Usporedba Redis i Memcached Baza Podataka

Suljić, Antonio

Sveučilište u Rijeci, Fakultet informatike i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Antonio Suljić

Usporedba Redis i Memcached Baza Podataka

Završni rad

Mentor: doc. dr. sc. Danijela Jakšić

Rijeka, rujan 2023

Zadatak

Sveučilište u Rijeci
FIDIT Fakultet informatike i digitalnih tehnologija
www.inf.uniri.hr

Rijeka, 12. lipanj 2023.

Zadatak za završni rad

Pristupnik: Antonio Suljić

Naziv završnog rada: NoSQL baze podataka - projekt usporedbe Redis i Memcached

Naziv završnog rada na engleskom jeziku: NoSQL data stores - Redis and Memcached comparison project

Sadržaj zadatka: U ovom završnom radu usporedit će se dvije popularne nerelacijske baze podataka: Redis i Memcached. Navedene baze podataka su posebno dizajnirane za pohranu podataka u memoriji radi brzog pristupa, no imaju razlike u načinu na koji se obrađuju podaci. Rad će detaljno objasniti arhitekturu, performanse i funkcionalnosti obje baze podataka. Usporedit će se njihova sposobnost skaliranja i kako se nose s problemima visokog opterećenja te će se pružiti dublji uvid u to kako funkcioniraju nerelacijske baze podataka i koje su prednosti i nedostaci upotrebe Redis-a ili Memcached-a u različitim kontekstima.

Mentor

Doc. dr. sc. Danijela Jakšić

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 12. lipanj 2023.

A Suljić

(potpis pristupnika)

Adresa: Radmile Matejčić 2
51000 Rijeka, Hrvatska
Tel: +385(0)51 584 700
E-mail: ured@inf.uniri.hr
OIB: 64218323816
IBAN: HR1524020061400006966

Sadržaj

1. Uvod

U današnjem svijetu brzina i efikasnost postaju ključne komponente uspjeha, brza i pouzdana pohrana i pristup podacima je minimalan kriterij za moderne aplikacije. U tom kontekstu, nerelacijske baze podataka su postale nezaobilazni alat programera i inženjera za rješavanje izazova performansi i skalabilnosti. Međutim, odabir pravog alata za privremenu pohranu podataka, koji igra ključnu ulogu u brzom pristupu podacima, može biti izazovan.

Ovaj završni rad usmjerava svoju pažnju na dvije istaknute nerelacijske baze podataka - Redis i Memcached. Oba ova sustava su popularni alati za privremenu pohranu podataka, no razlikuju se u nekim aspektima, od svojih arhitektura i funkcionalnosti do primjena u stvarnom svijetu.

U prvom dijelu rada pružit će se teorijski pregled nerelacijskih baza podataka, definirajući njihove osnovne koncepte i objašnjavajući potrebu za privremenom pohranom podataka. Nakon toga, detaljno će se analizirati Redis i Memcached, istražujući njihove arhitekture, funkcionalnosti, performanse i skalabilnost. Usporedba ovih sustava pomoći će razumjeti njihove prednosti i nedostatke u različitim scenarijima.

Zaključno, ovaj rad pruža dubok uvid u Redis i Memcached te će pomoći programerima i inženjerima da donesu informirane odluke o izboru alata za privremenu pohranu podataka. Performanse, skalabilnost i jednostavnost implementacije bit će neki od ključnih faktora koji će se istražiti kako bismo razumjeli kako ovi alati mogu poboljšati brzinu i učinkovitost modernih aplikacija.

2. Teorijski Pregled

2.1. NoSQL baze podataka

NoSQL (engl. "Not Only SQL") baze podataka su skupina baza podataka koje se razlikuju od tradicionalnih relacijskih baza podataka. One su osmišljene za rješavanje specifičnih problema u području skalabilnosti, fleksibilnosti, performansi i obrade velikih količina nestrukturiranih ili polustrukturiranih podataka [1]. NoSQL baze podataka postale su popularne u posljednjem desetljeću, kako zbog naglog porasta nestrukturiranih podataka, tako i zbog potrebe za bržim i učinkovitijim pristupom tim podacima [2].

U ovom odjeljku, pružit će se općeniti pregled NoSQL baza podataka, uključujući njihove osnovne karakteristike i vrste te će se najaviti kratki uvod za koncept privremene pohrane podataka.

2.1.1 Karakteristike NoSQL Baza Podataka

- **Bez sheme:** NoSQL baze podataka omogućuju pohranjivanje podataka bez potrebe za unaprijed definiranom shemom. To znači da se svaki zapis u bazi može razlikovati u svojoj strukturi i sadržavati različite atribute, što olakšava brzu i jednostavnu promjenu modela podataka kako se zahtjevi mijenjaju.
- **Skalabilnost:** NoSQL baze podataka dizajnirane su za jednostavno skaliranje kako vertikalno (povećanje resursa pojedinačnog poslužitelja) tako i horizontalno (dodavanje novih poslužitelja u grozd). To omogućuje obradu velike količine podataka i velik broj zahtjeva uz minimalan pad u radu.
- **Replikacija i Dostupnost:** Mnoge NoSQL baze podataka podržavaju automatsku replikaciju podataka na više čvorova kako bi se osigurala visoka dostupnost i otpornost na kvarove. To znači da će podaci biti dostupni i ako jedan čvor neuspješno odgovara.
- **Horizontalno dijeljenje:** Horizontalno dijeljenje omogućuje raspodjelu podataka na više poslužitelja kako bi se smanjilo opterećenje pojedinih čvorova. To omogućuje rješavanje problema s ograničenjem resursa pojedinih poslužitelja i omogućuje linearni rast kapaciteta [1].

2.1.2 Vrste NoSQL Baza Podataka

Postoji nekoliko vrsta NoSQL baza podataka, a neke od najčešćih uključuju:

- **Ključ-Vrijednost Baze Podataka:** Ove baze podataka pohranjuju podatke u obliku ključeva i vrijednosti. Ključevi su jedinstveni identifikatori koji se koriste za dohvat podataka, a vrijednosti mogu biti bilo kakvi podaci, poput tekstualnih nizova, binarnih datoteka ili struktura podataka.
- **Dokumentne Baze Podataka:** Ove baze podataka pohranjuju podatke u obliku JSON, BSON ili XML dokumenata. Svaki dokument može sadržavati različite atribute i strukture.
- **Stupčane Baze Podataka:** Ove baze podataka organizirane su u stupce umjesto retke. Podaci su grupirani u obitelji stupaca, što omogućuje brz pristup i upite nad određenim stupcima.
- **Grafovske Baze Podataka:** Grafovske baze podataka koriste strukturu grafa za pohranu podataka. Podaci se modeliraju kao čvorovi i bridovi, omogućujući jednostavno upravljanje povezanim podacima [2].

Ovaj pregled pruža osnovni uvid u NoSQL baze podataka, a kasnije u radu usporedit ćemo dvije popularne NoSQL baze podataka, Redis i Memcached, kako bismo bolje razumjeli njihove razlike i primjene.

2.2. Potreba za Privremenom Pohranom Podataka

Privremena pohrana (eng. "caching") je tehnika privremene pohrane kopije podataka ili rezultata operacija kako bi se poboljšala brzina pristupa tim podacima i smanjila opterećenja na glavnom izvoru podataka. Ova tehnika postaje posebno važna u kontekstu velikih baza podataka i sustava s visokom razinom prometa, gdje je brzina i učinkovitost od ključne važnosti za pružanje odgovora u realnom vremenu [8].

U ovom odjeljku, razmotrit će se osnovni koncepti i mehanizam privremene pohrane podataka te zašto je ona potrebna u modernim sustavima obrade podataka.

2.2.1 Koncept Privremene Pohrane Podataka

Privremena pohrana podataka temelji se na ideji da su nedavno pristupljeni podaci vrlo vjerojatno opet zatraženi u bliskoj budućnosti. Umjesto da svaki put pristupamo izvoru podataka (npr. bazi podataka), privremena pohrana podataka omogućuje pohranu nedavno dohvaćenih rezultata u brzu i pristupačnu memoriju. Kada se ponovno zatraži isti podatak, sustav prvo provjerava je li taj podatak već dostupan u memoriji. Ako je tako, podaci se dobivaju iz memorije, što je mnogo brže nego pristup izvoru baze podataka [9].

2.2.2 Mehanizam Privremene Pohrane Podataka

Mehanizam privremene pohrane podataka obično se sastoji od tri ključna koraka:

- **Dohvat:** Kada se podaci prvi put traže, sustav provjerava memoriju kako bi vidio je li podatak već dostupan. Ako je, podatak se dohvaća i vraća kao odgovor.
- **Pohrana:** Ako podatka nema u memoriji, podaci se dohvaćaju iz izvora (npr. baze podataka) i pohranjuju u memoriju. Svaki podatak u memoriji ima jedinstveni ključ koji se koristi za identifikaciju tog podatka.
- **Istjecanje Privremenog Podatka:** Privremeno pohranjeni podaci imaju vremensko ograničenje koliko će ostati u memoriji prije nego što se smatraju zastarjelim. Istjecanje privremenog podatka osigurava se algoritmima poput LRU(Least Recently Used) [9].

2.2.3 Potreba za Privremenom Pohranom Podataka

Privremena pohrana podataka ima nekoliko ključnih prednosti koje čine njegovu primjenu nužnom:

- **Poboljšanje Performansi:** Omogućuje brži pristup podacima, podaci se dohvaćaju iz memorije(RAM) baze podataka.
- **Smanjenje Opterećenja Izvora Podataka:** Smanjuje se broj upita i zahtjeva prema izvornom izvoru podataka, čime se oslobađaju resursi i smanjuje opterećenje na sustav.
- **Poboljšanje Responzivnosti:** Krajnji korisnici dobivaju brže odgovore na svoje zahtjeve, što poboljšava korisničko iskustvo i zadovoljstvo.

Prema tome, privremena pohrana podataka je ključni element u modernim sustavima obrade podataka. Omogućuje brže i efikasnije pružanje informacija krajnjim korisnicima te smanjuje opterećenje na glavnom izvoru podataka. U idućem dijelu rada, usporedit će se dvije popularne tehnologije privremene pohrane podataka, Redis i Memcached, kako bismo bolje razumjeli njihove prednosti i primjene [10].

3. Redis

3.1. Arhitektura

Redis je dizajniran kao klijent-poslužitelj (eng. "client-server") sustav, što znači da se komunikacija odvija između klijenata i poslužitelja kako bi se izvršile naredbe i upiti nad podacima. Redis poslužitelj predstavlja centralni repozitorij podataka koji podržava rad s različitim vrstama struktura podataka. S druge strane, Redis klijent predstavlja korisničku aplikaciju koja komunicira s Redis poslužiteljem kako bi obavljala operacije dohvaćanja i pohrane podataka [11].

3.1.1 Ključ-Vrijednost Model

Redis pohranjuje podatke u memoriji kao parove ključeva i vrijednosti. Svaki ključ je jedinstven i koristi se za identifikaciju odgovarajuće vrijednosti. Vrijednost može biti različitih tipova podataka, uključujući tekstualne nizove, brojeve, liste, skupove, redove i hash mape [12].

3.1.2 Jednonitna Arhitektura

Redis koristi jednonitnu arhitekuru, što je iznenađujuće s obzirom na njegovu brzinu. Međutim, Redis se smatra jednonitnim do neke granice, zato što je njegova implementacija napravljena tako da se izbjegne nestabilnost višenitne arhitekture te maksimalno utilizira izvođenje procesa jednom niti [13].

3.1.3 Struktura Podataka

Redis podržava različite strukture podataka, što ga čini svestranom bazom podataka za različite potrebe. Neke od struktura su:

1. **Nizovi (engl. “lists”):** Povezani popis vrijednosti koji omogućuju brzo umetanje i dohvat elemenata s početka i kraja liste.
2. **Znakovni nizovi (engl. “strings”):** Niz koji čine slovno-brojčani znakovi.
3. **Skupovi (engl. “sets”):** Kolekcija različitih vrijednosti koje omogućuju operacije unije, presjeka i razlike između skupova.
4. **Sortirani Skupovi (engl. “sorted sets”):** Slično skupovima, ali svaka vrijednost ima dodijeljen poredak, što omogućuje sortiranje elemenata po tim vrijednostima.

5. **Hash Mape (engl. “hashes”):** Kolekcija polja i vrijednosti, gdje se polja mogu koristiti za jednostavno indeksiranje i dohvat vrijednosti.
6. **Bit Mape (engl. “bitmaps”):** Optimizirana struktura podataka za rad s bitovima i izvođenje bitnih operacija [14].

3.1.4 Redis Poslužitelj

Redis poslužitelj predstavlja središnji dio Redis arhitekture. Glavna uloga Redis poslužitelja je obrada naredbi poslanih od strane klijenata i upravljanje strukturama podataka u memoriji. Kada Redis poslužitelj primi naredbu od klijenta, on je izvršava u skladu s definiranim operacijama te ažurira podatke u memoriji ili vraća rezultate natrag klijentu [3].

