Optimiranje delovanja zbirke podatkov časovne vrste

Igor Mernik, Franc Klauzner

Informatika, Informacijske storitve in inženiring, d.o.o., Maribor, Slovenija
igor.mernik@informatnika.si, franc.klauzner@informatika.si

Podatkovna zbirka je eden izmed kamenčkov v mozaiku celotne rešitve obračuna omrežnine, ki smo jo uvedli z rešitvijo podatkovne zbirke časovnih vrst. Ko smo začeli s projektom, smo pričakovali, da bo delo z velikimi količinami podatkov zahtevalo več dela kot je običajno z operativno-transakcijskimi podatkovnimi zbirkami. Predvidevali smo, da sistem ne bo dovolj odziven vsakič, ko bomo v aplikaciji napisali poljuben SQL stavek in ga izvedli. Ugotovili smo, da je delo z velikimi količinami podatkov dokaj drugačno. Da je delovanje podatkovne zbirke optimalno, je potrebno ustrezno nasloviti vsak delček procesa. Za vsak napisan SQL in vsak proces je potrebno razumeti, kako se bo odrazil na odzivnosti podatkovne zbirke. Ker se posamezne operacije izvajajo nekaj milijon-krat, vsak drobec neoptimalne izvedbe lahko pripelje do večurnega podaljšanja obdelav ali pa celo do nikoli končanih obdelav, če so le-te napisane tako, da zavzamejo preveč računalniških virov.

V članku opisujemo, katere programske rešitve smo uporabili pri optimiranju podatkovne zbirke časovnih vrst in zakaj smo se odločili za takšno rešitev. V jedru članka je opisan postopek optimizacij podatkovne zbirke, najprej na ravni arhitekture rešitve, nato kako smo prilagodili aplikacijsko okolje in nastavitve na ravni podatkovne zbirke same ter uvedba nadzora nad optimalnim delovanjem podatkovne zbirke.

Ključne besede:
velepodatki
optimiranje podatkovne zbirke
podatkovna zbirka časovnih vrst
TimescaleDB
obračun omrežnine

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.8

1 Uvod

Dne 25. 11. 2022 je Agencija za energijo sprejela Akt o metodologiji za obračunavanje omrežnine za elektro operaterje [1]. Skladno s tem aktom bo stopil v veljavo nov tarifni sistem za obračunavanje omrežnine. Posledica tega je precej spremenjen sistem obračunavanja omrežnine za električno energijo.

Glavne značilnosti novega tarifnega sistema so:

- obračun, ki temelji na 15-minutnih meritvah,
- uvedba dveh sezon, višje med novembrom in februarjem ter nižje med marcem in oktobrom,
- uvedba pet časovnih blokov,
- razločevanje med dogovorjeno in presežno obračunsko močjo.

Že v preteklosti smo v podjetju Informatika d.o.o. s svojimi informacijskimi rešitvami obvladovali to področje obračunavanja omrežnine za električno energijo. Kljub temu je bil izziv vse obstoječe rešitve prilagoditi zahtevam novega akta. Zaradi lažjega obvladovanja tega izziva smo potrebne prilagoditve razdelili v tri sklope in sicer:

- Platforma za obdelavo merilnih podatkov (POMP) - vzpostavitev rešitve za shranjevanje, validacijo in nadomeščanje merilnih podatkov.
- Obračun omrežnine – implementacija vseh algoritmov novega obračuna za različne vrste merilnih mest.
- Nadgradnja Enotne vstopne točke – EVT (prilagoditev portala Moj Elektro, portala CEEPS in B2B storitev) skratka prenove vse B2B in B2C komunikacije.

Rešitev POMP na najnižji ravni predstavljajo merilni centri elektrodistribucijskih družb (Elektro Celje, Elektro Gorenjska, Elektro Ljubljana, Elektro Maribor in Elektro Primorska), ki pridobivajo merilne podatke iz merilnih naprav odjemalcev preko komunikacije po energetskih linijah (PLC – Power-line communication) ali s protokoli P2P (Peer-to-Peer). Merilni podatki se nato po mehanizmu potiskanja (»push«) v rezinah prenesejo v centralno podatkovno zbirko PostgreSQL s TimescaleDB, ki je vzpostavljena na infrastrukturi Informatike.

V tokratnem prispevku se osredotočamo na POMP in sicer vam želimo predstaviti naše tehnične izzive ob optimizaciji te platforme na ravni podatkovne zbirke.

V okviru te rešitve tečejo kompleksne obdelave sprotne validacije in nadomeščanja podatkov, izračuna obračunskih količin, kot tudi izračuna dogovorjenih moči, ki potekajo nad podatki večjega, v določenih primerih celo celoletnega obdobja. Ob tem pa morajo podatki biti ves čas na voljo tudi zunanjim deležnikom. Sistem mora biti pripravljen tudi za naprednejšo analitiko. Vse skupaj je bil velik izziv, reševanje katerega skušamo predstaviti v okviru prispevka.

