KEŠIRANJE U WEB OKRUŽENJU

KEŠIRANJE

ŠTA JE KEŠIRANJE?

- Čuvanje aplikativnih podataka na lokaciji kojoj je obezbeđen optimizovan pristup za brže korišćenje u odgovarajućem sloju n-slojne arhitekture

KEŠIRANJE NIJE UVEK NEOPHODNO

- Iako postoje čitave kategorije kandidata za keširanje, ti podaci se ne moraju nužno keširati jer sama izvedba može biti zahtevna, skupa ili može negativno uticati na druge operacije koje barataju tim podacima

SCENARIJI ZA KEŠIRANJE

KORIŠĆENJE SKUPIH RESURSA

- Bilo koja operacija koja je skupa u pogledu korišćenih resursa (memorija/CPU/propusni opseg) može biti kandidat za keširanje
  - Podaci dobijeni čitanjem iz baze podataka
  - Datoteke i binarni sadržaj dobijen sa udaljenih servera
  - Izveštaji dobijeni iz sistema za generisanje izveštaja
  - Pozivi internih i eksternih web servisa

SCENARIJI ZA KEŠIRANJE

RESURSI KOJI PREDSTAVLJAJU USKO GRLO SISTEMA

- Neke softverske ili hardverske komponente mogu da postanu usko grlo sistema pod velikim opterećenjem
  - Podaci dobijeni od 3rd party proizvođača koji nudi servis sa jednog servera
  - Biblioteka ili stylesheet datoteka koji se učitava uvek sa iste lokacije proizvođača

SCENARIJI ZA KEŠIRANJE

POŠTOVANJE STRIKTNIH UGOVORA PO PITANJU PERFORMANSI

- Engl. Service Level Agreement (SLA)
- Neke operacije koje utiču na biznis aktivnosti i direktan prihod mogu biti podložni posebnim ugovorima
    - Početna ili neka druga specifična stranica bitna za poslovanje mora da se učita za određeni broj sekundi
    - Transakcija plaćanja mora da se obavi za do 30-45 sekundi

OPTIMIZACIJE PRIMENJIVE NA VIŠE TIPOVA UREĐAJA

- Optimizacije koje treba da uključe i operacije koje se mogu izvršavati na mobilnim uređajima, a ne samo u web pretraživačima

SCENARIJI ZA KEŠIRANJE

ČESTO POZIVANI SERVISI ILI KORIŠĆENI PODACI

- Sve operacije čiji pozivi rezultuju istim podacima dobri su kandidati za keširanje
    - Često korišćene vrednosti dobijene kompleksnim proračunima
    - Često korišćeni podaci iz baze podataka
    - Fiksne liste podržanih jezika, država, opcija na nivou aplikacije

STATIČKI PODACI

- Bilo koji podaci koji nisu zavisni od korisnika i njegovog konteksta korišćenja dobri su kandidati za keširanje
    - Sadržaj koji ide u *header* i *footer* stranica
    - FAQ stranica
    - Kontakt stranica
    - ...

UTICAJ KEŠIRANJA

UTICAJ NA SKALABILNOST APLIKACIJE

- Različitim tehnikama keširanja se mogu rešiti problemi koje unose komponente koje predstavljaju usko grlo sistema (prefetch, on-demand fetch)
- Ako postoje integracione tačke sistema koje nisu podložne skaliranju, keširanjem rezultata koji se dobijaju kroz njih može se minimizovati njihov uticaj na celokupnu arhitekturu
- Keširanje se može upotrebiti na nivou web servera, baze podataka, servisa, itd. i problem skalabilnosti rešavati na nivou individualnih slojeva aplikacije

UTICAJ KEŠIRANJA

UTICAJ NA PERFORMANSE APLIKACIJE

- Najveća prednost keširanja se ogleda u performansama aplikacija
  - Smanjuje se vreme za pravljenje poziva ka bazi podataka, web servisu, itd.
  - Keširanje rezultata kompleksnih operacija smanjuje ukupno utrošeno vreme za računanje
  - Keširanjem fajlova izbegava se ponovno dovlačenje istih preko mreže, kreiranje, parsiranje, itd.