Redis poslužitelj je implementiran u programskom jeziku ANSI C i iznimno je brz zbog toga što podatke čuva u memoriji. Međutim, kako bi se osigurala sigurnost i trajnost podataka, Redis podržava opcionalno spremanje podataka na disk [15].

3.1.5 Redis Klijent

Redis klijent je aplikacija koja komunicira s Redis poslužiteljem putem mreže kako bi izvršavala naredbe i upite nad podacima. Klijent šalje naredbe Redis poslužitelju i prima rezultate kao odgovor. Klijent može biti implementiran u raznim programskim jezicima, što omogućuje jednostavno integriranje Redis baze podataka u različite vrste aplikacija.

Klijent se povezuje s Redis poslužiteljem putem TCP/IP veze. Kad klijent šalje naredbe, Redis poslužitelj izvršava te naredbe i vraća rezultate u tekstualnom formatu. Redis klijenti obično koriste jednostavan tekstualni protokol komunikacije kako bi smanjili složenost komunikacije.

Jedan Redis poslužitelj može istovremeno podržavati više klijenata, što omogućuje paralelno izvršavanje naredbi i rukovanje višestrukim zahtjevima u stvarnom vremenu [16].

3.2. Funkcionalnosti

Redis nudi širok raspon naprednih funkcionalnosti. Ova raznolikost omogućuje Redisu da bude popularan izbor za razne primjene, od privremene pohrane podataka do obrade događaja u stvarnom vremenu. U ovom odjeljku, istražit će se neke od ključnih funkcionalnosti Redis-a.

3.2.1 Transakcije

Redis podržava transakcije kako bi omogućio skup radnji koje se izvršavaju kao jedna logička cjelina. Transakcije se koriste za osiguranje dosljednosti podataka u skupu operacija. Uobičajene naredbe poput MULTI, EXEC, DISCARD i WATCH koriste se za upravljanje transakcijama u Redisu.

Transakcije se koriste kada se želi osigurati izvršavanje više zavisnih promjena. Naprimjer, korisnik na internet trgovini kupi neki proizvod što potiče ažuriranje podataka o inventaru i kupovini. Zahtjev aplikacije je da se te dvije promjene izvrše skupa. U tom slučaju, transakcije omogućuju da se dogode ili obje promjene ili niti jedna promjena, dakle, izbjegava se događaj u kojem je samo jedan ključ ažuriran.

MULTI naredba započinje transakciju te omogućava početak izvršavanja više naredbi. WATCH naredba se može izvršiti neposredno prije MULTI naredbe. Svrha WATCH naredbe je zaključavanje ključa kako ne bi došlo do situacije u kojoj se vrijednost ključa promijenila, od strane drugog procesa, prije samog izvršavanja transakcije. EXEC naredba služi kako bi izvršili transakciju nakon što smo u nju stavili sve potrebne naredbe. DISCARD naredba služi kako bi prekinuli transakciju prije njenog završetka [17].

3.2.2 Pub/Sub Model

Redis podržava model publikacija/pretplata (pub/sub) koji omogućuje komunikaciju između različitih dijelova aplikacije ili različitih aplikacija. Klijenti mogu se pretplatiti na kanale i primati poruke koje se objavljuju na tim kanalima. Ovo je korisno za implementaciju mehanizama slanja poruka, obavijesti i sinkronizacije događaja u stvarnom vremenu [18].

3.2.3 Ključevi s Istekom

Redis omogućuje postavljanje vremenskog ograničenja za ključeve. To znači da ključevi mogu automatski isteći nakon određenog vremena. Ova funkcionalnost je korisna za privremenu pohranu podataka koji su relevantni samo za određeno vrijeme i za kontrolu veličine privremenih podataka kako bi se pravovremeno oslobodio prostor u memoriji.

Naredba TTL omogućuje uvid u preostali vijek ključa, dok naredbom EXPIRE postavljamo preostalo vrijeme ključa [19].

3.2.4 Skripte

Redis podržava izvršavanje Lua skripti izravno na poslužitelju. Ovo omogućuje kombiniranje više operacija u jednu, što može smanjiti mrežni promet i poboljšati učinkovitost.

Lua je jednostavni dinamički programski jezik koji je, kao takav, savršen za izvršavanje složenih operacija u Redis-u. Primjer složene operacije može biti praćenje statistike pregleda članaka na web stranici. Svaki put kad se članak pregleda, potrebno je ažurirati statistiku za taj članak, kao što su ukupan broj pregleda i prosječno trajanje pregleda [20].

3.2.5 Pipelining

Pipelining omogućuje klijentima slanje više naredbi Redisu bez čekanja na odgovor za svaku pojedinu naredbu. To značajno smanjuje kašnjenje i poboljšava izvedbu u situacijama gdje klijent šalje seriju naredbi.

Motivacija za uvođenje “Pipelining-a” u Redis je Round Trip Time(RTT). Ono označava vrijeme potrebno za dolazak zahtjeva od rutera do poslužitelja te odgovora od poslužitelja do rutera. Ako se uzme u obzir da se šalju zahtjevi jedan po jedan, onda je očekivano kako će se RTT nagomilati.

Samim time, očito je kako “Pipelining” rješava problem gomilanja vremena potrebnog za odgovor [21].

3.3. Performanse i Memorija

Redis se ističe svojom iznimnom brzinom i performansama. Ključni faktori koji doprinose visokim performansama Redis-a uključuju upotrebu memorije za pohranu podataka i optimizirane mehanizme za obradu naredbi. U ovom odjeljku istražit će se ključne značajke performansi i memorije Redis-a i razloge za njegovu brzinu.

3.3.1 Rad u Memoriji

Jedan od glavnih razloga za brzinu Redis-a leži u njegovom pristupu pohrani podataka - koristi RAM (Random Access Memory) za pohranu. Umjesto pisanja podataka na disk, Redis zadržava sve svoje podatke u memoriji, čime eliminira kašnjenje koje se obično javlja kod operacija pisanja i čitanja s diska. Ovo omogućuje Redisu postizanje iznimno niskih vremena odaziva za operacije čitanja i pisanja podataka [15].

3.3.2 Optimizacija Memorije

Budući da RAM ima ograničeni kapacitet, potrebno je voditi računa o veličini podataka kako bi se izbjegao preveliki zahtjev za memorijom koji bi mogao dovesti do problema s performansama ili iscrpljivanja memorije.

Jedan od načina kako se mogu preventirati problemi s veličinom podataka je **eksplicitno postavljanje limita memorije;** Moguće je postaviti gornji limit za korištenje memorije u Redis konfiguraciji kako bi se osiguralo da Redis ne premašuje dostupnu RAM memoriju.

Isto tako, Redis koristi LRU(engl. “least recently used”) algoritam za **iseljavanje ključeva** koji obavlja sljedeće:

- Klijent pošalje podatke
- Redis pregleda dostupnost memorije, ako je dostupnost manja od predviđene, brišu se stari ključevima prema odabranoj konfiguraciji [22]

Neke od dostupnih opcija pri odabiru konfiguracije **iseljavanja ključeva:**

- **noeviction:** Novi podaci bit će odbačeni ako je limit memorije dosegnut
- **allkeys-lru:** Briše samo najstarije ključeve

- **volatile-lru:** Briše najstarije ključeve koji imaju atribut **expire** u vrijednosti **true**
- **allkeys-random:** Nasumično briše podatke kako bi se oslobodilo mjesta u memoriji [23]

3.3.3 Optimizacija Struktura Podataka

Redis je osmišljen za rad s naprednim strukturama podataka, što dodatno poboljšava njegove performanse. Strukture podataka, poput nizova, skupova i hash mapa, omogućuju brz pristup i izmjenu podataka. Ove strukture su implementirane kako bi se minimizirala složenost operacija, čime se postižu visoke brzine i efikasnost [14].

3.3.4 Virtualna Memorija i Perzistencija

Redis omogućuje opcionalno spremanje podataka na disk kako bi se izbjegao gubitak podataka u slučaju nestanka napajanja ili neočekivanog prekida. Također, može se konfigurirati kao sustav s virtualnom memorijom, gdje se rijetko korišteni podaci automatski prebacuju na disk kako bi se oslobodio prostor u RAM-u.

Opcije za perzistenciju uključuju "snapshot" (RDB - Redis DataBase) mehanizam i "append-only file" (AOF) mehanizam.

- **RDB mehanizam** omogućuje periodično snimanje trenutnog stanja baze podataka na disk u obliku binarnog "snapshot"-a. Ovo je brz i učinkovit mehanizam, ali podaci mogu biti izgubljeni ako se dogodi neočekivani prekid rada.
- **AOF mehanizam** bilježi sve naredbe koje mijenjaju stanje baze podataka u tekstualnu datoteku. Ova datoteka se redovito sinkronizira s diskom. AOF mehanizam pruža trajnost podataka jer omogućuje obnovu baze podataka nakon neočekivanog prekida rada, ali može biti malo sporiji od RDB mehanizma zbog dodatnog pisanja na disk [24].

3.3.5 Konzistencija

Redis ne obećava strogu konzistenciju. To znači da postoji mogućnost gubitka dijela podataka, zato što Redis koristi asinkronu replikaciju prema čvorovima. Situacija u kojoj će asinkrona replikacija uzrokovati gubitak podataka je sljedeća:

- Klijent šalje podatke čvoru B

- Čvor B vraća odgovor u kojem potvrđuje primitak
- Čvor B šalje podatke prema replikama B1, B2, B3

Dakle, problem u ovoj situaciji je taj da Čvor B ne zahtijeva povratni odgovor od svojih replika. Što implicira da u slučaju da se čvor B ugasi zbog greške prije nego pošalje podatke prema replikama, ona replika koja preuzme ulogu čvora B neće sadržavati podatke koje je klijent poslao [26].

3.4. Skalabilnost

Skalabilnost je ključan aspekt kada je riječ o obradi velike količine podataka i postizanju visokih performansi s povećanim prometom. Redis nudi nekoliko mehanizama za skaliranje, što ga čini pogodnim za projekte s različitim zahtjevima za resursima.

3.4.1 Vertikalno Skaliranje

Ovaj pristup uključuje nadogradnju samog Redis poslužitelja tako da se namjeni više resursa kao što su CPU, RAM i prostor za disk. To može poboljšati performanse za pojedinačni poslužitelj, ali postoji ograničenje koliko resursa možete dodati na jedan poslužitelj [25].

3.4.2 Horizontalno Skaliranje

Ovo uključuje dodavanje više Redis poslužitelja kako bi se podaci raspodijelili i omogućila veća paralelna obrada. Postoje dvije glavne metode horizontalnog skaliranja u Redisu:

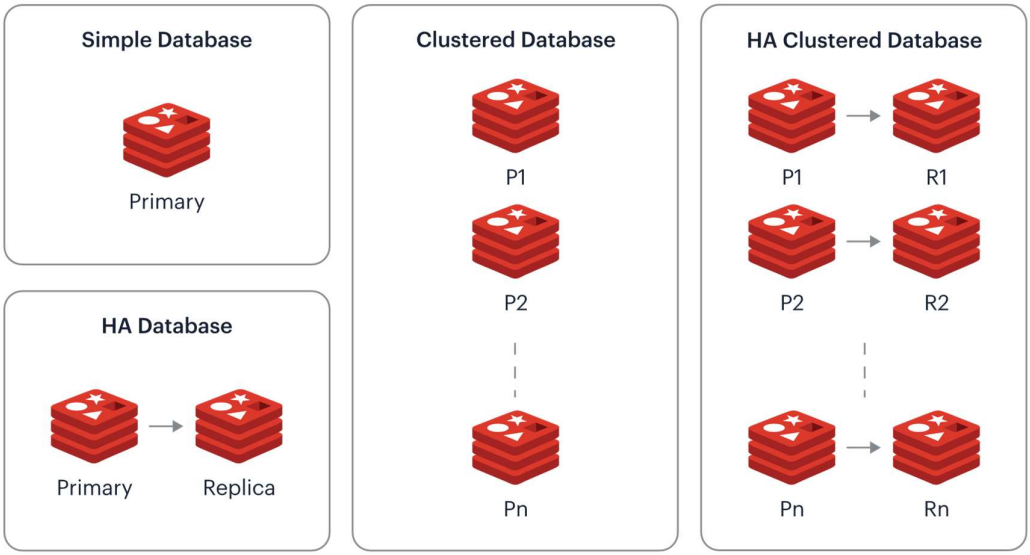
- **Distribucija Podataka(engl. “sharding”)**: Distribucija Podataka podataka znači da se podaci dijele na više poslužitelja (često nazvanih shardovi) prema određenom pravilu. Svaki shard drži dio podataka, a aplikacija mora znati kojim shardom upravlja za koje podatke. Ovo omogućuje raspodjelu ključeva i upita među više čvorova, čime se postiže bolja ravnoteža opterećenja
- **Repliciranje**: Ova metoda uključuje stvaranje više replika primarnog Redis poslužitelja. Replike sadrže kopije podataka iz primarnog poslužitelja i koriste se za visoku dostupnost i skalabilnost čitanja. Primarni poslužitelj odgovoran je za pisanje,

dok se zadaci čitanja mogu distribuirati između replika. Sekundarni čvorovi mogu preuzeti ulogu primarnog čvora ako dođe do problema na poslužitelju. Tako se osigurava kontinuirani pristup podacima i minimalno vrijeme nedostupnosti u slučaju kvara [26].