2 Arhitektura rešitve

2.1. Arhitekturna slika procesov obdelav podatkov v podatkovni zbirki

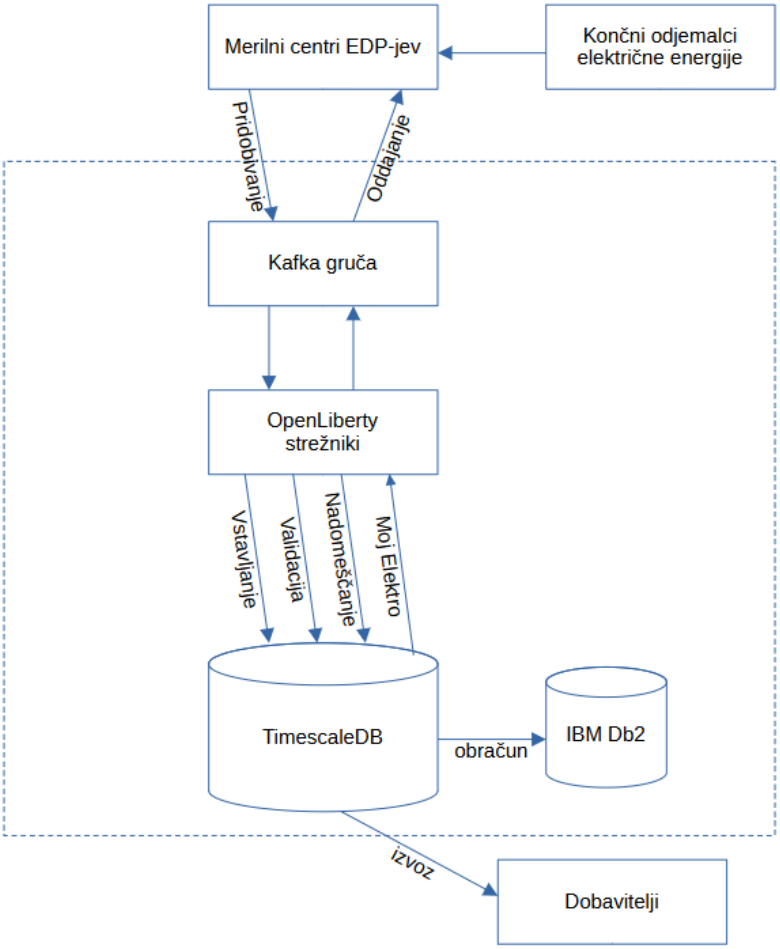
Podatke 15 minutnih odčitavanj količin porabe električne energije merilnih mest končnih odjemalcev električne energije pridobivamo od elektro-distribucijskih podjetij. Izmenjava podatkov je obojestranska.

Podatki se s pomočjo Kafkine gruče strežnikov in OpenLiberty aplikacijskih strežnikov prepisujejo v podatkovno zbirko. Ta postopek imenujemo »Vstavljanje« podatkov oziroma tudi nastanek podatkov. Pri tem postopku gre za množičen vnos velepodatkov.

Nad vnesenimi podatki se izvede postopek »Validacije« podatkov, s katerim se preveri konsistentnost in kakovost pridobljenih podatkov. Temu postopku sledi postopek »Nadomeščanja«, ki po dogovorjenem algoritmu polni manjkajoče vrzeli in počisti morebitne anomalije v podatkih.

Obdelani podatki so na voljo preko spletne in mobilne aplikacije Moj Elektro. Podatke pa izvažamo tudi za potrebe dobaviteljev električne energije.

Kot glavni ponor pa je priprava agregatnih podatkov za obračun električne energije. Ta postopek obračunavanja izvajamo že vrsto desetletij na IBM Db2 podatkovni zbirki. Na TimescaleDB podatkovni zbirki pripravimo agregatne podatke, ki jih potisnemo na IBM Db2 podatkovno zbirko, kjer se izvedejo obdelave obračuna.



Slika 1: Arhitektura obdelave podatkov na podatkovni zbirki časovnih vrst.

2.2. Uporaba podatkovne zbirke

Za poslovne procese uporabljamo dvoje programja in sicer IBM Db2 podatkovno zbirko in Microsoft SQL Server. Za obe vrsti podatkovnih zbirk imamo nekaj deset programskih primerkov podatkovnih zbirk in v vsakem primerku po nekaj podatkovnih zbirk. Ko smo proučevali, katero podatkovno zbirko uporabiti za velepodatke, smo ugotovili, da nas bodo kupljene lastniške podatkovne zbirke licenčno preveč obremenile in hkrati nas precej ovirale pri določanju potrebnih računalniških virov. Odločili smo se za podatkovno zbirko PostgreSQL z razširitvijo TimescaleDB. Slednja je licenčno omejevana le, če bi zunanjemu partnerju ponujali rešitev podatkovne zbirke kot storitev. Ker pa partnerjem ponujamo storitev preko aplikacijskih klicev, smo skladni z licenco in tako popolnoma prilagodljivi glede potreb po razširitvi infrastrukture.

TimescaleDB je razširitev PostgreSQL podatkovne zbirke. Prednost tovrstne rešitve pred posebnimi za velepodatke namenjenimi podatkovnimi zbirkami, je robustnost PostgreSQL podatkovne zbirke in uporaba SQL jezika za obdelavo podatkov. Slednje je zlasti pomembno, da lahko razvijalci aplikacij uporabijo obstoječe znanje, ki ga že imajo pri uporabi dosedanjih rešitev. Za skrbnike podatkovnih zbirk pa je pomembno, da se uporablja preizkušena podatkovna zbirka PostgreSQL.

TimescaleDB je posebej prilagojena za delo z velepodatki. Ima številne prilagoditve, na primer lahko uporabljamo standardni SQL, lahko pa uporabimo tudi funkcije podatkovne zbirke, ki pohitrijo izvajanje SQL-a. S stališča skrbništva podatkovne zbirke je upravljanje poenostavljeno, saj sama podatkovna zbirka skrbi za particioniranje tabel, stiskanje podatkov v tabelah in pa za samodejno odstranitev podatkov, ko le-ti niso več potrebni. Podatkovna zbirka omogoča številne prilagoditve in optimiranja, kar je opisano v nadaljevanju.

2.3. Nadzor nad zanesljivim delovanjem podatkovne zbirke

Za nadzor nad podatkovno zbirko smo uporabili že obstoječo infrastrukturo za spremljanje strežnikov in sicer nadzorni sistem Zabbix [2]. Izdelek ima vtičnik za PostgreSQL podatkovno zbirko in že privzeto spremlja na stotine parametrov podatkovne zbirke. Izdelali smo preglede - grafikone, iz katerih prepoznamo morebitne težave in tako lažje odreagiramo. Izdelali smo preglede za nadzor porabe procesorske moči in pomnilnika ter časa vhodno izhodnimi operacij, iz česar lahko ugotovimo ali je težave morda z vhodno izhodnimi operacijami diska. Naredili smo pregled učinkovitosti zapisovanja transakcijskih logov, iz katerih ugotovimo nepričakovane povečane aktivnosti podatkovne zbirke. Prikazujemo učinkovitost čiščenja podatkov z »autovacuum« procesi, kar je pomembno, da podatkovna zbirka ne prenaša v pomnilnik nepotrebnih podatkov. Izdelali smo pregled, s katerim prikazujemo število uspešnih in neuspešnih zaključitev transakcij. Naredili smo tudi svoje lastne preglede zaklepanja objektov podatkovne zbirke, s katerimi prepoznamo, če aplikacija nepravilno sprošča zaklepanja objektov. Imamo še številne preglede, ki kažejo učinkovitost notranjih procesov podatkovne zbirke, iz katerih zaznamo potrebo po spremembi parametrov podatkovne zbirke.

