UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

Vuk Cvetković

Interna struktura i organizacija skladišta podataka u MongoDB bazi

Seminarski rad

Studijski program: Računarstvo i informatika

Modul: Softversko inženjerstvo

Mentor:

Doc. dr Aleksandar Stanimirović

Student:

Vuk Cvetković 1667

Niš, april 2024.

Sadržaj

Uvod

Baze podataka predstavljaju ključni deo savremenog informacionog sistema, omogućavajući organizaciju, skladištenje i efikasno upravljanje podacima. One čine osnovu mnogih aplikacija, poslovnih procesa i informacionih sistema.

Baze podataka se koriste za čuvanje različitih vrsta podataka, kao što su tekstualne informacije, slike, zvukovi i drugi tipovi podataka. Osim toga, baze podataka omogućavaju brz pristup podacima i olakšavaju njihovo ažuriranje, brisanje i pretragu, što ih čini neophodnim alatom u svim oblastima gde je potrebno upravljati velikim količinama informacija.

Kroz razvoj tehnologije, baze podataka su postale sve složenije i moćnije, pružajući napredne funkcionalnosti kao što su transakcije, kontrola pristupa, podrška za više korisnika istovremeno i replikacija podataka. U današnjem digitalnom dobu, baze podataka su neizostavni deo poslovnih procesa, istraživanja, upravljanja informacijama i mnogih drugih oblasti.

U ovom seminarskom radu će se istražiti MongoDB baza podataka, sa posebnim fokusom na njen interni sistem i način na koji upravlja podacima. Biće proučene osnovne komponente MongoDB-a, na primer Storage Engine, kao i kako one doprinose organizaciji i skladištenju podataka.

Za potrebe ovog seminarskog, koriste se najnovije verzije MongoDB servera i MongoDB shell-a. Za informacije o verzijama, u okviru command prompt-a, koristi se komanda mongosh.

Slika 1. Verzije MongoDB-a i Mongo Shell-a

NoSQL baze (MongoDB)

Za razliku od strukturalnog okruženja RDBMS-ova (relacionih baza podataka), NoSQL baze podataka [1] nude fleksibilnije rešenje za skladištenje i upravljanje podacima. One su se pojavile kao odgovor na ekspanziju velikih količina podataka i ograničenja RDBMS-ova u efikasnom rukovanju velikim obimom neoblikovanih podataka. NoSQL baze podataka, uključujući popularne opcije kao što su MongoDB i Cassandra, podržavaju različite modele podataka, poput ključ-vrednost parova, dokumentno-orijentisanih, grafovskih i široko-kolonskih modela. Ova fleksibilnost omogućava im da upravljaju složenim strukturama podataka i raznovrsnim tipovima podataka - od jednostavnog teksta do složenih ugnježdenih dokumenata - bez potrebe za unapred definisanom šemom.

One su izuzetno korisne u okruženjima koja zahtevaju brzo skaliranje, visoku dostupnost i mogućnost rukovanja velikim količinama raznovrsnih podataka, čineći ih idealnim za moderne aplikacije u oblastima kao što su real-time analitika, upravljanje sadržajem, e-trgovina i Internet stvari (IoT). Njihova priroda bez šeme, distribuirana arhitektura i visoke performanse čine NoSQL baze podataka preferiranim izborom za kompanije koje moraju brzo da se prilagode promenljivim zahtevima za podacima.

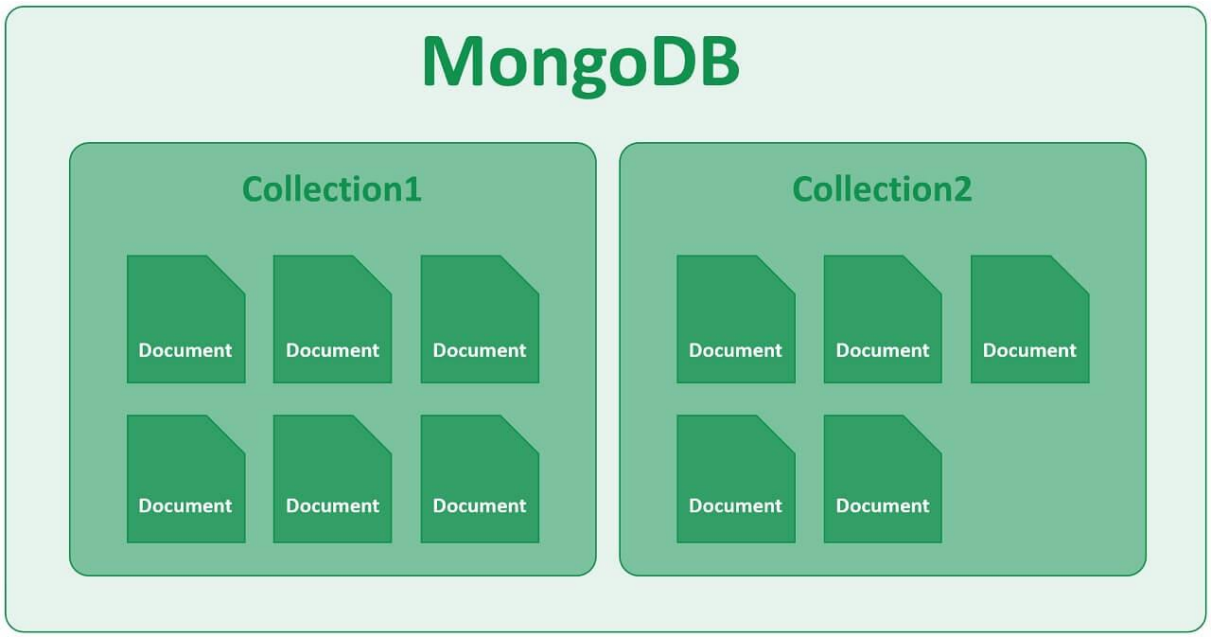
Kolekcije

Kolekcije su osnovne strukture za organizaciju podataka u MongoDB-u. One predstavljaju skup dokumenata i slične su tabelama u tradicionalnim relacionim bazama podataka, ali sa većom fleksibilnošću. Kolekcije nemaju unapred definisanog šablona ili strukturu, što znači da dokumenti unutar iste kolekcije mogu imati različite skupove polja i struktura.

Svaka kolekcija se sastoji od:

- Dokumenata: Dokumenti su osnovne jedinice podataka u MongoDB-u i sastoje se od polja sa različitim tipovima podataka kao što su tekst, brojevi, datumi, binarni podaci i drugi. Dokumenti su zapisani u BSON formatu, koji je sličan JSON formatu, ali sa dodatnim mogućnostima poput binarnih podataka.
- Indeksa: Kolekcije mogu imati indekse, koji ubrzavaju pretragu podataka i poboljšavaju performanse upita. Indeksi se koriste za brzo pretraživanje dokumenata po određenim poljima.

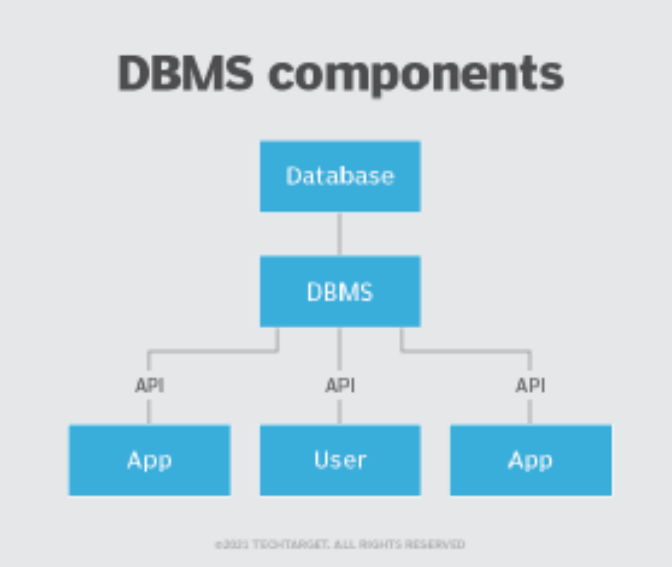
Kolekcije ne moraju imati unapred definisanu strukturu i mogu dinamički prihvatiti nove dokumente sa različitim poljima i vrednostima. Ova fleksibilnost čini MongoDB prikladnim za raznolike i složene skupove podataka.