3.4.3 Vrste Arhitektura

Razlikujemo neke od arhitektura:

- **Jednostavna arhitektura** sadrži jedan primarni čvor.
- **Jednostavna visoko dostupna arhitektura** sadrži jedan primarni čvor i njegove replike.
- **Grozd arhitektura** je grozd koji sadrži više shard čvorova.
- **Visoko dostupna grozd arhitektura** je grozd koji sadrži više shard čvorova zajedno s replikama tih čvorova [27].



Slika 1. Arhitektura Redis Sustava (Izvor: https://redis.com/redis-enterprise/technology/redis-enterprise-cluster-architecture/)

3.5. Sigurnost i Autorizacija

Sigurnost igra ključnu ulogu u svakoj bazi podataka, uključujući i Redis. Redis nudi nekoliko mehanizama za osiguravanje sigurnosti podataka i ograničavanje pristupa kako bi se zaštitili podaci od neovlaštenog pristupa ili zlonamjernih napada.

3.5.1 Autentikacija

Redis podržava jednostavnu autentikaciju putem lozinki. Ova značajka omogućuje postavljanje lozinke koju klijent mora pružiti prije nego što može pristupiti Redis poslužitelju. Postavljanje lozinke može se obaviti u konfiguracijskoj datoteci Redis poslužitelja. Klijenti koji žele pristupiti Redis poslužitelju moraju pružiti ispravnu lozinku pomoću AUTH naredbe kako bi bili autentificirani i mogli izvršavati naredbe.

Međutim, važno je napomenuti da ova metoda autentikacije nije kriptografski najjača opcija zato što se lozinka skladišti u tekstualnom formatu. Ako je potrebna sigurnija autentikacija, preporučuje se korištenje SSL/TLS enkripcije i drugih naprednijih metoda autentikacije [28].

3.5.2 Mrežno ograničenje

Redis omogućuje ograničavanje mrežnih sučelja na kojima osluškuje. Ova značajka omogućuje postavljanje mrežne adrese na kojoj Redis sluša dolazne zahtjeve. Na taj način se može ograničiti pristup Redis poslužitelju samo na određenoj mreži, što povećava sigurnost i sprječava neovlašteni pristup s javnih mreža [29].

3.5.3 Role i Autorizacija

Redis uključuje napredne mogućnosti upravljanja autorizacijom i ulogama korisnika. Pomoću ovih značajki, administratorski korisnici mogu postaviti granice i dozvole pristupa za različite korisnike i klijente. Ulogama se može upravljati putem posebnih naredbi, što omogućuje precizno definiranje što pojedini klijenti ili korisnici mogu raditi unutar Redis baze podataka.

Ova razina kontrole pristupa pruža bolju sigurnost i omogućuje administraciju različitih razina pristupa, što je osobito važno u višekorisničkim okruženjima [30].

3.6. Kompatibilnost

Redis je razvijen kao višenamjenski sustav koji podržava različite programske jezike i platforme. Ova sekcija istražuje kompatibilnost Redis-a s raznim tehnologijama i jezicima, što ga čini pristupačnim i fleksibilnim rješenjem za različite razvojne scenarije.

3.6.1 Podrška za Različite Programske Jezike

Redis nudi veliku podršku za različite programske jezike. Ovo je omogućeno putem klijentskih biblioteka koje su dostupne za mnoge popularne jezike. Ove klijentske biblioteke pružaju programerima jednostavan način za komunikaciju s Redis poslužiteljem iz njihovih aplikacija.

Evo nekoliko primjera popularnih klijentskih biblioteka za Redis u različitim programskim jezicima, neki od njih su:

- Python: redis-py
- Node.js: node-redis
- Java: Jedis
- .NET: NRedisStack

Uz to, Redis nudi i jednostavnu implementaciju koja omogućuje komunikaciju s Redis poslužiteljem koristeći osnovne TCP naredbe, što znači da se može komunicirati s Redis poslužiteljem iz bilo kojeg programskog jezika koji podržava TCP protokol [31].

3.6.2 API Kompatibilnost(Application Programming Interface)

Redis implementira jednostavan API koji olakšava integraciju s aplikacijama. API naredbe za pohranu, dohvaćanje i brisanje ključ-vrijednost parova su intuitivne i lako razumljive, što olakšava razvoj aplikacija koje koriste Redis [32].

3.6.3 Redis CLI

Redis sadrži interaktivnu naredbenu ljusku kojom korisnici mogu upravljati Redis poslužiteljem iz naredbene linije operativnog sustava. Neke od osnovnih naredbi su:

- **SET**: Postavlja vrijednost za određeni ključ
- **GET**: Dohvaća vrijednost pohranjenu pod određenim ključem
- **DEL**: Briše ključ i njemu pridruženu vrijednost
- **INCR**: Povećava vrijednost pohranjenu pod određenim ključem za 1 ili za broj koji specificiramo u argumentu
- **DECR**: Smanjuje vrijednost pohranjenu pod određenim ključem za 1 ili za broj koji specificiramo u argumentu
- **EXPIRE**: Postavlja vremensko ograničenje (TTL) za ključ
- **HSET**: Postavlja vrijednost za određeni ključ u hash mapi
- **HGET**: Dohvaća vrijednost za određeni ključ u hash mapi
- **RPUSH**: Dodaje elemente na kraj liste
- **LRANGE**: Dohvaća elemente liste unutar određenog raspona
- **SADD**: Dodaje članove u skup
- **SMEMBERS**: Dohvaća sve članove skupa
- **ZADD**: Dodaje članove u uređeni skup s pripadajućim ocjenama
- **ZRANGE**: Dohvaća članove uređenog skupa unutar određenog raspona ocjena [33]

3.6.4 Podrška za Različite Platforme

Redis je dizajniran da bude neovisan o platformi, što znači da se može izvoditi na raznim operacijskim sustavima. Redis je testiran i podržan na popularnim operacijskim sustavima kao što su:

- Linux
- macOS
- Windows

To omogućuje fleksibilnost u odabiru platforme za implementaciju Redis poslužitelja prema specifičnim potrebama projekta [34].

3.7. Prednosti i Nedostaci Redis-a

Redis nudi mnoge prednosti, ali također ima i neke nedostatke. U ovom odjeljku navest će se ključne prednosti i nedostatci Redis-a za potrebe cjelovitog uvida u ovu bazu podataka.

3.7.1 Prednosti Redis-a

- **Visoke Performanse:** Redis je poznat po iznimnoj brzini i performansama zbog činjenice da podatke pohranjuje u memoriji, što omogućuje brze odazive na upite i naredbe.
- **Jednostavnost Korištenja:** Redis nudi jednostavan i intuitivan API, koji je lako naučiti i koristiti. Ovo olakšava razvoj aplikacija koje koriste Redis kao backend.
- **Podrška za Različite Strukture Podataka:** Redis nudi različite strukture podataka poput nizova, skupova, hash mapa i drugih, što ga čini pogodnim za različite vrste aplikacija.
- **Replikacija za Visoku Dostupnost:** Redis podržava replikaciju koja omogućuje stvaranje primarnog i sekundarnih čvorova za osiguranje visoke dostupnosti podataka.
- **Distribucija Podataka s Redis Clusterom:** Redis Cluster omogućuje horizontalno skaliranje podataka i upita kako bi se zadovoljile potrebe projekata s velikim prometom i količinom podataka.

3.7.2 Nedostaci Redis-a

- **Ograničen Kapacitet Memorije:** Budući da Redis pohranjuje sve podatke u memoriji, kapacitet memorije može biti ograničavajući faktor za velike količine podataka.
- **Potencijalni Gubitak Podataka:** Budući da Redis nudi opcionalno pohranjivanje podataka na disku, postoji potencijal za gubitak podataka u slučaju neočekivanog prekida rada.
- **Nema Naprednih Query Mogućnosti:** Redis nije dizajniran za napredne upite i analize podataka kao što su SQL baze podataka. To može biti ograničavajući faktor za projekte koji zahtijevaju kompleksne upite.

- **Ovisnost o Repliciranju i Grozdovima:** Visoka dostupnost i skalabilnost Redis-a zahtijevaju konfiguraciju replikacije, Redis Clustera ili oboje, što može povećati kompleksnost implementacije i održavanja [35].

3.8. Primjeri korištenja Redis-a

Redis je pronašao primjenu u raznim kompanijama i projektima diljem svijeta. Njegove brze performanse i raznolike funkcionalnosti omogućuju mu da se koristi u različitim scenarijima kako bi se poboljšala učinkovitost i skalabilnost aplikacija. U ovom odjeljku istražit će se neke od stvarnih primjera korištenja Redis-a od strane poznatih kompanija.

1. Twitter

Twitter koristi Redis za različite svrhe, uključujući privremenu pohranu tweetova i korisničkih profila kako bi smanjili vrijeme potrebno za dohvaćanje tih podataka iz baze podataka. Također, Redis se koristi kao red poruka za obradu i distribuciju tweetova u stvarnom vremenu [36].

2. GitHub

GitHub, kao jedan od najvećih i najutjecajnijih servisa za upravljanje izvornim kodom, koristi Redis kao ključnu komponentu svoje infrastrukture kako bi pružio brze i pouzdane usluge svojim korisnicima. GitHub koristi Redis kao trajno pohranjivanje ključeva i vrijednosti za potrebe pohranjivanja informacija o usmjeravanju (routing information) te različitih drugih podataka [38].

3. Craigslist

Craigslist, jedan od najpoznatijih online oglasnika za razne usluge i proizvode, koristi Redis kako bi poboljšao performanse i odgovorio na veliki broj zahtjeva korisnika [39].

4. Stackoverflow

Stackoverflow, popularna internetska platforma koja omogućava korisnicima postavljanje pitanja i dobivanje odgovora na različite teme vezane uz programiranje, računalne znanosti i tehničke izazove, koristi Redis na različite načine kako bi poboljšao

performanse svoje platforme. To uključuje privremenu pohranu podataka o često izvođenim upitima kako bi se ubrzao pristup podacima, pohrana sesija i autentikacija korisnika, praćenje statistike, upravljanje redom poruka za asinkrone zadatke i mnoge druge svrhe [40].

4. Memcached

4.1. Arhitektura

Memcached je jednostavna i visoko performantna memorijska baza podataka koja se koristi za privremenu pohranu podataka kako bi se ubrzalo dohvaćanje i obrada podataka u web aplikacijama i drugim sustavima. Njegova arhitektura je jednostavna i usmjerena na pohranu i dohvaćanje ključ-vrijednost parova u RAM memoriji [41].

4.1.1 Jednostavna Struktura Podataka

Memcached koristi jednostavan model ključ-vrijednost pohrane podataka. Ova jednostavna struktura podataka omogućuje brže izvršavanje operacija, jer nema složenih veza između različitih vrsta podataka. Klijenti lako mogu izvršavati upite pomoću jednostavnih API naredbi, što olakšava integraciju i korištenje Memcached-a.

Naprimjer, korištenjem python biblioteke *memcache* postavljamo ključ-vrijednost pomoću metode *set("ključ1", "vrijednost1")*. Slično i dohvaćamo vrijednost ključa pomoću metode *get("ključ1")* [42].

4.1.2 Višenitna Arhitektura

Memcached koristi višenitnu arhitekturu kako bi omogućio bolje iskorištavanje višejezgrenih procesora i poboljšava odziv i performanse aplikacije. Svaki čvor u Memcached grozdu može istovremeno obraditi više upita i zahtjeva, što poboljšava skalabilnost i omogućuje visok promet bez pada performansi [43].

4.1.3 Otpornost na Kvarove

Zbog svoje jednostavne strukture, Memcached ne podržava replikaciju, stoga ako dođe do greške na poslužitelju, vrlo vjerojatno će doći do gubitka podataka s tog čvora. Kako bi se umanjio gubitak podataka, preporuka je da grozd sadrži veći broj čvorova, što je izvedivo pomoću mogućnosti horizontalne distribucije podataka [44].

4.2. Funkcionalnosti

Memcached svoju popularnost duguje brzini i jednostavnosti, iz tog razloga nudi manji opseg funkcionalnosti. U ovom odjeljku istražit će se ključne funkcije Memcached-a.

4.2.1 Privremena Pohrana

Glavna funkcionalnost Memcached-a je privremena pohrana podataka. Omogućuje pohranjivanje često korištenih podataka u brzu RAM memoriju kako bi se ubrzao pristup tim podacima. Privremena pohrana pomaže smanjiti vrijeme potrebno za dohvaćanje podataka iz spore i skladištene baze podataka, čime se poboljšava performansa aplikacija [41].

