LRU Sistem Keširanja

Seminarski rad u okviru kursa
Konstrukcija i Analiza Algoritama 2
Matematički fakultet

Dušan Trtica, Matematički Fakultet, Beograd

Januar 2024

1 Abstrakt

U ovom radu biće prikazan LRU kešing sistem. Biće opisana njegova upotreba, njegova implementacija kao i data analiza vremenske i prostorne složenosti. Praktična verzija rada se može videti na stranici: https://dusantrtica.github.io/kiaa2-lru/ a kod je dostupan na stranici: https://github.com/dusantrtica/kiaa2-lru

2 Keširanje

Keširanje kao koncept je naširoko poznat u računarskim sistemima. Sistemi za keširanje nam služe da u najkraćem roku dodjemo do podatka koji nam je potreban za odredjenu operaciju. Pod pojmom u najkraćem roku, misli se na to da, ako je moguće zaobidjemo čitanje podataka sa udaljenih servera, praviti složene SQL upite, čitanje podataka sa sporijih uredjaja za čuvanje podataka. U slučaju hardvera, keširanje podrazumeva smeštanje podataka u memorijski prostor koji ima veliku brzinu čitanja i pisanja.

Postoji mnogo načina na koji se sistem keširanja može realizovati, a takodje, često se kombinuju keš sistemi i implementiraju se na više nivoa (L1, L2 keš kod procesora).

3 Keširanje na Webu

Keširanje, kada su u pitanju web aplikacije, uključuje složene sisteme i implementira se na više nivoa. Pod pojmom web aplikacija, možemo podrazumevati aplikaciju koja se, uprošćeno govoreći, sastoji od serverskog dela aplikacije (takozvani bekend sistemi) i frontend dela, klijentske aplikacije, najčešće napisane u Javascript ili Typescript jeziku korišćenjem nekog od frejmvorka (VueJS, Angular, React).
Keširanje u takvim sistemima može uključivati sledeće elemente:

- Keš na nivou baze podataka, implementiran u samoj bazi koji omogućava efikasno dohvatanje stranica tabele, njenih delova, delova particije i slično
- Kako je čitanje i pisanje u bazu u cloud distribuiranim sistemima izuzetno skupo (finansijski gledano), postoje kešing sistemi koji predstavljaju sloj iznad baze podataka i aplikacija prilikom obrade zahteva, prvo konsultuje keš, pa tek ako podatak nije nadjen u kešu, formira se upit prema bazi. Tehnologije koje se često koriste u ovu svrhu su: Redis, Memcache, Hazelcast.
- Keširanje zahteva koji su već u skorijoj prošlosti obradjeni od strane bekenda i velika je verovatnoća da se nisu promenili. Ovakav kešing sistem često daju na raspolaganje Web frejmvorci, poput Java Springa. Nakon obradjenog zahteva sa frontenda, podaci se čuvaju u memoriji neko vreme (unapred zadato).
- Keširanje statičkih fajlova i generisane frontend aplikacije. Inicijalno, kad posećujemo neku aplikaciju na webu, dohvata se index.html fajl koji u sebi sadrži reference na ostale delove aplikacije: Javascript, CSS fajlove potrebne da bi aplikacija funkcionisala. Keširanje ovih fajlova se realizuje kako na serverskoj strani (web serveri, poput Nginxa već imaju ugradjene ove funkcionalnosti), tako i na klijentskoj strani: browseri čuvaju ove fajlove lokalno. Izazov je da osiguramo dobre performanse sistema s jedne strane, ali i da korisnik dobije najsvežiju verziju aplikacije. To se postiže time što index.html se nikad ne kešira, uvek se dakle, čita sa web servera, a linkovi na Javascript i CSS fajlove u sebi imaju heš koja se prilikom svakog narednog builda menja.

4 LRU Keš Sistem i njegova upotreba

LRU je skraćenica od: Least Recently Used - najdavnije korišćen - i podrazumeva keš u kojem su podaci u kešu uredjeni po tome kad im se poslednji put pristupalo. Ovaj keš sistem se razlikuje od ostalih po tome što se podaci, ne samo čuvaju u memoriji kako bi im se brše pristupilo, već i po tome što redosled podataka u kešu direktno odražava poredak u kojem se tim podacima pristupalo. Takodje, kako se keš popunjava, redosled izbacivanja ide u obrnutom poretku od onog kojem je podacima pristupano. Drugim rečima, podatak kojem se pristupalo najdavnije, biva izbačen iz keša prvi. Primeri su mnogobrojni:

- Sistemi za kupovinu najčešće uredjuju artikle po tome kad ste pregledali odredjene artikle, pa tako, aritikli kojima ste pristupali najskorije, prikazaće se prvi.
- Društvene mreže isto tako favorizuju kontakte kojima ste najskorije pristupali
- Operativni sistemi koriste LRU kešing sistem da segmente memorije kojima se najčešće pristupalo održe u keš memoriji što je moguće duže

- Kod baza podataka, stranica koja sadrži podatke kojima se skoro pristupalo, biće u kešu ispred one kojoj se ranije pristupalo.

Ukratko, gotovo svi keš sistemi danas, sadrže neku logiku koja selektivno izabacuje podatke iz keša a ne proizvoljno, kad se keš memorija popuni. I ne samo to, sadržaj keša se često periodično evaluira kako se dobili podaci o tome koji je proizvod popularan i u koje vreme, kad se on izabacuje iz keša i tako dalje. Kako su podaci u LRU kešu uredjeni po tzv. popularnosti, ovo je vrlo kvalitetan podatak koji može biti iskorišćen u realnom vremenu u analitičke svrhe.

