Međuoprema za komunikaciju porukama u okolinama računarstva na rubu

Mrkonjić, Ana

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 536

MEĐUOPREMA ZA KOMUNIKACIJU PORUKAMA U OKOLINAMA RAČUNARSTVA NA RUBU

Ana Mrkonjić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 536

MEĐUOPREMA ZA KOMUNIKACIJU PORUKAMA U OKOLINAMA RAČUNARSTVA NA RUBU

Ana Mrkonjić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 536

Pristupnica:	**Ana Mrkonjić (0036514185)**
Studij:	Računarstvo
Profil:	Znanost o mrežama
Mentorica:	prof. dr. sc. Ivana Podnar Žarko

Zadatak: **Međuoprema za komunikaciju porukama u okolinama računarstva na rubu**

Opis zadatka:

U raspodijeljenoj okolini Interneta stvari danas se sve češće za obradu podataka uz računalni oblak koriste uređaji na rubu mreže (engl. edge computing) koji su hijerarhijski organizirani od lokalne mreže u blizini izvora podataka do računalnog oblaka. Uređaji na rubu imaju ograničena sredstava u odnosu na računalni oblak, ali su mrežno bliže izvoru podataka, čime se smanjuje količina podataka prenesena do računalnog oblaka i kašnjenje obrade podataka s IoT-uređaja zbog smanjenog vremena za prijenos podataka do obradnog čvora na rubu mreže. Međuoprema za komunikaciju porukama omogućuje fleksibilno usmjeravanje i filtriranje podataka s IoT-uređaja u ovoj okolini do odgovarajućih usluga pokrenutih na različitim uređajima na rubu mreže i u računalnom oblaku. Vaš je zadatak analizirati i ispitati mogućnosti korištenja međuopreme za komunikaciju porukama Apache Kafka u okruženju računarstva na rubu te predložiti rješenje u kojem su Kafkini posrednici postavljeni na različitim slojevima okruženja računarstva na rubu za primjenu u kontekstu Interneta stvari. Posebno istražite funkcionalnosti fleksibilnog usmjeravanja i filtriranja podataka s IoT-uređaja u ovoj okolini do odgovarajućih obradnih usluga. Potrebno je eksperimentalno ispitati ograničenja izvođenja Kafkinih klijenata i posrednika na različitim uređajima kako bi se identificirali minimalni zahtjevi za izvođenje Kafkine konfiguracije u okruženju računarstva na rubu. Ispitajte predloženo rješenje u emuliranom okruženju računarstva na rubu, gdje kašnjenje raste od ruba prema oblaku, s Kafkinim posrednicima pokrenutim na različitim slojevima hijerarhije računarstva na rubu.

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

Uvod

Pojavom Interneta stvari došlo je do generiranja velikih količina podataka na tzv. rubu mreže. Prije slanja velikih količina podataka na računalni oblak, pojavila se potreba za obradom i pohranom takvih podataka što bliže njihovom izvoru odakle i potječe naziv „računarstvo na rubu“. Uređaji na rubu mreže imaju manje resursa za razliku od računalnog oblaka, ali zbog blizine izvoru podataka smanjuju količinu podataka koja se prenosi u oblak. Time smanjuju mrežno opterećenje te skraćuju vrijeme potrebno za obradu podataka s IoT-uređaja jer je prijenos do obradnog čvora na rubu mreže brži. Obradni čvorovi na rubu mreže koriste se u ovome radu za postavljanje i pokretanje međuopreme za komunikaciju porukama i trebaju omogućiti fleksibilno usmjeravanje i filtriranje poruka s IoT-uređaja prema servisima za obradu tih podataka.

Popularni sustav za razmjenu poruka Apache Kafka tako je neizbježno svoju primjenu dobio i u računarstvu na rubu mreže kao dio međuopreme za komunikaciju. Korištenje Kafke na rubu mreže više nije iznimka već učestala pojava, pa se zbog toga primjenjuje u raznim domenama i za različite primjene. Cilj je ovog rada ispitati i analizirati korištenje Apache Kafke u okolini računarstva na rubu i predložiti rješenje u kojem su Kafkini posrednici raspoređeni na različitim slojevima okruženja računarstva na rubu za primjenu u kontekstu Interneta stvari.

Rad je organiziran u 4 poglavlja i zaključak. U prvom poglavlju je objašnjen pojam računarstva na rubu te su navedeni izazovi koji se javljaju u takvim okruženjima. Potom je objašnjeno što je međuoprema za razmjenu porukama te su navedena postojeća rješenja i primjene međuopreme. U drugom poglavlju su opisana obilježja sustava Apache Kafka te su analizirane mogućnosti za usmjeravanje i filtriranje poruka koje pruža uz koje prednosti Kafka donosi za primjenu u okruženju računarstva na rubu.

Treće poglavlje predstavlja razvijeno rješenje za filtriranje poruka u sustavu Apache Kafka prilagođeno okruženju računarstva na rubu. U četvrtom poglavlju su navedeni rezultati provedenog ispitivanja razvijenog rješenja i klasične primjene Kafkinog grozda u dva scenarija radi usporedbe resursnog opterećenja i generiranog mrežnog prometa.

1. Računarstvo na rubu

Kada govorimo o računarstvu na rubu, dobro je definirati na što se točno odnosi izraz „rub mreže“. Izraz „računarstvo na rubu“ (engl. *edge computing*) odnosi se na proces premještanja obrade i pohrane podataka bliže uređajima koji su izvor tih podatka i korisnicima koji koriste te podatke. Aplikacije Interneta stvari u nedavnoj su prošlosti prenosile podatke uglavnom od senzora i mobilnih uređaja do središnjeg podatkovnog centra [1]. Takvi centralizirani sustavi uz dostignuće novih tehnologija mogu se skalirati tako da rukuju s milijunima poruka u sekundi. Međutim, takva se rješenja danas nalaze pred izazovom uz sve strože zahtjeve za odgovarajućom kvalitetom usluge (engl. *Quality of Service, QoS*) i zahtjeva za privatnošću modernih IoT-rješenja [2].

Kao rješenje za takve probleme prelazi se sve više na računarstvo na rubu. Prenošenjem obrade podataka bliže njihovom izvoru, smanjuje se količina podataka koja se šalje u oblak, također podaci s uređaja se brže obrađuju jer se skraćuje vrijeme prijenosa podataka do čvora za obradu na rubu mreže. Time se može znatno smanjiti opterećenje mreže i poboljšati performanse sustava, ubrzati vrijeme odgovora i smanjiti zahtjeve za propusnošću (engl. *throughput*). Dodatne prednosti koje ovakva arhitektura sustava donosi su smanjenje operativnih troškova, prilagodljiva arhitektura i jasno razdvajanje odgovornosti.

1.1. Međuoprema za komunikaciju porukama

Međuoprema za komunikaciju porukama (engl. *Message-Oriented Middleware*) u kontekstu Interneta stvari odnosi se na softversku infrastrukturu koja omogućuje komunikaciju između IoT-uređaja i krajnjih aplikacija razmjenom poruka. MOM igra ključnu ulogu u IoT-sustavima jer pruža pouzdanu, skalabilnu i sigurnu komunikaciju između različitih dijelova raspodijeljenog okruženja (uređaja, aplikacija i platforme). MOM sustavi pružaju značajke poput spremanja poruka u redove (engl. *message queuing*), usmjeravanje, transformaciju i zajamčenu isporuku poruka, osiguravajući tako siguran protok poruka kroz heterogene okoline [3].