4.2.2 Istjecanje Podataka

Memcached omogućuje postavljanje vremena isteka za pohranjene podatke. Ovo znači da se podaci automatski uklanjaju iz memorijske baze podataka nakon što istekne određeno vrijeme. Ova značajka je korisna za osiguravanje da se privremeno pohranjeni podaci redovito osvježavaju i oslobađaju memoriju za nove podatke [42].

4.2.3 Distribucija Podataka(Sharding)

Memcached nudi mogućnost distribucije podataka na više čvorova, što omogućuje horizontalno skaliranje i uravnoteženost opterećenja. Svaki čvor u Memcached grozdu neovisno upravlja svojim dijelom podataka, a klijenti koriste algoritme za raspodjelu opterećenja kako bi pristupili odgovarajućem čvoru. Ova distribuirana arhitektura omogućuje efikasno korištenje resursa i visoku dostupnost [44].

4.3. Performanse i Memorija

Memcached je poznat po iznimnim performansama i brzini pristupa podacima zahvaljujući svom jednostavnom modelu pohrane u memoriji. U ovom odjeljku istražit će se ključne značajke performansi i memorije Memcached-a koje ga čine popularnim izborom za privremenu pohranu podataka.

4.3.1 Brzina Dohvaćanja i Pohrane Podataka

Memcached je dizajniran za brzo pohranjivanje i dohvaćanje podataka. Svi podaci se pohranjuju u RAM memoriji, što omogućuje iznimno brzi pristup podacima bez potrebe za sporo čitanje i pisanje na disk. Brzina dohvaćanja i pohrane ključ-vrijednost odvija se ispod milisekunde pri normalnim uvjetima [42].

4.3.2 Očekivano Ponašanje Pri Iznimno Visokim Opterećenjem

Memcached je sposoban rukovati sa 200,000+ zahtjeva po sekundi ukoliko se radi o jakom poslužitelju s iznimnom brzinom interneta. Povećanjem performansi poslužitelja, Memcached može potencijalno podnijeti dodatno veću količinu zahtjeva. Iako se radi o vrlo sporom poslužitelju, performanse će biti zadovoljavajuće[43].

4.3.3 Zadržavanje Procesa

Gotovo sve Memcached operacije su složenosti O(1), što označava da operacije imaju konstantnu vremensku složenost. To znači da je vrijeme potrebno za izvršavanje operacije uvijek isto i ne ovisi o veličini unesenog podatka. Bilo kakvo kašnjenje je najvjerojatnije rezultat slabe konfiguracije hardvera ili problem mrežnog pristupa [43].

4.3.4 Preciznost Istjecanja Ključa

S obzirom na to da Memcached svake sekunde ažurira unutarnju vrijednost sata, istjecanje ključa se izvrši do unutar sekunde preciznosti. To znači da će se ključ obrisati sekundu prije, točno u definirano vrijeme ili sekundu kasnije od definirane vrijednosti [43].

4.3.5 Rukovanje Neuspjelim *set* Operacijama

Postoje uvjeti u kojima će operacija *set* rezultirati greškom:

- Nedovoljno memorije
- Preveliki objekt
- Ključ već postoji

U većini slučajeva kada se poziva *set* nad ključem koji već postoji, greška će pobrisati stari ključ. Ako se koristi *set*, onda Memcached naslućuje da je namjera korisnika ažurirati ključ te da se u njemu nalaze zastarjeli podaci [43].

4.3.6 Efikasna Upotreba Memorije

Memcached koristi efikasne algoritme poput Least Recently Used (LRU) za upravljanje memorijom kako bi se osiguralo što učinkovitije korištenje dostupne RAM memorije. Upravljanje memorijom uključuje alokaciju, oslobađanje i ponovno korištenje memorije za pohranu podataka. Ovo osigurava minimalno zadržavanje zastarjelih podataka i maksimalno iskorištenje raspoloživih resursa [45].

4.3.7 Kapacitet Memorije

Kapacitet memorije Memcached-a ovisi o resursima dostupnim na čvorovima u grozdu. Svaki čvor ima svoju vlastitu RAM memoriju za pohranu podataka. Stoga, ukupan kapacitet memorije u Memcached grozdu određen je ukupnim zbrojem RAM memorije svih čvorova. Ako se grozd sastoji od više čvorova, ukupan kapacitet memorije će biti veći, što omogućuje pohranu veće količine podataka [43].

4.3.8 Fleksibilno Konfiguriranje Memorije

Memcached omogućuje konfiguriranje i prilagodbu veličine memorije na razini pojedinog čvora. To znači da se može postaviti maksimalna veličina memorije koju će svaki čvor koristiti za pohranu podataka. Ovo prilagodljivo konfiguriranje omogućuje optimizaciju upotrebe resursa prema potrebama aplikacije [46].

4.4. Skalabilnost

Skalabilnost Memcached-a je ključna značajka koja omogućuje ovom sustavu brzo i pouzdano odgovarati na rastuće zahtjeve aplikacija. Uz horizontalno skaliranje i replikaciju podataka, Memcached nudi dodatne značajke koje pridonose njegovoj skalabilnosti.

4.4.1 Sljedivost(engl. „consistent hashing“)

Sljedivost je tehnika raspodjele ključeva i podataka unutar Memcached grozda. Umjesto tradicionalnog hashinga koji može uzrokovati promjenu lokacije podataka prilikom dodavanja ili uklanjanja čvorova, sljedivost osigurava minimalne promjene lokacije podataka pri izmjenama grozda. To omogućuje bolju ravnotežu opterećenja i smanjuje potrebu za ponovnim raspodjelom podataka pri promjeni grozda [47].

4.4.2 Iseljenje Ključeva(engl. „key eviction“)

Kada se grozd približi maksimalnom kapacitetu, a novi podaci dolaze za pohranu, Memcached ima mogućnost izbacivanja dijela podataka iz memorije pomoću LRU algoritma kako bi oslobodio prostor za nove podatke. Ova značajka omogućuje prilagodbu veličine memorije bez prekida rada i osigurava nesmetan rad sustava i uz puno iskorištenje dostupnih resursa [48].

4.4.3 Horizontalno Skaliranje

Memcached postiže skalabilnost pomoću horizontalnog skaliranja. To znači da se grozd Memcached-a sastoji od više nezavisnih čvorova, a ne samo jednog velikog čvora. Svaki čvor može biti dodan ili uklonjen iz grozda kako bi se prilagodio promjenama u zahtjevima. Ovakva arhitektura omogućuje efikasno korištenje resursa i bolje iskorištavanje dostupnih kapaciteta [7].

4.4.4 Jednostavno Dodavanje Čvorova

Dodavanje novih čvorova u Memcached grozd relativno je jednostavan postupak. S obzirom na horizontalnu arhitekturu, novi čvorovi mogu se lako uključiti u grozd bez potrebe za zaustavljanjem cijelog sustava. Ovo omogućuje dinamičko prilagođavanje kapaciteta kako bi se rješavali zahtjevi aplikacije [7].

4.4.5 Vertikalno Skaliranje

Vertikalno skaliranje se odnosi na povećanje resursa (poput procesora, memorije, diska) na pojedinačnom čvoru (poslužitelju) kako bi se poboljšao ukupan kapacitet i performanse Memcacheda.

Međutim, važno je napomenuti da Memcached nije tako lako skalabilan vertikalno kao neki drugi sustavi, jer se svaka promjena mora provesti na pojedinačnom čvoru. Horizontalno skaliranje, s dodavanjem više čvorova, često je efikasniji način za rješavanje potreba za većom skalabilnošću i performansama [49].

4.5. Sigurnost i Autorizacija

S obzirom na to da je Memcached projektiran s fokusom na brzinu i jednostavnost, on ne nudi složene mehanizme sigurnosti i autorizacije. U osnovnoj konfiguraciji, Memcached ne zahtijeva autentikaciju, što ga čini ranjivim na neovlašteni pristup. No, postoji nekoliko načina kako se može poboljšati sigurnost Memcached grozda.

4.5.1 IP Filtriranje

Jedan od pristupa kako poboljšati sigurnost Memcached-a je korištenje IP filtriranja (IP whitelisting). To znači da se konfigurira Memcached grozd tako da dopušta samo pristup iz određenih pouzdanih IP adresa. Na ovaj način, samo klijenti s ovlaštenih adresa mogu pristupiti grozdu [50].

4.5.2 Korištenje Proxy Poslužitelja

Kako bi se osigurala bolja kontrola pristupa, moguće je koristiti Memcached-ov proxy poslužitelj ispred Memcached grozda. Proxy poslužitelj može provesti autentikaciju i autorizaciju klijenata prije nego što proslijede zahtjeve prema grozdu. Ovo pruža mogućnost fleksibilnije kontrole pristupa i upravljanje dozvoljenim operacijama [51].

4.5.3 Implementacija Vlastitih Rješenja

Za organizacije koje zahtijevaju naprednije sigurnosne mehanizme, moguće je implementirati vlastita rješenja za sigurnost i autorizaciju. To može uključivati integraciju s postojećim sustavima autentikacije, sustava jedinstvene prijave (SSO), kako bi se omogućilo provođenje autentikacije prije pristupa Memcached grozdu.

4.6. Kompatibilnost

Memcached je dizajniran s ciljem jednostavne integracije s različitim programskim jezicima i platformama. Ova kompatibilnost čini Memcached popularnim izborom za privremenu pohranu podataka u raznim vrstama aplikacija. U ovom odjeljku istražit će se ključni aspekti vezani uz kompatibilnost Memcached-a.

4.6.1 Podržani Programski Jezici

Memcached ima podršku za širok raspon programskih jezika, uključujući, ali ne ograničavajući se na, sljedeće:

- Java
- Python
- PHP
- C#
- Node.js
- Go

Ovo omogućuje razvojnim programerima da integriraju Memcached u svoje aplikacije bez obzira na programski jezik koji koriste [52].

4.6.2 Platforme i Operacijski Sustavi

Memcached je radi na većini sustava, uključujući:

- Linux
- macOS
- Unix

To znači da se Memcached može koristiti na različitim poslužiteljima i računalnim okruženjima bez problema s kompatibilnošću. Iznimno tome, Memcached ne pruža nativnu mogućnost rada na Windows sustavu, već to zahtijeva poseban postupak [53].

4.6.3 Biblioteke

Za olakšavanje integracije, postoji mnogo biblioteka koji omogućuju razvojnim programerima da jednostavno komuniciraju s Memcached-om iz svojih aplikacija. Te biblioteke dostupne su za različite programske jezike, što olakšava korištenje Memcached-a u aplikacijama napisanim u različitim tehnologijama.

4.6.4 API Kompatibilnost(Application Programming Interface)

Memcached implementira jednostavan API koji olakšava integraciju s aplikacijama. API naredbe za pohranu, dohvaćanje i brisanje ključ-vrijednost parova su intuitivne i lako razumljive, što olakšava razvoj aplikacija koje koriste Memcached.

4.6.5 Memcached CLI(Command-Line Interface)

Memcached pruža mogućnost korištenja naredbenu ljusku kako bi se omogućio ručni pregled i upravljanje ključevima u bazi podataka. Neke od osnovnih naredbi koje se koriste u naredbenoj ljusci su:

- ***get*** – dohvaća vrijednost ključa.
- ***set*** – ažuriranje postojećeg ključa.
- ***add*** – dodavanje novog ključa.
- ***delete*** – brisanje postojećeg ključa [54].

4.7. Prednosti i Nedostaci

Memcached, kao i svaka tehnologija, ima svoje prednosti i nedostatke koje je važno uzeti u obzir prilikom odabira baze podataka za konkretnu primjenu. U ovom odjeljku će se istražiti prednosti i nedostaci Memcached-a.

4.7.1 Prednosti

1. **Brzina**: Memcached je iznimno brz zbog svoje jednostavne arhitekture i činjenice da podatke drži u memoriji. To omogućuje brze operacije čitanja i pisanja podataka, što je ključno za ubrzavanje performansi aplikacija.
2. **Jednostavnost**: Memcached je jednostavan za postavljanje i korištenje. Njegova jednostavna arhitektura omogućuje brzo integriranje u postojeće aplikacije.

3. **Horizontalna Skalabilnost**: Memcached podržava horizontalno skaliranje, omogućujući dodavanje novih čvorova kako bi se povećao ukupni kapacitet i poboljšala skalabilnost.
4. **Nisko Kašnjenje**: Zahvaljujući držanju podataka u memoriji, Memcached osigurava nisko kašnjenje i brze odazive na upite.
5. **Visoka Dostupnost**: Korištenjem replikacije podataka, Memcached osigurava visoku dostupnost u slučaju kvara jednog čvora.