UTICAJ KEŠIRANJA

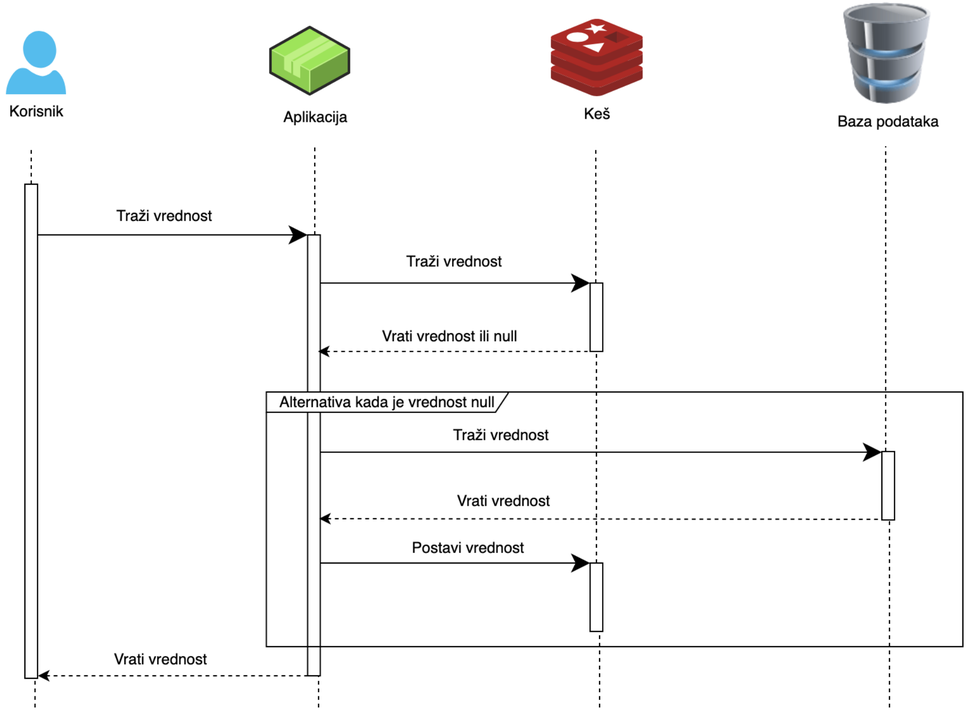
UTICAJ NA DOSTUPNOST APLIKACIJE

- U aplikacijama koje se zasnivaju na višeslojnoj arhitekturi, dostupnost se zasniva na dostupnosti svake komponente koja učestvuje u obavljanju biznis operacija
    - Ako je baza podataka ili mreža nedostupna, keširanje može pomoći da se privremeno nastavi neometan rad

STRATEGIJE ZA KEŠIRANJE

CACHE-ASIDE (LAZY-LOAD)

- **Aplikacija preuzima odgovornost** za učitavanje podataka u keš
- Kada se podaci zatraže, prvo se proverava keš
- Ako podaci nisu u kešu (*cache miss*), **aplikacija učitava podatke** iz glavnog skladišta (baze podataka ili drugih izvora), a zatim ih stavlja u keš za buduće zahteve



