https://doi.org/10.63356/stes.ing.2026.003 УДК 004.891:004.451.3

ANALIZA UTICAJA DVOSLOJNOG KEŠIRANJA I PROREĐENIH INDEKSA NA LATENCIJU ČITANJA U LSM SKLADIŠTIMA PODATAKA

Autor: ALEKSA DEJANOVIĆ, Petar Popović, Boško Vasilić, Sara Stojkov
e-mail: aleksa@dejanovic.online
Mentor: doc. Miloš Simić
Departman za računarstvo i automatiku, Fakultet tehničkih nauka Univerziteta u Novom Sadu

Uvod: Skladišta podataka zasnovana na LSM stablu (Log-Structured Merge-Tree) omogućavaju visoku propusnost upisa, ali pretraga više nivoa i pojačano čitanje sa diska mogu povećati latenciju čitanja.

Cilj: Eksperimentalno ispitati uticaj dvoslojnog keširanja (aplikativni LRU Cache i diskovni Block Cache), te različitih nivoa proređenosti Summary/Index struktura na performanse tačkastih GET upita u prilagođenom LSM Key-Value sistemu.

Materijal i metode: Implementiran je Key-Value sistem sa dvoslojnim keširanjem i podesivim Summary/Index strukturama. Evaluacija je obuhvatila 10^6 operacija čitanja sa Zipfovom i uniformnom distribucijom, nad postojećim i nepostojećim ključevima. Varirane su veličine blokova od 4, 8 i 16 KB i odnosi uzorkovanja Summary/Index struktura od 1 : 4 do 1 : 64.

Rezultati: Najbolje ukupne performanse postignute su blokovima od 4 KB, odnosom uzorkovanja Summary strukture 1 : 16 i aktivnim L1 i L2 kešom. L2 keš je, u odnosu na konfiguraciju bez keširanja, smanjio prosečnu latenciju sa 1.048,2 na 214,7 μs (79,5%), a kombinacija L1 i L2 keša na 178,4 μs (83,0%). U mešovitom opterećenju od 10^6 zahteva dvoslojno keširanje je, u odnosu na samo L2 keš, smanjilo prosečnu GET latenciju sa 220,30 na 182,36 μs (17,2%) i povećalo propusnost sa 4.539 na 5.484 GET/s (20,8%). L1 keš ostvario je stopu pogodaka od 37,5%, uz 16,5% manje pristupa L2 kešu.

Zaključak: Dvoslojno keširanje značajno unapređuje performanse čitanja, a veličina blokova i proređenost indeksa omogućavaju balansiranje latencije, propusnosti i memorijskih zahteva. U evaluiranom sistemu najpovoljnija je kombinacija blokova od 4 KB, odnosa uzorkovanja 1 : 16 i aktivnih L1 i L2 keševa.

Ključne reči: LSM stablo ; Key-Value skladište ; Block Cache ; LRU Cache ; latencija čitanja

UVOD

Log-Structured Merge-Tree (LSM) predstavlja pristup organizaciji podataka koji optimizuje obradu velikog broja upisa tako što se izmene najpre akumuliraju u memoriji, a zatim sekvencijalno zapisuju u sortirane strukture na disku. Osnovni princip LSM arhitekture omogućava efikasno izvršavanje operacija pisanja, ali istovremeno uvodi potrebu za pretragom više diskovnih struktura tokom čitanja, naročito kada se podaci nalaze na različitim nivoima stabla [1]. Zbog toga performanse LSM sistema predstavljaju kompromis između efikasnosti upisa, troška kompakcije i troška čitanja. Savremene LSM implementacije, poput LevelDB-a, koriste memorijske bafere i nepromenljive SSTable strukture organizovane po nivoima, pri čemu SSTable obično sadrži blokove podataka, indeksne i filterske strukture koje usmeravaju *point lookup* prema relevantnom delu podataka i mogu smanjiti broj nepotrebnih pristupa disku [2–4].

Kod tačkastih upita, pronalaženje vrednosti može obuhvatiti više komponenata *read path*-a, uključujući memorijske strukture, *Bloom* filtere, indeksne strukture i blokove podataka *SSTable* datoteka. Broj potencijalnih diskovnih pristupa zavisi od organizacije *LSM* nivoa, raspodele podataka i uspešnosti mehanizama koji omogućavaju rano isključivanje *SSTable* datoteka. Posledično, *read amplification* predstavlja jedan od važnih faktora koji ograničavaju performanse *LSM Key-Value* sistema [5].

Jedan od ključnih mehanizama za smanjenje broja nepotrebnih pristupa *SSTable* datotekama jesu probabilistički filteri. *Bloom* filter omogućava da se sa određenom verovatnoćom utvrdi da ključ nije prisutan u skupu, čime se može izbeći nepotrebno čitanje podataka, uz mogućnost lažno pozitivnog odgovora [5]. Dayan i saradnici analiziraju raspodelu memorije između *Bloom* filtera i drugih komponenti *LSM* sistema i pokazuju da izbor memorijskog budžeta može značajno uticati na trošak tačkastih upita [6]. Njihovi rezultati ukazuju da optimizacija memorijskih struktura predstavlja važan deo smanjenja *read amplification*-a, ali se razmatrani mehanizam razlikuje od konfiguracije ispitane u ovom radu.

