path*), koja je opisana u potpoglavlju *A.* Operacija upisa i promjene podataka (write path)
2. operacija čitanja podataka (*read path*), koja je opisana u potpoglavlju B. Operacija čitanja podataka (read path)

U cilju zaštite sistema od prevelikog broja zahtjeva i zlonamjernih napada implementiran je mehanizam za ograničavanje broja zahtjeva koje korisnik može poslati. Ovaj mehanizam zaštite se pokreće prije slanja svakog zahtjeva od strane korisnika. Algoritam koji je korišten u ovu svrhu zove se *Token Bucket* (potpoglavlje *A.*).

A. Operacija upisa i promjene podataka (write path)

Operacija upisa i promjene podataka realizuje se komandom „PUT”. Kao ulazni podatak, uz komandu se navode ključ i vrijednost objekta kojeg dodajemo ili mijenjamo. Izlaz je indikator uspješnosti operacije upisa. Operacija neće biti uspješna ako nije zadovoljen odgovarajući format komande. Cilj ove operacije jeste skladištnje podataka u ključ-vrijednost bazu podataka, pri čemu se obezbjeđuje njihov integritet.

Primjer komande za ovu operaciju je „PUT|k1|Pozdrav svijete”, gdje „k1“ predstavlja ključ, a „Pozdrav svijete“ predstavlja vrijednost koju želimo da upišemo. Povratna vrijednost operacije bi bila *true*, odnosno da je operacija uspješno izvršena. Primjer kada operacija ne bi bila uspješno izvrešena je komanda „PUT|k1” jer nije naveden treći parametar komande.



Slika 1 Putanja pisanja (write path)

Slika 1 Putanja pisanja (write path) prikazuje operaciju upisa podataka i tok kojim se podaci kreću pri toj operaciji. Zahtjev se prvo upisuje u Keš (*Cache)* memoriju koja je implementirana korištenjem *Least Recently Used* algoritma [7]. Ovo nam omogućava brži pristup objektima koji su korišteni u N^2 poslednjih zahtjeva. Za obezbjeđivanje integriteta podataka, koristimo strukturu WAL. Ova struktura nam omogućava da, u slučaju otkaza sistema, možemo reprodukovati komande koje su upisane u WAL i povratiti staro stanje sistema. WAL se čuva na disku.

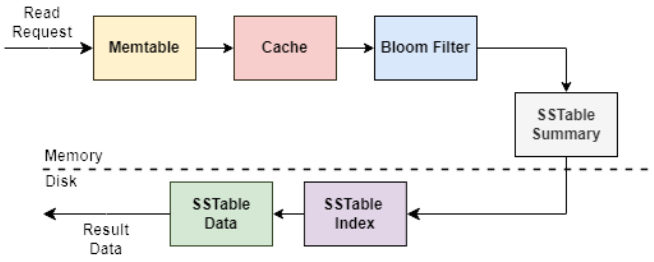
Kada WAL potvrdi da se izvršio zapis, podatak se dodaje u strukturu *Memtable*, koja predstavlja privremeno skladište podataka u radnoj memoriji. Kada se dostigne unaprijed zadata veličina[2] *Memtable-a,* formira se *SSTable* i podaci se upisuju na disk (*Flush* operacija*)*. Na ovaj način smo omogućili da se disku ne pristupa pri svakom zahtjevu za upis podataka, nego samo tokom *Flush* operacije. Nakon uspješno izvršene operacije *Flush*, *Memtable* se prazni i ponovo je dostupna za prijem novih podataka.

B. Operacija čitanja podataka (read path)

Operacija čitanja podataka realizuje se uz pomoć komande „GET”. Kao ulazni podatak, uz komandu se navodi ključ objekta koji se traži. Izlaz je objekat koji ima ključ dobijen na ulazu. Cilj operacije je efikasno dobaviti objekat koji odgovara zadatom ključu.

Primjer komande za ovu operaciju je „GET|k1”, gdje „k1“ predstavlja ključ, za kojeg želimo da dobavimo vrijednost. Povratna vrijednost operacije bi bila „Pozdrav svijete“ (iz Primjer komande za ovu operaciju je „PUT|k1|Pozdrav svijete”, gdje „k1“ predstavlja ključ, a „Pozdrav svijete“ predstavlja vrijednost koju želimo da upišemo. Povratna vrijednost operacije bi bila *true*, odnosno da je operacija uspješno izvršena. Primjer kada operacija ne bi bila uspješno izvrešena je komanda „PUT|k1” jer nije naveden treći parametar komande.

[2] Potrebni parametri za sve algoritme i strukture podataka se definišu iz konfiguracionog fajla.