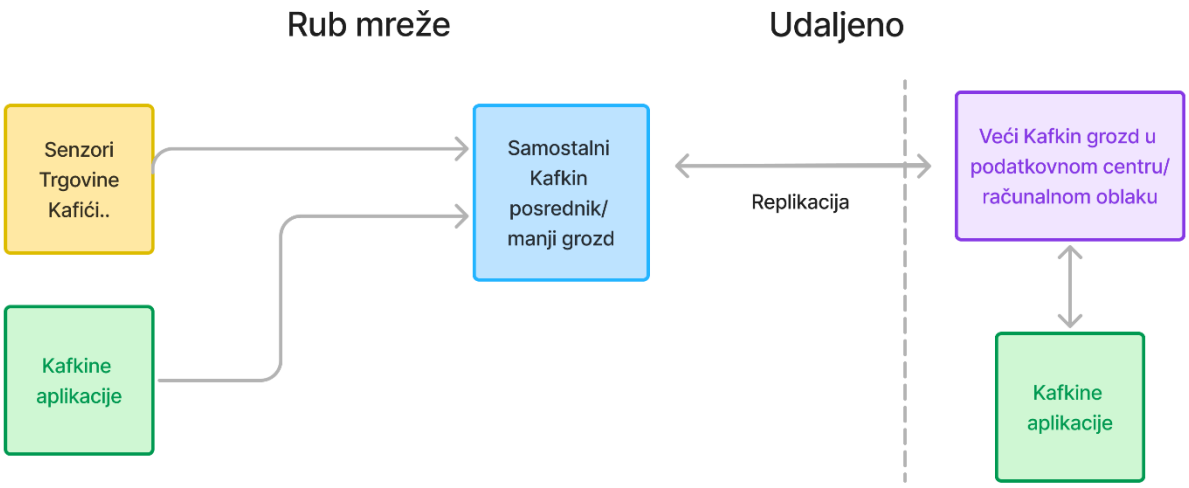
1.2. Postojeća rješenja

MOM se pokazuje ključnim u različitim scenarijima primjerice iz domene mobilnog zdravstva (engl. *mobile health, mHealth*) u slučajevima nekih katastrofa. U takvim slučajevima pacijenti na sebi imaju bio-senzore koji neprekidno objavljuju zdravstvene podatke na mobilne uređaje medicinskog osoblja u blizini koje dobiva obavijesti o stanju pacijenata. U ovom slučaju centralizirano rješenje koje se temelji na oblaku (engl. *cloud-based*) bilo bi nepouzdano zbog vremenske kritičnosti. Boljim rješenjem pokazuje se obrada podataka blizu izvora, tako da se podaci sa senzora šalju odmah na mobilne uređaje osoblja, zatim na male centre podataka ili jaka računala koji se mogu nalaziti na primjer u vozilima hitne pomoći. Tamo se podaci mogu obrađivati u stvarnom vremenu, omogućujući timu hitne pomoći da vidi podatke o hitnosti pacijenta. Podaci se po potrebi nakon toga mogu slati i u obližnje bolnice kako bi se pripremio doček pacijenta i za trajnu pohranu se mogu poslati i u računalni oblak (engl. *cloud*) kada se uspostavi stabilna internetska veza. MOM igra kritičnu ulogu u ovakvim scenarijima jer omogućuje malo kašnjenje u komunikaciji [2].

Primjeri tehnologija i protokola koji se koriste kao međuoprema za komunikaciju porukama su protokol MQTT, AMQP, CoAP, Apache Kafka, ZeroMQ, RabbitMQ i drugi.

Apache Kafka počinje se sve češće koristiti i na rubu mreže, kao međuoprema za komunikaciju porukama. Kada govorimo o Kafki u kontekstu računarstva na rubu (engl. *Kafka at the Edge*), u većini slučajeva mislimo na scenarij kada se Kafkini klijenti, ali i brokeri nalaze na rubu mreže. Arhitektura Kafke na rubu mreže tipično je troslojna arhitektura, kao što je prikazano na slici (Sl. 1), koja se sastoji od:

- klijentskog sloja – IoT uređaji, aplikacije
- rubnog sloja (engl. *edge layer*)- Kafkini posrednici koji se nalaze na rubu mreže i posrednici koji se nalaze u podatkovnim centrima na rubu mreže, primaju podatke s IoT-uređaja i mogu biti u više slojeva, odnosno u hijerarhiji
- sloja računalnog oblaka – Kafkin grozd u računalnom oblaku koji prima podatke s rubnog sloja za daljnje agregiranje i analizu podataka



Sl. 1 Apache Kafka na rubu mreže

Apache Kafka raspodijeljena je *streaming* platforma koja se koristi za razvoj pouzdanih i skalabilnih podatkovnih cjevovoda (engl. *data pipelines*) i koja omogućuje pretplaćivanje i objavljivanje na tokove podataka. Dobro je prilagođena za scenarije korištenja računarstva na rubu jer je „lagana“, otporna na pogreške i skalabilna.

Kafka se u računarstvu na rubu često koristi za omogućavanje strojnog učenja na rubu jer se koristi za prikupljanje, obradu i slanje podataka od rubnih uređaja do modela za strojno učenje u svrhu zaključivanja [4]. Neki primjeri strojnog učenja u kojima se koristi su autonomna vozila, industrijska automatizacija i video-nadzor.

Još neki od slučajeva korištenja Kafke u okolini računarstva na rubu su postavljanje Kafke na lokacije poput maloprodajnih trgovina, baznih stanica, vlakova, malih tvornica ili restorana [5]. Koristi se u proizvodnim industrijama, farmaceutskim industrijama, telekomunikacijama, energetskim sustavima, autoindustriji, zdravstvu, zrakoplovstvu, transportu, i ostalima.

Izazovi koji se susreću na rubu mreže su:

- rad izvan pristupa internetu – važno je da se komunikacija nastavi i kada dođe do ispada mreže
- resursno ograničenje – računarstvo na rubu je ograničeno resursima

- minimalna potreba za tehničkom intervencijom – problematično bi bilo za svaku izmjenu imati stručnjaka na licu mjesta
- integracija - velike količine različitih povezanih uređaja – senzori, strojevi, mobilni uređaji i ostalo
- obrada u stvarnom vremenu – potrebno u mnogim slučajevima

2. Primjena Kafke u računarstvu na rubu

2.1. Apache Kafka

Kao što je ranije spomenuto, Apache Kafka je platforma koja implementira komunikaciju na načelu objavi/pretplati uz mogućnost trajne pohrane podataka. U slučajevima kada se koristi u okolini Interneta stvari donosi prednosti time što povećava propusnost i osigurava malo kašnjenje tijekom obrade velikih količina poruka u stvarnom vremenu. Osim toga ima sposobnost dobre integracije s mnogim drugim sustavima [6].

Arhitektura Apache Kafke sastoji se od posrednika koji mogu funkcionirati samostalno ili biti dio tzv. Kafkinog grozda, tema, particija, proizvođača i potrošača. Dijelovi sustava koji objavljuju poruke prema Kafkinom grozdu (čini ga više Kafkinih posrednika) ili samostalnom Kafkinom posredniku, nazivaju se proizvođači ili objavljivači (engl. *producers*), a pretplatnici na poruke nazivaju se potrošačima (engl. *consumers*). Poruke se pohranjuju na teme (engl. *topics*) koje apstrahiraju pohranu podataka.

Svaka se tema sastoji od jedne ili više particija (engl. *partitions*). Particija je podskup podataka teme spremljenih u redoslijedu kojim su pristigli. Potrošači se dakle pretplaćuju na teme i dohvaćaju poruke kada budu objavljene na temu. Potrošači koji pripadaju istoj grupi (engl. *consumer group*) mogu čitati poruke s jedne teme istovremeno tako što se svakom potrošaču iz grupe dodijeli jedna particija čime se omogućuje paralelno čitanje poruka s teme. Tako se osigurava podjednaka raspodjela poruka među potrošačima iste grupe. Različite grupe potrošača također mogu istovremeno čitati poruke neovisno jedna o drugoj. S pomoću particija i grupa potrošača Kafka postiže tako veliku propusnost (engl. *throughput*). Redoslijed poruka nije garantiran na razini teme, jedino na razini particija. Stoga kada se želi osigurati pravilan redoslijed poruka, uz poruku se postavlja ključ jer sve poruke s istim ključem dolaze na istu particiju.

Kada se teme stvaraju, treba odabrati koliko će particija tema imati, što nije uvijek jednostavno. U obzir treba uzeti koliko će poruka stizati na temu jer u slučaju većeg broja poruka, potrebno je kreirati i veći broj particija ako se želi osigurati bolja propusnost. Međutim, veći broj particija povećava operativne troškove sustava. Gruba formula koja može biti smjernica za definiranje broja particija s obzirom na propusnost je sljedeća. Izmjeri se propusnost koja se može postići na jednoj particiji za objavu poruka (engl. *production*,

označimo s *p*) i dohvaćanje poruka (engl. *consumption*, označimo s *c*). Ako željenu propusnost označimo s t, onda bi trebali imati najmanje *max(t/p, t/c)* particija za tu temu [7].

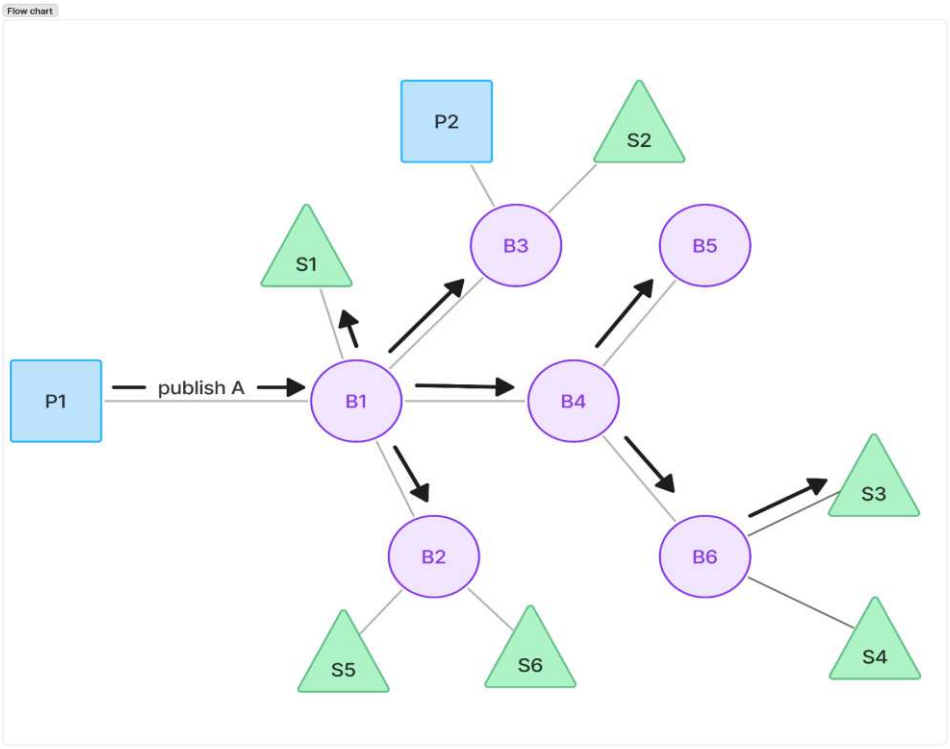
Objavljivači i pretplatnici nisu direktno povezani i ne moraju znati za međusobno postojanje. Kafka podržava replikaciju podataka unutar grozda. Replikacija se ostvaruje postavljanjem replikacijskog faktora teme koji označava na koliko će se kopija (replika) svake particije napraviti u grozdu. Ako se replikacijski faktor postavi na tri, a u grozdu se nalaze tri posrednika, to će značiti da će particija imati svoju kopiju na svakom posredniku. Replikacija omogućuje otpornost na pogreške u sustavu.