Slika 2. Kolekcije u MongoDB [2]

DBMS

Sistem za upravljanje bazama podataka (DBMS) [3] je softverski sistem koji omogućava kreiranje, organizaciju i manipulaciju podacima unutar baze podataka. DBMS omogućava korisnicima da kreiraju, čitaju, ažuriraju i brišu podatke unutar baze podataka. On predstavlja interfejs između baze podataka i korisnika ili aplikativnih programa, osiguravajući doslednu organizaciju podataka i olakšavajući njihov pristup.



Slika 3. DBMS kao interfejs baze podataka

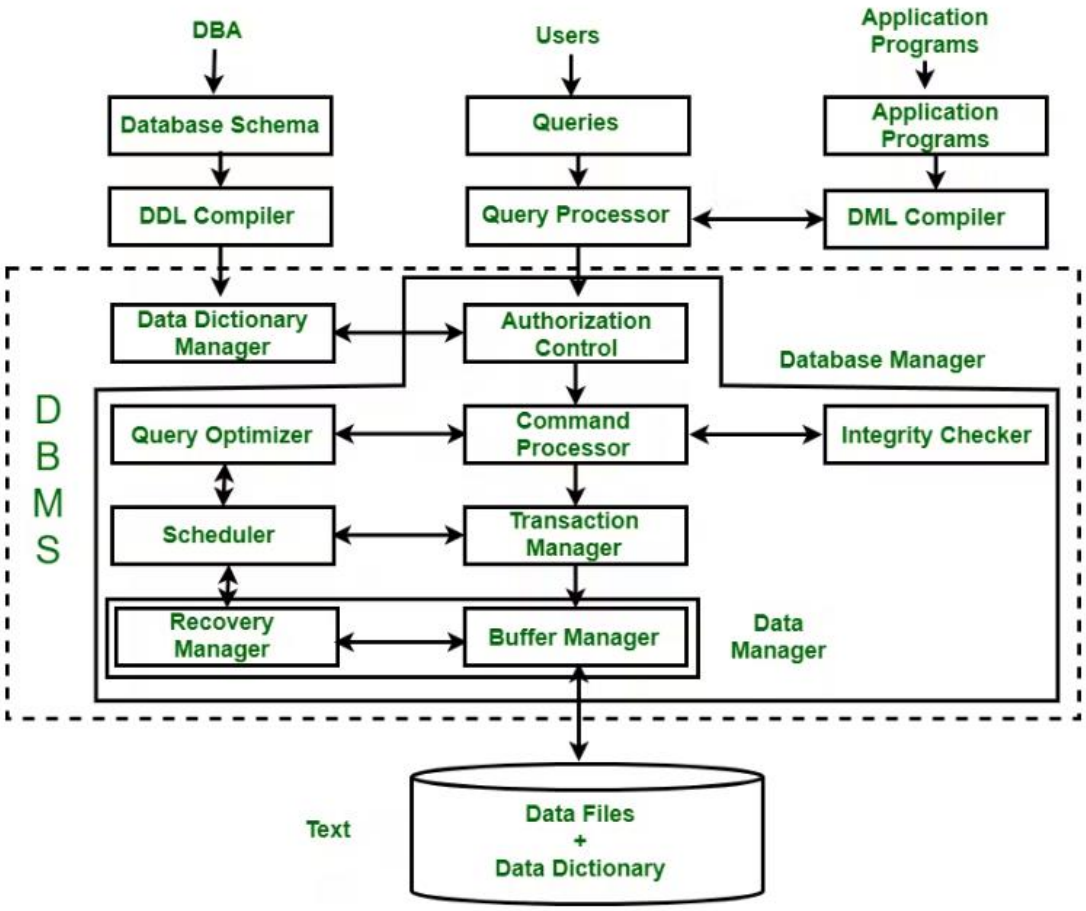
Funkcije DBMS-a

DBMS [3] upravlja podacima pružajući mehanizam za pristup, zaključavanje i modifikaciju podataka. Struktura baze podataka definiše se kroz shemu, što omogućava logičku organizaciju podataka. Ovi osnovni elementi doprinose istovremenosti, bezbednosti, integritetu podataka i jedinstvenim administrativnim procedurama. DBMS podržava različite zadatke administracije baze podataka, uključujući upravljanje promenama, nadgledanje i optimizaciju performansi, bezbednost, rezervne kopije i oporavak. Većina DBMS-a takođe pruža automatsko vraćanje i ponovno pokretanje, kao i evidenciju i reviziju aktivnosti u bazama podataka i aplikacijama koje im pristupaju.

DBMS pruža centralizovan prikaz podataka koji mogu biti dostupni više korisnicima odjednom. Mogu ograničiti vidljivost i pristup podacima korisnicima, pružajući različite poglede na jednu šemu baze podataka. Krajnji korisnici i softverski programi ne moraju znati gde se fizički nalaze podaci ili na kojem medijumu su smešteni, jer DBMS obrađuje sve zahteve.

DBMS može pružiti i logičku i fizičku nezavisnost podataka kako bi zaštitio korisnike i aplikacije od potrebe za poznavanjem lokacije podataka ili promena u fizičkoj strukturi. Sve dok programi koriste API za bazu podataka koji pruža DBMS, programeri ne moraju menjati aplikacije zbog promena u bazi podataka.

Struktura DBMS-a



Slika 4. Struktura DBMS-a [1]

Database Administrator (DBA)

DBA je ključna osoba odgovorna za projektovanje, upravljanje i održavanje baze podataka. On definiše šemu baze podataka, koja predstavlja njenu strukturu, i osigurava da baza podataka funkcioniše pravilno i efikasno. DBA takođe obavlja zadatke poput optimizacije performansi, upravljanja bezbednošću podataka, nadgledanja integriteta i pružanja podrške korisnicima baze podataka.

Users

Korisnici su oni koji interaguju sa sistemom baze podataka preko query-a i aplikativnih programa. Korisnici koriste upite da bi dobili podatke, manipulisali njima i interagovali sa podacima baze podataka prema svojim potrebama.

Application Programs	Ove aplikacije predstavljaju interfejs za korisnike i često sadrže poslovnu logiku. Aplikativni programi stupaju u interakciju sa DBMS-om preko DML-a (Data Manipulation Language) kako bi izvodili operacije nad podacima, uključujući umetanje, ažuriranje i brisanje podataka.
Database Manager	Centralna komponenta koja upravlja operacijama celog sistema baze podataka, uključujući kreiranje, čitanje, ažuriranje i brisanje podataka. Database Manager koordinira interakcije između različitih delova sistema, kao što su Query Processor, Transaction Manager i Buffer Manager.
DDL Compiler	Obrada DDL (Data Definition Language) iskaza, koji definišu šemu baze podataka i ažuriraju Data Dictionary. DDL Compiler tumači i primenjuje iskaze koji definišu strukturu baze podataka, njene tabele, indekse i druge elemente.
Query Processor	Tumači i kompajlira korisničke upite iz query jezika u operacije koje Database Manager može da izvrši. Query Processor često koristi Query Optimizer za identifikaciju najefikasnijih metoda izvršavanja upita.
DML Compiler	Kompajlira DML (Data Manipulation Language) iskaze iz aplikativnih programa u operacije koje Database Manager može da izvrši. To uključuje zadatke poput umetanja, ažuriranja i brisanja podataka.
Data Dictionary Manager	Čuva informacije o strukturi baze podataka, njenim objektima, tipovima podataka i ograničenjima. Data Dictionary Manager je izvor istine o strukturi baze podataka i osigurava da svi delovi sistema koriste konzistentne informacije.
Authorization Control	Proverava da li korisnik ima potrebne dozvole za izvršavanje upita i operacija u bazi podataka. Authorization Control osigurava da samo ovlašćeni korisnici mogu pristupati određenim podacima i operacijama.
Command Processor	Izvršava DDL i DML komande, obezbeđujući da se izmene podataka primenjuju pravilno i efikasno. Command Processor koordinira sa Database Manager-om kako bi izvršio operacije koje su definisane u upitima.
Query Optimizer	Određuje najefikasniji način za izvršavanje upita. Query Optimizer analizira moguće strategije izvršavanja upita i bira onu koja minimizuje troškove i vreme potrebno za izvršenje.