Pored probabilističkih filtera, značajnu ulogu u *read path*-u imaju indeksne strukture i mehanizmi keširanja. Keširanje omogućava da se često korišćeni podaci ili delovi *SSTable* datoteka ponovo koriste bez ponovnog pristupa sporijem nivou memorijske hijerarhije. Wu i saradnici analiziraju adaptivno keširanje u *Key-Value* sistemima baziranim na *LSM* stablu i razlikuju keširanje parova ključ–vrednost, pokazivača i blokova [7]. Njihovi rezultati pokazuju da različite vrste keširanih objekata imaju različite troškove i koristi, što ukazuje na značaj zajedničkog posmatranja više nivoa keširanja.

Dodatni aspekt predstavlja organizacija indeksnih i filter struktura unutar blokovskog keša. RocksDB analiza particionisanih indeksa i filtera pokazuje da indeksni i filter blokovi mogu zauzimati značajan deo raspoloživog prostora i konkurisati blokovima podataka za mesto u *Block Cache*-u [8]. Time izbor veličine i organizacije indeksnih struktura postaje povezan sa efikasnošću keširanja. Ovaj odnos je posebno značajan kada se indeksne strukture proređuju: smanjenje njihovog memorijskog otiska može smanjiti potrošnju memorije, ali istovremeno povećati količinu podataka koju je potrebno pretražiti između dva susedna indeksna zapisa.

Na performanse *LSM* sistema utiče i način kompaktovanja podataka. Sarkar i saradnici analiziraju prostor dizajna *LSM* kompaktovanja i pokazuju da različite strategije predstavljaju različite kompromise između *read amplification*-a, *write amplification*-a i zauzeća prostora [2]. Dostoevsky dodatno pokazuje da izbor politike kompaktovanja može promeniti odnos između troška upisa, troška upita i prostorne efikasnosti [9]. Ovi rezultati potvrđuju da performanse LSM sistema zavise od međusobno povezanih arhitektonskih odluka, a ne od jednog izolovanog parametra.

Postojeća istraživanja, međutim, uglavnom posmatraju pojedinačne aspekte ovog problema: optimizaciju *Bloom* filtera i memorije [6], adaptivno upravljanje različitim vrstama keša [7], organizaciju indeksnih i filter struktura [8] ili strategije kompakcije [2,9]. U ovom radu fokus je na eksperimentalnom ispitivanju međusobnog uticaja aplikativnog *Least Recently Used* (*LRU*) keša, blokovskog *LRU* keša, veličine bloka i proređenosti *Summary/Index* struktura u okviru jedne implementacije *LSM Key-Value* sistema. Cilj nije direktno poređenje sa rezultatima prethodnih radova, već izolovanje efekata navedenih parametara i određivanje konfiguracije koja u posmatranom eksperimentalnom okruženju ostvaruje najpovoljnije performanse tačkastih *GET* upita.

U ovom radu pokušaćemo da odgovorimo na sledeća pitanja: koliko L1 i L2 keš pojedinačno i zajedno menjaju izmerenu latenciju? Kako veličina bloka i gustina Summary utiču na jednako ponderisani prosek četiri dijagnostička opterećenja? Koliki je dodatni efekat L1 u milionskom jednonitnom mešovitom nizu u odnosu na samo L2?

MATERIJAL I METODE

Materijal i eksperimentalno okruženje

Eksperimentalna evaluacija je sprovedena nad implementacijom sistema za skladištenje parova ključ–vrednost zasnovanom na *LSM* stablu. Sistem je implementiran u programskom jeziku Go. Eksperiment je izveden na računaru sa procesorom AMD Ryzen 9 7900, koji ima 12 fizičkih jezgara i 24 logičke niti, baznu frekvenciju od 3,7 GHz i maksimalnu fabričku boost frekvenciju od 5,4 GHz uz 32GB DDR5-4800 RAM memorije. Računar je koristio Kingston NVMe SSD (SNV2S2000G) nominalnog kapaciteta 2 TB i operativni sistem EndeavourOS sa Linux jezgrom 7.2.11-arch1-1.

Predmet merenja bila je tačkasta operacija *GET* nad stablom sa tri popunjena nivoa i ukupno 14 *SSTable* datoteka. Nivo L0 sadržao je dve preklapajuće tabele, pri čemu je novija tabela obuhvatala 10% prostora ključeva, a starija se preklapala sa polovinom njenog sadržaja. Nivo L1 sadržao je četiri nepreklapajuće tabele sa narednih 30% ključeva, a L2 osam nepreklapajućih tabela sa preostalih 60%. Takav raspored omogućio je da se uspešna čitanja završavaju na različitim nivoima, uz proveru novijih verzija zapisa pre starijih na nivou L0. Pri neuspešnom čitanju ispitivani su *Bloom* filteri svih 14 tabela i pripadajuće putanje pristupa metapodacima.

Skup podataka obuhvatao je 10.000 jedinstvenih ključeva sa vrednostima veličine 256 bajtova. Testne *SSTable* datoteke generisane su neposredno u formatu koji koristi čitač; njihova izrada nije bila uključena u mereno vreme. Merenja su obuhvatila postojeću putanju čitanja kroz *LSM* stablo, bez konkurentnih upisa i pozadinske kompakcije. Latencija operacije *GET* merena je pomoću monotonog sata. U testovima nisu bili zastupljeni zapisi brisanja (*tombstones*). Eksperiment je izvođen u jednoj niti.