2.2. Usmjeravanje i filtriranje poruka

Jedno od osnovnih načela usmjeravanja poruka u raspodijeljenim sustavima je tzv. preplavljivanje. Preplavljivanje podrazumijeva da posrednici svaku primljenu poruku, koja može biti obavijest, pretplata ili odjava pretplate, prosljeđuju svim susjednim posrednicima osim onome od koga su poruku primili.

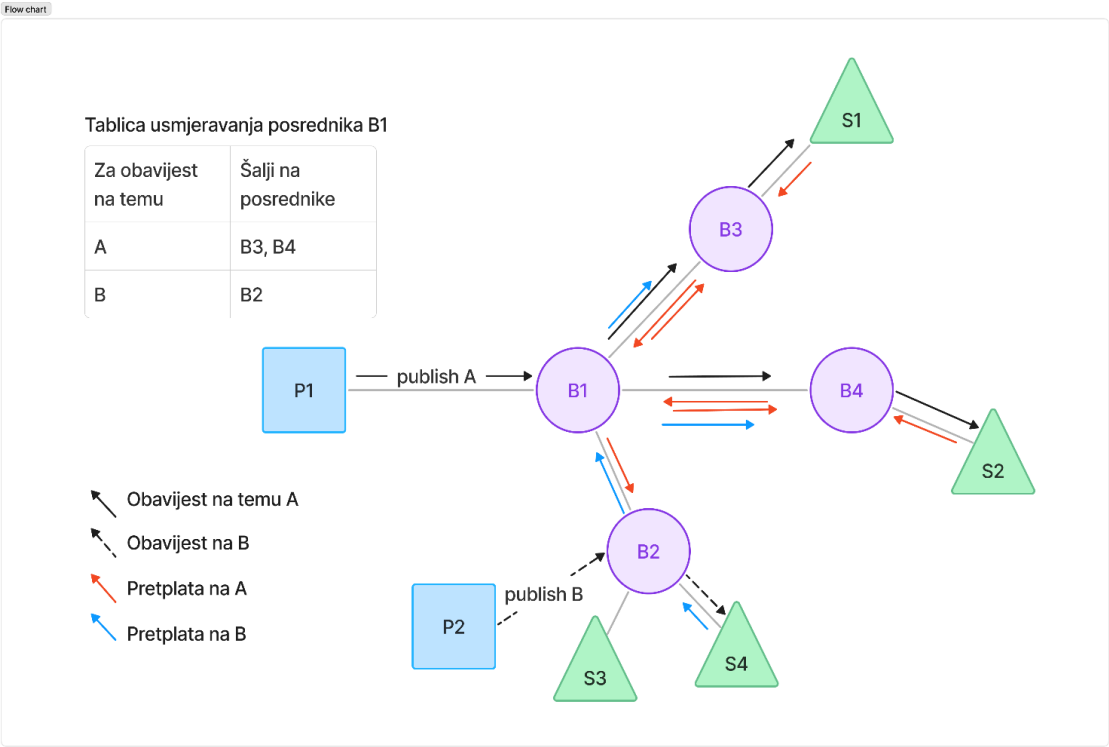
Primjer dvaju algoritama usmjeravanja poruka u sustavima modela objavi-pretplati su preplavljivanje obavijestima i preplavljivanje pretplatama [8].

Primjer preplavljivanja obavijestima prikazuje slika Sl. 2. Oznakama B1, B2, B3, B4, B5 i B6 označeni su posrednici, oznakama S1, S2, S3, S4, S5, S6 pretplatnici, a oznakama P1 i P2 objavljivači poruka u sustavu. Svaka poruka prosljeđuje se svim susjednim posrednicima bez obzira imaju li zainteresirane pretplatnika ili ne, te se tako stvara suvišan mrežni promet što je najveći nedostatak ovog algoritma.



Sl. 2 Preplavljivanje obavijestima [8]

Drugi je primjer preplavljivanje pretplatama, kao što prikazuje slika Sl. 3. Posrednici međusobno izmjenjuju samo poruke s pretplatama, a obavijesti se prosljeđuju samo onim posrednicima koji imaju zainteresirane pretplatnike. Posrednici pohranjuju informaciju zainteresiranih susjednih pretplatnika i ažuriraju stanje s obzirom na primljene pretplate. Prednost ovog slučaja smanjenje je generiranog mrežnog prometa jer poruke ne dolaze do posrednika koji nemaju zainteresirane pretplatnike, kao što je to bio slučaj kod preplavljivanja obavijestima. Preplavljivanje pretplatama je algoritam koji je korišten za rješenje implementiran u ovome radu jer se održavanjem stanja pretplata želi izbjeći nepotrebno prosljeđivanje obavijesti u sustavu posrednika.



Sl. 3 Preplavljivanje pretplatama [8]

Predviđeni i najčešći način korištenja Kafke je u obliku grozda koji podržava replikaciju i tako osigurava pouzdanost sustava i osigurava otpornost sustava na pogreške i ispade. U Kafkinom grozdu kada je replikacijski faktor jednak broju posrednika u grozdu, poruke se repliciraju na sve posrednike, što znači da svaki posrednik prima sve poruke koje stižu na sustav. Poruke se dakle prosljeđuju svim posrednicima.

Ako želimo podržati drugačije usmjeravanje poruka u sustavu ili čak filtriranje poruka, Kafka to automatski ne podržava. Zbog toga se filtriranje može „premjestiti“ tako da se obavlja na klijentu tako da se doda logika kojom primljene obavijesti klijent prvo filtrira te ih potom neće koristiti u svom radu. Ako se filtriranje poruka prebaci na klijenta, mreža se nepotrebno opterećuje porukama koje će klijent odbaciti. Ovaj scenarij ima smisla jedino ako je količina poruka u sustavu relativno malena i nema značajnijih zahtjeva na propusnost.

Bolja je opcija filtriranje poruka prije isporuke klijentu ili drugom posredniku, odnosno poruke se može filtrirati na samome posredniku ili mrežno što bliže njemu. Neki od načina kako se funkcionalnost filtriranja može ostvariti su korištenjem Kafkinih nativnih komponenti (koje su uključene prilikom instalacije Kafke) kao što su *Kafka Connect* i

MirrorMaker2, prilagođeni Kafka klijenti (objavljivači/pretplatnici koji će sadržavati logiku filtriranja) i komponente za obradu tokova podataka *Kafka Streams* i *ksqlDB*.

Mogući načini filtriranja poruka u sustavima koji koriste Kafku su sljedeći:

1. Filtriranje na strani klijenata

Kao što je ranije spomenuto, ovakvo filtriranje opterećuje mrežu zbog velike količine razmijenjenih poruka pa je poželjno izbjegavat ovaj pristup.

2. Korištenje zaglavlja i metapodataka

Prilikom slanja poruka mogu se dodati zaglavlja poruka i metapodaci koje će se koristiti za filtriranje na klijentskoj strani. Primjerice može se dodati informacija o identifikacijskom broju aplikacije koja je proizvela poruku ili vrsti podataka koje poruka sadrži. Te informacije mogu olakšati pretplatniku ili Kafka Streams aplikaciji pri odluci o prihvaćanju poruke za obradu.

3. Korištenje uređaja za računarstvo na rubu

Za filtriranje poruke u sustav se mogu dodati uređaji koji će obrađivati podatke s IoT-uređaja prije slanja na posrednike. Ovakvi uređaji mogu filtrirati i agregirati podatke lokalno na rubu mreže prije isporuke do Kafkinog grozda. Nedostatak je dodatna oprema i dodano kašnjenje poruka.

4. Kafka Streams [11] i ksqlDB [12]

Opcije za filtriranje kada se obrađuje tok podataka, a podatke je potrebno obraditi u stvarnom vremenu.

5. Kafka MirrorMaker 2

Omogućuje replikaciju tema između dva Kafka grozda uz što se može dodati jednostavna logika za filtriranje poruka.

6. Kafka Connect [13]

Kafka Connect okvir je za povezivanje Kafke s vanjskim sustavima. Pokreće se kao grozd s jednim ili više sustavnih procesa. Može se prilagoditi za komunikaciju između posrednika s pomoću tzv. konektora (engl. *connectors*). Jednostavna logika filtriranja može se postići samo dodavanjem konfiguracijskih datoteka, a za složenije načine korištenja može se razviti vlastiti *plug-in*, a to je zapravo vlastita implementacija Javinih klasa postojećih konektora.