Scheduler	Planer. Upravlja izvršavanjem istovremenih transakcija kako bi se obezbedila doslednost i integritet baze podataka. Scheduler koordinira sa Transaction Manager-om kako bi osigurao da se istovremene operacije izvršavaju bez konflikata.
Transaction Manager	Osigurava pravilno izvršavanje transakcija, garantujući ACID (Atomicity, Consistency, Isolation, Durability) svojstva. Transaction Manager vodi računa o vođenju evidencije transakcija kako bi omogućio proces oporavka u slučaju neuspeha.
Recovery Manager	Rukuje oporavkom baze podataka u slučaju kvara sistema ili neuspeha transakcija. Recovery Manager koristi informacije iz journaling sistema i snimaka (snapshots) kako bi vratio bazu podataka u konzistentno stanje.
Buffer Manager	Upravlja keš memorijom za podatke kojima se često pristupa, smanjujući potrebu za učitavanjem podataka sa diska. Buffer Manager optimizuje performanse sistema tako što ubrzava pristup podacima.
Integrity Checker	Osigurava da izmene podataka ne krše ograničenja integriteta baze podataka. Integrity Checker obavlja provere i validacije kako bi se osiguralo da podaci ostanu tačni i konzistentni.
Data Manager	Rukuje fizičkim skladištenjem podataka, uključujući datoteke sa podacima i Data Dictionary. Data Manager nadgleda pristup podacima na disku i osigurava efikasno korišćenje resursa skladišta.

Komponente DBMS-a

DBMS je složen skup komponenti [1], svaka dizajnirana da obavlja specifične funkcije koje zajedno osiguravaju efikasno, sigurno i pouzdano upravljanje podacima. Razumevanje ovih komponenti pruža uvid u to kako DBMS obrađuje složene zahteve skladištenja i povlačenja podataka.

Database Engine

Database Engine upravlja podacima, objektima baze podataka, samom bazom podataka i odnosima među njima. On upravlja svim interakcijama sa bazom podataka, od osnovnog skladištenja i povlačenja podataka do složenijih operacija poput upravljanja transakcijama. Baza podataka obezbeđuje integritet i doslednost podataka, čak i u slučaju istovremenih pristupa korisnika ili kvarova sistema, implementirajući ACID (atomicity, consistency, isolation, durability) osobine.

Database access language

Ovaj interfejs je sredstvo komunikacije korisnika i aplikacija sa bazom podataka. Strukturirani upitni jezik (SQL) je najčešće korišćen jezik pristupa, poznat po efikasnosti u upravljanju relacionim bazama podataka. SQL omogućava korisnicima izvođenje različitih operacija, od jednostavnog povlačenja podataka (korišćenjem SELECT naredbi) do složenih transakcija i analitike. Njegova univerzalna upotreba u različitim DBMS-ovima govori o njegovoj efikasnosti i robusnosti u upravljanju bazama podataka.

Za razliku od SQL-a, NoSQL jezik se koristi za interakciju sa NoSQL bazama podataka, koje se razlikuju od tradicionalnih relacionih baza podataka po svom modelu podataka i načinu skladištenja. NoSQL jezik pristupa omogućava korisnicima da obavljaju raznovrsne operacije nad podacima, uključujući upite, dodavanje novih podataka, ažuriranje postojećih zapisa i brisanje podataka. Ovaj jezik pruža fleksibilnost i skalabilnost potrebnu za rad sa velikim količinama podataka i dinamičkim strukturama podataka koje su karakteristične za NoSQL baze podataka.

Database structure

Database structure je centralna komponenta DBMS-a koja definiše logičku strukturu baze podataka. Data Definition Language (DDL) je ključan u ovom delu. On omogućava kreiranje, modifikaciju i brisanje objekata baze podataka poput tabela (kod SQL), kolekcija i fajlova (kod NoSQL), indeksa i pogleda. Ova struktura određuje kako su podaci skladišteni i organizovani, kao i kako se različiti elementi podataka međusobno povezuju. Efikasno dizajnirana struktura baze podataka ključna je za postizanje optimalne performanse, integriteta podataka i jednostavnosti upravljanja podacima.

Query processor

Query processor je zapravo mozak DBMS-a, koji interpretira i optimizuje korisničke upite radi efikasnog izvršavanja. Ima ključnu ulogu u ukupnoj performansi DBMS-a, transformišući upite visokog nivoa u niz niskih operacija koje baza podataka može izvršiti. Procesor uključuje optimizator upita, koji procenjuje više strategija za izvršenje upita i bira najefikasniju na osnovu faktora poput distribucije podataka, kompleksnosti upita i trenutnog opterećenja sistema.

Storage Engine

Storage engine [4] (mehanizam za skladištenje) je komponenta baze podataka koja je odgovorna za upravljanje načinom na koji se podaci skladište, kako u memoriji, tako i na disku. MongoDB podržava više različitih storage engine-a, jer se pojedini engine-i bolje ponašaju za specifične zadatke. Svaki storage engine može na različite načine uticati na performanse aplikacije.

Svaki storage engine ima različite karakteristike u pogledu upravljanja podacima, performansi, efikasnosti i podrške za transakcije. Neki engine-i su optimizovani za brzinu pristupa podacima, dok su drugi fokusirani na pouzdanost i održivost podataka. Upotreba odgovarajućeg storage engine-a može povećati efikasnost i stabilnost baze podataka, kao i smanjiti vreme odziva i troškove skladištenja.

Tipovi Storage Engine-a

WiredTiger Storage Engine

WiredTiger je podrazumevani storage engine od verzije 3.2 u MongoDB-u. On je idealan za većinu radnih opterećenja i preporučuje se za nove implementacije. WiredTiger pruža model konkurentnosti na nivou dokumenata, automatski vrši checkpointing i koristi kompresiju podataka za efikasno skladištenje. U MongoDB Enterprise verziji, WiredTiger podržava šifrovanje podataka u mirovanju.

In-Memory Storage Engine

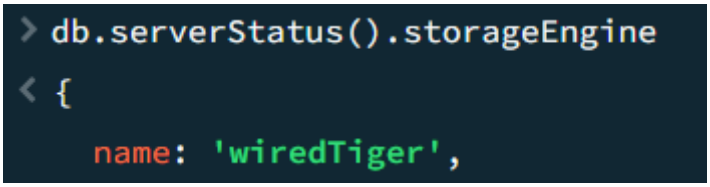
In-Memory storage engine je dostupan u MongoDB Enterprise verziji. Umesto da skladišti podatke na disku, ovaj engine drži ih u memoriji, što omogućava niže kašnjenje i brži pristup podacima. Ovaj engine je koristan za aplikacije koje zahtevaju brzinu i performanse, ali nije pogodan za trajno skladištenje podataka.