4.7.2 Nedostaci

1. **Nema Autentikacije**: Memcached na tvorničkim postavkama ne nudi mehanizme autentikacije, što ga čini osjetljivim na neovlašteni pristup. Dodatne mjere sigurnosti trebaju se implementirati kako bi se osigurao samo ovlašten pristup grozdu.
2. **Nema Trajnog Pohranjivanja**: Memcached je namijenjen samo za privremenu pohranu podataka u memoriji i ne pruža trajno pohranjivanje podataka na disku. U slučaju pada sustava, podaci se gube i potrebno ih je ponovno generirati.
3. **Memorija kao Ograničavajući Faktor**: Budući da Memcached drži sve podatke u memoriji, količina dostupne memorije postaje ograničavajući faktor. Veliki skupovi podataka mogu zahtijevati značajnu količinu memorije, što može predstavljati izazov za velike sustave.
4. **Manjak funkcionalnosti**: Memcached-ova jednostavnost je pridonijela činjenici da nema puno funkcionalnosti.
5. **Ovisnost o Aplikaciji**: Memcached ne pruža složene funkcije baze podataka, poput indeksiranja i pretraživanja. Aplikacija mora pažljivo upravljati i organizirati podatke kako bi ih učinkovito koristila [55].

4.8. Primjeri korištenja Memcached-a

Memcached se široko koristi u pravom svijetu kako bi se poboljšale performanse i smanjio pritisak na backend infrastrukturu. Evo nekoliko primjera kako kompanije koriste Memcached za poboljšanje performansi i skalabilnosti svojih aplikacija.

1. Facebook

Facebook, jedna od najvećih društvenih mreža na svijetu, koristi Memcached za privremenu pohranu rezultata upita na bazu podataka. Kada korisnici pristupaju profilima prijatelja ili njihovim vlastitim profilima, često se generira veliki broj upita na bazu podataka kako bi se dohvatili podaci. Memcached drži rezultate tih upita u memoriji, što smanjuje broj stvarnih upita na bazu podataka i ubrzava prikazivanje sadržaja korisnicima [56].

2. YouTube

YouTube, najveća platforma za dijeljenje videozapisa, također koristi Memcached za privremenu pohranu sadržaja videozapisa. Kada korisnik pristupi videozapisu, Memcached drži kopije videozapisa i metapodataka u memoriji kako bi se smanjilo opterećenje na backend sustav i osigurala brza isporuka sadržaja korisnicima [57].

3. Airbnb

Airbnb, popularna platforma za rezervaciju smještaja, koristi Memcached za privremenu pohranu rezultata pretrage smještaja. Kada korisnici pretražuju dostupne smještaje za određeni datum i lokaciju, Memcached drži rezultate pretrage u memoriji kako bi se smanjio broj upita na backend bazu podataka i osiguralo brže i učinkovitije pretraživanje [58].

4. Reddit

Reddit, popularna platforma za dijeljenje i raspravu o sadržaju, koristi Memcached za privremenu pohranu korisničkih podataka o sesiji. Kada korisnik pristupa Redditu i prijavljuje se, Memcached drži informacije o sesiji u memoriji, što omogućuje brže dohvaćanje i upravljanje sesijama, smanjujući opterećenje na backend sustav [59].

5. Konkretna usporedba Redis-a i Memcached-a i Jednostavan Primjer Korištenja

5.1. Usporedba Redis-a i Memcached-a

U ovom odjeljku usporedit će se Redis i Memcached. Kriteriji, koji se nalaze u *Tablici 1*., odabrani su na temelju jednog izvora te se smatraju važnim aspektima pri razmatranju ove dvije baze podataka za različite primjene.

Kroz te kriterije usporedit će se:

- **Arhitektura i Struktura Podataka**: Ovaj kriterij odnosi se na način na koji su podaci organizirani i kako se pristupa njima u Redisu i Memcachedu. Arhitektura igra ključnu ulogu u brzini i fleksibilnosti baze podataka.
- **Funkcionalnosti**: Funkcionalnosti se odnose na razne mogućnosti i alate koje nude ove baze podataka. To uključuje mehanizam zaključavanja, transakcije, ključ s istekom, itd. Ova kategorija pomaže razumjeti opseg upotrebe ovih baza podataka.
- **Performanse i Memorija**: Performanse su od važnog značaja. Ovaj kriterij uključuje brzinu čitanja i pisanja podataka, kao i sposobnost rada sa složenim i jednostavnim podacima. Također, pitanje memorije je važno jer utječe na sposobnost pohrane i brzine pristupa podacima.
- **Skalabilnost i Visoka Dostupnost**: Skalabilnost se odnosi na sposobnost baze podataka da se prilagodi zahtjevima aplikacije. Važno je razmotriti kako se svaka baza podataka može proširiti kako bi podržala veći broj korisnika ili veću količinu podataka. Uz to, treba razmotriti i kakve mehanizme baza podataka nudi u slučaju kvara.
- **Primjena u Pravom Svijetu**: Realni primjeri korištenja su ključni jer pokazuju kako baza podataka funkcionira u pravom svijetu. To omogućuje da se bolje razumije kako se mogu primijeniti u specifičnom okruženju.

Tablica 1. Kriteriji i kratki opis kao uvod u usporedbu

Kriterij	Redis	Memcached
Arhitektura i Struktura Podataka	• Jednonitna arhitektura • Bogat skup struktura podataka – limit za vrijednost pojedinačnog podatka je 512 MB	• Višenitna arhitektura • Jednostavna struktura podataka – limit za vrijednost pojedinačnog podatka je 1 MB
Funkcionalnosti	• Mehanizam zaključavanja • Transakcije • Pub/Sub • Ključevi s istekom • Lua skripte • Pipelining • Distribucija podataka	• Ključevi s istekom • Distribucija podataka
Performanse i Memorija	• Iznimno brz sa složenijim strukturama podataka • Napredni algoritmi za oslobađanje memorije • Perzistencija	• Iznimno brz s jednostavnom strukturom podataka(znakovni nizovi) • LRU algoritam za oslobađanje memorije
Skalabilnost i Visoka Dostupnost	• Horizontalno i horizontalno dijeljenje • Vertikalno • Replikacija	• Horizontalno i horizontalno dijeljenje • Vertikalno
Primjena u Pravom Svijetu	• Primjena za složenije potrebe	• Primjena za jednostavnije potrebe

5.1.1 Arhitektura i Struktura Podataka

Redis koristi jednonitnu arhitekturu zato što njegova primjena nije opterećujuća za procesor, stoga jedna nit je dovoljna kako bi se postigli zadovoljavajući rezultati. Uz to, višenitna arhitektura je kompleksnija te zahtjeva pažljivo rukovanje kako bi sveukupni proces bio što stabilniji. Upravo zbog jednonitne arhitekture je preporuka Redis skalirati horizontalno.
Redis pruža bogat skup struktura podataka, uključujući nizove, skupove, hash mape, uređene liste i još mnogo toga. Ova raznovrsnost struktura omogućuje kompleksnu privremenu pohranu i rješavanje različitih problema.

Memcached koristi višenitnu arhitekturu, što mu omogućava rukovanje s više operacija u isto vrijeme, stoga se za Memcached preporučuje vertikalno skaliranje, međutim to može značiti gubitak dijela podataka. Memcached podržava samo jednostavnu strukturu ključ-vrijednost [60].

5.1.2 Funkcionalnosti

Redis nudi napredne funkcionalnosti poput *transakcija* za višestruke naredbe, *mehanizma zaključavanja* kako bi se spriječio višestruki pristup izmjeni ključa*, Pub/Sub* mehanizma za slanje poruka između klijenata, *ključeve s Istekom* za oslobađanje memorije, *skriptiranje* pomoću Lua skripti, *pipelining* za bolji odaziv te distribucija podataka kako bi se osigurala dosljedna raspodjela podataka i rasterećenje sustava.

Memcached nudi osnovne operacije čitanja i pisanja vrijednosti u memoriju. Memcached ne pruža složenije funkcionalnosti koje nudi Redis, poput transakcija, složenih struktura podataka ili mehanizma zaključavanja [60].

5.1.3 Performanse i Memorija

Prema podacima iz 2016. godine, Redis je sporiji od Memcacheda kada je u pitanju radnja pisanja. Dok Memcached s druge strane ima izvanrednu brzinu pisanja s obzirom na količinu podataka [61].

The calculated time to write key-value pairs (ms)

	Number of records			
Database	1,000	10,000	100,000	1,000,000
Redis	34	214	1,666	14,638
Memcached	23	100	276	2,813

Slika 2. Operacija pisanja(Izvor: https://scalegrid.io/blog/redis-vs-memcached-2021-comparison/)

Redis je brži kada se radi o čitanju podataka, unatoč uvećanoj količini podataka. Memcached je osjetno sporiji od Redisa kada se radi o čitanju podataka [61].

The elapsed time to read value corresponding to a given key per database (ms)

	Number of records			
Database	1,000	10,000	100,000	1,000,000
Redis	8	6	8	8
Memcached	9	14	14	30

Slika 3. Operacija čitanja(Izvor: https://scalegrid.io/blog/redis-vs-memcached-2021-comparison/)

Redis drži sve podatke u memoriji, no nudi opciju perzistencije, odnosno mogućnost pohranjivanja na disk kako bi osigurao trajnost podataka. Uz to, Redis nudi i napredne algoritme za upravljanje memorijom poput LRU(engl. Least Recently Used) i LFU(engl. Least Frequently Used).

Memcached također drži sve podatke u memoriji i ne nudi opciju pohranjivanja na disk. To znači da podaci nisu trajni i da će se izgubiti u slučaju pada sustava. Međutim, ova jednostavnost čini Memcached vrlo brzim i pogodnim za privremenu pohranu podataka koji nisu kritični za trajno pohranjivanje.

5.1.4 Skalabilnost i Visoka Dostupnost

Redis podržava horizontalno skaliranje putem distribucije podataka, što omogućuje podjelu podataka na više čvorova za ravnomjernu raspodjelu opterećenja. Međutim, distribucija podataka može biti složena za postavljanje i upravljanje, a potrebno je i pažljivo razmotriti kako podijeliti podatke. Redis isto tako podržava vertikalno skaliranje i replikaciju, odnosno repliciranje čvorova unutar grozda kako bi se osigurala visoka dostupnost u slučaju greške.

Memcached također podržava distribuciju podataka. Budući da je Memcached jednostavniji od Redis-a, distribucija podataka može biti jednostavnija za implementaciju i upravljanje. Memcached podržava vertikalno i horizontalno skaliranje, no ne nudi mogućnost *replikacije* što znači da nema visoke dostupnosti u slučaju kvara [63].

5.1.5 Primjena u Pravom Svijetu

Redis se često koristi za zadatke koji zahtijevaju napredne strukture podataka, mehanizme zaključavanja i zahtjevne operacije na podacima, kao što su redovi događaja, praćenje statistika, privremena pohrana sesija i druge složene aplikacije. Isto tako, Redis je koristan u slučajevima kada je potrebna perzistencija. Svakako, Redis je fleksibilan te može zadovoljiti sva očekivanja i potrebe.

Memcached se najčešće koristi za privremenu pohranu jednostavnih podataka kojima se često pristupa, kao što su česti upiti na bazu podataka, privremena pohrana sesija, rezultati pretrage i sl. Memcached je dobar izbor kada je potrebno jednostavna i brza privremena pohrana bez složenih funkcionalnosti [62].

5.1.6 Zaključak Usporedbe

Odluka između Redis-a i Memcached-a ovisi o specifičnim potrebama i zahtjevima aplikacije. Ako je potrebna napredna struktura podataka i složene funkcionalnosti, Redis može biti bolji izbor. S druge strane, ako je potrebna jednostavna i brza privremena pohrana podataka bez dodatnih funkcionalnosti, Memcached može biti prikladniji.

Tablica 2. prikazuje ocjene Redisa i Memcacheda na temelju dojma dobivenog od strane različitih izvora koji su referencirani kroz ovaj rad. Obrazloženje ocjena prikazuje prema kojim stavkama su baze podataka ocijenjene.

Tablica 2. Ocjena Redis-a i Memcached-a

Kriterij	Ocjena - Redis	Ocjena- Memcached	Obrazloženje
Arhitektura i Struktura Podataka	4	3	**Redis:** vrlo bogata struktura podataka, no jednonitna arhitektura je ograničavajuća za operaciju pisanja. **Memcached:** prejednostavna struktura podataka koja ograničava odabir kod ove baze podataka, no višenitna arhitektura ide u korist pri operaciji pisanja.
Funkcionalnost	5	2	**Redis:** dostupan velik izbor funkcionalnosti koje omogućuju fleksibilnost korištenja ove baze podataka. **Memcached:** siromašan izbor funkcionalnosti koji eventualno pridonosi jednostavnosti ove baze podataka.
Performanse i Memorija	5	3	**Redis:** Vrlo visoke performanse pri čitanju i pisanju podataka, što se dobro slaže sa složenim strukturama podataka koje Redis pruža. Uz to, memorija pruža mogućnost perzistencije što apsolutno pridonosi fleksibilnosti korištenja ove baze podataka. **Memcached:** Vrlo visoke performanse pri čitanju, ali i posebno visoke pri pisanju podataka, no radi se o jednostavnoj strukturi podataka. Memorija pruža osnovne operacije te ne nudi mogućnost perzistencije.
Skalabilnost	5	4	**Redis:** Uz horizontalno i vertikalno skaliranje, Redis omogućuje i replikaciju kako bi se povećava visoka dostupnost. Isto tako, horizontalno dijeljenje je jedna od mogućnosti koja pomaže raspodjeli opterećenja sustava. **Memcached:** Nudi horizontalno i vertikalno skaliranje, uz horizontalno dijeljenje. No, ne nudi mogućnost replikacije, što ne osigurava nastavaka rada pri kvaru.
Primjena u Pravom Svijetu	5	2	**Redis:** Svojim velikim opsegom mogućnosti pridonosi tome da se Redis može ukomponirati u različite svrhe te ga to čini popularnim alatom koji je prisutan u mnogim današnjim aplikacijama.