Z IBM Instana produktom sledimo aplikacijskim klicem do podatkovnih zbirk. Pri tem proučimo medsebojno odvisnost aplikacijskih klicev in zaporedje izvedbo le-teh ter vpliv posameznih klicev na poizvedbe podatkov iz podatkovnih zbirk. Iz grafikonov razberemo razmerje trajanj posameznih klicev in ugotovimo, katerim klicem posvetimo pozornost za izboljšavo odzivnosti aplikacije in pripadajočih SQL stavkov.

Kot najučinkovitejše sredstvo za prepoznavo težav, ki se odrazijo na podatkovni zbirki, pa je spremljanje izvedenih SQL-ov na podatkovni zbirki. V ta namen uporabljamo PostgreSQL razširitev pg_stat_statements, s katerim prepoznamo SQL-e, ki zasedajo največ računalniških virov, ki so kandidati za optimiranje.

3 Optimizacijski postopki – arhitekturni vidik

3.1. Poznavanje arhitekture procesa polnjenja podatkovne zbirke

Na podatkovni zbirki se izvajajo številni procesi in poznavanje le-teh je nujno za učinkovito izvajanje optimizacij. Pri nas imamo tri velike sklope procesov:
a) inicialno nastajanje podatkov iz merilnih mest,
b) validacija z nadomeščanjem manjkajočih in napačnih podatkov ter
c) zapis iz podatkovne zbirke časovne vrste v podatkovno zbirko operativnega-transakcijskega sistema.

Pomembno je razumevanje arhitekture procesa nastajanja in obdelave podatkov v podatkovni zbirki. Le-to namreč odločilno vpliva na odzivnost ekipe, da je sposobna hitro določiti izvor težave in predlagati možne rešitve. Na strani aplikacijskega strežnika smo ustrezno poimenovali aplikacije, ki s prijavo na podatkovno zbirko le-tej pošljejo

podatek o imenu aplikacije. Če nastanejo težave, lahko iz imena aplikacije zelo hitro omejimo izvor težave in iskanje samo znotraj točno določenega področja.

3.2. Obremenitev podatkovne zbirke

Tudi če imamo zelo zmogljiv računalnik, ne moremo s preprosto akcijo pospešiti odzivnost podatkovne zbirke. Hitrost oziroma odzivnost podatkovne zbirke je omejena z drugimi dejavniki, kot so hitrost shranjevanja, omrežna pasovna širina ali zasnova same podatkovne zbirke. Če se pričakuje, da se bo odzivnost podatkovne zbirke povečala za n-krat, potem je potreba temeljita analiza vsega kar se dogaja v podatkovni zbirki. Dodajanje več virov ne bo pospešilo dela, če je osnovni proces počasen. Delati moramo drugače, precej drugače. Samo povečevanje računalniških virov ni rešitev. Kvečjemu obratno, zmanjšaš vire, da lažje vidiš, kaj pomoli glavo iz povprečja.

3.3. Politika odstranitve podatkov

Ker moramo, kot je določeno z zakonom, poskrbeti za uničenje podatkov [3], je bilo potrebno dobro razmisliti, kako se bodo podatki ustrezno odstranili iz podatkovne zbirke. Sprva smo načrtovali, da bi ročno odstranjevali podatke po mesecih, ki jih več ne potrebujemo, vendar sedaj se nagibamo k uvedbi politike samodejnega odstranjevanja podatkov. Samega odstranjevanja podatkov še nismo rešili in sicer iz razloga, da za enkrat še nismo prenašali podatkov za obdobje, za katero bi jih že morali samodejno odstranjevati. Imamo pa že jasno nastavljene okvirje, kako bomo to izvedli, da bomo zakonsko skladni.

4 Optimizacijski postopki – podatkovni vidik

4.1. Spremljanje SQL-ov

Ko pride do težav ali bolje preden pride do njih, je potrebno razumeti, kaj povzroča obremenitev podatkovne zbirke. Podatkovna zbirka sama po sebi ne povzroča obremenitve sistema, obremenitev povzročajo SQL stavki. Pri tem ni pomembno, kateri posamezni SQL-i se najdlje izvajajo, ampak je potrebno gledati agregatno, kateri SQL-i skupaj zasedajo največ računalniških virov [4]. Čeprav bi se na prvi pogled zdelo, da je dolgotrajna petminutna izvedba SQL poizvedbe problematična, lahko predstavlja še večjo težavo poizvedba, ki se sicer izvede hitro v 50 milisekundah, vendar se ponovi milijardo krat. Za spremljanje obremenitve smo uporabili PostgreSQL razšitiev pg_stat_statement [5], s katerim smo pridobili SQL-e, ki so agregatno zasedli največ računalniških virov. Nato smo se osredotočili le na te SQL-e in jih preverili, v čem je njihova posebnost, da zasedajo računalniške vire.

4.2. Kompleksni SQL-i

Če je za operativno-transakcijske SQLe še nekako sprejemljivo, da so SQL-i kompleksni, za podatkovne zbirke časovnih vrst, le-ti ne pridejo v poštev. Če imamo na primer »update« stavek, ki traja pet ur, da se konča in le-tega prekineš po treh urah, si izgubil tri ure časa. Veliko bolje je napisati milijon majhnih »update« stavkov, ki vsak spremeni majhen del podatkov in če po treh urah prekineš proces in je le-ta naredil na primer 40% dela, lahko nato nadaljuješ z naslednjimi obdelavami tam, kjer si ostal in tako imaš le še 60% obdelav za dokončati. Z namenom izboljšanja smo izvedli temeljito preoblikovanje kompleksnih SQL poizvedb v nabor manjših bolj preglednih in lažje vzdrževanih poizvedb. Že nekaj desetletij znano pravilo, da mora biti SQL enostaven in razumljiv, pri podatkovnih zbirkah časovnih vrst pride še toliko bolj do izraza. Namreč izvedba kompleksnega SQL-a preko 100-milijard zapisov bo zanesljivo povzročilo izjemno obremenjenost računalniških virov in verjetno bomo prisiljeni celo v prekinitev obdelave.