MMAPv1 Storage Engine

MMAPv1 je stariji storage engine koji se koristio u ranijim verzijama MongoDB-a. Iako pruža osnovne funkcije za rad sa podacima, MMAPv1 ima ograničenja u pogledu skalabilnosti i performansi u poređenju sa WiredTiger-om. Nije preporučljiv za nove projekte, ali se i dalje može naći u upotrebi u nekim starijim sistemima.

Wired Tiger Storage Engine

Od verzije 3.2, WiredTiger storage engine [5] je postao podrazumevani storage engine u MongoDB-u. Ovo znači da su sve nove MongoDB instance, koje se pokreću od ove verzije, podrazumevano podešene da koriste WiredTiger engine za skladištenje podataka. Ovaj storage engine pruža mnoge prednosti, uključujući efikasno upravljanje podacima, napredne mogućnosti kompresije i poboljšane performanse.



Slika 5. Komanda za prikaz default Storage Engine-a

WiredTiger je poznat po svojoj sposobnosti da optimizuje resurse i smanji prostor za skladištenje zahvaljujući mogućnostima kompresije podataka. Takođe, omogućava paralelno čitanje i pisanje podataka bez konflikata.

Jedna od ključnih prednosti WiredTiger-a je podrška za transakcije sa svojstvima ACID (atomicity, consistency, isolation, durability), što doprinosi pouzdanosti i integritetu podataka u bazama.

Zbog svog efikasnog rada, WiredTiger je preporučen za upotrebu u širokom spektru aplikacija, od manjih projekata do velikih sistema sa složenim radnim opterećenjem. Zahvaljujući ovim prednostima, WiredTiger je postao ključni element u modernim MongoDB implementacijama.

Document Level Concurrency

WiredTiger koristi kontrolu konkurentnosti na nivou dokumenta (document level concurrency) [5] za operacije upisa, što omogućava da više klijenata istovremeno modifikuju različite dokumente unutar iste kolekcije. Ova karakteristika poboljšava performanse sistema i omogućava paralelne promene podataka.

Za većinu operacija čitanja i upisa, WiredTiger primenjuje optimističku kontrolu konkurentnosti, što znači da se podrazumeva da neće biti konflikata pri pristupu podacima. WiredTiger koristi namerno zaključavanje na globalnom nivou, nivou baze podataka i kolekcije. Kada storage engine otkrije konflikte između dve operacije, jedna od njih će naići na konflikt pisanja, što će prouzrokovati da MongoDB automatski pokuša ponovno izvršenje te operacije.

Neke globalne operacije, obično kratkotrajne operacije koje uključuju više baza podataka, i dalje zahtevaju globalni "instance-wide" lock kako bi se osigurala konzistentnost podataka. Neke druge operacije, poput collMod (koje su povezane s promenom strukture kolekcija), i dalje zahtevaju ekskluzivni lock na nivou baze podataka kako bi se osiguralo da operacija prođe bezbedno i efikasno.

Snapshots and Checkpoints

WiredTiger koristi MultiVersion Concurrency Control (MVCC) [5], odnosno sistem kontrole konkurentnosti sa više verzija za rad sa podacima. Pri početku svake operacije, WiredTiger kreira trenutni snimak podataka i pruža konzistentan prikaz memorisanih informacija.

Kada se podaci upisuju na disk, WiredTiger piše sve podatke iz snimka na disk u konzistentan način kroz sve datoteke. Ovi trajno zabeleženi podaci služe kao checkpoint-i u datotekama, osiguravajući konzistentnost podataka do poslednjeg checkpoint-a. Ovi checkpoint-i takođe služe kao tačke oporavka podataka.

Počevši od verzije 3.6, MongoDB konfiguriše WiredTiger da automatski kreira checkpoint-e svakih 60 sekundi. Prethodne verzije postavljale su checkpoint-e ili svakih 60 sekundi ili nakon 2 GB zabeleženih podataka u journal-u, u zavisnosti od toga šta se desi prvo.

Tokom kreiranja novog checkpoint-a, prethodni ostaje važeći. Ako dođe do prekida ili greške tokom procesa kreiranja novog checkpoint-a, MongoDB može da se oporavi iz poslednjeg važećeg checkpoint-a.

Novi snimak postaje dostupan i trajan kada WiredTiger ažurira tabelu metapodataka kako bi ukazala na novi checkpoint. Kada je novi snimak dostupan, WiredTiger oslobađa resurse iz prethodnih checkpoint-a, optimizujući korišćenje resursa.

Čuvanje istorije snimaka

Počevši od verzije MongoDB 5.0, može se koristiti parametar *minSnapshotHistoryWindowInSeconds* da se odredi koliko dugo WiredTiger čuva istoriju snimaka.

Ovaj parameter se može postaviti na jedan od sledeća 3 načina:

- Preko komande:
  mongod -- minSnapshotHistoryWindowInSeconds **<seconds>**
  Ova komanda se poziva pri samom pokretanju MongoDB servera.
  -- minSnapshotHistoryWindowInSeconds se postavlja na vrednost **<seconds>**

- Preko konfiguracionog fajla
  storage:
    wiredTiger:
      engineConfig:
        minSnapshotHistoryWindowInSeconds: <seconds>
  Parametri su identični kao parametri koji se unose putem komande.
- Preko MongoDB shell-a

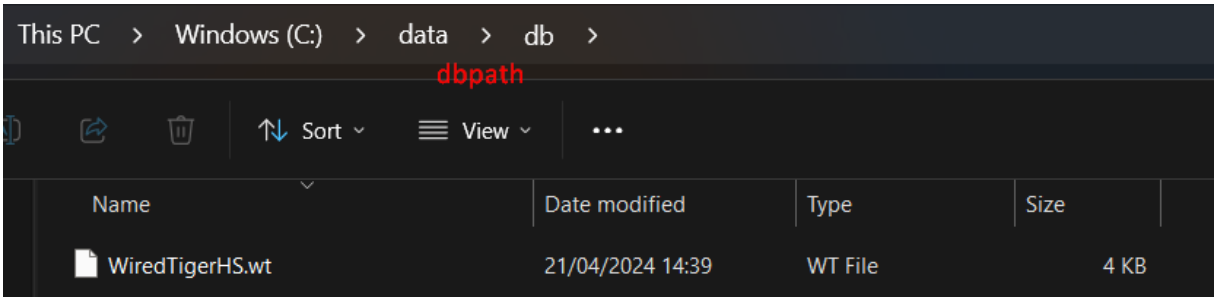
```
> db.adminCommand( { setParameter: 1, minSnapshotHistoryWindowInSeconds: 600 } )
< { was: 300, ok: 1 }
```

Slika 6. Promena parametra minSnapshotHistoryWindowInSeconds

Sa slike vidimo komandu kojom se menja parameter *minSnapshotHistoryWindowInSeconds*. Default vrednost je 300, u konkretnom primeru je postavljeno na 600.

Povećanje vrednosti parametra *minSnapshotHistoryWindowInSeconds* povećava potrošnju diska jer server mora da održava istoriju starijih promena unutar određenog vremenskog okvira. Količina zauzetog prostora na disku zavisi od radne opterećenosti - veća opterećenja zahtevaju više prostora.

MongoDB čuva istoriju snimaka u datoteci “WiredTigerHS.wt”.



Slika 7. WiredTigerHS fajl

Ovaj parametar je koristan kada su duže vremenske serije podataka potrebne, jer omogućava zadržavanje duže istorije za snimke podataka, što može biti korisno za analizu i oporavak. Međutim, treba biti oprezan pri podešavanju ove vrednosti jer može značajno uticati na korišćenje diska i performanse sistema.