7. Prilagođena „agentska“ aplikacija

Izradom vlastite aplikacije koja služi kao posrednik u komunikaciji između posrednika može se dodati željena logika usmjeravanja i filtriranje poruka.

Od navedenih rješenja, potencijalne opcije za usmjeravanje i filtriranje poruka između posrednika su korištenje komponente Kafka Connect ili implementacija prilagođene vlastite „agentske“ aplikacije. Za jednostavnije slučajeve kada se samo želi dohvaćati poruke s jedne teme i objavljivati na drugu dovoljno je koristiti vlastitu aplikaciju koja će raditi kao objavljivač i pretplatnik na teme uz dodatnu jednostavnu logiku. Korištenje komponente Kafka Connect s prilagođenim *plug-in*-om u ovakvom scenariju može donijeti dodatno resursno opterećenje te kompleksnost u razvoju. Zbog jednostavnosti, fleksibilnosti i potencijalnih dodatnih zahtjeva koje komponenta Kafka Connect ne bi podržavala, u ovom radu će se razvijati vlastita agentska aplikacija za svaki posrednik.

2.3. Podrška za računarstvo na rubu

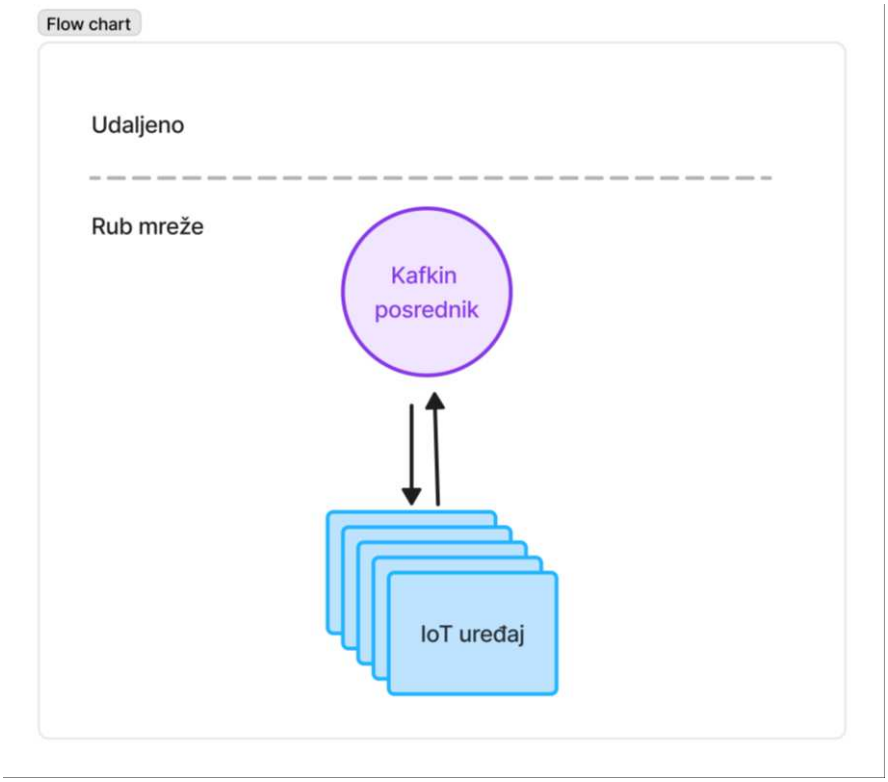
Kafka može biti postavljena na različitim infrastrukturama, poput računala, odnosno poslužitelja, virtualnih strojeva, kontejnera, *Kubernetes*-a, *Raspberry Pi*-ja itd.

Prednost Kafke je što samostalan Kafka posrednik može podržavati veliki broj klijenata pa nije potrebno imati grozd za najniži sloj posrednika na rubu mreže. Česta je primjena stoga samostalnih posrednika na krajnjem rubu koji zatim repliciraju dio podataka slanjem na centralni Kafkin grozd. Mogućnosti koje samostalni posrednik donosi su pohrana na disk, podrška Kafkinih nativnih komponenti (Kafka Connect, Kafka Streams, ksqlDB itd.), sposobnost obrade velikog obujma podataka i mnoge druge. Samostalni posrednici mogu se koristiti za lokalno procesiranje, a dodatni Kafka grozd može se dodati kao posrednik između lokalnih posrednika i Kafka grozda u računalnom oblaku.

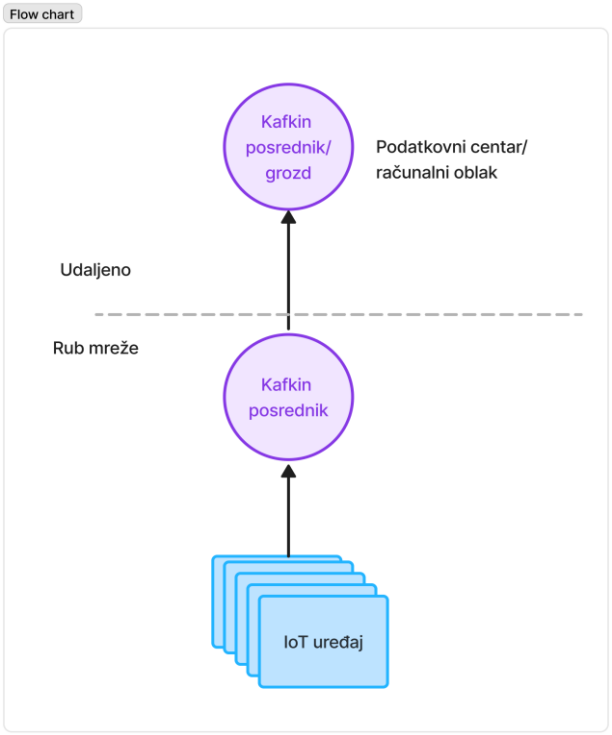
Kafkini klijenti često se izvode na ugrađenim uređajima (engl. *embedded devices*) ograničenih sredstava koji koriste programske jezike C ili C++ uz Kafkine klijentske biblioteke za direktnu komunikaciju s posrednicima ili posrednikom REST za komunikaciju putem HTTP-a. Ostali programski jezici poput Jave, Python-a, Javascripta i drugih, također podržavaju implementaciju Kafkinih klijenata.

Vrste komunikacije koje Kafka podržava za računarstvo na rubu su:

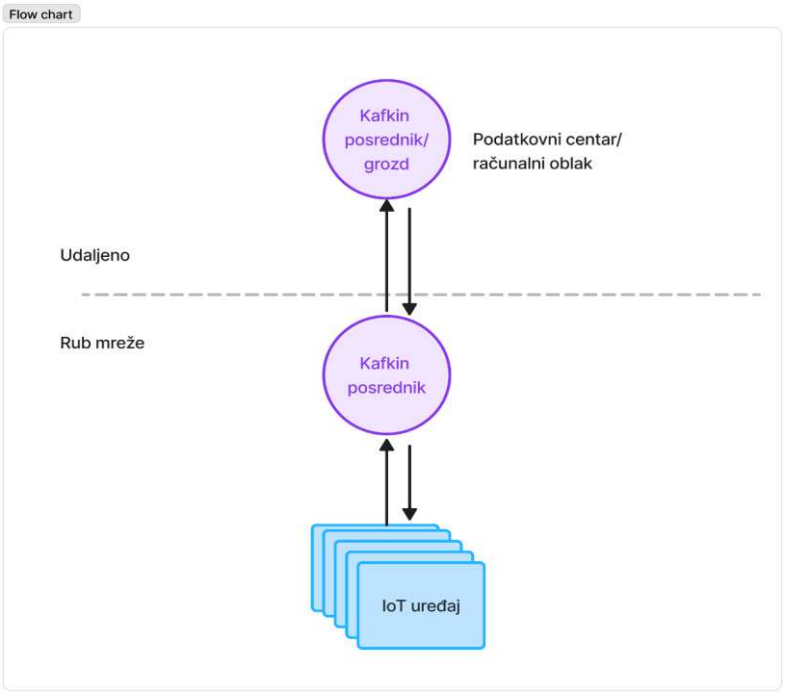
1. Samo na rubu *(*engl. *edge-only*): uređaj > Kafka na rubu > uređaj (Sl. 4)
2. Od ruba do udaljenog (engl. *edge-to-remote*): uređaj > Kafka na rubu > Kafka (podatkovni centar/računalni oblak) (Sl. 5)
3. Dvosmjerno (engl. *bidirectional*): slučajevi 1) i 2) s komunikacijom u oba smjera (Sl. 6)



Sl. 4 Komunikacija samo na rubu mreže



Sl. 5 Jednosmjerna komunikacija od ruba mreže do udaljenog dijela sustava



Sl. 6 Dvosmjerna komunikacija od ruba mreže do udaljenog dijela sustava

3. Razvijeno rješenje

Za demonstraciju usmjeravanja i filtriranja podataka korištenjem Kafke u računarstvu na rubu razvijeno je rješenje u kojem je implementirano slanje poruka između samostalnih Kafkinih posrednika u mreži. Za tu svrhu razvijene su Javine agentske aplikacije za svakog posrednika u sustavu. Posrednici su u hijerarhiji, jedan posrednik nalazi se na vrhu hijerarhije, on izvodi u računalnom oblaku, a na njega se spaja *n* posrednika na rubu. Posrednici na rubu mreže mogu komunicirati međusobno preko posrednika na vrhu hijerarhije, na kojeg se također mogu spajati klijenti.