CACHE-ASIDE

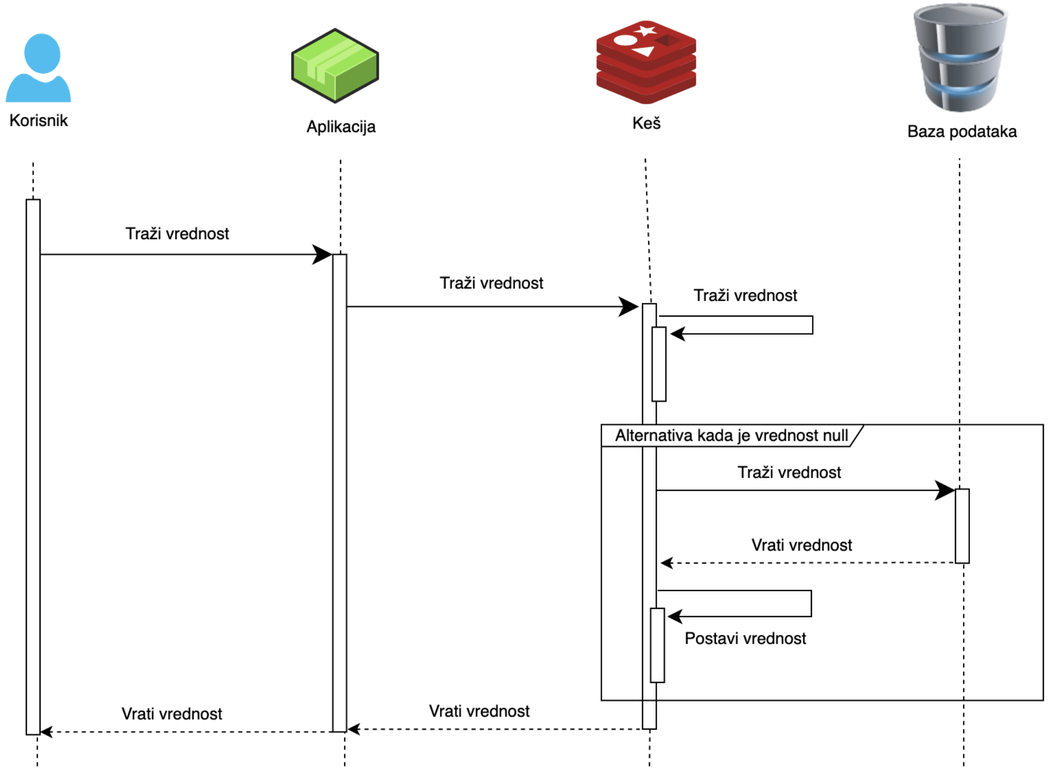
SCENARIJI KORIŠĆENJA

- Ova strategija je idealna za podatke koji se retko menjaju, ali se često čitaju
- U aplikacijama koje koriste više izvora podataka (npr. mikroservisi) može biti efikasna

STRATEGIJE ZA KEŠIRANJE

READ-THROUGH

- Aplikacija uvek pokušava da pročita podatke iz keša pre nego iz izvora podataka (npr. baze podataka)
- Ako podaci nisu prisutni u kešu, **keš postaje medijator** koji učitava podatke iz izvora i smešta ih u keš za buduću upotrebu



READ-THROUGH

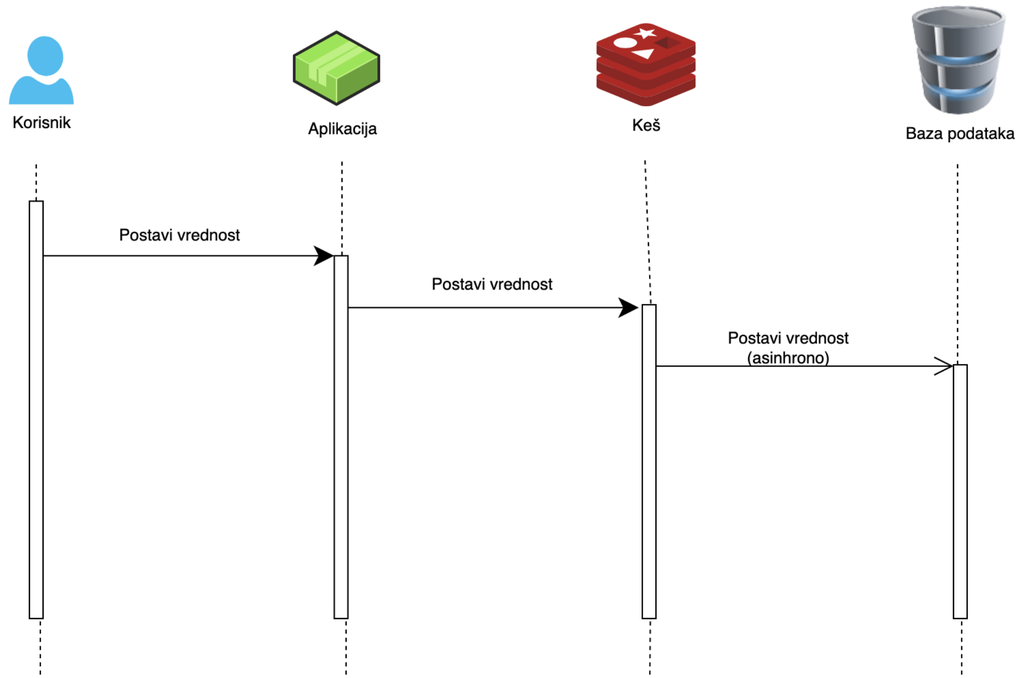
SCENARIJI KORIŠĆENJA

- Ako su podaci često čitani, ali retko menjani, *Read-Through* strategija je efikasna jer keširanje smanjuje potrebu za ponovnim učitavanjem podataka
- Strategija je korisna kada postoji veliki broj korisnika koji često traže iste podatke, jer smanjuje opterećenje ka izvorima podataka