Putanja čitanja i keširanje

Zahtev *GET* najpre prolazi kroz proveru memorijskih struktura, a zatim kroz aplikativni *LRU* keš (L1). Ako tražena vrednost nije dostupna u L1 kešu, pretraga se nastavlja kroz *SSTable* datoteke. Za svaku tabelu *Bloom* filter može da isključi dalje pretraživanje; kada ga ne isključi, strukture *Summary* i *Index* usmeravaju pretragu ka odgovarajućem podatkovnom bloku. Pristup blokovima obavlja *Block Manager*, uz zaseban *LRU* keš blokova (L2). Promašaj u L2 dovodi do čitanja datoteke. L1 čuva rezultate uspešnih čitanja, ali ne kešira odsustvo ključa, pa neuspešni *GET* zahtevi prolaze kroz putanju pretrage *SSTable* datoteka. Pomoćne strukture *SSTable* datoteka čuvaju se na disku, odnosno pristupa im se kroz L2 keš.

Kapacitet L1 keša iznosio je 1.000 zapisa. Memorijski budžet L2 keša bio je približno 2 MiB i ostao je fiksan pri promeni veličine bloka; za blokove od 4 KB odgovarao je kapacitetu od 512 blokova. U eksperimentu su razmatrana četiri režima: bez oba keša, samo L1, samo L2 i istovremeno L1 i L2. Time je omogućeno da se uticaj keša rezultata razlikuje od uticaja keša blokova.

Nezavisne promenljive i protokol merenja

Veličine blokova bile su 4, 8 i 16 KB, a odnosi uzorkovanja strukture *Summary* 1 : 4, 1 : 16 i 1 : 64. Ovi odnosi određuju gustinu zapisa u strukturi *Summary*. *Bloom* filter i gustina strukture *Index* nisu menjani kao nezavisne promenljive.

Evaluacija je imala dve faze. Prvu je činila dijagnostička matrica svih 144 kombinacija tri veličine bloka, tri odnosa uzorkovanja *Summary*, četiri režima keširanja i četiri vrste

opterećenja: čitanja iz često korišćenog skupa ključeva, uniformna uspešna čitanja, uspešna čitanja sa Zipfovom raspodelom i čitanja nepostojećih ključeva. Pre izvršavanja svake kombinacije keš je pražnjen kako bi merenja počinjala bez zapisa iz prethodne kombinacije. Za svaku kombinaciju izvršeno je 100 zahteva za zagrevanje, a zatim 300 merenih GET zahteva. Broj od 100 zahteva izabran je kao kratko zagrevanje za dijagnostičko poređenje svih 144 kombinacija, tako da merenje u svakoj kombinaciji počne nakon istog broja početnih zahteva. Stepen popunjenosti keša nakon zagrevanja mogao je da se razlikuje među vrstama opterećenja. Ova faza služila je za poređenje konfiguracija i izbor parametara za duži test. Ova faza služila je za poređenje konfiguracija i izbor parametara za duži test. Pri poređenju srednjih latencija četiri pojedinačna opterećenja imala su jednaku težinu; tako dobijeni prosek nije srednja latencija mešovitog testa.

U prvoj fazi eksperimenta svaka kombinacija veličine bloka i odnosa uzorkovanja strukture Summary merena je u deset nezavisnih ponavljanja. Za odnos 1 : 16 srednja latencija iznosila je 178,4 μs (95% interval poverenja od 172,0 do 184,8 μs), a za odnos 1 : 64 iznosila je 187,3 μs (95% interval poverenja od 180,9 do 193,7 μs). Standardna devijacija srednjih latencija između ponavljanja iznosila je približno 9 μs za obe konfiguracije.

Drugu fazu činio je deterministički mešoviti niz od 1.000.000 merenih *GET* zahteva, kojem je prethodilo 10.000 zahteva za zagrevanje. Niz je sadržao 50% uniformnih uspešnih čitanja, 30% uspešnih čitanja prema Zipfovoj raspodeli sa parametrom s=1,2, 10% čitanja usmerenih na često korišćeni skup koji obuhvata 1% ključeva i 10% čitanja nepostojećih ključeva. Isti niz korišćen je za poređenje režima samo L2 i L1+L2, uz blokove od 4 KB i odnos *Summary* 1 : 16.

Pokazatelji i obrada rezultata

Osnovni pokazatelj bila je srednja latencija *GET* zahteva, izražena u mikrosekundama (μs). Za mešoviti test praćeni su i propusnost u zahtevima u sekundi (*GET/s*), medijana (p50), p95 i p99 latencija, stopa pogodaka L1 i L2 keša, broj pristupa L2 kešu, broj čitanja datoteka ispod L2, kao i alocirana memorija i broj alokacija po zahtevu. Stopa pogodaka keša odnosi se na udeo pogodaka među odgovarajućim proverama keša. Stopa pogodaka L2 izračunata je kao odnos broja pogodaka i ukupnog broja pojedinačnih provera blokovskog keša. Jedan GET zahtev može da obuhvati više provera L2 tokom pretrage SSTable datoteka. Prosečan obim alokacija po GET zahtevu dobijen je deljenjem ukupnog broja bajtova alociranih tokom merenog dela testa brojem merenih zahteva. Pokazatelj obuhvata i privremene alokacije čija se memorija može osloboditi tokom testa. Pokazatelj p99 predstavlja prag ispod kojeg se nalazi približno 99% izmerenih latencija. Vrednosti p99 iz dijagnostičke matrice, sa 300 merenih zahteva po kombinaciji, treba tumačiti kao orijentacione; poređenje repa raspodele prvenstveno se zasniva na dužem testu.