			Memcached: Svojom brzinom i jednostavnom primjenom, Memcached se koristi u manje zahtjevne svrhe, stoga neće biti prvi izbor kada se razmatra rješenje za naprednu privremenu pohranu podataka.

5.2 Jednostavan Primjer Korištenja Redis-a i Memcached-a

U ovom odjeljku provest će se obje baze podataka kroz jednostavan primjer koji će prikazati korištenje ovih tehnologija unutar Flask aplikacije.

5.2.1 Opis Primjera

Neke od osnovnih funkcionalnosti ovih baza podataka prikazat će se na ovom primjeru koji predstavlja jednostavnu Flask web aplikaciju koja koristi Redis ili Memcached za privremenu pohranu podataka o najboljim filmovima i komunicira s vanjskim API-em (OMDb API) kako bi dobila detalje o filmovima.
Konkretnije, logika aplikacije će demonstrirati kako koristiti privremenu pohranu i provjeru postojanosti ključa. Isto tako, konkretno će se usporediti vrijeme potrebno za dohvaćanje podataka iz izvorne baze podataka i vrijeme potrebno za dohvaćanje podataka iz Memcached ili Redis baze podataka.

Kako bi se mogle koristiti Redis i Memcached baze podataka, za potrebe ovoga primjera, pokrenuta su dva kontejnera u Docker-u. Docker je otvorena platforma za razvoj, isporuku i upravljanje aplikacijama u kontejnerima.

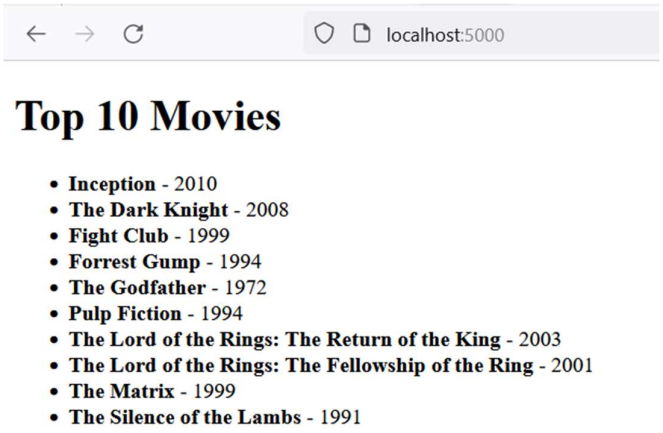
5.2.2 Redis – Flask Aplikacija

```
from flask import Flask, request, render_template
from redis import Redis
import json
import requests

app = Flask(__name__)
r = Redis(host='localhost', port=6379)

connection = r.ping()
print(connection)

def get_movie_details(movie_id):
    """Pomocna funkcija za dohvacanje podataka o filmovima sa vanjskog API-ja"""
    url = f'http://www.omdbapi.com/?i={movie_id}&apikey=4c28fb5f'
    response = requests.get(url)
    return json.loads(response.text)

@app.route('/')
def index():
    # Provjeri jesu li podaci o filmovima vec privremeno pohranjeni
    cached_movies = r.get('top_movies')
    if cached_movies:
        movies = json.loads(cached_movies)
    else:
        # Ako podaci o filmovima nisu pohranjeni, dohvati podatke sa vanjskog
API-ja
        top_10_movies = ['tt1375666', 'tt0468569', 'tt0137523', 'tt0109830',
'tt0068646', 'tt0110912', 'tt0167260', 'tt0120737', 'tt0133093', 'tt0102926']
        movies = []
        for movie_id in top_10_movies:
            movie_details = get_movie_details(movie_id)
            movies.append(movie_details)

        # Privremeno pohrani podatke o filmovima za buduce upite
        r.set('top_movies', json.dumps(movies))

    # Prikazi podatke na pocetnoj stranici
    return render_template('index.html', movies=movies)

if __name__ == '__main__':
    app.debug = True
    app.run(host='localhost', port=5000)
```

Slika 4. Programski kod aplikacije za testiranje

1. **Uvoz modula i postavljanje temeljnih varijabli**:
    - Uvoz **Flask**, **request**, **render_template** modula za izradu web aplikacije.
    - Također se uvoze **Redis**, **json** i **requests** za rad s Redis bazom podataka, obradu JSON podataka i slanje HTTP zahtjeva.
    - Inicijalizira se Flask aplikacija i Redis klijent koji će se koristiti za pristup Redis bazi podataka.
2. **Funkcija za dobivanje detalja o filmu**:
    - **get_movie_details**(**movie_id**) je pomoćna funkcija koja prima ID filma i koristi OMDb API za dohvaćanje detalja o filmu. Detalji se vraćaju u JSON formatu.
3. **Ruta za početnu stranicu**:
    - **@app.route('/')** označava glavnu rutu web aplikacije.
    - Prvo se provjerava je li lista najboljih 10 filmova privremeno pohranjena u Redisu.
    - Ako je, onda se koristi privremeno pohranjena lista.
    - Ako nije, tada se koristi **get_movie_details** funkcija za dohvaćanje detalja o svakom od 10 filmova iz vanjskog API-ja, a zatim se ta lista privremeno pohranjuje za buduće zahtjeve.
4. **Prikazivanje rezultata**:
    - Na kraju, rezultati se šalju predlošku za prikazivanje na web stranici. Ovo je statički popis 10 najboljih filmova s informacijama poput naslova, ocjene, godine izdanja, itd.
5. **Pokretanje aplikacije**:
    - Aplikacija se pokreće ako je skripta izravno izvršena (ako nije uvezena iz druge skripte).

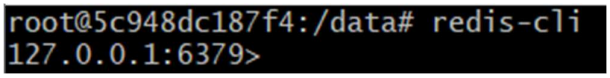
Pri pokretanju programskog koda, aplikacija se može pronaći u pregledniku na poveznici *http://localhost:5000,* gdje će se ispisati najboljih 10 filmova.



Slika 5. Izgled aplikacije na početnoj stranici

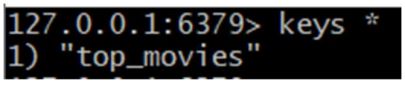
Logika programskoga koda nalaže kako se prvo izvršila provjera ukoliko podaci postoje u Redis-u, zatim kada je aplikacija determinirala suprotno, povlače se podaci iz izvorne baze podataka te se rezultat sprema u Redis.

Unutar kontejnera, za pristup naredbenoj ljusci Redis-a potrebno je pokrenuti naredbu „*redis-cli*“.



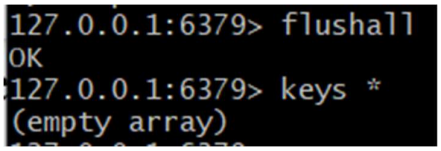
Slika 6. Pristup komandnoj ljusci Redis-a

Kako bi se provjerilo jesu li podaci stigli u Redis, unutar Redis kontejnera će se pokrenuti naredba „*keys **“.



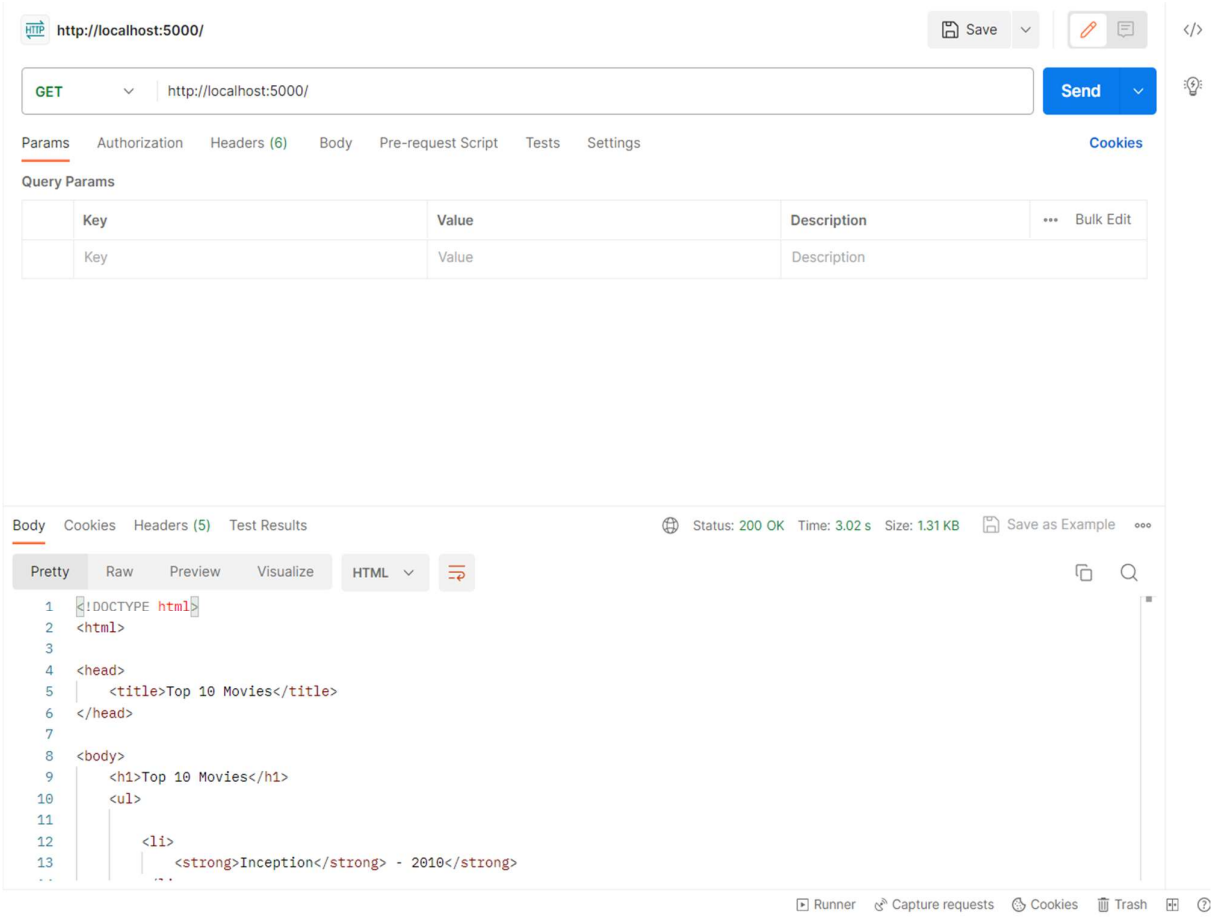
Slika 7. Dohvaćanje svih ključeva unutar Redis-a

Za potrebe testiranja pokreće se naredba „flushall“ koja briše sve ključeve iz Redis-a.



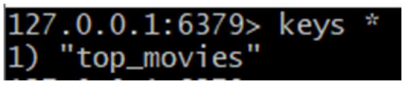
Slika 8. Brisanje svih ključeva unutar Redis-a i potvrda uspješnosti brisanja

Kako bi se prikazala razlika između dohvata podataka iz izvorne baze podataka i dohvata podataka iz Redis-a, koristit će se Postman. Postman je posebno koristan u razvoju web aplikacija, jer omogućuje brzo slanje HTTP zahtjeva i analizu odgovora, uključujući brzinu odgovora.