Poruke (obavijesti) se prosljeđuju onim posrednicima u mreži koji imaju pretplaćene klijente na te obavijesti. Ako ostali posrednici u sustavu nemaju pretplatnike na obavijest, ona se zadržava na lokalnom posredniku i ne prosljeđuje se dalje u sustavu. Posrednici se dakle neće preplavljivati obavijestima kao što je prikazano na slici (Sl. 2), nego će se obavijesti slati preko posrednika u računalnom oblaku samo onim posrednicima koji imaju zainteresirane klijente, a to će se postići tako što će se pratiti pretplate lokalnih klijenata i prema tome će se agentska aplikacija svakog posrednika pretplatiti na teme posrednika na rubu ili u računalnom oblaku, na teme koje sadrže obavijesti pristigle s ostalih posrednika, a za koje su lokalni klijenti zainteresirani. Obavijesti s udaljenih posrednika dohvaćat će se preko posrednika u računalnom oblaku i pohranjivati na interne istoimene teme na koje se pretplaćuju lokalni klijenti. Na ovaj način želi se postići manje opterećenje komunikacijske mreže između posrednika, tj. smanjuje se komunikacija između posrednika na rubu mreže i onoga u računalnom oblaku. Navedeno rješenje implementirano je u kontekstu Interneta stvari tako što su klijenti sustava simulirane punionice električnih vozila. Za ovaj scenarij koristi se Kafka jer se želi podržati veliki broj punionica, što znači veliko opterećenje posrednika u računalnom oblaku svim porukama generiranim na nivou punionica.

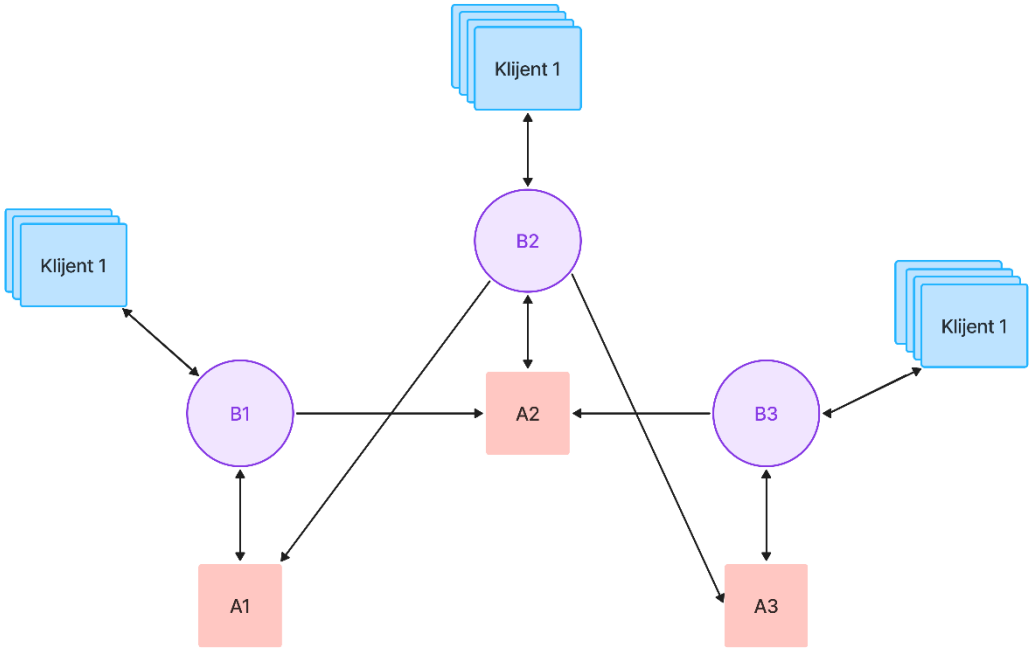
3.1. Arhitektura rješenja

Osnovne komponente sustava su sljedeće:

- *n* samostalnih Kafkinih posrednika na rubu mreže
- samostalni Kafkin posrednik u računalnom oblaku
- klijenti (objavljivači i pretplatnici)
- agentska aplikacija (objavljivač i pretplatnik) za svakog posrednika

Planirano ponašanje sustava:

Informacije koje se izmjenjuju u sustavu odnose se na punionice električnih vozila. Iz tog razloga svaki posrednik definira teme pod nazivom „*p-zg-N-b{ID posrednika}*“, gdje *N* označava identifikacijski broj regije, odnosno područja, u kojem se punionica nalazi. Tako će u sustavu u kojem postoje posrednici s ID-jevima 1, 2 i 3 koji razmjenjuju poruke vezane za punionice zagrebačkih regija s identifikacijskim brojevima 1, 2 i 3 postojati teme *p-zg-1-b1, p-zg-1-b2, p-zg-1-b3* itd. U nastavku će biti objašnjeno zašto je potrebno imati teme iste regije s nastavkom za svaki posrednik. Posrednici imaju skup tema za svaku regiju punionica. Svi posrednici imaju iste teme, ali posrednici dobivaju poruke istoimene teme s drugih posrednika samo ako postoje pretplaćeni lokalni klijenti za tu regiju. Slika (Sl. 7) prikazuje osnovne komponente sustava i komunikacijske kanale među njima na primjeru dva posrednika na rubu mreže i jednog posrednika u računalnom oblaku (B2 sa slike).



Sl. 7 Osnovne komponente razvijenog rješenja

Klijenti su uređaji koji komuniciraju međusobno pomoću Kafke. Svaki je klijent ujedno i proizvođač i potrošač, odnosno objavljivač i pretplatnik.

3.2. Kafkini posrednici

Kafkini posrednici su virtualne instance s operacijskim sustavom Ubuntu na kojima je pokrenuta Kafka uz servis Zookeeper. Kafkini posrednici u sustavu rade samostalno tj. ne pripadaju istom grozdu. Svaki posrednik razmjenjuje poruke s vlastitom agentskom Javinom aplikacijom. Aplikacija posrednika u računalnom oblaku (A2 na slici Sl. 7) ima podatke za spajanje na sve posrednike na rubu mreže, a aplikacija svakog posrednika na rubu mreže (A1 i A3 na slici Sl. 7) ima podatak o IP adresi i vratima samo posrednika u računalnom oblaku. Svi posrednici u sustavu sadrže iste teme koje se vrlo jednostavno mogu programski kreirati udaljeno od strane administratora s pomoću Kafkinog sučelja *AdminClient*. Sučelje pruža administratorske operacije za Kafkine posrednike. Prilikom stvaranja tema replikacijski faktor postavlja se na 1 jer se radi o samostalnim posrednicima. Broj particija postavljen je na 4 kako bi se poboljšala propusnost pri objavljivanju poruka jer se na različite particije poruke mogu objavljivati paralelno. Ovaj broj particija dovoljno je nizak da se ne bi trebale

narušavati performanse zbog dodanog resursnog opterećenja koji može uzrokovati prevelik broj particija.

3.3. Usmjeravanje i filtriranje poruka

Za usmjeravanje i filtriranje poruka koriste se kao što je već spomenuto prilagođene agentske Javine aplikacije. Svaka aplikacija je ujedno i objavljivač i pretplatnik, a svaki posrednik u sustavu ima vlastitu agentsku aplikaciju. S obzirom radi li se o aplikaciji namijenjenoj posredniku na rubu mreže ili onome u računalnom oblaku, određeno je ponašanje aplikacije. Aplikacija se u oba slučaja pretplaćuje na vlastitu temu *subscriptions* jer s pomoću nje provjerava ima li pretplaćenih lokalnih klijenata i prema tome određuje treba li se pretplatiti na istoimene teme drugih posrednika u sustavu i ažurirati lokalne teme dobivenim obavijestima. Svaka regija u sustavu ima skup tema sa svojim *ID*-jem, odnosno postoji onoliko tema za jednu regiju punionica koliko je posrednika u sustavu kako bi se moglo razlikovati s kojeg su posrednika došle poruke kako ne bi došlo do zatvorene petlje u razmjeni poruka. Tako će se, primjerice, za regiju s ID-jem 2 stvoriti teme *p-zg-2-b1*, *p-zg-2-b2* i *p-zg-2-b3*, jedna tema za svakog posrednika u sustavu.