Relativno smanjenje latencije u odnosu na referentnu konfiguraciju računato je kao

$$R_L = \frac{L_0 - L_1}{L_0} \cdot 100\%$$

gde su L_0 i L_1 srednje latencije referentne i poređene konfiguracije. Relativna promena propusnosti računata je kao

$$R_T = \frac{T_1 - T_0}{T_0} \cdot 100\%$$

gde su T_0 i T_1 odgovarajuće propusnosti.

Datoteke su tokom testa ostale u kešu stranica operativnog sistema. Stoga, broj čitanja datoteka ispod aplikativnog L2 keša označava pozive putanji čitanja datoteka, a ne broj fizičkih čitanja skladišnog uređaja.

REZULTATI

Rezultati su prikazani odvojeno za dijagnostičku matricu, u kojoj je za svaku kombinaciju izmereno po 300 *GET* zahteva nakon 100 zahteva za zagrevanje i za mešovito opterećenje sa po 1.000.000 merenih zahteva u dva režima keširanja. Tabela br. 1 prikazuje srednju latenciju četiri vrste opterećenja pri veličini bloka od 4 KB i odnosu uzorkovanja *Summary* strukture 1 : 16. Bez aplikativnih keševa aritmetička sredina četiri prikazane latencije iznosila je 1.048,2 μs. Sa uključenim samo blokovskim kešom L2 iznosila je 214,7 μs, što je za 79,5% manje. Kada su oba keša bila aktivna, iznosila je 178,4 μs, odnosno 83,0% manje nego bez keša i 16,9% manje nego sa samo L2 kešom. Ove vrednosti predstavljaju jednako ponderisane proseke četiri dijagnostička opterećenja, a ne merenja jednog niza *GET* zahteva (Tabela br. 1).

Opterećenje	Bez keša	Samo L1	Samo L2	L1 + L2
Često korišćeni ključevi	439,9	0,1	74,9	0,2
Uniformna uspešna čitanja	1.849,6	1.761,5	347,8	308,4
Uspešna čitanja sa Zipfovom raspodelom	501,5	301,7	92,3	58,5
Nepostojeći ključevi	1.401,9	1.412,9	343,9	346,6
Aritmetička sredina četiri opterećenja[1]	1.048,2	869,1	214,7	178,4

Tabela br. 1 Srednja latencija *GET* zahteva po vrsti opterećenja i režimu keširanja (μs; blok 4 KB, *Summary* 1 : 16)

Razlike među vrstama opterećenja bile su izrazite. Pri čitanju često korišćenih ključeva L1 je, nakon zagrevanja, omogućio latenciju od 0,1 μs u režimu samo L1, odnosno 0,2 μs kada su oba keša bila uključena. Kod uniformnih uspešnih čitanja najveće smanjenje u odnosu na režim bez keša ostvareno je uključivanjem L2, sa 1.849,6 na 347,8 μs; dodavanje L1 smanjilo je latenciju na 308,4 μs. Kod čitanja sa Zipfovom raspodelom dodavanje L1 uz aktivni L2 smanjilo je latenciju sa 92,3 na 58,5 μs. Za nepostojeće ključeve latencije režima samo L2 i L1+L2 bile su bliske, 343,9 i 346,6 μs, što je u skladu sa činjenicom da L1 ne kešira neuspešna čitanja. Male razlike u okviru ovog kratkog testa ne predstavljaju dokaz prednosti jednog od ta dva režima za neuspešna čitanja.

Mešovito opterećenje od milion GET zahteva

U dužem testu poređeni su režimi samo L2 i L1+L2 pri bloku od 4 KB i odnosu *Summary* 1 : 16. Oba režima izmerena su na istom determinističkom nizu od 1.000.000 zahteva nakon 10.000 zahteva za zagrevanje. Uključivanje L1 smanjilo je srednju latenciju sa 220,30 na 182,36 μs, odnosno za 37,94 μs (17,2%), dok je propusnost porasla sa 4.539 na 5.484 *GET/s* (20,8%).