4.3. Optimiranje podatkovnega modela

Ko smo imeli seznam najbolj računalniško intenzivnih SQL-ov, smo opazili, da kompleksni SQL-i izvajajo funkcije podatkovne zbirke in izvajajo transformacije podatkov. Rešitev je v prilagoditvi podatkovnega modela. Tako smo spremenili šifrante in jih napolnili s »stored procedurami«, ki smo jih samo enkrat izvedli in zapisali podatke v spremenjen podatkovni model. Tako smo SQL-e, ki se izvedejo več milijon-krat, precej poenostavili in večkratno pohitrili izvajanje celotne obdelave.

4.4. Analiza dostopne poti SQL-ov

Že za preprost SQL lahko napačno oceniš, kako hitro se bo izvajal. Podatkovne zbirke imajo orodja za analizo dostopnih poti posameznih SQL ukazov. Uporabili smo orodje EXPLAIN [6],[7], ki zelo natančno pokaže, kakšne izvajalne operacije načrtuje podatkovna zbirka in kakšne operacije je podatkovna zbirka dejansko izvedla. Iz analize dostopnih poti smo naredili sklepe, kaj je potrebno na podatkovni zbirki spremeniti. Če je smo zaznali, da lahko SQL bolje napišemo, smo ga optimirali. Če se na tabeli izvajajo številna branja podatkov, smo ustvarili manjkajoči indeks in podobno. Torej z EXPLAIN smo pridobili ključne podatke o tem, kje v izvajanju SQL-a so težave.

5 Optimizacijski postopki – aplikacijski vidik

5.1. Sprememba podatkov

Zaradi izvajanja sprememb podatkov, kar odstopa od običajnih praks v okviru časovnih vrst, nismo uspeli najti ustreznih spletnih virov za uspešno izvedbo procesa. Zato smo bili primorani izvesti testiranja delovanja podatkovne zbirke, da bi prepoznali ustrezne, neoptimalne in neprimerne funkcionalnosti. Proces spreminjanja podatkov smo začeli tako, da smo najprej pobrisali podatke, ki smo jih morali spremeniti in jih ponovno vstavili v podatkovno zbirko. Ta metoda je dokaj hitra, dokler gre za manjše spremembe nad podatki. Ko smo ugotovili, da za določeni procesi zahtevajo večje spremembe, smo testirali t.i. »upsert« stavek [8], ki v enem samem SQL ukazu naredi vstavljanje podatka v podatkovno zbirko, če pa podatek v podatkovni zbirki že obstaja, pa se izvede sprememba podatka.

5.2. Množično popravljanje podatkov

Med procesom optimizacije, smo na testnem okolju izvedli t.i. množično spremembo podatkov (ang. bulk-update) [9], ki pa se je na produkcijskem okolju izkazala kot prepočasna rešitev, ker podatkovna zbirka časovnih vrst žal nima ali pa še nima, rešeno ustrezno množično popravljanje. S sodelovanjem z razvijalci podatkovne zbirke smo ugotovili, da želena funkcionalnost ne dosega pričakovane odzivnosti. Zato smo razvili aplikacijsko rešitev, kjer smo uporabili t.i. »upsert«, ki z enim ukazom izvede vstavljanje podatkov v podatkovno zbirko, če podatki v tabeli podatkovne zbirke še ne obstajajo po podatkovnem ključu, ali spremeni podatke, če le-ti že obstajajo v podatkovni zbirki, ter programsko zagotovili, da se vedno izvede popravljanje podatkov.

5.3. Bazen povezav do podatkovne zbirke

Do podatkovne zbirke lahko vzpostavimo povezavo v aplikaciji namensko za izvajanje posameznega SQL-a ali skupine SQL-ov. To je pri večkratnem ponavljanju neoptimalno, ker je prijavljanje na podatkovno zbirko časovno potratna operacija, saj mora podatkovna zbirka najprej vzpostaviti mrežno povezavo, nato izvesti avtentikacijski postopek, avtorizacijski postopek in šele nato sledi izvedba SQL-a. Da se izognemo ponavljanju vseh teh postopkov pri prijavi na podatkovno zbirko, s pomočjo aplikacijskega strežnika vzpostavimo t.i. bazen povezav do podatkovne zbirke. Nato aplikacija za dostop do podatkovne zbirke uporabi že vzpostavljeno povezavo. Pri

več milijonih izvedenih operacijah prijavljanja na podatkovno zbirko, se s tem ustvari precejšen časovni prihranek. Natančno smo preverili vse aplikacije, da uporabljajo bazen prijav do podatkovne zbirke.

5.4. Pobegle povezave podatkovne zbirke

Zaradi množice izvajajočih hkratnih vzporednih procesov in hitrega razvoja aplikacij, se lahko zgodi, da nastane kakšna napaka. Razvili smo metodo, s katero zaznamo pobegle povezave v podatkovni zbirki, ki lahko pripeljejo do odzivnostnih težav [10] ali pa celo do neodzivnosti podatkovne zbirke zaradi pretiranega zaklepanja. Za preprečevanje prenosa težav v produkcijsko okolje smo najprej vzpostavili sistem na razvojnem okolju, kar omogoča zgodnje odkrivanje in odpravljanje napak v času razvoja aplikacij. Sistem smo vzpostavili tudi na produkcijskem okolju, s katerim zagotavljamo nemoteno delovanje aplikacij.