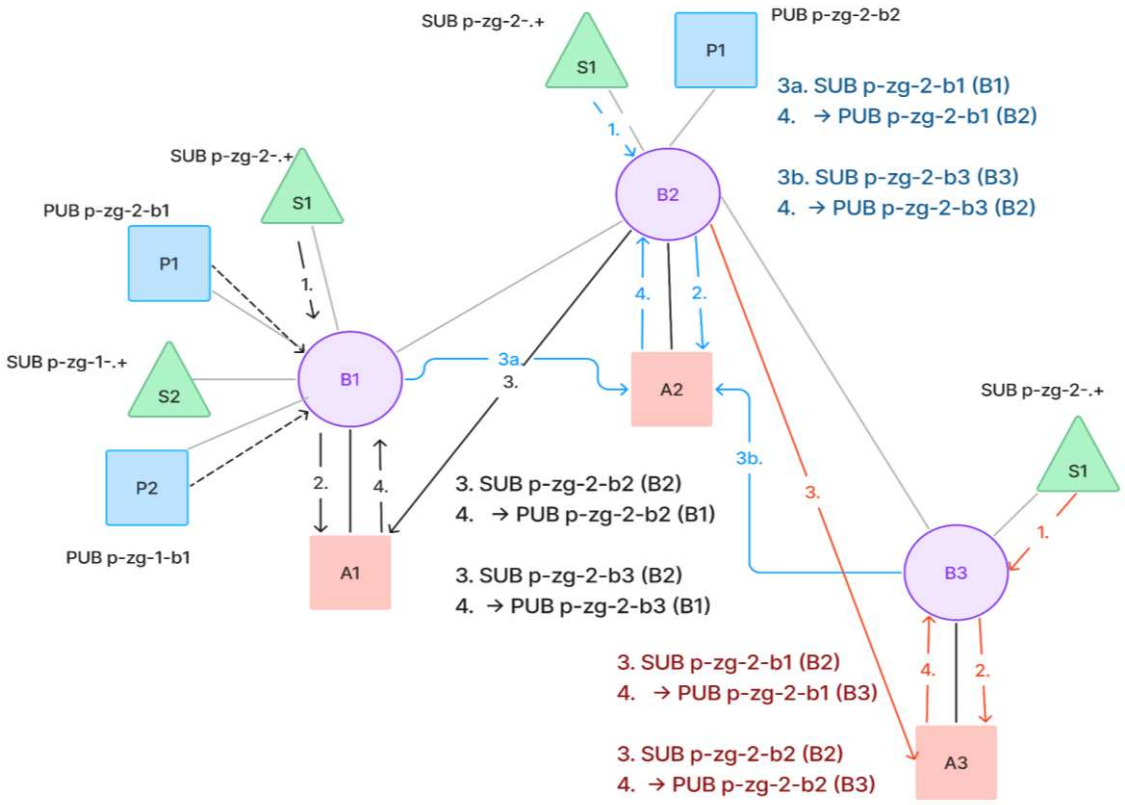
Kada se lokalni klijent pretplati na obavijesti određene regije, on nakon toga šalje poruku o svojoj pretplati na temu *subscriptions* lokalnog posrednika. Ova poruka sadrži informaciju o skupu tema na koje se klijent pretplatio. Ovo vrijedi i za klijente posrednika na rubu mreže i za klijente posrednika u računalnom oblaku. Primjerice, za obavijesti o regiji 2 klijent šalje u poruci regularni izraz *p-zg-2-.+*. Kada poruka o pretplati lokalnog klijenta (Sl. 11) stigne na temu *subscriptions* posrednika na rubu, agentska aplikacija tog posrednika registrira primitak pretplate. Aplikacija se prethodno pretplatila na lokalnu temu *subscriptions* prilikom pokretanja. Nakon registriranog primitka pretplate, aplikacija posrednika na rubu pretplaćuje se (ako već nije pretplaćena) na podskup tema. Ako se radi o posredniku 1, podskup tema će biti sve teme koje zadovoljavaju regularni izraz *p-zg-2-.+*, izuzev one s nastavkom „b1“ jer su tamo spremljene lokalne obavijesti. Podskup tema na koje će se pretplatit će biti *p-zg-2-b2* i *p-zg-2-b3* na posredniku „roditelju“ – posredniku u računalnom oblaku. Posrednik roditelj za posrednike na rubu mreže je posrednik u računalnom oblaku, a za posrednika u računalnom oblaku „djeca“ su svi posrednici na rubu mreže. Pretplaćivanjem na taj skup tema aplikacija dohvaća obavijesti/poruke o željenoj regiji koje

su sa svih posrednika na rubu stigle na posrednik u oblaku, a pohranjene su na zasebne teme za svakog posrednika. Primjerice, u temi p-zg-2-b3 na posredniku 2 dohvaćaju se poruke s posrednika 3 i tako posrednik u računalnom oblaku ažurira stanje za svakog posrednika u sustavu.

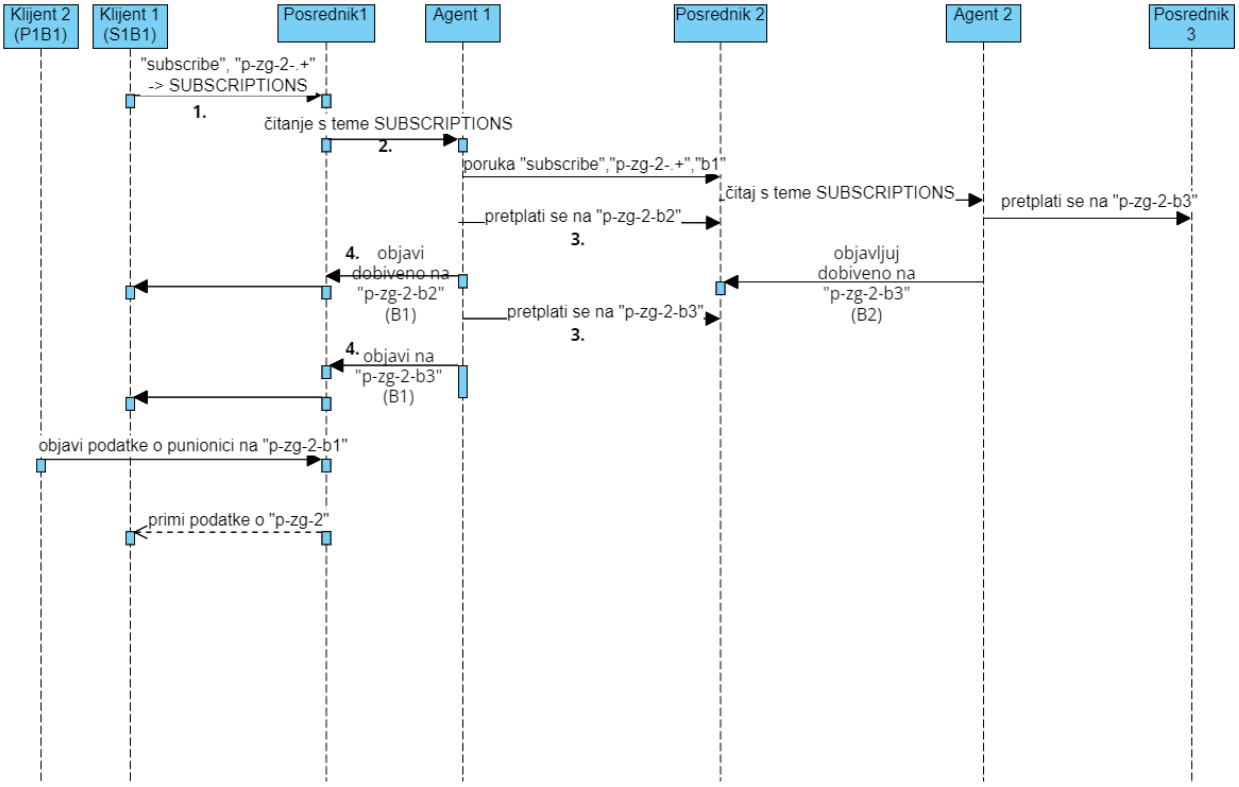
Nakon što prvi klijent na posredniku na rubu definira svoju pretplatu, agentska aplikacija tog posrednika, također šalje poruku s pretplatom i informaciju od kojeg posrednika je došla poruka, posredniku u računalnom oblaku kako bi njegova agentska aplikacija zabilježila informaciju da se posrednik „dijete“ pretplatio. Primjer poruke o pretplati agentske aplikacije namijenjene posredniku u računalnom oblaku prikazuje slika (Sl. 10). Ta informacija je potrebna agentu posrednika u računalnom oblaku kako bi znao za koje sve posrednike na rubu (djecu) treba ažurirati lokalne teme. Kada neki klijent objavi poruku na temu lokalnog posrednika, dobit će ju samo oni posrednici čija se agentska aplikacija pretplatila na tu temu posrednika roditelja/posrednika djece i one će po primitku svake poruke s tih tema ažurirati stanje istoimenih lokalnih tema. Agentske aplikacije posrednika na rubu se pretplaćuju na teme drugih posrednika u sustavu samo ako postoje njihovi zainteresirani lokalni klijenti. Tako smanjuju komunikacijski mrežni promet jer poruke neće pristizati na posrednik ako nema pretplaćenih klijenata.