Slika 2 Putanja čitanja (read path)

Slika 2 Putanja čitanja (read path) prikazuje operaciju čitanja podataka i tok kojim se podaci kreću pri toj operaciji. Zbog optimizacije i brzine sistema, prvo pokušavamo da pronađemo podatak u radnoj memoriji. Provjeravamo da li je ključ već prisutan u *Memtable* strukturi. Ova struktura je implementirana uz pomoć *Skip List*-e, koja nam omogućava efikasniju pretragu. Ako ključ nije prisutan, provjerava se Keš memorija. Nakon što podatak nije pronađen u Keš memoriji, pretražujemo strukturu *Bloom Filter* (potpoglavlje C. Algoritmi i strukture podataka potrebni za rješenje problema). Ova struktura nam daje indikaciju da li ključ postoji na disku. Upit vraća vrijednost „možda je na disku" ili „sigurno nije na disku" [8]. Ukoliko *Bloom Filter* signalizira da podatak ne postoji, zaključujemo da nije potrebno pristupati disku, čime se završava pretraga.

Ako nam signalizira da podatak možda postoji, sa diska učitavamo *SSTable Summary* fajl koji odgovara opsegu traženog ključa. Iz ovog fajla saznajemo kome *SSTable Index* fajlu treba da pristupimo. *SSTable Index* nam omogućava bržu pretragu tako što nam govori tačnu lokaciju traženog podatka u okviru *SSTable Data* fajlova. Strukture ovih fajlova su objašnjene u potpoglavlju C. Algoritmi i strukture podataka potrebni za rješenje problema. Ako je podatak pronađen, on se dodaje u Keš memoriju i vraća korisniku.

IV. REZULTATI I DISKUSIJA

Cilj ovog poglavlja je da ustanovimo da li ključ-vrijednost baza podataka efikasno rješava probleme koje smo identifikovali kod tradicionalnih relacionih baza podataka. U narednim potpoglavljima će biti opisani eksperimenti koji su izvršeni, njihovi rezultati i diskusija rezultata.

A. Metodologija

Za sprovođenje eksperimenta korišten je računar opremljen *Ryzen5 5600* procesorom, sa 16GB RAM memorijom i 512GB skladišnog kapaciteta. Operativni sistem na kojem su vršeni eksperimenti je *Windows 11*.

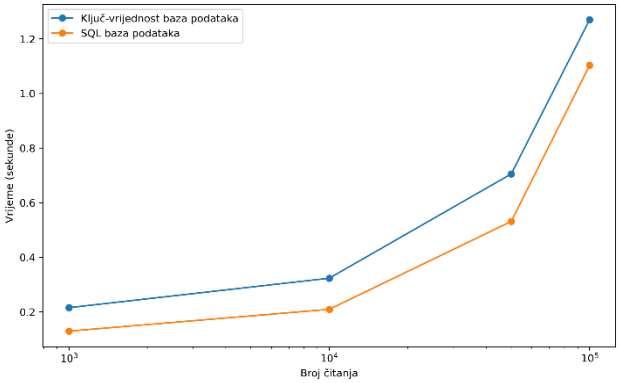
Za testiranje performansi koristili smo *Postgres SQL* relacionu bazu podataka naspram naše implementacije ključ-vrijednost baze podataka. Testiranje je obavljeno pokretanjem SQL skripti za *Postgres SQL* bazu podataka, a za ključ-vrijednost bazu podataka je pisana skripta u *Go* programskom jeziku. Performanse su testirane na 1000, 10000, 50000 i 100000 podataka. Ove veličine podataka su odabrane kako bismo identifikovali kako se svaka baza nosi sa skaliranjem i povećanjem opterećenja. U obje baze podataka su upisani isti podaci.

Za *Postgres SQL* relacionu bazu podataka smo ostavili originalnu konfiguraciju. Konfiguraciju ključ-vrijednost baze podataka smo mijenjali da bismo obezbijedili efikasno testiranje performansi. Kapacitet *Memtable* strukture smo postavili na 3072 bajta. Da bi omogućili uspješan prolazak svih zahtjeva, vrijednost maksimalnog broja tokena za *Tocken Bucket* smo postavli na neograničeno.

Obavljena su dva eksperimenta. Prvi eksperiment je testirao performanse operacije upisa velike količine podataka, dok je drugi eksperiment služio za testiranje performansi operacije čitanja velike količine podataka. U sledećem potpoglavlju se mogu vidjeti rezultati navedenih eksperimenata.