STRATEGIJE ZA KEŠIRANJE

WRITE-BACK (WRITE-BEHIND)

- Podaci najpre bivaju zapisani u keš, a zatim se **asinhrono ili periodično** upisuju u izvor podataka (npr. bazu podataka)
- To znači da promena podataka nije odmah vidljiva u izvornim podacima, već se ažuriranje dešava kasnije, tokom procesa *write-back*



WRITE-BACK

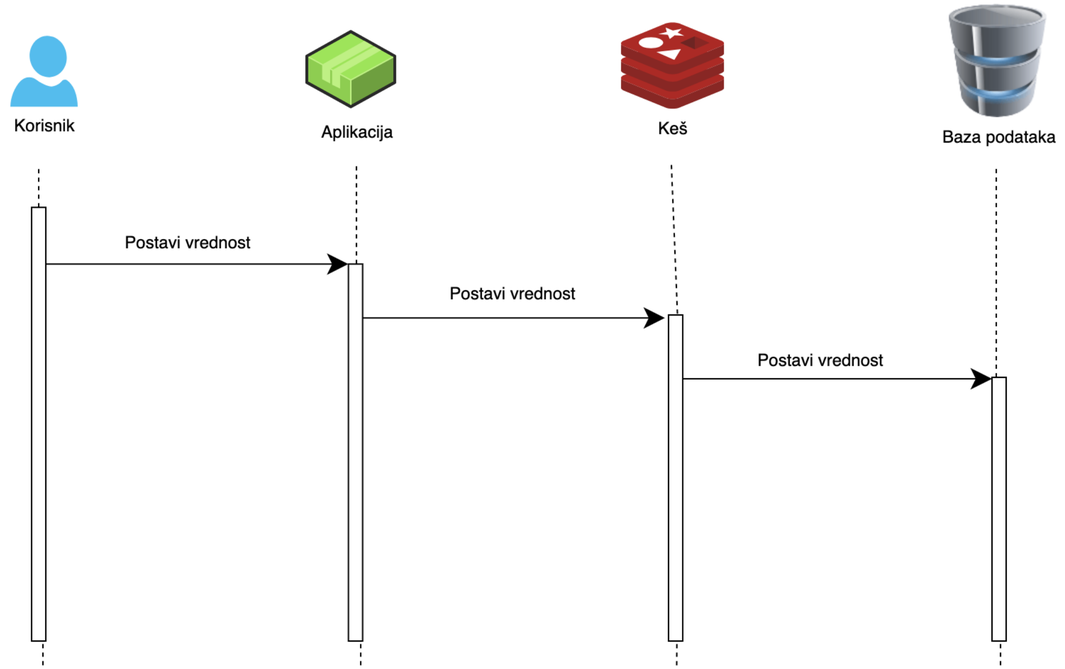
SCENARIJI KORIŠĆENJA

- Kada sistem mora da upisuje velike količine podataka, *Write-Back* smanjuje broj upita prema bazi, jer se zapisi izvršavaju u grupama
- Ako je potrebno brzo izvršavanje operacija pisanja, *Write-Back* omogućava brz zapis podataka u keš dok se zapisivanje u izvor podataka dešava kasnije

STRATEGIJE ZA KEŠIRANJE

WRITE-THROUGH

- Podaci koji se pišu u izvor podataka (npr. bazu podataka) takođe **odmah** bivaju zapisani u keš
- Ovaj pristup osigurava da keš bude uvek u sinhronizaciji sa izvorom podataka, jer se pisanje u keš dešava istovremeno sa pisanjem u izvor podataka



WRITE-THROUGH

SCENARIJI KORIŠĆENJA

- Ako je potrebno da podaci u kešu budu uvek u sinhronizaciji sa izvorom podataka, *Write-Through* je dobra opcija
- Kada se podaci često čitaju, ali se retko menjaju, čime se smanjuje opterećenje na izvor podataka
- Ako sistem nije podložan velikim brojevima zapisa u kratkom vremenskom periodu (malo opterećenje), *Write-Through* može biti jednostavno rešenje