Razmjena poruka u sustavu prikazana je na slici (Sl. 8) na primjeru pretplaćivanja jednog klijenta i objave poruka drugog klijenta na posredniku 1 na rubu mreže. Skraćenica PUB označava objavu poruke (engl. *publish*), a SUB označava pretplaćivanje (engl. *subscribe*). Na slici se može vidjeti kada se klijent S1 pretplati na teme pomoću regularnog izraza, u koraku 1. (crna strelica) šalje informaciju o pretplati (poruka na slici Sl. 11) na temu *subscriptions* na posredniku 1 (B1). Nakon toga agentska aplikacija A1 pročita tu pretplatu (korak 2., crna strelica), i proslijedi informaciju o pretplati na posrednika 2, uz dodatak s kojeg je posrednika stigla poruka (poruka na slici Sl. 10). Nakon toga se posrednik 2 pretplaćuje na temu *p-zg-2-b3* na posredniku 3 (korak 3b) i po primitku obavijesti objavljuje ih na istoimenu lokalnu temu. Istovremeno se i agent A1 pretplaćuje na teme *p-zg-2-b2* i *p-zg-2-b3* posrednika 2 (korak 3. označen crnom strelicom) i po primitku poruka s te teme objavljuje ih na istoimenu lokalnu temu (korak 4. označen crnom strelicom). Nakon ovih koraka posrednik 1 može pratiti stanje svih poruka u sustavu i pretplaćeni lokalni klijent će dobiti poruke koje objavljuju klijenti na posrednicima 2 i 3. Dodatno se može vidjeti da kada klijent 2 (P1) posrednika 1 pošalje obavijest na temu *p-zg-2-b1*, ona stiže na pretplaćenog klijenta na posredniku 1. Slijed ovih poruka prikazuje i sekvencijski dijagram na slici Sl. 9.

Na slici *Sl. 8* također je prikazano da kada se klijent S1 posrednika 2 pretplati na skup tema *p-zg-2-.+* i pošalje o tome obavijest na temu *subscriptions* posrednika 2 (korak 1. označen plavom strelicom), agentska aplikacija A2 čita tu poruku (korak 2., plava strelica) i pretplaćuje se na teme *p-zg-2-b1* posrednika 1 (korak 3a., plava strelica) i *p-zg-2-b3* posrednika 3 (korak 3b., plava strelica) te po primitku poruka na te teme ažurira istoimene lokalne teme (korak 4. plava strelica). Crvene strelice odnose se na razmjenu poruka kada dođe do pretplate klijenata na posredniku 3 na rubu mreže. Razmjena poruka slijedi logiku kao i za svakog drugog posrednika na rubu. Kada se klijent S1 pretplati, šalje obavijest o tome na lokalnu temu *subscriptions* (korak 1., crvena strelica), agentska aplikacija čita poruku (korak 2., crvena strelica), pretplaćuje se na teme *p-zg-2-b1* i *p-zg-2-b2* na posredniku 2 jer želi obavijesti s druga dva posrednika u sustavu (korak 3., crvena strelica) te po primitku poruka s tih tema objavljuje ih na lokalne istoimene teme (korak 4.).



Sl. 8 Razmjena poruka u sustavu



Sl. 9 Pretplata i objava klijenata posrednika 1 na rubu mreže

```
{
    "action": "subscribe",
    "topic": "p-zg-2-.+",
    "fromBroker": "b1"
}
```

Sl. 10 Poruka o pretplati agentske aplikacije posrednika na rubu mreže

Agentske aplikacije posrednika 1 i 3 stvaraju po jednu dretvu za pretplaćivanje po regiji. Broj stvorenih pretplatnika odgovara broju regija za koje postoje lokalni pretplatnici.

Koraci u kôdu agentske aplikacije posrednika na rubu mreže kod pretplaćivanja klijenta (vrijedi za posrednike 1 i 3, uz zamjenu ID-ja, ID 2 u primjeru zamjenjuje se ID-jem posrednika u računalnom oblaku) prikazani su pseudokodom (*Kôd 1*).

```
1. Glavna petlja:
    - Kreiraj Kafkinog pretplatnika za posrednika 1
    - Pretplati se na temu "subscriptions"
    - Beskonačna petlja:
        - Dohvati novu poruku sa posrednika 1
        - Pozovi funkciju `obradiPoruku` s dobivenom porukom

2. Funkcija `obradiPoruku` (poruka):
    - Ako poruka sadrži akciju "subscribe" i naziv teme:
        - Povećaj brojač pretplaćenih klijenata za teme tog
prefiksa za jedan (npr. teme prefiks p-zg-2)
        - Ažuriraj mapu koja pohranjuje liste tema prema
prefiksu regije (npr. za p-zg-2 pohrani listu [p-zg-2-b2, p-
zg-2-b3])
            - Pokreni funkciju `kreirajNoviPretplatnik` s
prefiksom tema (npr. p-zg-2)

3. Funkcija `kreirajNoviPretplatnik` (prefiksTema):
    - Kreiraj novu dretvu:
        - U dretvi, stvori Kafka pretplatnika za posrednika s
ID-jem 2
        - Dohvati listu tema na koje se treba pretplatiti
klijent na temelju naziva teme (za regiju     2 prefiks je p-
zg-2)
        - Pretplati se na teme iz `listaTema`
        - Beskonačna petlja:
            - Kada stigne nova poruka na posrednik 2:
                - Objavi tu poruku na istoimenu temu na
posredniku 1
```

Kôd 1 – Koraci kod pretplaćivanja klijenta u agentskoj aplikaciji posrednika na rubu mreže

Agentska aplikacija posrednika 2 (posrednik roditelj) stvara po jednu dretvu za pretplaćivanje po regiji te za svaku regiju dodatno onoliko dretvi za pretplaćivanje koliko ima posrednika u sustavu.

Koraci u kôdu agentske aplikacije posrednika u računalnom oblaku kod pretplaćivanja klijenta (ID posrednika je u primjeru 2) prikazani su pseudokodom (*Kôd 2*).

```
1. Glavna petlja:
    - Kreiraj Kafkinog pretplatnika za posrednika 2
    - Pretplati se na temu "subscriptions"
    - Beskonačna petlja:
        - Dohvati novu poruku sa posrednika 2
        - Pozovi funkciju `obradiPoruku` s dobivenom porukom

2. Funkcija `obradiPoruku` (poruka):
    - Ako poruka sadrži akciju "subscribe" i naziv teme:
        - Povećaj brojač klijenata za taj prefiks za jedan
(npr. teme p-zg-2)
       - Pokreni funkciju `obradiTemu` s prefiksom (npr. p-
zg-2)

3. Funkcija `obradiTemu` (prefiksTema):
    - Kreiraj novu dretvu:
- Dohvati listu tema na koje se treba pretplatiti klijent na
temelju naziva prefiksa teme (za regiju 2 prefiks je p-zg-2)
           - za svaku `temu` iz `listaTema` (listaTema=[p-
zg-2-b1, p-zg-2-b3])
                - Kreiraj novu dretvu:
- Stvori Kafkinog pretplatnika za posrednika s ID-jem čije
poruke tema sadrži (određeno sufiksom teme npr. "-b1" u p-zg-
2-b1 rezultira stvaranjem pretplatnika za posrednika 1)
                    - Pretplati se na `temu` (u prvoj
iteraciji primjera to je p-zg-2-b1)
                    - Beskonačna petlja:
                        - Kada stigne nova poruka s
posrednika, objavi
 tu poruku na istoimenu temu na posredniku 2
```

Kôd 2 – Koraci kod pretplaćivanja klijenta u agentskoj aplikaciji posrednika u računalnom oblaku

Lokalni klijenti odjavljuju pretplatu slanjem poruke s odjavom na temu *subscriptions*, a kada agentska aplikacija posrednika na rubu registrira da nema više lokalnih klijenata, šalje poruku za odjavu posredniku roditelju.

3.4. Integracija IoT uređaja

Klijentski kod koji simulira slanje poruka s punionica za električna vozila pisan je u obliku Python skripti zbog jednostavnosti. Ako klijenti podržavaju samo programske jezike niže razine, isti se kod može implementirati u C/C++ uz uvjet da se mogu koristiti Kafkine biblioteke za komunikaciju s posrednikom.

Svaki klijent može objavljivati poruke i pretplaćivati se na postojeće teme. Za to su potrebne informacije o IP adresi i vratima lokalnog posrednika preko kojih se spaja na njega te mu je potreban uzorak tema na koji se želi pretplatiti, odnosno na koje želi objaviti poruke. Klijent mora unaprijed znati kako se zovu teme na posredniku. Vrata koja Kafka koristi za razmjenu su 9092. Ako se primjerice radi o podacima za punionice regije s ID-jem 2, klijent će se pretplatiti na sve teme koje počinju s „p-zg-2“, a to je moguće ostvariti korištenjem regularnog izraza „p-zg-2-.+“ u kôdu.

Kao što je već navedeno svaki klijent koji se želi pretplatiti na informacije o punionicama određene regije (na slici Sl. 8 to su pretplatnici *S1* i *S2* posrednika 1, pretplatnik *S1* posrednika 2 i pretplatnik *S1* posrednika 3) pretplaćuje se na sve teme na lokalnom posredniku vezane za tu regiju. Nakon toga šalje poruku prikazanu na slici Sl. 11 na temu *subscriptions* na lokalnom posredniku. Ta tema služi agentskoj aplikaciji da prati stanje pretplaćenih klijenata i ako postoje pretplaćeni klijenti, agent će se pretplatiti na istoimene teme s posrednika roditelja/posrednika djece.

```
{
    'action': 'subscribe',
    'topic': 'p-zg-2-.+'
}
```

Sl. 11 Poruka o pretplati klijenta

Komunikacija klijenata prikazana na slici *Sl. 8* ostvarena je pokretanjem opisanih skripti koje simuliraju pretplatnike i objavljivače poruka u sustavu. U nazivu skripti „B{*ID_posrednika*}“ označava kojem posredniku klijent pripada.