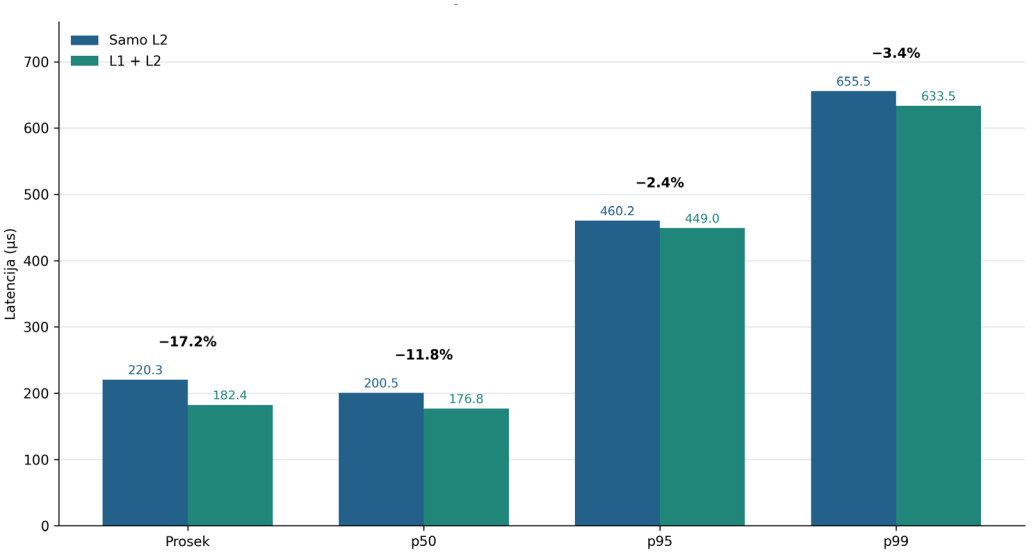
Stopa pogodaka L1 u režimu L1+L2 iznosila je 37,5%. Istovremeno je broj provera L2 opao za približno 30,6 miliona, dok se broj čitanja datoteka ispod L2 smanjio za 7.099. Stopa pogodaka L2 ostala je približno ista u oba režima. Ovi podaci ukazuju na to da je dobitak od L1 u ovom testu pre svega povezan sa izbegavanjem dela pristupa keširanim blokovima i obrade struktura *Summary* i *Index*, a u znatno manjoj meri sa smanjenjem broja čitanja datoteka.

Tabela br. 2 prikazuje r**ezultate mešovitog opterećenja od 1.000.000 GET zahteva po režimu keširanja. Broj čitanja datoteka u Tabeli br. 2 označava pozive ispod aplikativnog L2 keša: datoteke su ostale u kešu stranica operativnog sistema, pa se te vrednosti ne mogu tumačiti kao broj fizičkih čitanja skladišnog uređaja (Tabela**

br. 2). Smanjenje p99 latencije iznosilo je 21,96 µs, odnosno 3,4%, i bilo je manje od relativnog smanjenja srednje latencije. Slično tome, p95 latencija smanjena je za 2,4%. Prema tome, dobitak u prosečnom vremenu odziva nije bio praćen jednako velikim dobitkom u repu raspodele latencija (Grafikon br. 1).

Pokazatelj	Samo L2	L1 + L2	Promena pri dodavanju L1
Srednja latencija (µs)	220,30	182,36	−17,2%
p50 latencija (µs)	200,48	176,82	−11,8%
p95 latencija (µs)	460,18	449,02	−2,4%
p99 latencija (µs)	655,46	633,50	−3,4%
Propusnost (*GET/s*)	4.539	5.484	+20,8%
Stopa pogodaka L1 (%)	0	37,5	+37,5 procentnih poena
Stopa pogodaka L2 (%)	99,876	99,856	−0,020 procentnih poena
Broj provera L2 (miliona)	185,03	154,43	−16,5%
Broj čitanja datoteka ispod L2	228.792	221.693	−3,1%
Alocirana memorija po *GET* zahtevu (KiB)	758,4	633,0	−16,5%
Broj alokacija po *GET* zahtevu	412,9	344,7	−16,5%

Tabela br. 2 Rezultati mešovitog opterećenja od 1.000.000 *GET* zahteva po režimu keširanja.