STRATEGIJE ZA KEŠIRANJE

KEŠIRANJE UNAPRED

- Podaci se keširaju kada se pokrene aplikacija

SCENARIJI KORIŠĆENJA

- Koristi se kada treba keširati vrednosti koje su globalno dostupne u aplikaciji
- Koristi se kada vrednosti treba koristiti na početku komunikacije

KANDIDATI ZA KEŠIRANJE

- Konfiguracija aplikacije
- Statičke vrednosti poput fajlova taksonomije sajta

STRATEGIJE ZA KEŠIRANJE

PREDIKTIVNO KEŠIRANJE UNAPRED

- Pravi se predikcija koji će se objekti keširati na osnovu ponašanja korisnika tokom korišćenja aplikacije

SCENARIJI KORIŠĆENJA

- Koristi se kada je izbor podataka koji će se keširati zasnovan na korisničkoj sesiji

KANDIDATI ZA KEŠIRANJE

- Statički elementi npr. slike, video zapisi, itd.

STRATEGIJE ZA KEŠIRANJE

KEŠIRANJE POMOĆU PROKSI SERVERA

- Koristi se zaseban server kako bi se ubrzalo slanje objekata

SCENARIJI KORIŠĆENJA

- Može se koristiti uvek kad se razmišlja o skaliranju aplikacije

KANDIDATI ZA KEŠIRANJE

- Statički fajlovi npr. slike, video zapisi, JS fajlovi, itd.

STRATEGIJE ZA KEŠIRANJE

CONTENT DELIVERY NETWORK (CDN)