5.5. Število povezav do podatkovne zbirke

Ker imamo številne aplikacijske strežnike in na vsakem strežniku vzpostavljen bazen povezav do podatkovne zbirke, je pomembno, da se najprej preveri morebitne pobegle povezave do podatkovne zbirke. Nato smo optimirali število povezav do podatkovne zbirke v posameznem bazenu povezav [11]. Opazili smo, da smo na začetku določili samo grobo oceno, koliko povezav do podatkovne zbirke bomo potrebovali, šele nato izkušnje pokažejo, da je v določenih bazenih povezav določenih preveč povezav, v naslednjih pa premalo. Optimirali smo število povezav v podatkovni zbirki. Ta postopek smo izvedli preden smo povečali število povezav na podatkovni zbirki. Povečanje povezav do podatkovne zbirke pomeni sprememba pomnilniških parametrov na podatkovni zbirki, saj vsaka povezava do podatkovne zbirke zahteva določen del lastnega pomnilnika, kjer se izvaja na primer razvrščanje podatkov in podobno.

5.6. Zaklepanja na podatkovni zbirki

Pri večjih spremembah podatkov v podatkovni zbirki se lahko hitro zgodi, da prihaja do zaklepanja objektov podatkovne zbirke [12]. To se hitreje zgodi, če se posamezni SQL-i izvajajo več kot nekaj milisekund. Dlje traja posamezen SQL, večja je verjetnost, da bo prišlo do medsebojnega zaklepanja objektov na podatkovni zbirki. Razvili smo mehanizem za zaznavo neustreznega zaklepanja. Hkrati smo v osrednji sistem nadzora izvedli integracijo, tako da lahko na grafu hitro zaznamo težave z zaklepanjem. Če se dovolj hitro ne ugotoviti, da je prišlo do neustreznega zaklepanja, se lahko na podatkovni zbirki pojavi zaklenjenih 50-tisoč in več objektov in je analiza vzroka zaklepanj precej zahtevnejša.

5.7. Vzporednost izvajanja SQL ukazov in vzporednost izvajanje zapisov

Ko zapisujemo vhodne podatke v podatkovno zbirko časovne vrste, smo ugotovili, da je izjemnega pomena, kako se izvaja zapisovanje podatkov v podatkovno zbirko [13]. Z merjenjem smo ugotovili, da v posameznem SQL stavku za vstavljanje podatkov lahko izvedemo do 3000 zapisov, pri popravljanju podatkov pa je optimalno število nekje tisoč zapisov v posameznem SQL-u. Hkrati vzporedno izvajamo procese iz več aplikacijskih strežnikov in sicer hkrati smo določili kot optimalno 30 hkratnih opravil, ki v podatkovno zbirko zapisujejo podatke. Optimalna zagotovitev vzporednosti zapisovanja je pri podatkovnih zbirkah časovne vrste velikega pomena, saj lahko za mnogokratnik skrajšamo čas izvedbe posameznega opravila podatkovne zbirke.

5.8. Uporaba parametrov v SQL-ih

Obstajata dva načina pisanja dinamičnih SQL-ov. Prvi način je enostavnejši in pri njem vpišemo celotni SQL in ga izvršimo nad podatkovno zbirko. Drugi način je uporaba parametrov [14] in sicer najprej napišemo SQL in v »where« pogojih navedemo parametre, ki jih kasneje v programu izpolnimo. Pri prvem načinu mora podatkovna zbirka za vsak SQL narediti dostopno pot do podatkov in če izvedemo nekaj milijonov SQL-ov, s tem povzročimo časovno zakasnitev in pa nepotrebno obremenitev procesorske moči. Drugi način pomeni, da podatkovna zbirka ustvari dostopno pot do podatkov samo enkrat in nato več milijon-krat uporabi isto dostopno pot. Preverili smo aplikacije, da smo se prepričali, da je povsod uporabljen način dostopa do podatkov s parametri.

5.9. Zakasnitev potrditve transakcije

Ugotovili smo, da se pri množičnem nastajanju podatkov, izvede tudi po deset-tisoč transakcij na sekundo. Vsakokrat, ko izvedemo potrditev transakcije, mora podatkovna zbirka zagotovi konsistentnost le-te in v transakcijski log prepisati podatke, s katerimi zagotavlja, da se v primeru sesutja programja podatkovne zbirke iz različnih vzrokov, lahko ob ponovnem zagonu podatkovne zbirke zagotovi konsistentnost podatkov v podatkovni zbirki. Pri zahtevku za potrditev transakcije pa je podatkovna zbirka prisiljena izvesti časovno potratno operacijo, s katero mora ne samo od operacijskega sistema ampak spodaj ležečega diskovja dobiti 100% zagotovilo, da je bila transakcija zapisana na disk. Ena izmed možnosti je, da nastavimo podatkovno zbirko tako, da le-tega ne zahteva, vendar tvegamo nekonsistentnost podatkov. Kljub temu da lahko v našem okolju del podatkov podatkovne zbirke obnovimo iz izvornih virov, je ključnega pomena zagotavljanje časovnega okna za morebitno obnovo podatkov in razpoložljivost sodelavcev, da to opravilo izvedejo. Druga možnost za pohitritev izvedbe transakcij je potrjevanje transakcij v skupini in ne posamično. To dosežemo z zakasnitvijo posameznih transakcij za delček sekunde [15]. V tem primeru podatkovna zbirka ne zapiše vsake transakcije posebej in čaka na odziv diska, ampak v skupini potrdi transakcije in podatke hkrati zapiše na disk. Pri tem žrtvujemo čas potrditve posamezne transakcije, s ciljem pohitriti skupino transakcij. Nastavili smo zakasnitev transakcije za delček sekunde.

6 Optimizacijski postopki – vidik podatkovne zbirke

6.1. Particioniranje

Največja tabela v našem okolju je velika 80-milijard zapisov, kar predstavlja podatke vseh merilnih mest v Sloveniji za obdobje dveh let in pol. Vsak mesec v podatkovni zbirki nastane dodatne tri milijarde zapisov. Vsi ti podatki ne morejo biti v eni sami veliki tabeli, ker bi SQL-i delovali občutno prepočasi. Hkrati se na tabelo ustvari indeks za pohitritev iskanja podatkov in tudi indeks bi bil prevelik, da bi deloval optimalno. Na podatkovni zbirki lahko uporabimo »partitioning« [16]. To je proces, kjer logično tabelo razdelimo na več fizičnih tabel. S stališča aplikacije obstaja ena tabela, fizično pa jih je lahko več tisoč. Pomembno je, da pravilno nastavimo »particioniranje« tabele, ki se pri podatkih časovne vrste praviloma nastavi na časovno značko nastanka dogodka.