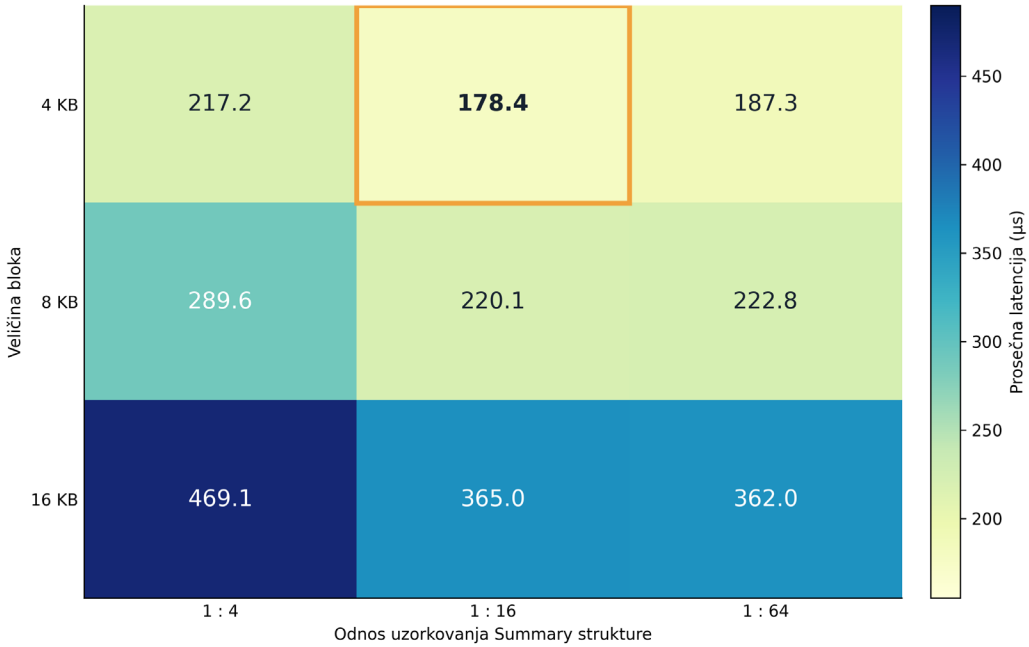


Grafikon br. 1 Latencija pri milion *GET* zahteva

Uticaj veličine bloka i uzorkovanja strukture Summary

Za svaku od devet kombinacija veličine bloka i odnosa uzorkovanja *Summary* strukture izračunata je aritmetička sredina latencija četiri dijagnostička opterećenja sa uključenim keševima L1 i L2. Najnižu vrednost u ovoj matrici, 178,4 µs, ostvarila je kombinacija bloka od 4 KB i odnosa 1 : 16. Pri istoj veličini bloka, odnos 1 : 64 dao je 187,3 µs, a odnos 1 : 4 dao je 217,2 µs. U sve tri ispitane vrednosti odnosa *Summary*, blok od 4 KB imao je

nižu srednju latenciju od blokova veličine 8 i 16 KB (Grafikon br. 2).



Grafikon br. 2 Veličina bloka i uzorkovanje *Summary* strukture (Latencija izvedena kao prosek četiri jednako ponderisana opterećenja)

Razlika između odnosa 1 : 16 i 1 : 64 pri bloku od 4 KB iznosila je 8,9 µs, odnosno latencija pri odnosu 1 : 16 bila je približno 4,8% niža u odnosu na 1 : 64. Međutim, prednost nije bila jednaka za sve vrste zahteva (Tabela br. 3): odnos 1 : 64 bio je brži za uspešna čitanja sa Zipfovom raspodelom, dok je 1 : 16 bio brži za uniformna uspešna čitanja. Zbog toga se 1 : 16 označava kao najbolja ispitana kombinacija prema jednako ponderisanoj srednjoj latenciji četiri opterećenja, a ne kao najbolji izbor za svaku pojedinačnu raspodelu zahteva.

Opterećenje	***Summary* 1 : 16**	***Summary* 1 : 64**
Često korišćeni ključevi	0,2	0,2
Uniformna uspešna čitanja	**308,4**	346,8
Uspešna čitanja sa Zipfovom raspodelom	58,5	**53,1**
Nepostojeći ključevi	346,6	349,1
Aritmetička sredina četiri opterećenja	**178,4**	187,3

Tabela br. 3 Srednja latencija po vrsti opterećenja za dva odnosa *Summary* uzorkovanja pri bloku od 4 KB i aktivnim keševima L1+L2 (µs)

DISKUSIJA

Dobijeni rezultati ukazuju da najveći doprinos smanjenju latencije u posmatranom sistemu dolazi od smanjenja troška pristupa diskovnim strukturama. Uključivanje L2 blokovskog keša dovelo je do izrazitog smanjenja prosečne latencije, dok je dodatni L1 keš ostvario još jedno, ali manje, poboljšanje. Ovakav odnos je očekivan za *LSM* arhitekturu, u kojoj point lookup može zahtevati pretragu više nivoa i više *SSTable* struktura. Prethodna

istraživanja takođe ističu read amplification kao jedan od osnovnih problema *LSM* sistema i keširanje kao važan mehanizam za njegovo ublažavanje [1,7]. Rezultati ovog rada zato ne ukazuju samo na korist samog keša, već i na činjenicu da se različiti nivoi keširanja obraćaju različitim delovima troška read path-a.

Dodatni efekat L1 keša može se objasniti činjenicom da on čuva kompletan zapis ključ–vrednost, čime se za pogođene zahteve izbegava nastavak pretrage prema *SSTable* strukturama. Njegova korist zato zavisi od lokalnosti pristupa, odnosno od toga koliko se često isti ključevi ponavljaju u kratkom vremenskom intervalu. Ovo je u skladu sa AC-Key pristupom, koji razlikuje *key-value*, *key-pointer* i *block cache* i pokazuje da trošak i korist keširanja nisu isti za sve vrste sadržaja [7]. U posmatranom sistemu L1 ostvaruje pogodak za deo zahteva, dok preostali zahtevi nastavljaju prema L2 i diskovnim strukturama. Zbog toga je razumno očekivati da bi promena raspodele popularnosti ključeva mogla promeniti relativni doprinos L1 sloja.

Režimi samo L2 i L1+L2 poređeni su pri različitim ukupnim memorijskim budžetima. Kapacitet L2 keša u oba režima iznosio je približno 2 MiB, dok je u režimu L1+L2 dodatni L1 keš mogao da čuva do 1.000 zapisa. Izmereno smanjenje latencije stoga pokazuje korist dodavanja L1 keša u ispitanoj konfiguraciji, pri čemu se doprinos dvoslojne organizacije i doprinos dodatne memorije na osnovu ovog poređenja ne mogu razdvojiti. U budućem kontrolnom eksperimentu režim L1+L2 trebalo bi uporediti sa režimom samo L2 čiji je kapacitet uvećan za memorijski ekvivalent L1 keša, tako da oba režima imaju isti ukupni memorijski budžet.