Journal

WiredTiger koristi sistem unapred napisanih logova (tj. Journal-a) [5] u kombinaciji sa checkpoint-ima kako bi osigurao trajnost podataka.

Journal WiredTiger-a čuva sve izmene podataka između checkpoint-a. Ako MongoDB napusti rad između checkpoint-a, koristi se journal kako bi se ponovile sve izmene podataka od poslednjeg checkpoint-a.

Proces oporavka:

- Pregledavaju se podaci u fajlovima kako bi se pronašao identifikator poslednjeg checkpoint-a.
- Pretražuju se journal fajlovi kako bi se pronašao zapis koji odgovara identifikatoru poslednjeg checkpoint-a.
- Primenjuju se operacije u journal fajlovima koje su se dogodile od poslednjeg checkpoint-a.

Kada se koristi journal, WiredTiger kreira jedan journal zapis za svaku operaciju upisivanja pokrenutu od strane klijenta. Journal zapis uključuje sve interne operacije upisivanja koje su rezultat početne operacije upisivanja. Na primer, ažuriranje dokumenta može rezultovati promenama indeksa; WiredTiger kreira jedan journal zapis koji uključuje i operaciju ažuriranja i njene odgovarajuće izmene indeksa.

WiredTiger Journal je kompresovan korišćenjem biblioteke za kompresiju pod imenom ***snappy***. Snappy je default biblioteka za kompresiju. Da bi se postavio drugačiji algoritam za kompresiju ili da se u potpunosti onemogući kompresija, koristi se opcija *storage.wiredTiger.engineConfig.journalCompressor*.

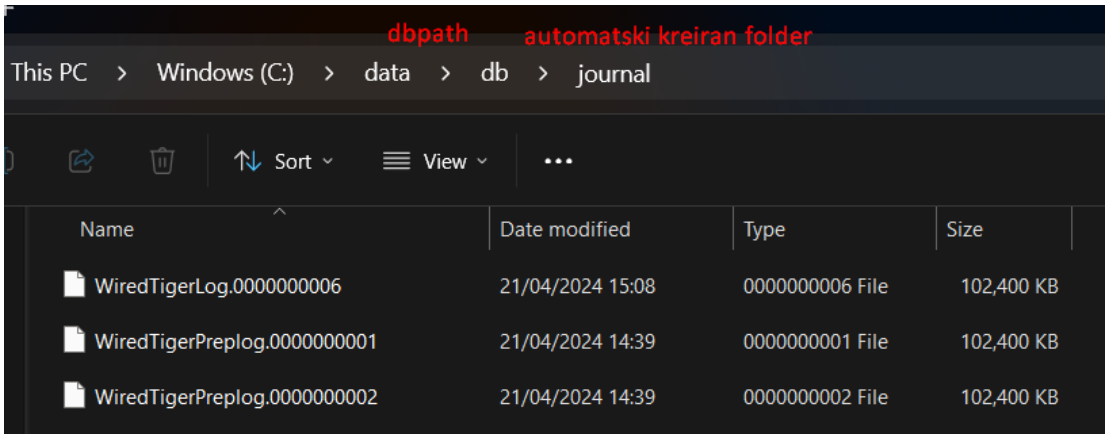
Ovaj parameter se može postaviti na jedan od sledeća 2 načina:

- Preko komande:
  mongod --wiredTigerJournalCompressor **<compressor>**
  Ova komanda se poziva pri samom pokretanju MongoDB servera.
  -- wiredTigerJournalCompressor se postavlja na vrednost **<compressor>**
- Preko konfiguracionog fajla
  ```
  storage:
    wiredTiger:
      engineConfig:
        wiredTigerJournalCompressor: <compressor>
  ```
  Parametri su identični kao parametri koji se unose putem komande.
  Moguće vrednosti su: **none**, **snappy**, **zlib**, **zstd**.

WiredTiger journal fajlovi imaju maksimalno ograničenje veličine od približno 100 MB. Kada fajl prelazi to ograničenje, WiredTiger kreira novi journal fajl. WiredTiger automatski uklanja stare journal fajlove i održava samo one koji su potrebni za oporavak od poslednjeg checkpoint-a.

Korišćenje journal-a u kombinaciji sa checkpoint-ima, osigurava se da su podaci trajni i da se mogu oporaviti u slučaju neočekivanih prekida. Snappy kompresija smanjuje veličinu journal-a, pospešujući efikasnost skladištenja.

Konkretan primer Journal fajlova



Slika 8. Journal fajlovi

Kompresija

WiredTiger omogućava kompresiju [5] za sve kolekcije i indekse u MongoDB-u, što smanjuje potrebu za skladišnim prostorom, ali uz povećanu potrošnju CPU resursa. Podrazumevano, WiredTiger koristi blokovsku kompresiju sa Snappy bibliotekom za kompresiju za sve kolekcije, kao i prefiks kompresiju za sve indekse. Opcije kompresije se takođe mogu konfigurisati za svaku kolekciju i indeks prilikom njihovog kreiranja.

Za kolekcije su dostupne i sledeće biblioteke za blokovsku kompresiju:

- zlib
- zstd (Dostupno počevši od MongoDB 4.2)

Da bi se uključio/isključio alternativni algoritam kompresije ili isključila kompresija, koristi se opcija *storage.wiredTiger.collectionConfig.blockCompressor*. WiredTiger koristi blokovsku kompresiju za kolekcije. To znači da podaci u kolekciji nisu kompresovani kao celina, već se dele na blokove (tipično veličine nekoliko kilobajta) i svaki blok se zasebno kompresuje.

Ovaj parameter se može postaviti na jedan od sledeća 2 načina:

- Preko komande:
  mongod --wiredTigerCollectionBlockCompressor **<compressor>**
  Ova komanda se poziva pri samom pokretanju MongoDB servera.
  -- wiredTigerCollectionBlockCompressor se postavlja na vrednost **<compressor>**
- Preko konfiguracionog fajla
  storage:
  wiredTiger:
  collectionConfig:
  blockCompressor: **<compressor>**
  Parametri su identični kao parametri koji se unose putem komande.

Praktičan primer

Kreiraćemo prvo kolekciju sa imenom snappy (snappy je podrazumevani način kompresije):

```
> db.createCollection('snappy')
< { ok: 1 }
```

Slika 9. Kreiranje kolekcije snappy

Zatim ćemo kreirati kolekciju sa imenom zlib i staviti da kompresija bude preko zlib-a

```
> db.createCollection('zlib', {storageEngine: {wiredTiger: {configString: 'block_compressor=zlib'}}})
< { ok: 1 }
```

Slika 10. Kreiranje kolekcije zlib

Sledeći korak je kreiranje jednog dummy dokumenta pod imenom doc

```
> doc = {_id:1, text: "Test test test test test test test"}
< { _id: 1, text: 'Test test test test test test test' }
```

Slika 11. Kreiranje dummy dokumenta

Insertovaćemo taj dokument u obe kolekcije

```
> db.snappy.insert(doc)
< {
    acknowledged: true,
    insertedIds: {
      '0': 1
    }
  }
> db.zlib.insert(doc)
< {
    acknowledged: true,
    insertedIds: {
      '0': 1
    }
  }
```

Slika 12. Insertovanje dokumenta u obe kolekcije

Na kraju provera veličine obe kolekcije

```
> db.snappy.stats().storageSize
< 4096
> db.zlib.stats().storageSize
< 20480
```

Slika 13. Prikaz veličina obe kolekcije

Možemo videti da u konkretnom promeru, snappy vrši bolju kompresiju u odnosu na zlib.

Za indekse, može se uključiti/isključiti prefiks kompresija pomoću opcije *storage.wiredTiger.indexConfig.prefixCompression*. WiredTiger koristi prefiks kompresiju za indekse, koja smanjuje veličinu indeksa kompresovanjem zajedničkih prefiksa među indeksima. Ovaj pristup je efikasan za indeksirane podatke koji sadrže mnogo sličnih ili ponavljajućih prefiksa.