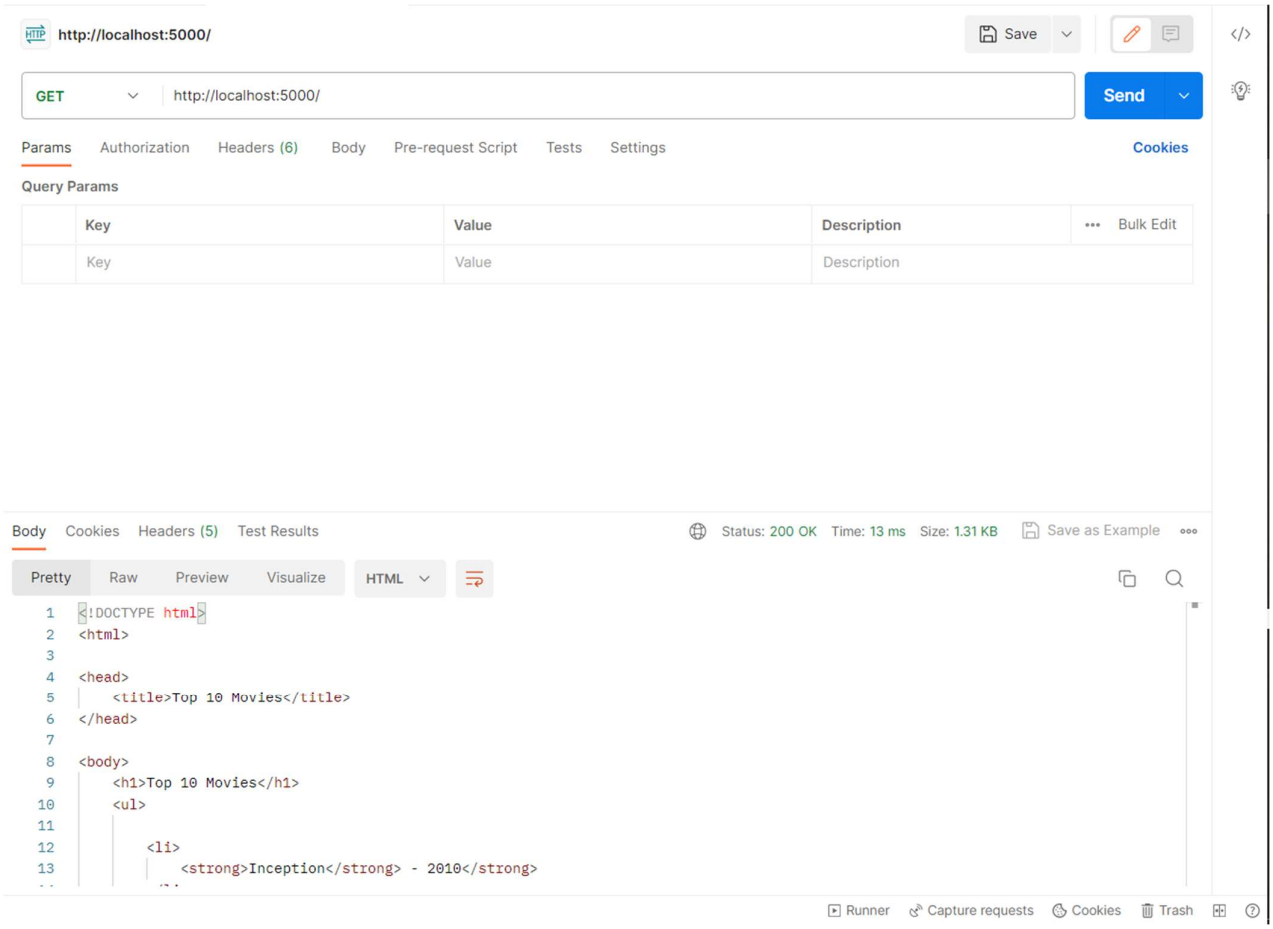
Slika 9. Prikaz vremena potrebnog za dohvaćanja podataka iz izvorne baze podataka u Postman-u

Postman prikazuje kako je odgovor na upit iz izvorne baze podataka stigao za tri sekunde. Provjerit će se jesu li podaci spremljeni u Redis nakon prvog pristupa ovoj poveznici.



Slika 10. Prikaz spremljenog ključa u Redis-u

Ponovit će se upit u Postmanu za provjeru brzine za slučaj kada se podatak nalazi u Redis-u.



Slika 11. Prikaz vremena potrebnog za dohvaćanje podataka iz Redis-a u Postman-u

Vidljivo je kako je brzina 13 milisekundi, što je znatno manje od tri sekunde dohvata iz izvorne baze podataka. Važno je napomenuti kako se brzina od 13 milisekundi u pravom svijetu smatra lošim rezultatom. No, ovaj primjer koristi lokalno testiranje te se rezultat ovog testiranja znatno razlikuje od rezultata koji bi bio izazvan od strane hardverski jakog poslužitelja.

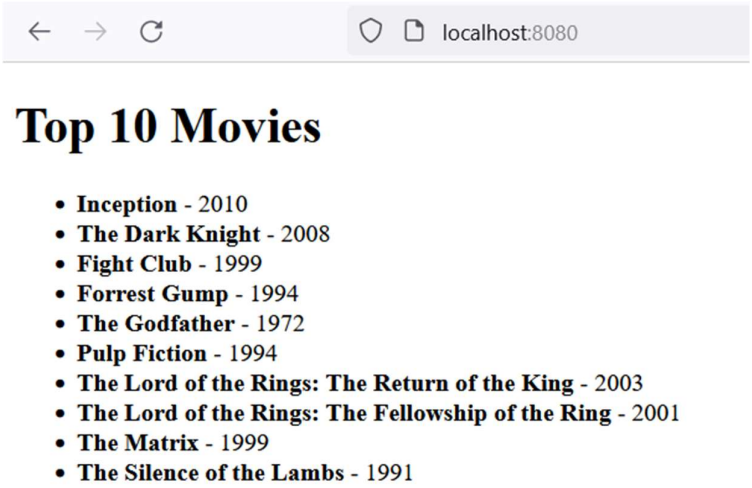
5.2.3 Memcached – Flask Aplikacija

```
from flask import Flask, request, render_template
from pymemcache.client import base
import json
import requests

app = Flask(__name__)
client = base.Client(('localhost', 11211))

def get_movie_details(movie_id):
    """Pomocna funkcija za dohvacanje podataka o filmovima sa vanjskog API-ja"""
    url = f'http://www.omdbapi.com/?i={movie_id}&apikey=4c28fb5f'
    response = requests.get(url)
    return json.loads(response.text)

@app.route('/')
def index():
    # Provjeri jesu li podaci o filmovima vec privremeno pohranjeni
    cached_movies = client.get('top_movies')
    if cached_movies:
        movies = json.loads(cached_movies.decode('utf-8'))
    else:
        # Ako podaci o filmovima nisu pohranjeni, dohvati podatke sa vanjskog
API-ja
        top_10_movies = ['tt1375666', 'tt0468569', 'tt0137523', 'tt0109830',
'tt0068646','tt0110912', 'tt0167260', 'tt0120737', 'tt0133093', 'tt0102926']
        movies = []
        for movie_id in top_10_movies:
            movie_details = get_movie_details(movie_id)
            movies.append(movie_details)

        # Privremeno pohrani podatke o filmovima za buduce upite
        client.set('top_movies', json.dumps(movies))

    # Prikazi podatke na pocetnoj stranici
    return render_template('index.html', movies=movies)

if __name__ == '__main__':
    app.debug = True
    app.run(host='localhost', port=8080)
```

Slika 12. Programski kod aplikacije za testiranje

1. **Uvoz modula i postavljanje temeljnih varijabli**:
    - Uvoz **Flask**, **request**, **render_template** modula za izradu web aplikacije.
    - Također se uvoze **base**, **json** i **requests** za rad s Memcached bazom podataka, obradu JSON podataka i slanje HTTP zahtjeva.
    - Inicijalizira se Flask aplikacija i Memcached klijent koji će se koristiti za pristup Memcached bazi podataka.
2. **Funkcija za dobivanje detalja o filmu**:
    - **get_movie_details**(**movie_id**) je pomoćna funkcija koja prima ID filma i koristi OMDb API za dohvaćanje detalja o filmu. Detalji se vraćaju u JSON formatu.
3. **Ruta za početnu stranicu**:
    - **@app.route('/')** označava glavnu rutu web aplikacije.
    - Prvo se provjerava je li lista najboljih 10 filmova privremeno pohranjena u Memcached-u.
    - Ako je, onda se koristi privremeno pohranjena lista.
    - Ako nije, tada se koristi **get_movie_details** funkcija za dohvaćanje detalja o svakom od 10 filmova iz vanjskog API-ja, a zatim se ta lista privremeno pohranjuje za buduće zahtjeve.
4. **Prikazivanje rezultata**:
    - Na kraju, rezultati se šalju predlošku za prikazivanje na web stranici. Ovo je statički popis 10 najboljih filmova s informacijama poput naslova, ocjene, godine izdanja, itd.
5. **Pokretanje aplikacije**:
    - Aplikacija se pokreće ako je skripta izravno izvršena (ako nije uvezena iz druge skripte).

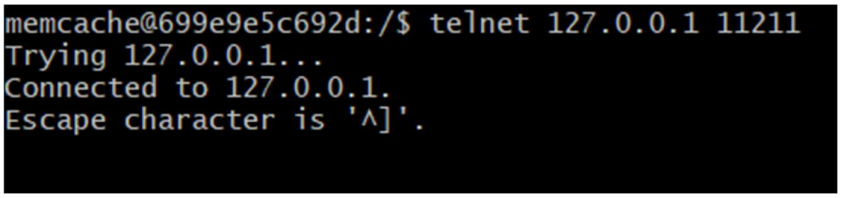
Pri pokretanju programskog koda, aplikacija se može pronaći u pregledniku na poveznici *http://localhost:8080,* gdje će se ispisati najboljih 10 filmova.



Slika 13. Izgled aplikacije na početnoj stranici

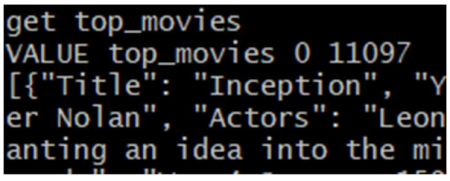
Kao i kod Redis-a, logika programskoga koda nalaže kako se prvo izvršila provjera ukoliko podaci postoje u Memcached-u, zatim kada je aplikacija determinirala suprotno, dohvaćaju se podaci iz izvorne baze podataka te se rezultat sprema u Memcached.

Unutar kontejnera, za pristup naredbenoj ljusci Memcached-a potrebno je pokrenuti naredbu „*telnet 127.0.0.1 11211*“.



Slika 14. Pristup komandnoj ljusci Memcached-a

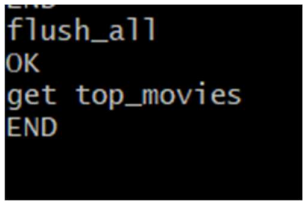
Kako bi se provjerilo jesu li podaci stigli u Memcached, u naredbenoj ljusci će se pokrenuti naredba „*get top_movies*“.



Slika 15. Dohvaćanje "top_movies" ključa unutar Memcached-a

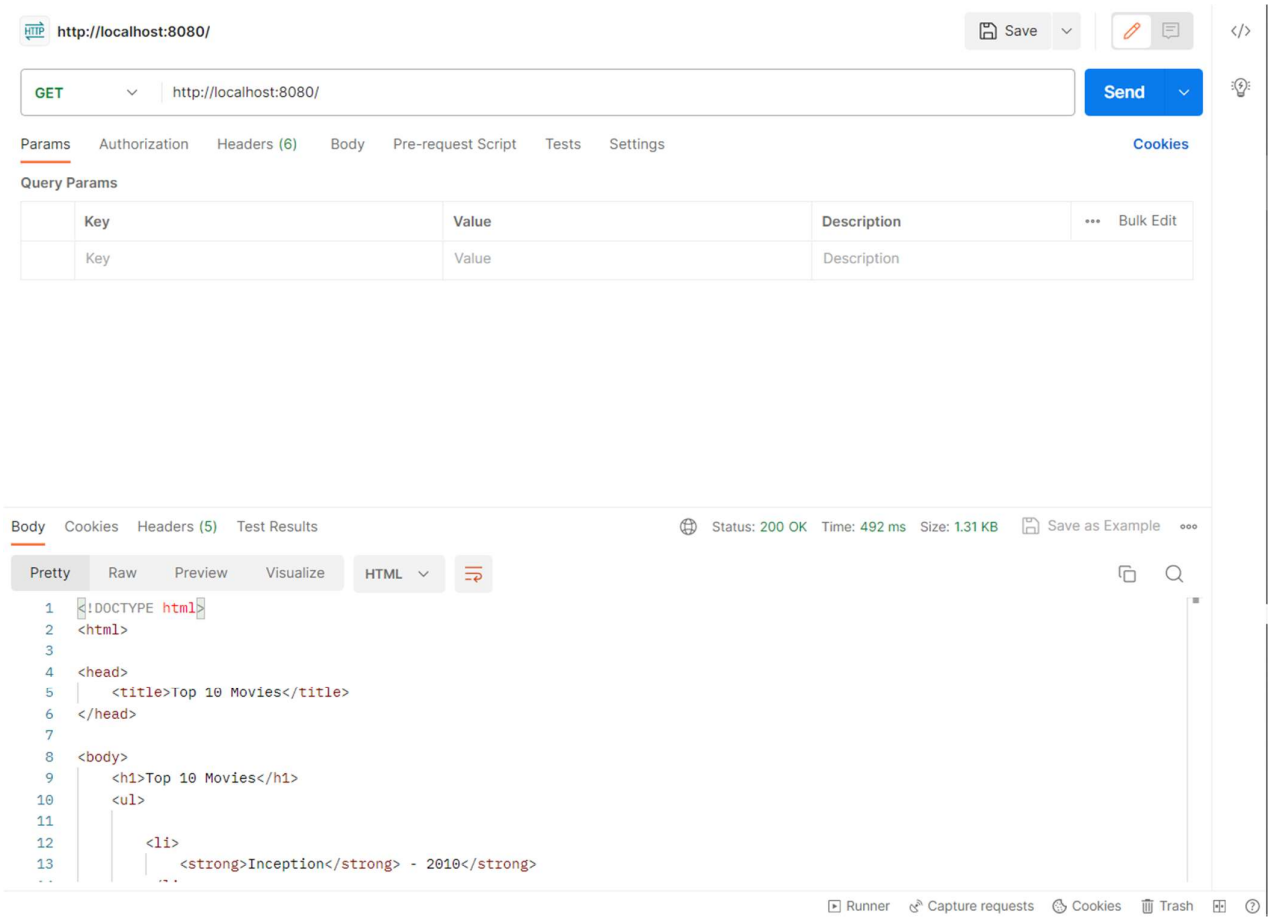
Naredbena ljuska Memcached-a nije pregledna kao ona od Redis-a, međutim, ovim ispisom je potvrđeno kako ključ „*top_movies*“ postoji.