Pri tumačenju rezultata važno je razlikovati stopu pogodaka L1 keša od smanjenja broja pristupa L2 kešu. Vrednosti od 37,5% i 16,5% ne predstavljaju dve mere istog događaja, jer se zasnivaju na različitim skupovima zahteva i opisuju različite tačke u read path-u. L1 pogodak završava obradu zahteva pre pristupa L2, dok smanjenje L2 pristupa zavisi i od toga koji bi zahtevi bez L1 nastavili do blokovskog keša. Ovakvo razlikovanje je značajno i u kontekstu prethodnih istraživanja, u kojima se trošak i korist pojedinačnih keš komponenti posmatraju odvojeno upravo zbog različitih uloga koje imaju u *LSM* sistemu [7].

Broj poziva putanji čitanja datoteka ispod aplikativnog L2 keša opisuje aktivnost na aplikativnoj putanji čitanja. Tokom merenja SSTable datoteke ostale su u kešu stranica operativnog sistema (page cache), pa je operativni sistem mogao da opsluži te pozive iz memorije. Izmerene latencije zato opisuju ponašanje sistema u uslovima zagrejanog keša stranica. Fizički I/O nije meren, pa se iz ovih rezultata ne može zaključiti koliko je fizičkih čitanja sa SSD/HDD uređaja izvršeno niti koliko su trajala. Za procenu broja i trajanja fizičkih čitanja bio bi potreban poseban eksperiment sa kontrolisanim stanjem keša stranica i praćenjem I/O brojača operativnog sistema na nivou uređaja.

Rezultati za veličinu bloka i proređenost Summary strukture ukazuju na postojanje kompromisa između memorijskog otiska pomoćnih struktura i troška njihove pretrage. Gušće uzorkovanje omogućava preciznije usmeravanje prema delu *SSTable* strukture, ali zahteva više prostora za indeksne informacije, dok ređe uzorkovanje smanjuje broj pomoćnih zapisa uz potencijalno veći interval koji je potrebno pretražiti. Ovakav kompromis je u skladu sa istraživanjima koja pokazuju da se lookup trošak u *LSM* sistemima ne može posmatrati nezavisno od raspoložive memorije i organizacije navigacionih struktura [6]. Činjenica da je među ispitanim konfiguracijama odnos 1 : 16 ostvario nižu latenciju od odnosa 1 : 64 zato predstavlja empirijski rezultat konkretne postavke, a ne dokaz da je 1 : 16 optimalna vrednost za druge sisteme ili radna opterećenja.

Ovaj rezultat je posebno značajan zbog toga što proređivanje indeksa ne utiče samo na veličinu same indeksne strukture. Ako pomoćne strukture nisu u memoriji, njihov pristup može postati dodatni deo troška *point lookup*-a. Istraživanje *LSM-trie* ukazuje na značaj veličine metadata i indeksnih struktura i pokazuje da njihovo izmeštanje iz memorije može dovesti do značajnog pogoršanja performansi čitanja [10]. U tom smislu, rezultat

dobijen u ovom radu može se tumačiti kao posledica ravnoteže između manjeg memorijskog otiska proređenog *Summary*-a i dodatnog rada potrebnog za lociranje odgovarajućeg dela podataka.

Istovremeno, uticaj keširanja ne treba posmatrati potpuno odvojeno od organizacije *LSM* stabla. Kompakcija određuje raspored *SSTable* datoteka po nivoima i utiče na broj i organizaciju struktura koje mogu biti relevantne tokom *point lookup*-a. Sarkar i saradnici pokazuju da izbor politike kompakcije fundamentalno utiče, između ostalog, na *point lookup* performanse i *read amplification* [2]. U eksperimentima ovog rada kompakcija nije izolovana kao nezavisni faktor, pa se deo posmatranog ponašanja read path-a ne može pripisati isključivo kešu ili *Summary* strukturi. Ovo predstavlja važan razlog za oprez pri generalizaciji rezultata.

Razlika između prosečne i p99 latencije dodatno pokazuje da poboljšanje prosečnog vremena odgovora ne znači nužno proporcionalno smanjenje najsporijih zahteva. Uvođenje L1 keša smanjilo je p99 latenciju u manjoj meri nego prosečnu latenciju, što ukazuje da deo dugih zahteva ostaje prisutan i kada se povećava lokalnost kroz aplikativni keš. Takvi zahtevi mogu zavisiti od promašaja u više nivoa strukture ili od pristupa blokovima koji nisu prethodno keširani. Zbog toga je za potpunu procenu read performance-a korisnije posmatrati prosečnu latenciju zajedno sa repom distribucije, a ne samo jednu agregatnu metriku.

Poređenje sa prethodnim istraživanjima treba tumačiti kvalitativno. Dayan i saradnici pokazuju značaj zajedničkog razmatranja memorijskog budžeta i troška navigacije kroz *LSM* strukturu, dok AC-Key pokazuje da različite vrste keširanih objekata imaju različite troškove i koristi [6,7]. Dobijeni rezultati podržavaju isti širi zaključak: performanse *point lookup*-a zavise od međusobnog odnosa keširanja i organizacije pomoćnih struktura. Međutim, apsolutne vrednosti latencije i propusnosti nije opravdano direktno porediti sa rezultatima tih istraživanja, jer se razlikuju implementacija, hardversko okruženje, veličina podataka, konfiguracija memorije i karakteristike radnog opterećenja.

Na osnovu postojećih podataka može se identifikovati najbolja konfiguracija za konkretnu testnu implementaciju, ali ne i dokazati univerzalno optimalna vrednost pojedinačnog parametra. Za preciznije razdvajanje doprinosa keša, veličine bloka i indeksnih struktura bilo bi potrebno sprovesti ponovljena merenja uz kontrolisane veličine keša i detaljno pratiti percentile latencije, memorijski otisak pomoćnih struktura i ponašanje za različite raspodele popularnosti ključeva.