Ovaj parameter se može postaviti na jedan od sledeća 2 načina:

- Preko komande:
  mongod -- wiredTigerIndexPrefixCompression **true|false**
  Ova komanda se poziva pri samom pokretanju MongoDB servera.
  -- wiredTigerIndexPrefixCompression se postavlja na vrednost **true ili false**
- Preko konfiguracionog fajla
  storage:
  wiredTiger:
  indexConfig:
  prefixCompression: **true|false**
  Parametri su identični kao parametri koji se unose putem komande.

Korišćenje memorije

U okviru MongoDB-a, WiredTiger koristi kombinaciju unutrašnjeg keša WiredTiger-a i keša datotečnog sistema. [5]

Veličina unutrašnjeg keša WiredTiger-a

Od verzije MongoDB 3.4, podrazumevana veličina unutrašnjeg keša WiredTiger-a određuje se kao veća vrednost između:

- 50% od (RAM - 1 GB)
- 256 MB

Na primer, na sistemu sa ukupno 4 GB RAM-a, WiredTiger keš će koristiti 1,5 GB RAM-a

0,5 * (4 GB - 1 GB) = 1.5 GB

Nasuprot tome, sistem sa ukupno 1,25 GB RAM-a će alocirati 256 MB za WiredTiger keš, jer je to više od polovine ukupnog RAM-a minus jedan gigabajt (0,5 * (1,25 GB - 1 GB) = 128 MB, što je manje od 256 MB).

Ove postavke omogućavaju WiredTiger-u da optimalno iskoristi dostupnu memoriju za keširanje, uzimajući u obzir količinu RAM-a koja je raspoloživa na sistemu. Keširanje pomaže ubrzavanju pristupa podacima i poboljšava performanse sistema.

Podrazumevano, WiredTiger koristi Snappy blok kompresiju za sve kolekcije i prefiks kompresiju za sve indekse. Podešavanja kompresije su konfigurabilna na globalnom nivou i mogu se takođe postaviti po kolekciji i indeksu tokom kreiranja kolekcije i indeksa.

Različite reprezentacije podataka

Podaci u unutrašnjem kešu WiredTiger-a i podaci na disku predstavljeni su na različite načine:

- Podaci u kešu datotečnog sistema: Oni su isti kao i podaci na disku, uključujući prednosti bilo koje kompresije koja se koristi za datoteke sa podacima. Keš datotečnog sistema koristi operativni sistem kako bi smanjio učitavanje diska (disk I/O).
- Indeksi učitani u unutrašnji keš WiredTiger-a: Imaju drugačiju reprezentaciju podataka u odnosu na format na disku, ali i dalje mogu iskoristiti prednosti prefiks kompresije indeksa kako bi se smanjila upotreba RAM-a. Prefiks kompresija indeksa eliminiše uobičajene prefikse iz indeksiranih polja, smanjujući upotrebu memorije.
- Podaci kolekcija u unutrašnjem kešu WiredTiger-a: Ovi podaci su nekonvertovani i koriste drugačiju reprezentaciju od one na disku. Blok kompresija može pružiti značajne uštede u prostoru za skladištenje na disku, ali podaci moraju biti dekompresovani kako bi server mogao njima da rukuje.

Korišćenje slobodne memorije

Preko keša datotečnog sistema, MongoDB automatski koristi svu slobodnu memoriju koja nije iskorišćena od strane keša WiredTiger-a ili drugih procesa, omogućavajući optimizaciju iskorišćenja memorije i poboljšanje performansi sistema.

Da bi se prilagodila veličinu unutrašnjeg keša WiredTiger-a, može se koristiti parametar *storage.wiredTiger.engineConfig.cacheSizeGB* u konfiguracionom fajlu ili *--wiredTigerCacheSizeGB* kao komandna linija prilikom pokretanja mongod procesa.

Ovaj parameter se može postaviti na jedan od sledeća 2 načina:

- Preko komande:
  mongod – wiredTigerCacheSizeGB **<sizeInGB>**
  Ova komanda se poziva pri samom pokretanju MongoDB servera.
  -- wiredTigerCacheSizeGB se postavlja na vrednost **<sizeInGB>**

- Preko konfiguracionog fajla
  ```
  storage:
    wiredTiger:
      engineConfig:
        cacheSizeGB: <sizeInGB>
  ```
  Parametri su identični kao parametri koji se unose putem komande.

Napomena: Preporučuje se da se ne povećava veličina unutrašnjeg keša WiredTiger-a iznad podrazumevane vrednosti, jer to može izazvati probleme sa performansama ili resursima na sistemu. Podrazumevane vrednosti su pažljivo odabrane kako bi obezbedile efikasan rad i stabilnost sistema.

Konfiguracija

Postoje različite vrste konfiguracija u bazama podataka koje zavise od potreba i zahteva aplikacije ili sistema. Konfiguracija može uključivati podešavanja Storage Engine-a, optimizaciju performansi, upravljanje memorijom i keširanjem, kao i sigurnosne postavke. Izbor odgovarajuće konfiguracije zavisi od veličine i prirode podataka, očekivanog opterećenja sistema, kao i prioriteta u performansama i pouzdanosti. Pravilno podešena konfiguracija može značajno poboljšati performanse i efikasnost baze podataka, stoga je važno izabrati odgovarajuće opcije prema specifičnim potrebama projekta.

WiredTiger Storage Engine može imati dve glavne konfiguracije: B-Tree i LSM (Log-Structured Merge-Tree) [6].

- B-Tree konfiguracija omogućava efikasno skladištenje i pretragu podataka koristeći balansirana stabla. Ovaj pristup obezbeđuje organizaciju podataka u hijerarhijsku strukturu, omogućavajući brza čitanja i pisanja uz minimalne promene strukture podataka.
- LSM konfiguracija koristi princip log-structured merge-stabla kako bi optimizovala performanse za aplikacije sa velikim opterećenjem podataka i učestalim operacijama pisanja. Ova konfiguracija omogućava bolju efikasnost za sekvencijalne zapise i pruža brzu upotrebu diska putem slojeva.

Izbor između B-Tree i LSM konfiguracija zavisi od specifičnih potreba i preferencija projekta.

B-Tree Based Engine

Ovo je jedan od drevnih konfiguracija Storage Engine-a iz kojeg su izvedene druge sofisticiranije konfiguracije. To je struktura podataka u obliku samouravnoteženog stabla koja osigurava sortiranje podataka i omogućava pretrage, sekvencijalni pristup, umetanje i brisanje na logaritamski način. To je skladište zasnovano na redovima, pri čemu se svaki red smatra kao jedan zapis u bazi podataka.

Prednosti B-Tree Storage Engine-a:

- Velika propusnost i nisko kašnjenje pri čitanju. B-Tree stabla imaju tendenciju rasta u širinu, tako da se vrlo malo čvorova pretražuje.
- Održavanje ključeva u sortiranom redosledu za sekvencijalno pretraživanje, dok su indeksi uravnoteženi pomoću rekurzivnog algoritma.
- Unutrašnji čvorovi uvek su barem napola puni, što generalno smanjuje rasipanje prostora.
- Lako rukovanje velikim brojem umetanja i brisanja u kratkom vremenskom periodu.
- Hijerarhijsko indeksiranje se koristi s ciljem smanjenja čitanja sa diska.
- Ubrzava umetanja i brisanja korišćenjem delimično punih blokova.

Ograničenja B-Tree skladišnog motora:

- Loše performanse upisivanja zbog potrebe za osiguranjem dobro uređenih podataka s nasumičnim upisima. Nasumični upisi su skuplji od sekvencijalnih upisa na skladište.
- Penali pri čitanju, modifikovanju i pisanju čitavog bloka čak i za manje izmene reda bloku.