Potrebno je sprejeti odločitev, da posamezna particionirana tabela ni kljub temu prevelika. Glede na izračun priporočene velikosti particionirane tabele, smo se odločili, da nastavimo razmejitev tako, da so za vsak dan podatki v svoji particijski tabeli.

Dodatno obstaja možnost, da se določi particioniranje tabele še po dodatnem particioniranem ključu. V jeziku produkta to pomeni dimenzioniranje oz. dodajanje dimenzij. Zaenkrat ne uporabljamo dodatnih dimenzij, pa ne zato, ker ne bi bilo koristno, ampak predvsem zaradi tega, ker je dimenzioniranje dokaj nova funkcionalnost produkta in zahteva, da se dimenzioniranje nastavi na prazni tabeli. Zaradi izjemne velikosti logične tabele, bi prepis podatkov v novo particionirano okolje trajal nekaj dni in pri tem bi bila podatkovna zbirka toliko obremenjena, da

si poslovno tega ne moremo privoščiti. Imamo pa v načrtu particioniranje tabele tudi po dodatnih ključih, vendar moramo sedaj najti ustrezen čas, ko bo poslovno najbolj sprejemljivo izvesti zahtevno operacijo premika podatkov.

6.2. Razvrščanje podatkov v tabelah

Če je za transakcijsko-operativne podatkovne zbirke priporočljivo, da se izvaja redno skrbništvo z razvrščanjem podatkov v tabelah, pa je v podatkovnih zbirkah časovne vrste to že nujnost [17]. Od tega, kako so podatki fizično zloženi v tabelah, bo odvisno, koliko diskovnih dostopov bo podatkovna zbirka morala izvesti, da bo pridobila podatke. Če so podatki v tabeli neustrezno razvrščeni, bo podatkovna zbirka za dostop do podatkov potrebovala tudi mnogokratnik časa kot v primeru lepo razvrščenih podatkov. Napisali smo optimizacijske skripte, ki se dnevno izvajajo in sproti vsako noč razvrščajo podatke v ustrezno optimalno razvrstitev, tako da dnevne obdelave delujejo bistveno hitreje. Ker pa lahko pridobimo do osem dni stare podatke, mesečno še enkrat razvrščamo podatke.

6.3. Prazen prostor podatkovnih strani tabel

V teoriji podatkovnih zbirk časovnih vrst, podatki v podatkovnih zbirkah samo nastajajo. Senzor proizvaja podatke in jih zapisuje v podatkovno zbirko. Na primer imamo senzor temperature ozračja, ki meri temperaturo in jo pošilja podatkovni zbirki. Če kakšen podatek manjka ali je celo napačen, ni takšne velike tragedije, ker pač tak podatek nima velikega poslovnega vpliva. V našem okolju, pa se na osnovni podatkov merilnikov električne energije obračunava omrežnina. Zato je potrebno zagotoviti, da so podatki ustrezne kakovosti. To pa pomeni, da nad obstoječimi podatki izvajamo spremembe. Podatki v tabelah so fizično zapisani v tako imenovanih podatkovnih straneh. Ker vnaprej vemo, da bodo na tabeli sledile spremembe za zagotavljanje kakovosti podatkov, smo to predvideli in podatkovne strani tabele nastavili tako, da smo nastavili določen prostor, kjer bodo spremembe zapisane [18]. S tem znatno zmanjšamo razpršenost podatkov v podatkovnih straneh, kar zmanjša količino diskovnih dostopov in zagotavlja hitrejše izvajanje podatkovne zbirke.

6.4. Indeksiranje tabel

Iz arhitekture aplikacije se predvidi, kakšni SQL-i se bodo nad podatkovno zbirko izvajali. Na osnovi SQL predikatov, se na podatkovni zbirki določi indeks. Ustrezno indeksiranje na podatkovni zbirki je ključnega pomena za hitrost izvajanja SQL-ov. Pri samem procesu indeksiranja smo bili zelo pozorni, kako bomo posamezen indeks ustvarili, ker ustvarjanje indeksa na nekaj milijardah zapisov tabele, lahko traja več dni. Torej skrbna izbira indeksa [19] in šele nato izvedba. Če ne bi naredili najbolj optimalnega indeksa, potem kasnejša sprememba pri nas iz poslovnih razlogov ni več mogoča, ker bi morali za kakšen dan ustaviti produkcijsko okolje, da bi ustrezno spremenili indeksiranje.

6.5. Agregacija podatkov z materializiranimi pogledi

Materializirani pogledi v podatkovnih zbirkah omogočajo pohitritev SQL poizvedb tako, da v naprej napolnimo podatke v materializiran pogled. Le-tega lahko napolnimo s postopnim polnjenjem ali z enkratnim polnjenjem. V začetku projekta smo uporabljali materializirane poglede [20] in ta funkcionalnost je dobro delovala. Nato smo ugotovili, da smo zaradi narave procesa prisiljeni ustrezno spreminjati podatke in s tem bi morali neprestano na novo osveževati materializirane poglede. S tega vidika, smo se odločili, da je to preveliko breme in smo spremenili dizajn podatkovnega modela, da smo to odpravili. Ali uporabiti materializirane poglede, je predvsem odvisno od tega ali imamo pri posameznem projektu izključno nastajanje novih podatkov, kar je po definiciji pri nastajanju podatkov časovne vrste ali pa se morajo izvajati dokaj intenzivne spremembe nad podatki.

6.6. Stiskanje podatkov

Iz poslovnih zahtev smo razbrali, katere podatke moramo imeti žive in hitro delujoče in katere podatke moramo časovno hraniti določeno število let in katere podatke moramo obvezno po zakonu uničiti. Izvedli smo stiskanje [21] podatkov na podatkovni zbirki, kar je zmanjšalo velikost tabel za 90%. Povpraševanja nad stisnjenimi podatki lahko celo delujejo hitreje, kot nad običajnimi nestisnjenimi podatki. Ugotovili pa smo, da obsežnejša sprememba podatkov v stisnjenih tabelah povzroča opazno poslabšanje odzivnosti. Odločili smo se, da samo tiste podatke, na katerimi bomo izvajali intenzivne spremembe, ponovno razširimo v osnovno obliko, izvedemo spremembe in nato jasno določimo časovno politiko stiskanja podatkov. Sedaj podatkovna zbirka sama skrbi za stiskanje podatkov, za katere smo prepričani, da se ne bodo več spreminjali.