ZAKLJUČAK

Rezultati pokazuju da keširanje značajno utiče na performanse tačkastih *GET* operacija u evaluiranom *LSM Key-Value* sistemu. L2 blokovski keš smanjio je prosečnu latenciju sa 1.048,2 na 214,7 μs, dok je kombinacija L1 i L2 keša dodatno smanjila latenciju na 178,4 μs. U mešovitom opterećenju dodavanje L1 keša smanjilo je prosečnu latenciju za 17,2% i povećalo propusnost za 20,8%. Među ispitanim konfiguracijama, najbolji rezultat ostvaren je sa blokovima od 4 KB i odnosom uzorkovanja *Summary* strukture 1 : 16. Rezultati ukazuju da zajedničko podešavanje keširanja i pomoćnih indeksnih struktura može značajno uticati na performanse čitanja u *LSM* sistemima, dok su za širu generalizaciju nalaza potrebna ponovljena merenja i potpunija evaluacija različitih konfiguracija.

LITERATURA

[1] O'Neil PE, Cheng E, Gawlick D, O'Neil EJ. The log-structured merge-tree (LSM-tree). *Acta Informatica*. 1996;33(4):351-385. doi:10.1007/s002360050048.

[2] Sarkar S, Staratzis D, Zhu Z, Athanassoulis M. Constructing and analyzing the LSM compaction design space. *Proc VLDB Endow*. 2021;14(11):2216-2229. doi:10.14778/3476249.3476274.

[3] Google. LevelDB implementation notes. https://github.com/google/leveldb/blob/main/doc/impl.md (pristupljeno 25. 9. 2026.).

[4] Google. LevelDB file format. https://github.com/google/leveldb/blob/main/doc/table_format.md (pristupljeno 25. 9. 2026.).

[5] Bloom BH. Space/time trade-offs in hash coding with allowable errors. *Commun ACM*. 1970;13(7):422–426. doi:10.1145/362686.362692.

[6] Dayan N, Athanassoulis M, Idreos S. Monkey: Optimal Navigable Key-Value Store. In: *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD)*. New York: ACM; 2017. p.7994. doi:10.1145/3035918.3064054.

[7] Wu F, Yang MH, Zhang B, Du DHC. AC-Key: Adaptive caching for LSM-based Key-Value Stores. In: *Proceedings of the 2020 USENIX Annual Technical Conference*. Berkeley: USENIX Association; 2020. p.603–615.

[8] Yabandeh, M. Partitioned Index/Filters. RocksDB Blog 2017 https://rocksdb.org/blog/2017/05/12/partitioned-index-filter.html (pristupljeno 25. 9. 2026.).

[9] Dayan N, Idreos S. Dostoevsky: Better space-time trade-offs for LSM-tree based key-value stores via adaptive removal of superfluous merging. In: *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*. New York: ACM; 2018. p.505–520. doi:10.1145/3183713.3196927.

[10] Wu X, Xu Y, Shao Z, Jiang S. LSM-trie: An LSM-tree-based ultra-large key-value store for small data items. In: *Proceedings of the 2015 USENIX Annual Technical Conference*. Berkeley: USENIX Association; 2015. p.71–82.

ANALYSIS OF THE EFFECTS OF TWO-LEVEL CACHING AND SPARSE INDEXES ON READ LATENCY IN LSM DATA STORES

Author: ALEKSA DEJANOVIĆ, Petar Popović, Boško Vasilić, Sara Stojkov
e-mail: aleksa@dejanovic.online
Mentor: Assist. Prof. Miloš Simić
Department of Computing and Control Engineering
Faculty of Technical Sciences, University of Novi Sad

Introduction: Log-Structured Merge-Tree (LSM) stores provide high write throughput, but multi-level searches and read amplification can increase read latency.

Aim: To experimentally evaluate the effects of two-level caching, an application LRU cache and a disk block cache, and Summary/Index sparsity on point GET performance in a custom LSM key-value system.

Material and Methods: A key-value system with two-level caching and configurable Summary/Index structures was implemented. We evaluated 10^6 reads with Zipfian and uniform distributions over existing and non-existing keys, using 4, 8, and 16 KB blocks and Summary/Index sampling ratios from 1:4 to 1:64.

Results: Best overall performance was achieved with 4 KB blocks, a 1:16 Summary sampling ratio, and both L1 and L2 caches. Compared with no caching, L2 reduced mean latency from 1,048.2 to 214.7 μs (79.5%), while L1+L2 reduced it to 178.4 μs (83.0%). Under 10^6 mixed requests, L1+L2 reduced mean GET latency from 220.30 to 182.36 μs (17.2%) and increased throughput from 4,539 to 5,484 GET/s (20.8%) versus L2 alone. L1 achieved a 37.5% hit rate and reduced L2 accesses by 16.5%.

Conclusion: Two-level caching substantially improves read performance, while block size and index sparsity balance latency, throughput, and memory use. The best configuration used 4 KB blocks, a 1:16 sampling ratio, and both L1 and L2 caches.

Keywords: LSM tree ; key-value store ; block cache ; LRU cache ; read latency