- Korišćenje geografski distribuiranih servera koji keširaju podatke

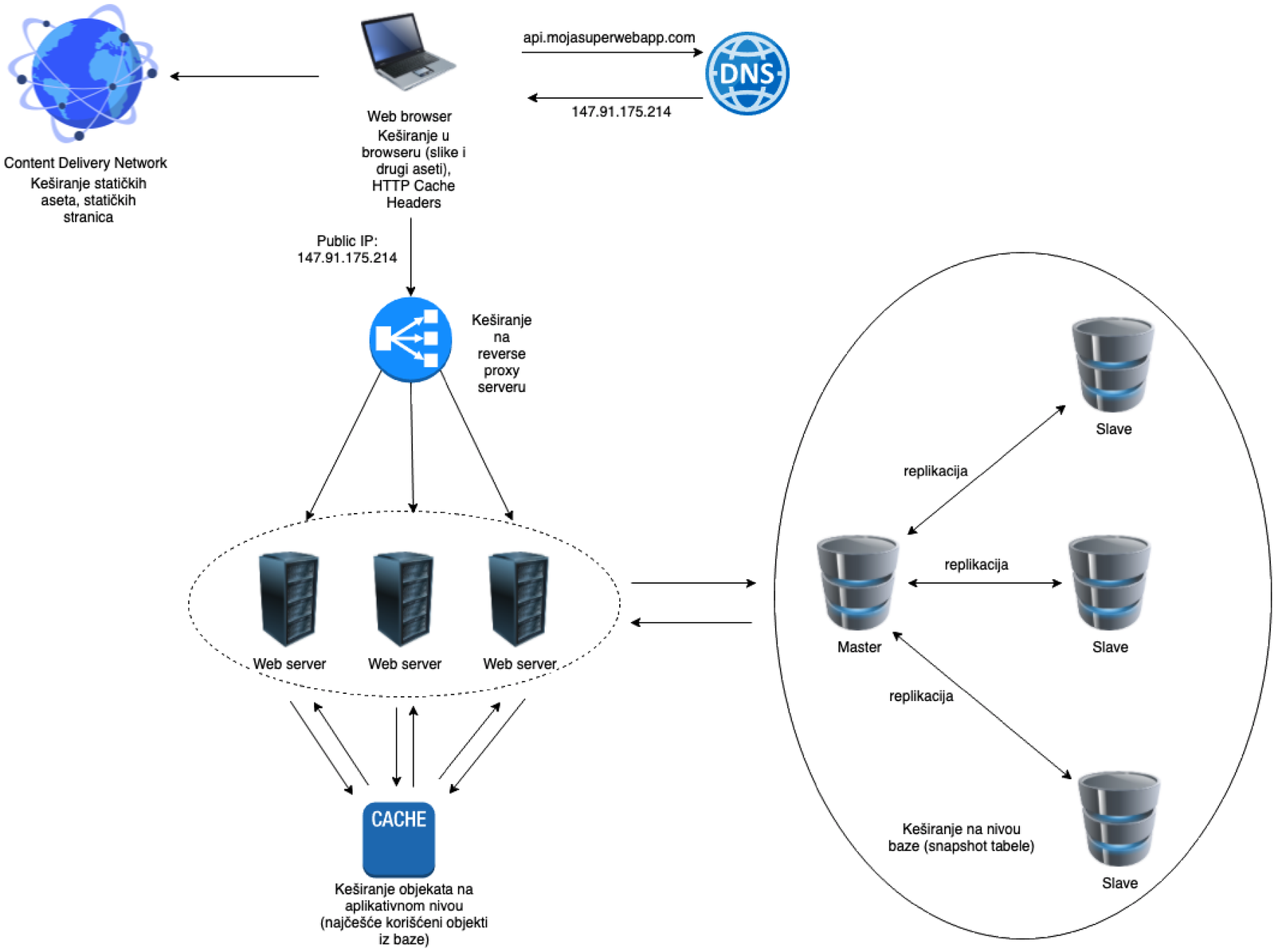
SCENARIJI KORIŠĆENJA

- Koristi se kada treba dodatno poboljšati performanse optimizovanjem pristupa na osnovu geografske lokacije

KANDIDATI ZA KEŠIRANJE

- Statički fajlovi npr. slike, video zapisi, JS fajlovi, itd.

KEŠIRANJE NA VIŠE SLOJEVA



BRISANJE KEŠA

DVE STRATEGIJE ZA INVALIDACIJU KEŠA

- Sistemi za keširanje bi trebali da obezbede načine za brisanje keša
  - Invalidacija bazirana na vremenu – za keš ili delove keša konfiguriše se TTL (time-to-live) i/ili TTI (time-to-idle)
  - Invalidacija na eksplicitan poziv

STRATEGIJE ZA IZBACIVANJE KANDIDATA IZ KEŠA

- LFU (Least Frequently Used)
- LRU (Least Recently Used)
- FIFO (First In First Out)
- Nešto četvrto?

METRIKE ZA PRAĆENJE

ŠTA PRATITI?

- Cache hit ratio – što je veći broj to znači da je strategija keširanja bolja
    - Cache hit ratio – ukupan broj pogodaka / ukupan broj zahteva
- Cache miss ratio – što je veći broj znači da je strategija keširanja lošija
    - Cache miss ratio = 1 – Cache hit ratio
- Cache size – ukupna memorija koju koristi keš

METRIKE ZA PRAĆENJE

CACHE HIT RATIO

- Najvažnija metrika kada se posmatra keširanje
- Meri koliko puta se može ponovo iskoristiti keširani podatak

TRI BITNA FAKTORA KOJA UTIČU NA HIT RATIO

- Veličina skupa podataka
- Zauzeće memorije keširanim podacima
- Trajnost podataka u kešu

POGREŠNE STRATEGIJE ZA KEŠIRANJE

POGREŠNA SELEKCIJA KANDIDATA ZA KEŠIRANJE

- Primeri podataka koji se ne bi trebalo keširati
  - ”Dinamički podaci” – podaci koji se često menjaju te sistem mogu dovesti u nekonzistentno stanje (npr. količina proizvoda na stanju, cene, itd)
  - Lični podaci – podatke o korisniku kojima bi mogli pristupiti i drugi korisnici
  - Poverljivi podaci o poslovanju – podaci koji bi trebali biti dostupni samo nalozima koji imaju specifične uloge u sistemu
  - Veliki ili ugnježdeni objekti – podaci koji mogu da potroše mnogo memorije (npr. čitava kolekcija stranica aplikacije)

PROBLEMI KEŠIRANJA

PAD KEŠ SERVISA (CACHE CRASH)

- Keš servis ne radi pa se svi zahtevi prosleđuju bazi podataka

REŠENJA

- Postavljanje klastera keš servisa kako bi se poboljšala dostupnost keša u slučaju otkaza
- Implementacija *circuit breaker* šablona[1] tako da kada je keš memorija nedostupna, servisi aplikacije ne mogu da posete keš ili bazu podataka