6.7. Sprememba parametrov podatkovne zbirke

Zaradi agresivnih procesov, ki se izvajajo v podatkovni zbirki, smo natančno proučili notranje delovanje podatkovne zbirke [22]. Podatkovna zbirka ima številne podprocese, s katerimi zagotavlja konsistentnost podatkov in pa odzivnost podatkovne zbirke. Izdelali smo mehanizme za poučevanje ustreznosti nastavitev internih procesov. Le-te smo povezali s sistemom za nadzor upravljanja okolja in tako z grafikoni opazujemo spremembe in zaznamo, kdaj je potrebno spremeniti parametre podatkovne zbirke. Podatkovna zbirka ima številne procese, na primer prilagajali smo proces čiščenja pomnilnika umazanih podatkov. Nato smo pohitrili proces izvajanja konsistentnosti podatkov z dodatnim stiskanjem podatkov transakcijskih logov. Optimirali smo proces sinhronizacije podatkov podatkovne zbirke iz pomnilnika na diskovje. Prilagodili smo več deset parametrov in aktivno spremljamo in ustrezno prilagajamo parametre podatkovne zbirke.

6.8. Optimizacija odstranjevanja mrtvih zapisov podatkovnih tabel

Ker smo bili prisiljeni zgraditi sistem spreminjanja in nadomeščanja podatkov, je s tem nastala množica t.i. mrtvih zapisov v podatkovni zbirki. Proučili smo, kako optimialno izvesti čiščenje mrtvih zapisov v podatkovni zbirki s t.i. autovacuum procesom [23]. Pri tem smo izvedli več korakov optimizacije, ker smo tehtali med hitrostjo čiščenja mrtvih zapisov in pa diskovno obremenitvijo. Če čistiš prehitro, pretirano obremenjuješ diskovje, če prepočasi, leto vpliva na odzivnost povpraševanj na podatkovni zbirki. Ob validaciji podatkov za celotno leto, smo čiščenje začasno povsem izključili in tako pohitrili obdelave.

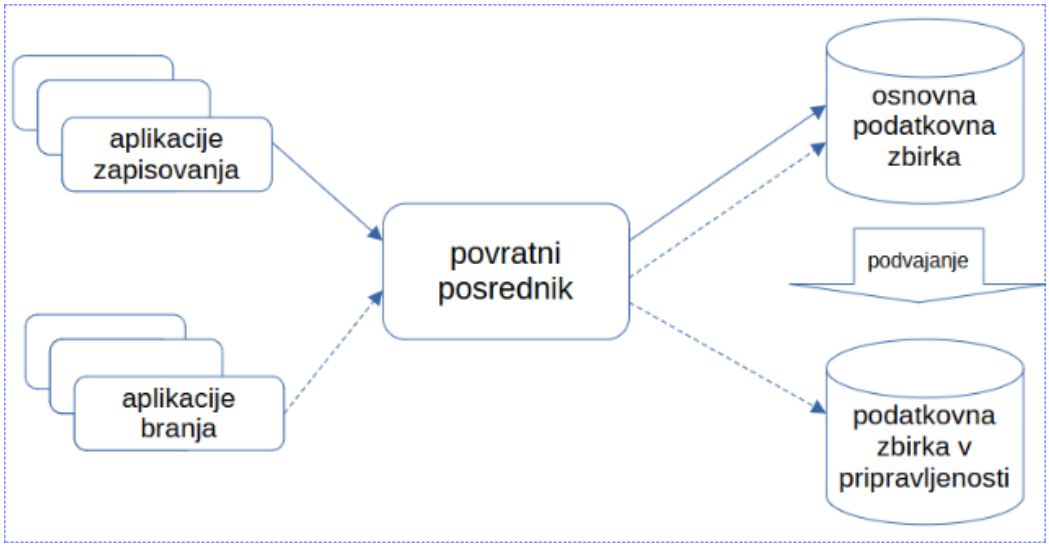
6.9. Spremembe parametrov pomnilniških enot

Večino časa pri optimiranju podatkovne zbirke smo posvetili diskovnim dostopom. Diskovni dostop je lahko tisoč-krat ali več počasnejši od dostopa do pomnilnika. Zato proces optimiranja vidimo kot zmanjševanje potrebe po dostopanju do diskovja. Pri velikih količinah podatkov je lahko težava že sama razdrobljenost podatkov v pomnilniku, ki ga uporablja podatkovna zbirka. Sprva smo mesečno ponovno zaganjali operacijski sistem, ker smo opazili, da deluje vedno počasneje. Z merjenjem pa smo ugotovili, da procesorska jedra tako zelo intenzivno dostopajo do pomnilnika, da so podatki v njem izjemno razdrobljeni in samo iskanje podatkov po pomnilniku pripelje do tega, da se proces delovanja podatkovne zbirke upočasni. Procesor dostopa do pomnilnika do t.i. pomnilniških strani. Privzeto velikost teh pomnilniških strani ni prilagojena količini podatkov, s katerimi upravlja podatkovna zbirka časovnih vrst. Zato smo povečali velikost pomnilniških strani z Linux »kernel« parametri in nastavili nastavitev, da podatkovna zbirka uporablja te velike pomnilniške strani [24]. S tem smo povsem odpravili potrebo po rednem ponovnem zagonu operacijskega sistema, kar je na produkcijskem okolju zelo pomembno, da ne prekinjamo procesov.

7 Optimizacijski postopki – vzporednost izvajanja

7.1. Podvajanje okolja podatkovne zbirke

Šele v fazi, ko smo izvedli vsa ostala optimiranja na podatkovni zbirki, je napočil čas za povečanje računalniške razpoložljivosti z vzporednim izvajanjem podatkovne zbirke. Zahteva naročnika je, da podatki čim prej nastanejo v podatkovni zbirki in so na voljo za uporabo. Posledica tega je, da se v kratkem časovnem razponu pojavlja množično nastajanje in spreminjanje podatkov. Le-to pa ima vpliv na odzivnost klicev podatkov iz podatkovne zbirke.