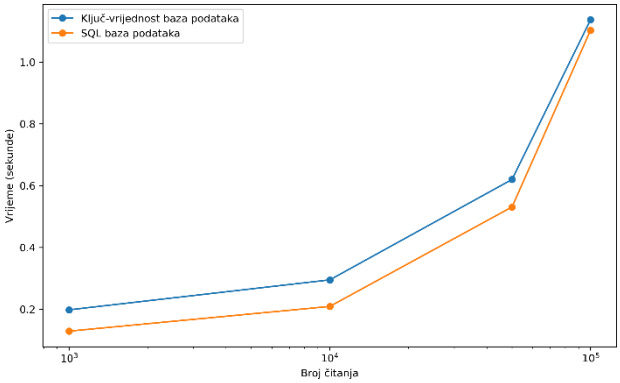
B. Rezultati

Rezultati eksperimenta su prikazani na Slika 3 Rezultati upisa velike količine podataka i Slika 4 Rezultati čitanja velike količine podataka. Na x-osi je predstavljen broj podataka koji je upisan u bazu podataka, dok je na y-osi prikazano vrijeme potrebno za izvršavanje operacije u sekundama.



Slika 3 Rezultati upisa velike količine podataka

Sa Slika 3 Rezultati upisa velike količine podataka možemo da vidimo da ključ-vrijednost baza podataka ima lošije vrijeme upisa podataka u odnosu na relacionu bazu podataka. Prosječno vrijeme potrebno za upis podatka u ključ-vrijednost bazu podataka je 1.03 sekunde.



Slika 4 Rezultati čitanja velike količine podataka

Sa Slika 4 Rezultati čitanja velike količine podataka možemo da vidimo da je vrijeme izvršenja operacije čitanja za ključ-vrijednost bazu podataka nešto manje nego za operaciju upisa. Kako broj podataka raste, tako se smanjuje razlika u vremenu potrebnom za izvršenje operacije između ove dvije baze podataka. Prosječno vrijeme za čitanje podatka u ključ-vrijednost bazi podataka je 0.92 sekunde.

B. Diskusija

Iako smo očekivali bolje performanse od ključ-vrijednost baze podataka zbog njene fleksibilnosti i skalabilnosti, rezultati su pokazali suprotno. Mogući razlozi za ovo su:

- nedovoljno optimizovani upiti
- nedostatak indeksiranja u ključ-vrijednost bazi podataka
- zbog distribuirane prirode sistema, ključ-vrijednost baza podataka je podložna sporijem upisu podataka
- hardverska ograničenja (RAM memorija)

Dodatna istraživanja su neophodna kako bi se utvrdili uzroci ovih razlika i identifikovali potencijalni načini za poboljšanje performansi ključ-vrijednost baze podataka.

1. LITERATURA

[1] The Big Data Guide. (n.d.). MongoDB. Retrieved March 16, 2024, https://www.mongodb.com/basics/big-data-explained

[2] What is NoSQL? | Nonrelational Databases, Flexible Schema Data Models. (n.d.). AWS. Retrieved March 2, 2024, https://aws.amazon.com/nosql/

[3] NoSQL — Википедија. (n.d.). Википедија. Retrieved March 2, 2024, https://sr.wikipedia.org/wiki/NoSQL

[4] Schaefer, L. (n.d.). What Is NoSQL? NoSQL Databases Explained. MongoDB. Retrieved March 2, 2024, https://www.mongodb.com/nosql-explained

[5] Ključ-vrednost baza podataka — Википедија. (n.d.). Википедија. Retrieved March 2, 2024, https://sr.m.wikipedia.org/sr-el/Ključ-vrednost_baza_podataka

[6] Internet stvari. (n.d.). Wikipedia. Retrieved March 2, 2024, https://bs.wikipedia.org/wiki/Internet_stvari

[7] Algoritmi keširanja — Википедија. (n.d.). Википедија. Retrieved April 10, 2024, https://sr.wikipedia.org/wiki/Algoritmi_keširanja

[8] Bloom filter. (n.d.). Wikipedia. Retrieved April 10, 2024, https://en.wikipedia.org/wiki/Bloom_filter

[9] Jain, S. (2024, February 14). Need for Token Bucket Algorithm. GeeksforGeeks. Retrieved April 21, 2024, https://www.geeksforgeeks.org/token-bucket-algorithm

[10] Write-ahead logging. (n.d.). Wikipedia. Retrieved April 21, 2024, https://en.wikipedia.org/wiki/Write-ahead_logging

[11] Raj, P. (Ed.). (2014). Handbook of research on cloud infrastructures for big data analytics. IGI Global.

[12] Pugh, W. (1990). Skip lists: a probabilistic alternative to balanced trees. Communications of the ACM, 33(6), 668-676.

[13] Xu, Y., Jin, P., Lu, M., & Wang, X. (2023, December). LeanKV: Efficient Garbage Collection for LSM-Based Key-Value Stores on ZNS SSDs through Lifetime-Based SSTable Clustering. In 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS) (pp. 1895-1902).IEEE.