[1] Circuit Breaker Design Pattern https://en.wikipedia.org/wiki/Circuit_breaker_design_pattern

PROBLEMI KEŠIRANJA

PENETRACIJA KEŠ SERVISA (CACHE PENETRATION)

- Ključ ne postoji ni u kešu ni u bazi podataka te aplikacija ne može da preuzme relevantne podatke iz baze podataka da bi se ažurirao keš
- Time se stvara veliki pritisak i na keš i na bazu podataka

REŠENJA

- Keširanje i provera rezervisane vrednosti za nepostojeće ključeve (npr. *null*), čime se izbegava komunikacija sa bazom podataka
- Korišćenje *bloom filtera*[1] da se prvo proveri postojanje ključa, a ako ključ ne postoji, može se izbeći dalja komunikacija sa bazom podataka

[1] Bloom Filter https://en.wikipedia.org/wiki/Bloom_filter

PROBLEMI KEŠIRANJA

THUNDERING HERD PROBLEM[1]

- Veliki broj ključeva u kešu istekne istovremeno pa zahtevi direktno pogađaju bazu podataka, što izaziva preopterećenje

REŠENJA

- Primena postepenog isteka podataka u kešu (staggered expiration) i korišćenje dostupnih mehanizama zaključavanja ili redove poruka za upravljanje protokom zahteva
- Izbegavanje postavljanja istog vremena isteka za ključeve, dodavanjem slučajnog broja u konfiguraciju za vreme
- Dozvola samo zahtevima za bitnim podacima da komuniciraju direktno sa bazom podataka a da se ostalim zahtevima spreči pristup bazi dok keš ne postane ponovo dostupan

[1] Thundering Herd Problem https://www.ehcache.org/documentation/2.8/recipes/thunderingherd.html

PROBLEMI KEŠIRANJA

CACHE STAMPEDE PROBLEM (DOG-PILING, CACHE CHOKING)[1]

- Keširani ključ koji sadrži deo “vrućih” podataka (često traženih) istekne, a više zahteva istovremeno traži iste podatke, preplavljujući sistem zahtevima za ponovnim dobavljanjem/izračunavanjem tih vrućih podataka zbog velike konkurentnosti

REŠENJA

- Zaključavanje keš podatka
  - Da bi se sprečilo više istovremenih ponovnih izračunavanja iste vrednosti, nakon promašaja u kešu, proces će pokušati da dobije *lock* za taj ključ iz keša i ponovo izračuna vrednost samo ako dobije *lock*
- Eksterno izračunavanje
  - Izračunavanje vrednosti za keš se premešta u nezavisni eksterni proces (nit) od procesa (niti, zahteva) kome je potrebna vrednost keša, a zahtevi koji zavise od te vrednosti ne blokiraju sistem, već čekaju da eksterni proces osveži keš
- Probabilističko izračunavanje pre isteka
  - Svaki proces ponovo izračunava vrednost u kešu pre njenog isteka donošenjem nezavisne odluke zasnovane na verovatnoći, pri čemu se verovatnoća ponovnog izračunavanja pre isteka povećava kako se približava isteku vremena

[1] Cache Stampede Problem https://en.wikipedia.org/wiki/Cache_stampede

SOFTVERSKA REŠENJA

Valkey

KeyDB

HAZELCAST

Dragonfly

EHCACHE

redis

SQUID

MEMCACHED

VARNISH CACHE

apache Ignite

NGINX
Part of F5

REFERENCE

- SHIVAKUMAR K. S. ARCHITECTING HIGH PERFORMING, SCALABLE AND AVALIABLE ENTERPRISE WEB APPLICATIONS. https://doi.org/10.1016/B978-0-12-802258-0.00004-4
- MOZILLA. HTTP CACHING. https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching
- SQUID CACHE http://www.squid-cache.org/
- AMAZON. CACHING CHALLENGES AND STRATEGIES. https://aws.amazon.com/builders-library/caching-challenges-and-strategies
- SERVICE WORKERS https://developers.google.com/web/fundamentals/primers/service-workers
- EHCACHE https://ehcache.org

KOJA SU VAŠA PITANJA?