5 Implementacija LRU keš sistema i analiza složenosti

Podjimo od jednostavnog primera, veličina keša je 4, a vrednosti su celi brojevi. Unosimo redom brojeve: 17, 21, 8 i 12. Ako keš posmatramo kao listu, njen sadržaj treba da bude: 12, 8, 21, 17 - baš tako da odražava činjenicu da se podatku 12 poslednjem pristupalo i da se on uneo poslednji. Ovo nas odmah asocira na povezanu listu, i zaista, povezana lista se koristi u implementaciji LRU keša. Ako želimo da unesemo podatak 15, pošto je veličina keša 4, jedan podatak se mora izbaciti, a to je podatak 17. Sadržaj keša sada je: 15, 12, 8, 21. Kako bismo efikasno pristupili poslednjem elementu, umesto obične povezane liste, koristimo dvostruko povezanu listu i imamo referencu na tail, odnosno poslednji element liste. Unošenje elementa u listu kao i njegovo uklanjanje ima vremensku složenost O(1)

Dalje, pretpostavimo da pristupamo elementu 8. Nakon pristupanja tom elementu, on postaje tzv. najpopularniji i treba da dodje na čelo liste. Dakle nakon čitanja elementa 8, sadržaj keša sada treba da bude: 8, 15, 12, 21. Ako je poznat čvor liste sa vrednošću 8, njegovo prebacivanje na čelo liste, odnosno postavljanje za novog heada, se može realizovati u O(1) vremenskoj složenosti, jednostavnim razvezivanjem i ponovnim povezivanjem pokazivača. Medjutim, problem je kako naći element liste, jer znamo pretraga liste ima O(n) složenost. Ove nam u pomoć služi heš mapa, koja sadrži kluč-vrednost parove, gde je ključ element kojem se pristupa, a vrednosti su čvorovi povezane liste. Kad kažemo da je ključ element kojem se pristupa mislimo na njegovu primitivnu vrednost. U realnom svetu to su primarni ključevi artikala, heš vrednosti stringova, hex vrednosti adresa u memoriji itd. U našem slučaju, to je celobrojna vrednost, koja predstavlja podatak koji unosimo i čitamo.

Dakle, za implementaciju LRU keš sistema, koristimo:

- Dvostruko povezanu listu i čuvamo reference na početak i kraj liste.
- Heš tabelu koja sadrži parove: primitivna vrednost podatka i pokazivač na element liste.

Operacije nad LRU keš sistemom podrazumevaju:

- Inicijalizacija keša, u konstruktoru se zadaje kapacitet keša i pravimo objekte liste i mape.
- *get(element)*:
  ako se element nalazi u heš mapi onda
  1. postavi head liste na map[element]
  2. vrati traženi element
  inače:
  1. Dohvati podatak (recimo, iz baze),
  2. Umetni element u mapu
  3. Umetni element na početak liste
  4. Izbaci poslednji element iz liste ukoliko je kapacitet keša (postavljen u konstuktoru) manji od broja elemenata u mapi.
  5. Vrati traženi element
- evictFromCache: Brisanje iz keša realizujemo tako što uklanjamo poslednji element iz liste - pomerajući tail na tail.previousNode i brišući element iz mape.

Vremenska složenost za operaciju unosa kao i brisanja je O(1) Prostorna složenost je O(n): heš mapa i povezana lista veličine n.

6 Projekat

U praktičnom projektu koji je sastavni deo ovog rada, prikazana je pojednostavljena verzija LRU kešing sistema, koja ne uključuje dohvatanje podataka sa udaljenih servera. Radi se o selekciji emotikona kroz widget koji je naširoko prepoznatljiv korisnicima mobilnih aplikacija. LRU je implementiran kroz Recently used (nedavno korišćene) emotikone, tako da, kako biramo emotikone, oni postaju deo liste nedavno korišćenih. Kako im dalje pristupamo, redosled liste se ažurira da odražava opisani princip rada LRU keša. Veličina keša, zbog prakičnosti je postavljena na 5, ali se može lako izmeniti shodno potrebi.

Aplikacija je napravljena u Vue3 frontend frameworku, za testove je korišćen Vitest i vue test utils biblioteka. Jezik implementacije je Typescript.

Ključne klase u ovom projektu su:

- LinkedList - klasa koja enkapsulira ponašanje povezane liste pratećim operacijama umetanja, brisanja čvorova, kao i postavljanje proizvoljnog čvora za početak liste. Funkcionalnost klase pokrivena testovima. Klasa LinkedList implementira iterator,kako bismo mogli da iteriramo kroz elemente liste i prikazujemo njen sadržaj.
- LRUCache - klasa koja enkapsulira kešing sistem. Metode korišćene u ovom projektu su: insert, koja dodaje emotikon u keš i izbacuje poslednji, ako treba, daje mogućnost da iterira kroz sadržaj keša kako bismo mogli na stranici da prikažemo Recently used u poretku u kojem im se pristupalo.
- Emoji - klasa koja simulira podatak koji čuvamo u kešu.

Sve ključne funkcionalnosti su temeljno pokrivene testovima.

7 Zaključak

LRU keš sistem je naširoko korišćen sistem za keširanje podataka koji odražava vremensku dimenziju pristupa tim podacima i podaci su uredjeni u kešu kako im se pristupalo. Poslednji element keša kome se pristupalo biva postavljen na čelo liste a ukoliko je potrebno ukloniti neki element iz keša, izbacujemo element kojem smo najdavnije pristupali.

Za implementaciju LRU keša, najčešće koristimo dvostruko povezanu listu, koja odražava redosled elemenata, kao i običnu keš mapu za brzo pronalaženje čvorova liste.

Pristupanje elementima keša, kao i brisanje iz keša, ima vremensku složenost O(1) Implementacija keša ima prostornu složenos O(n)