Slika 2: Arhitektura podvajanja podatkovne zbirke.

Vzpostavili smo okolje podvajanja podatkovne zbirke z namenom pohitritve delovanja poizvedb iz podatkovne zbirke. Imamo dve vrsti aplikacij in sicer aplikacije za množično zapisovanje podatkov in aplikacije za branje. Množično zapisovanje podatkov se izvaja v osnovno podatkovno zbirko. Nato se izvaja proces podvajanja in prepis podatkov v podatkovno zbirko v pripravljenosti. Predstavitvene aplikacije pa berejo podatke iz obeh podatkovnih zbirk. Razporejanje obremenitve osnovne podatkovne zbirke s strani aplikacij za branje je odvisno od dneva v mesecu. V začetku meseca je osnovni podatkovni strežnik bolj obremenjen in tako se na njem izvaja manj aplikacij za branje podatkov. V preostalih dneh je razporeditev med osnovnim strežnikom in strežnikom v pripravljenosti bolj uravnovešeno. Takšna arhitektura omogoča zanesljivost delovanja strežnikov podatkovne zbirke in razporeditev obremenitve strežnikov. Hkrati je dodatna zaščita ob silovitem povečanju mrežnega prometa, da osnovna podatkovna zbirka deluje nemoteno.

8 Zaključek

Podatkovna zbirka in procesi, ki se izvajajo v njej, se neprestano spreminjajo in prilagajajo zahtevam naročnika projekta. Na spletu obstajajo številni viri za pohitritev delovanja podatkovne zbirke. Redno spremljanje dnevnikov podatkovne zbirke, gledanje video posnetkov in branje knjig, vodi do novih in novih idej, kako je mogoče optimirati podatkovno zbirko. Ni enega samega recepta za optimiziranje odzivnosti podatkovne zbirke. Le-ta je odvisna od številnih mehanizmov in procesov, ki tečejo v podatkovni zbirki. Vsem optimizacijam pa je skupno predvidevanje morebitnih težav, zaznava obstoječe težave, prepoznava ideje za izboljšavo, razvoj metode, s katero merimo proces izboljšave, izvedba same izboljšave in merjenje učinka.

V podjetju smo vzpostavili zaupanje med razvijalskimi, sistemskimi in ekipami podatkovnih zbirk. Na ravni baze pa razvili metode dela, ki neprestano uvajajo preizkušanje novih idej. Podatkovne zbirke velikih podatkov imajo

številne možnosti za izboljšanje odzivnosti delovanja podatkovne zbirke. V prispevku smo prikazali številne optimizacijske prijeme z arhitekturnega, aplikacijskega, podatkovnega vidika in z vidika podatkovnih zbirk.

Prišli smo do točke, da podatkovna zbirka deluje odzivno in stabilno. S tem smo pripravljeni na nove izzive, ki bodo vključevale podatkovno zbirko velikih podatkov, kot je strojno učenje na velikih količinah podatkov, napredna analitika in podobno.

Literatura

[1] Akt o metodologiji za obračunavanje omrežnine za elektrooperaterje, Uradni list RS, št. 146/2022 z dne 25. 11. 2022, https://www.uradni-list.si/glasilo-uradni-list-rs/vsebina/2022-01-3624/akt-o-metodologiji-za-obracunavanje-omreznine-za-elektrooperaterje, obiskano 19. 6. 2024.

[2] www.zabbix.com, Zabbix + Postgresql, obiskano 13. 6. 2024.

[3] docs.timescale.com, Create a data retention policy, obiskano 13. 6. 2024.

[4] www.youtube.com, PostgreSQL performance in 5 minutes, obiskano 13. 6. 2024.

[5] www.eversql.com, PostgreSQL – pg_stat_statements, explained, obiskano 13. 6. 2024.

[6] www.postgresqltutorial.com, PostgreSQL EXPLAIN, obiskano 13. 6. 2024.

[7] www.cybertec-postgresql.com, How to interpret PostgreSQL EXPLAIN ANALYZE output, obiskano 13. 6. 2024.

[8] www.tutorialsteacher.com, PostgreSQL - UPSERT Operation, obiskano 13. 6. 2024.

[9] www.alibabacloud.com, How Does PostgreSQL Implement Batch Update, Deletion, and Insertion, obiskano 13.6.2024.

[10] knowledge.enterprisedb.com, Idle and Idle in transaction connections and impact on performance and ways to handle it, obiskano 13. 6. 2024.

[11] www.enterprisedb.com, The Challenges of Setting max_connections and Why You Should Use a Connection Pooler, obiskano 13. 6. 2024.

[12] hevodata.com, PostgreSQL LOCKS: Comprehensive Guide 101, obiskano 13. 6. 2024.

[13] www.postgresqltutorial.com, PostgreSQL INSERT Multiple Rows, obiskano 13. 6. 2024.

[14] www.ibm.com, Parameter markers, obiskano 13. 6. 2024.

[15] medium.com, Delaying commits in PostgreSQL, obiskano 13. 6. 2024.

[16] docs.timescale.com, Hypertable partitioning, obiskano 13. 6. 2024.

[17] docs.timescale.com, Reorder_chunk, obiskano 13. 6. 2024.

[18] www.cybertec-postgresql.com, What is fillfactor and how does it affect PostgreSQL performance?, obiskano 13. 6. 2024.

[19] www.freecodecamp.org, Advanced Indexing Strategies in PostgreSQL, obiskano 13. 6. 2024.

[20] docs.timescale.com, Continuous aggregation, obiskano 13. 6. 2024.

[21] docs.timescale.com, Compression, obiskano 13. 6. 2024.

[22] ROGOV Egor, "Buffer cache and WAL", PostgreSQL 14 Internals", Postgres Professional, 2022, str. 168-209.

[23] pganalyze.com, Visualizing & Tuning Postgres Autovacuum, obiskano 13. 6. 2024.

[24] www.enterprisedb.com, Improving PostgreSQL performance without making changes to PostgreSQL, obiskano 13.6.2024.