Log Structured Merge (LSM) Tree Based Engine

Zbog loših performansi upisa B-Tree Storage Engine-a, programeri su morali smisliti način kako bi se nosili sa većim skupovima podataka u DBMS-u. Tako je Log Structured Merge Tree Storage Engine (LSM Tree) osmišljen kako bi poboljšao performanse indeksiranog pristupa datotekama s visokim volumenom upisa tokom dužeg perioda. U ovom slučaju, nasumični upisi u prvom nivou kaskadne memorije pretvaraju se u sekvencijalne upise u komponentu baziranu na disku.

Prednosti LSM Tree Storage Engine-a:

- Mogućnost brzih sekvencijalnih upisa omogućava brzo rukovanje velikim brzorastućim podacima.
- Dobro prilagođen za višeslojno skladištenje, pružajući organizacijama bolji izbor u pogledu troškova i performansi. SSD-ovi na bazi fleša pružaju odlične performanse u ovom slučaju.
- Bolja kompresija i efikasnost skladištenja, čime se štedi prostor i omogućava skoro potpuno popunjavanje skladišta.
- Podaci su uvek dostupni za upit odmah.
- Umetanja su vrlo brza.

Ograničenja LSM Tree skladišnog motora:

- Veća potrošnja memorije u poređenju sa B-Tree tokom operacija čitanja zbog čitanja i amplifikacije prostora. Međutim, neki pristupi, kao što su bloom filteri, ublažili su ovaj efekat u praksi, smanjujući broj fajlova koji se proveravaju tokom tačkaste pretrage.

WiredTiger tehnologija je dizajnirana tako da koristi prednosti i B-Tree i LSM-a, što je čini sofisticiranom i najboljim Storage Engine-om za MongoDB.

Ograničenja

U nastavku su data ograničenja kod WiredTiger Storage Engine-a [5].

Dokumenti se ne mogu fiksirati u kešu: WiredTiger storage engine ne omogućava fiksiranje (pinovanje) dokumenata u kešu. Ova ograničenja mogu uticati na performanse pri pristupu velikim količinama podataka ili pri radu sa često korišćenim dokumentima.

Keš nije podeljen na čitanje i pisanje: WiredTiger ne deli keš na posebne delove za čitanje i pisanje podataka. To znači da intenzivan upis podataka može povremeno smanjiti performanse sistema. Ipak, WiredTiger u takvim situacijama daje prioritet keširanju indeksa kako bi osigurao efikasniji rad baze podataka.

Prioritet indeksa u slučaju intenzivnog pisanja: Kada su operacije pisanja veoma intenzivne, WiredTiger fokusira svoj keš na indekse. Ova karakteristika pomaže u održavanju performansi sistema i ubrzava pristup podacima.

Keširanje na nivou instance: WiredTiger keš se dodeljuje celoj mongod instanci, umesto na nivou pojedinačne baze podataka ili kolekcije. Ova podešavanja mogu uticati na efikasnost keširanja u zavisnosti od konkretnih zahteva aplikacija.

In-Memory Storage Engine

Počevši od verzije 3.2.6 MongoDB Enterprise-a, In-Memory Storage Engine [7] postaje dostupan za opštu upotrebu u 64-bitnim izdanjima. Ova opcija omogućava brže i efikasnije rukovanje bazom podataka jer storage engine čuva sve podatke isključivo u memoriji, osim nekih metapodataka i dijagnostičkih podataka. In-Memory storage engine ne čuva nikakve podatke na disku, uključujući konfiguracione podatke, indekse, korisničke akreditive i druge podatke.

Kao rezultat izbegavanja diskovnih I/O operacija, ovaj storage engine pruža znatno predvidljivije i niže kašnjenje pri izvršavanju operacija u bazi podataka, što rezultira bržim performansama i efikasnijim radom baze. Ovaj pristup čini ga idealnim za rad sa podacima koji zahtevaju visoke performanse i minimalno kašnjenje, kao što su real-time analize i obrada podataka u stvarnom vremenu.

Postavljanje Engine-a

In-Memory Storage Engine se može postaviti na jedan od sledeća dva načina:

- Preko komande:
  mongod --storageEngine **inMemory** --dbpath **<path>**
  Ova komanda se poziva pri samom pokretanju MongoDB servera.
  --storageEngine se postavlja na vrednost **inMemory**
  --dbpath se postavlja na **path**, odnosno to je putanja gde će se skladištiti metapodaci
- Preko konfiguracionog fajla
  storage:
  engine: **inMemory**
  dbPath: **<path>**
  Parametri su identični kao parametri koji se unose putem komande.

In-Memory Storage Engine koristi kontrolu konkurentnosti na nivou dokumenata za operacije upisivanja. Ovo znači da više klijenata može istovremeno modifikovati različite dokumente u istoj kolekciji. Ovaj pristup omogućava paralelno rukovanje podacima, što dovodi do efikasnijeg korišćenja resursa i bržih operacija nad podacima. U praksi, ovo znači poboljšane performanse u aplikacijama koje rade s velikim količinama podataka i zahtevaju brz pristup i promene u bazama podataka.

Potrošnja memorije

In-memory Storage Engine zahteva da svi njegovi podaci (uključujući indekse) moraju stati unutar memorije čija je veličina specificirana opcijom komandne linije *--inMemorySizeGB* ili podešavanjem *storage.inMemory.engineConfig.inMemorySizeGB* u konfiguracionom fajlu.

Podrazumevano, In-Memory Storage Engine koristi 50% fizičke RAM memorije umanjene za 1 GB.

Ako operacija upisivanja izazove prekoračenje specificirane veličine memorije, MongoDB će vratiti grešku:

"WT_CACHE_FULL: operation would overflow cache"

Veličina memorije se može postaviti na jedan od sledeća dva načina:

- Preko komande:
  mongod --storageEngine **inMemory** --dbpath **<path>** --inMemorySizeGB **<newSize>**
  Ova komanda se poziva pri samom pokretanju MongoDB servera.
  --storageEngine se postavlja na vrednost **inMemory**
  --dbpath se postavlja na **path**, odnosno to je putanja gde će se skladištiti metapodaci
  --inMemorySizeGB se postavlja na vrednost **newSize**, odnosno novu veličinu memorije
- Preko konfiguracionog fajla
  storage:
  engine: **inMemory**
  dbPath: **<path>**
  inMemory:
  engineConfig:
  inMemorySizeGB: **<newSize>**
  Parametri su identični kao parametri koji se unose putem komande.

Trajnost (Durability)

In-Memory Storage Engine nije perzistentan i ne čuva podatke na trajno skladište. Neperzisntentni podaci uključuju podatke aplikacije i sistemske podatke, kao što su korisnici, dozvole, indeksi. Kao takav, koncept Journal-a ili čekanja da podaci postanu trajni se ne odnosi na In-Memory Storage Engine.

MMAPv1 Storage Engine

MMAPv1 [8] je originalni MongoDB Storage Engine koji se zasniva na fajlovima mapiranim u memoriju. Izvanredan je za radne zadatke koji zahtevaju visoki obim upisa, čitanja i ažuriranja.

Journal

Da bi se osiguralo da su sve izmene u MongoDB skupu podataka trajno zabeležene na disk, MongoDB po defaultu beleži sve izmene u journal na disku. MongoDB češće zapisuje u journal nego u skup fajlova. [8]

U podrazumevanoj konfiguraciji za MMAPv1 Storage Engine, MongoDB zapisuje skup podataka na disk svakih 60 sekundi i zapisuje u journal približno svakih 100 milisekundi.