Za potrebe testiranja pokreće se naredba „*flush_all*“ koja briše sve ključeve iz Memcached-a.



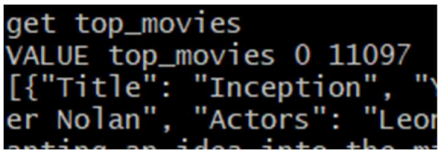
Slika 16. Brisanje svih ključeva unutar Memcached-a i potvrda uspješnosti brisanja

Kako bi se prikazala razlika između dohvata podataka iz izvorne baze podataka i dohvata podataka iz Memcached-a, koristit će se Postman.



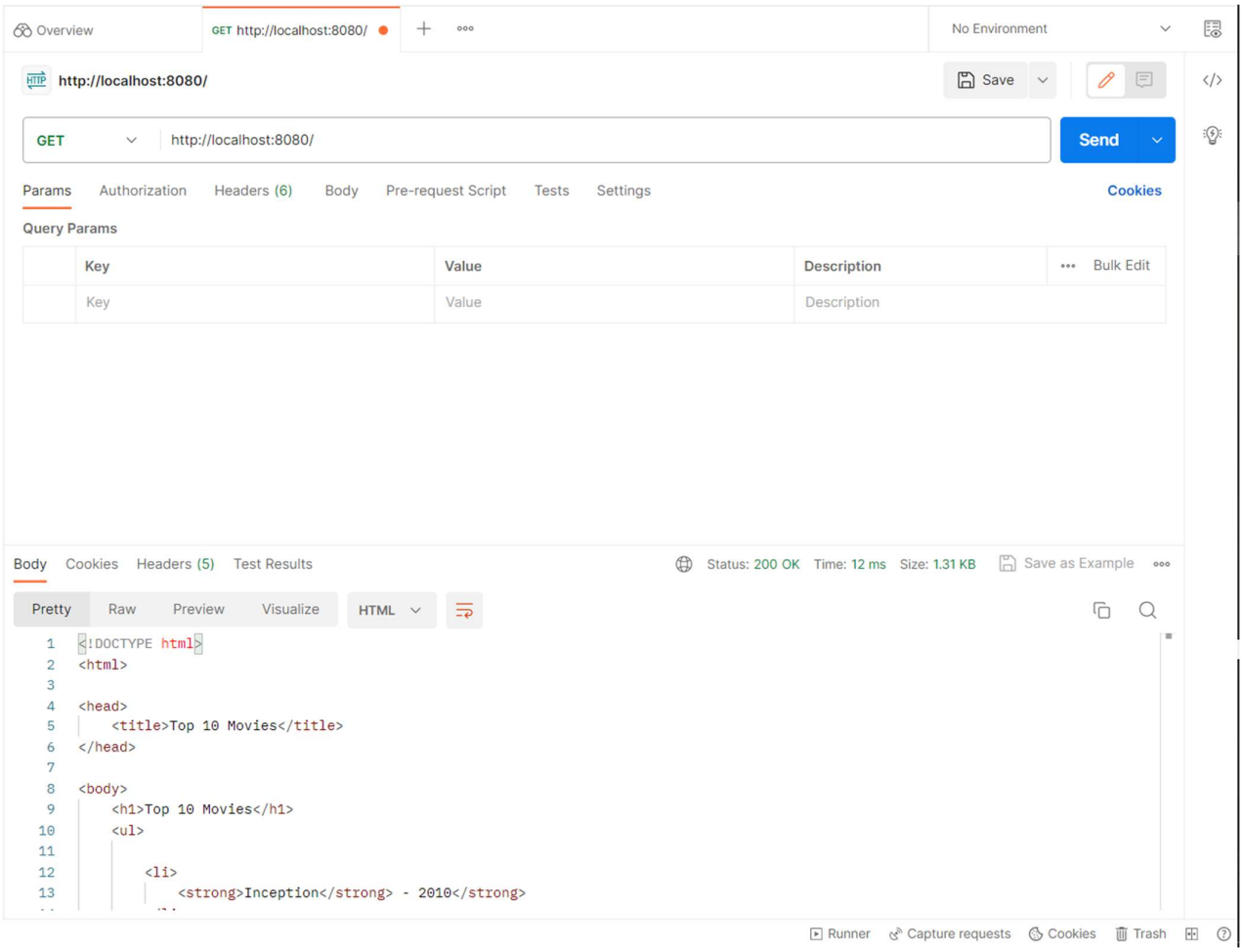
Slika 17. Prikaz vremena potrebnog za dohvaćanja podataka iz izvorne baze podataka u Postman-u

Postman prikazuje kako je odgovor na upit iz izvorne baze podataka stigao za 492 milisekunde. Provjerit će se jesu li podaci spremljeni u Memcached-u nakon prvog pristupa ovoj poveznici.



Slika 18. Prikaz spremljenog ključa u Memcached-u

Ponovit će se upit u Postmanu za provjeru brzine za slučaj kada se podatak nalazi u Memcached-u.



Slika 19. Prikaz vremena potrebnog za dohvaćanje podataka iz Memcached-a u Postman-u

Vidljivo je kako je brzina 12 milisekundi, što je znatno manje od 492 milisekunde dohvata iz izvorne baze podataka.

5.2.4 Zaključak Primjera

Primjerom je prikazano kako je dohvaćanje podataka iz baze podataka poput Redis-a ili Memcached-a znatno brže od dohvata podataka iz izvorne baze podataka, što prikazuje prednosti korištenja takvih baza podataka za privremenu pohranu. Dostupnost podataka svodi se na nekoliko milisekundi ili u idealnim uvjetima ispod milisekunde.

6. Zaključak

U ovom završnom radu duboko su istražene i uspoređene dvije popularne nerelacijske baze podataka, Redis i Memcached. Oba sustava služe kao alati za privremenu pohranu podataka i pružaju brz pristup podacima, ali imaju različite karakteristike i prednosti koje ih čine prikladnim za različite primjene.

U teorijskom pregledu, definirani su koncepti nerelacijskih baza podataka, objašnjena je potreba za privremenom pohranom podataka i istražene su osnovne karakteristike ovih baza podataka. Nakon toga, detaljno su analizirani Redis i Memcached u odvojenim sekcijama, uključujući njihove arhitekture, funkcionalnosti, performanse, upotrebljivost, skalabilnost, sigurnost i druge relevantne aspekte.

Nakon usporedbe tih dvaju sustava, može se zaključiti da oba imaju svoje prednosti i nedostatke. Redis se ističe svojom bogatom funkcionalnošću, podrškom za različite tipove podataka, kao i mogućnošću pohrane podataka na disku. S druge strane, Memcached se ističe svojom jednostavnošću i brzinom, što ga čini izvrsnim izborom za jednostavnu upotrebu privremene pohrane.

U konačnom zaključku, odabir između Redis-a i Memcached-a ovisi o specifičnim potrebama i zahtjevima aplikacije. Treba uzeti u obzir performanse, kompleksnost implementacije, skalabilnost i druge čimbenike. Važno je napomenuti da ovi sustavi često rade zajedno u složenim aplikacijama kako bi se postigle najbolje performanse i iskoristila kombinacija njihovih prednosti.

7. Popis Literature

1. Fowler, Adam: *NoSQL For Dummies* (2015)
2. Sullivan, Dan: *NoSQL for Mere Mortals* (2015)
3. Carlson, L. Josiah: *Redis in Action* (2013)
4. Macedo, Tiago; Oliveira, Fred: *Redis Cookbook* (2011)
5. Da Silva, Dayvson Maxwel; Tavares, Lopes Hugo: *Redis Essentials* (2015)
6. Farghal, Soliman Ahmed: *Getting started with Memcached* (2013)
7. Blokdyk, Gerardus: *Memcached A Complete Guide 2019 Edition* (2019)
8. https://aws.amazon.com/caching/web-caching/
9. https://medium.com/codex/server-side-caching-in-web-applications-a9145be1cfa0
10. https://www.torocloud.com/blog/using-caching-strategies-to-improve-api-performance
11. https://www.linkedin.com/pulse/system-design-distributed-cache-concepts-explained-using-arpan-das/
12. https://architecturenotes.co/redis/
13. https://www.sobyte.net/post/2022-08/redis-single-thread/
14. https://redis.io/docs/data-types/
15. https://redis.io/docs/about/
16. https://developer.redis.com/operate/redis-at-scale/talking-to-redis/redis-clients/
17. https://redis.io/docs/interact/transactions/
18. https://redis.io/docs/interact/pubsub/
19. https://redis.io/commands/expire/
20. https://redis.io/docs/interact/programmability/eval-intro/
21. https://redis.io/docs/manual/pipelining/
22. https://redis.io/docs/management/optimization/memory-optimization/
23. https://redis.io/docs/reference/eviction/
24. https://redis.io/docs/management/persistence/
25. https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/scaling-redis-cluster-mode-enabled.html
26. https://redis.io/docs/management/scaling/
27. https://redis.com/redis-enterprise/technology/redis-enterprise-cluster-architecture/
28. https://redis.io/docs/management/security/#authentication
29. https://redis.io/docs/management/security/#network-security
30. https://docs.redis.com/latest/rs/security/access-control/rbac/create-roles/

31. https://redis.io/docs/clients/
32. https://redis.io/docs/clients/python/
33. https://redis.io/docs/ui/cli/#command-line-usage
34. https://redis.io/docs/getting-started/installation/
35. https://alronz.github.io/Factors-Influencing-NoSQL-Adoption/site/Redis/Results/Strengths%20and%20Weaknesses/
36. https://www.infoq.com/presentations/Real-Time-Delivery-Twitter/
37. https://blog.twitter.com/engineering/en_us/topics/infrastructure/2019/improving-key-expiration-in-redis
38. https://github.blog/2009-10-20-how-we-made-github-fast/
39. https://blog.zawodny.com/2011/02/26/redis-sharding-at-craigslist/
40. https://nickcraver.com/blog/2019/08/06/stack-overflow-how-we-do-app-caching/
41. https://memcached.org/about
42. https://github.com/memcached/memcached/wiki/Overview
43. https://github.com/memcached/memcached/wiki/Performance
44. https://medium.com/geekculture/why-we-migrated-from-memcached-to-redis-2ecdd21d68d0
45. https://github.com/memcached/memcached/blob/master/doc/new_lru.txt
46. https://github.com/memcached/memcached/wiki/ConfiguringServer#commandline-arguments
47. https://github.com/memcached/memcached/wiki/ConfiguringClient#consistent-hashing
48. https://memcached.org/blog/modern-lru/
49. https://docs.aws.amazon.com/AmazonElastiCache/latest/mem-ug/Scaling.html#Scaling.Memcached.Vertically
50. https://github.com/memcached/memcached/wiki/ConfiguringServer#networking
51. https://github.com/memcached/memcached/wiki/Proxy
52. https://aws.amazon.com/memcached/
53. https://github.com/memcached/memcached/wiki/Hardware
54. https://github.com/memcached/memcached/wiki/Commands
55. https://www.ionos.com/digitalguide/hosting/technical-matters/what-is-memcached/
56. https://engineering.fb.com/2013/04/15/core-data/scaling-memcache-at-facebook/
57. https://ieeexplore.ieee.org/document/6838738

58. https://medium.com/airbnb-engineering/behind-the-scenes-airbnb-neighborhoods-cef63242eab7#.fo63n0iqm
59. https://medium.com/@sevebadajoz/exploring-cache-optimization-with-reddits-infrastructure-f5682325990d
60. https://scalegrid.io/blog/redis-vs-memcached-2021-comparison/
61. https://www.sciencedirect.com/science/article/pii/S1319157816300453
62. https://www.infoworld.com/article/3063161/why-redis-beats-memcached-for-caching.html
63. https://kinsta.com/blog/memcached-vs-redis/