Da bi se promenio interval za zapisivanje u skup fajlova, koristi se opcija *storage.syncPeriodSecs*.

- Preko komande:
  mongod --syncPeriodSecs **<timeInMiliseconds>**
  --syncPeriodSecs se postavlja na vrednost **timeInMiliseconds**
- Preko konfiguracionog fajla
  storage:
    syncPeriodSecs: **<timeInMilliseconds>**
  Parametri su identični kao parametri koji se unose putem komande.

Za journal se koristi opcija *storage.journal.commitIntervalMs*.

- Preko komande:
  mongod --journalCommitInterval **<timeInMiliseconds>**
  --journalCommitInterval se postavlja na vrednost **timeInMiliseconds**
- Preko konfiguracionog fajla
  storage:
    journal:
      commitIntervalMs: **<timeInMiliseconds>**
  Parametri su identični kao parametri koji se unose putem komande.

Ove vrednosti predstavljaju maksimalno vreme između završetka operacije upisivanja i trenutka kada MongoDB zapisuje skup fajlova ili journal. U mnogim slučajevima MongoDB i operativni sistem refreshuju podatke na disk češće, tako da gornje vrednosti predstavljaju teorijski maksimum. Journal omogućava MongoDB-u da uspešno oporavi podatke nakon što mongod instanca izađe bez refreshovanih promena.

Karakteristike skladištenja zapisa

Svi zapisi su neprekidno smešteni na disku, a kada dokument postane veći od dodeljenog zapisa, MongoDB mora dodeliti novi zapis. Nova dodeljivanja zahtevaju da MongoDB premesti dokument i ažurira sve indekse koji se odnose na dokument, što zahteva više vremena od ažuriranja na licu mesta i dovodi do fragmentacije skladišta.

Podrazumevano, MongoDB koristi dodelu veličine stepena dvojke, tako da je svaki dokument u MongoDB-u smešten u zapisu koji sadrži sam dokument i dodatni prostor, ili padding. Padding omogućava da dokument raste kao rezultat ažuriranja, a time se smanjuje verovatnoća ponovnog dodeljivanja.

Potrošnja memorije

Sa MMAPv1 Storage Engine, MongoDB automatski koristi sav slobodan memorijski prostor na mašini kao svoju keš memoriju. Monitori resursa sistema pokazuju da MongoDB koristi mnogo memorije, ali njegova upotreba je dinamična. Ako drugom procesu iznenada zatreba polovina memorije servera, MongoDB će osloboditi keš memoriju za taj proces.

Tehnički, podsistem virtuelne memorije operativnog sistema upravlja memorijom MongoDB-a. To znači da će MongoDB koristiti što više slobodne memorije, a prema potrebi će prelaziti na swapovanje na disk. Implementacije sa dovoljno memorije da smeste radni set aplikacije u RAM postići će najbolju performansu.

GridFS

GridFS [9] je specifikacija za skladištenje i preuzimanje datoteka koje prelaze ograničenje veličine BSON-dokumenta od 16 MB. Umesto da se datoteka skladišti u jednom dokumentu, GridFS deli datoteku na delove, ili fragmente, i svaki fragment skladišti kao poseban dokument. Podrazumevano, GridFS koristi zadanu veličinu fragmenta od 255 kB; to jest, GridFS deli datoteku na fragmente od 255 kB s izuzetkom poslednjeg fragmenta, koji je samo onoliko veliki koliko je potrebno. Slično tome, datoteke koje nisu veće od veličine fragmenta imaju samo završni fragment, koristeći samo onoliko prostora koliko je potrebno plus dodatne metapodatke.

GridFS koristi dve kolekcije za skladištenje datoteka. Jedna kolekcija skladišti fragmente datoteka, a druga skladišti metapodatke datoteka.

Kada se traži datoteka od GridFS, drajver će sastaviti fragmente prema potrebi. Mogu se pristupiti informacijama iz proizvoljnih sekcija datoteka, na primer da se "preskoči" na sredinu video ili audio fajla.

GridFS je koristan ne samo za skladištenje datoteka koje prelaze 16 MB već i za skladištenje bilo kojih datoteka kojima želite pristupiti bez potrebe da učitate celu datoteku u memoriju.

Zaključak

Na osnovu analize i pregleda NoSQL baza podataka, posebno fokusirajući se na MongoDB, možemo zaključiti nekoliko ključnih tačaka koje definišu savremene sisteme za upravljanje bazama podataka.

MongoDB kao predstavnik NoSQL baza podataka nudi fleksibilnost i skalabilnost koja je neophodna za savremene aplikacije sa velikim količinama podataka i složenim zahtevima. Njegova sposobnost da skladišti podatke u različitim formatima, uključujući JSON-like dokumente, omogućava efikasno rukovanje složenim podacima.

Različiti Storage Engine-i pružaju različite benefite u zavisnosti od potreba aplikacije. WiredTiger, kao podrazumevani storage engine u MongoDB, nudi napredne karakteristike poput snimaka (snapshots), kontrolu verzija, journaling i efikasne kompresije podataka. Ove karakteristike poboljšavaju performanse i pouzdanost sistema.

In-Memory Storage Engine je pogodan za situacije gde je brzina prioritet i gde se podaci mogu u potpunosti smestiti u memoriju, dok MMAPv1 Storage Engine koristi prednosti operativnog sistema za upravljanje memorijom, što može biti korisno za aplikacije sa specifičnim potrebama.

GridFS je koristan alat za rukovanje velikim datotekama koje prelaze ograničenje veličine BSON dokumenta, omogućavajući efikasno skladištenje i dohvat delova velikih datoteka.

Konačno, važno je naglasiti da izbor odgovarajuće baze podataka i storage engine-a zavisi od specifičnih zahteva aplikacije, kao i od očekivanih performansi i pouzdanosti. MongoDB pruža širok spektar opcija i konfiguracija koje omogućavaju prilagođavanje različitim potrebama i olakšavaju rukovanje složenim podacima. Kroz pažljivo ispitivanje i optimizaciju, moguće je izvući maksimum iz MongoDB-a i njegovih različitih Storage Engine-a.

Literatura

[1] MongoDB, Database Management Systems,
Datum pristupa: 09.04.2024.
Dostupno na: https://www.mongodb.com/database-management-system

[2] Pragimtech, Relational and non relational databases
Datum pristupa: 09.04.2024.
Dostupno na: https://www.pragimtech.com/blog/mongodb-tutorial/relational-and-non-relational-databases/

[3] Craid S. Mullins, TechTarget, Database Management System (DBMS)
Datum pristupa: 09.04.2024.
Dostupno na: https://www.techtarget.com/searchdatamanagement/definition/database-management-system

[4] MongoDB, Storage Enginees,
Datum pristupa: 11.04.2024.
Dostupno na: https://www.mongodb.com/docs/manual/core/storage-engines/

[5] MongoDB, WiredTiger Storage Engine,
Datum pristupa: 12.04.2024.
Dostupno na: https://www.mongodb.com/docs/manual/core/wiredtiger/

[6] Onyancha Brian Henry, An Overview of WiredTiger Storage Engine for MongoDB, (21.05.2019.)
Datum pristupa: 14.04.2024.
Dostupno na: https://severalnines.com/blog/overview-wiredtiger-storage-engine-mongodb/

[7] MongoDB, In-Memory Storage Engine
Datum pristupa: 16.04.2024.
Dostupno na: https://www.mongodb.com/docs/manual/core/inmemory/

[8] MongoDB, MMAPv1 Storage Engine
Datum pristupa: 16.04.2024.
Dostupno na: https://www.mongodb.com/docs/v4.0/core/mmapv1/

[9] MongoDB, GridFS
Datum pristupa: 17.04.2024.
Dostupno na: https://www.mongodb.com/docs/manual/core/gridfs/