Pokrenute su sljedeće skripte:

clientS1B1, clientS1B2, clientS1B3, clientS2B1, clientP2B1, clientP1B1, clientP1B2

Objavljivači poruka (P1 i P2 posrednika 1 i P1 posrednika 2) šalju informaciju o ID-ju punionice, njenoj lokaciji, broju dostupnih punjača te njihovom statusu na temu s ID-jem regije u kojoj se ti punjači nalaze. Na primjer, u ovom slučaju za regiju 2 u Zagrebu, poruke se šalju na temu *p-zg-2-b*{*ID* lokalnog posrednika}. Poruke se objavljuju samo na teme koje završavaju s *ID*-jem lokalnog posrednika. Objavljivači P1 i P2 posrednika 1 objavljuju stoga poruke na teme *p-zg-1-b1* i *p-zg-2-b1*. Slike Sl. 12, Sl. 13 i Sl. 14 prikazuju na koje teme se šalju poruke. Klijent 2 posrednika 1 pretplatio se samo na obavijesti za regiju 1 i ne dobiva poruke o regiji 2 (Sl. 15). Ostali klijenti pretplaćeni su na obavijesti regije 2 i dobivaju informacije samo o toj regiji kao što je prikazano na slikama Sl. 16, Sl. 17 i Sl. 18.

Svaki Kafkin pretplatnik treba pripada različitoj grupi pretplatnika ako želi čitati sve poruke od trenutka pretplaćivanja. Stoga svaki pretplatnik postavlja jedinstveni identifikacijski broj grupe (engl. *consumer group ID*). Pri objavljivanju poruka na teme, objavljivači postavljaju ID punionice za ključ poruke kako bi se osigurao redoslijed poruka za tu punionicu jer se poruke s istim ključem šalju na istu particiju. Broj u uglatim zagradama pored naziva teme na slikama Sl. 12, Sl. 13 i Sl. 14 označava na koju particiju teme se šalju poruke.

```
PS D:\aDiplProjekt\dipl2> python .\clientP1B2.py
Sending message: {"station_id": "22P1S01", "location": "Downtown", "chargers_available": 5, "status": "active"}
Sending message: {"station_id": "22P1S01", "location": "Airport", "chargers_available": 7, "status": "active"}
Message delivered to p-zg-2-b2 [0]
Message delivered to p-zg-2-b2 [0]
Sending message: {"station_id": "22P1S01", "location": "Downtown", "chargers_available": 3, "status": "active"}
Sending message: {"station_id": "22P1S01", "location": "Airport", "chargers_available": 6, "status": "active"}
Message delivered to p-zg-2-b2 [0]
Message delivered to p-zg-2-b2 [0]
Sending message: {"station_id": "22P1S01", "location": "Downtown", "chargers_available": 4, "status": "active"}
Sending message: {"station_id": "22P1S01", "location": "Airport", "chargers_available": 6, "status": "active"}
Message delivered to p-zg-2-b2 [0]
Message delivered to p-zg-2-b2 [0]
```

Sl. 12 Poruke objavljivača P1 spojenog na posrednika 2 o regiji 2

```
PS D:\aDiplProjekt\dipl2> python .\clientP1B1.py
Sending message: {"station_id": "11P1S1001", "location": "Downtown", "chargers_available": 5, "status": "active"}
Sending message: {"station_id": "11P1S1002", "location": "Airport", "chargers_available": 7, "status": "active"}
Message delivered to p-zg-2-b1 [1]
Message delivered to p-zg-2-b1 [3]
Sending message: {"station_id": "11P1S1001", "location": "Downtown", "chargers_available": 4, "status": "active"}
Sending message: {"station_id": "11P1S1002", "location": "Airport", "chargers_available": 5, "status": "active"}
Message delivered to p-zg-2-b1 [3]
Message delivered to p-zg-2-b1 [1]
Sending message: {"station_id": "11P1S1001", "location": "Downtown", "chargers_available": 4, "status": "active"}
Sending message: {"station_id": "11P1S1002", "location": "Airport", "chargers_available": 7, "status": "active"}
Message delivered to p-zg-2-b1 [3]
Message delivered to p-zg-2-b1 [1]
```

Sl. 13 Poruke objavljivača P1 spojenog na posrednika 1 o regiji 2

```
PS D:\aDiplProjekt\dipl2> python .\clientP2B1.py
Sending message: {"station_id": "11P2S1001", "location": "Downtown", "chargers_available": 4, "status": "active"}
Sending message: {"station_id": "11P2S1002", "location": "Airport", "chargers_available": 7, "status": "active"}
Message delivered to p-zg-1-b1 [1]
Message delivered to p-zg-1-b1 [3]
Sending message: {"station_id": "11P2S1001", "location": "Downtown", "chargers_available": 4, "status": "active"}
Sending message: {"station_id": "11P2S1002", "location": "Airport", "chargers_available": 7, "status": "active"}
Message delivered to p-zg-1-b1 [1]
Message delivered to p-zg-1-b1 [3]
Sending message: {"station_id": "11P2S1001", "location": "Downtown", "chargers_available": 4, "status": "active"}
Sending message: {"station_id": "11P2S1002", "location": "Airport", "chargers_available": 5, "status": "active"}
Message delivered to p-zg-1-b1 [3]
Message delivered to p-zg-1-b1 [1]
```

Sl. 14 Poruke objavljivača P2 spojenog na posrednika 1 o regiji 1

```
PS D:\aDiplProjekt\dipl2> python .\clientS2B1.py
Received message from p-zg-1-b1: {'station_id': '11P2S1002', 'location': 'Airport', 'chargers_available': 7, 'status': 'active'}
Received message from p-zg-1-b1: {'station_id': '11P2S1001', 'location': 'Downtown', 'chargers_available': 4, 'status': 'active'}
Received message from p-zg-1-b1: {'station_id': '11P2S1002', 'location': 'Airport', 'chargers_available': 7, 'status': 'active'}
Received message from p-zg-1-b1: {'station_id': '11P2S1001', 'location': 'Downtown', 'chargers_available': 4, 'status': 'active'}
Received message from p-zg-1-b1: {'station_id': '11P2S1001', 'location': 'Downtown', 'chargers_available': 4, 'status': 'active'}
Received message from p-zg-1-b1: {'station_id': '11P2S1002', 'location': 'Airport', 'chargers_available': 5, 'status': 'active'}
Received message from p-zg-1-b1: {'station_id': '11P2S1001', 'location': 'Downtown', 'chargers_available': 5, 'status': 'active'}
```

Sl. 15 Pretplatnik S2 posrednika 1 prima obavijesti regije 1

```
PS D:\aDiplProjekt\dipl2> python .\clientS1B3.py

Received message from p-zg-2-b1: {'station_id': '11P1S1002', 'location': 'Airport', 'chargers_available': 7, 'status': 'active'}
Received message from p-zg-2-b1: {'station_id': '11P1S1001', 'location': 'Downtown', 'chargers_available': 5, 'status': 'active'}
Received message from p-zg-2-b2: {'station_id': '22P1S01', 'location': 'Downtown', 'chargers_available': 5, 'status': 'active'}
Received message from p-zg-2-b2: {'station_id': '22P1S01', 'location': 'Airport', 'chargers_available': 7, 'status': 'active'}
Received message from p-zg-2-b1: {'station_id': '11P1S1002', 'location': 'Airport', 'chargers_available': 5, 'status': 'active'}
Received message from p-zg-2-b1: {'station_id': '11P1S1001', 'location': 'Downtown', 'chargers_available': 4, 'status': 'active'}
Received message from p-zg-2-b2: {'station_id': '22P1S01', 'location': 'Downtown', 'chargers_available': 3, 'status': 'active'}
Received message from p-zg-2-b2: {'station_id': '22P1S01', 'location': 'Airport', 'chargers_available': 6, 'status': 'active'}
```

Sl. 16 Pretplatnik S1 posrednika 3 prima obavijesti regije 2

```
PS D:\aDiplProjekt\dipl2> python .\clientS1B2.py
Received message from p-zg-2-b1: {'station_id': '11P1S1001', 'location': 'Downtown', 'chargers_available': 5, 'status': 'active'}
Received message from p-zg-2-b1: {'station_id': '11P1S1002', 'location': 'Airport', 'chargers_available': 7, 'status': 'active'}
Received message from p-zg-2-b2: {'station_id': '22P1S01', 'location': 'Downtown', 'chargers_available': 5, 'status': 'active'}
Received message from p-zg-2-b2: {'station_id': '22P1S01', 'location': 'Airport', 'chargers_available': 7, 'status': 'active'}
Received message from p-zg-2-b1: {'station_id': '11P1S1002', 'location': 'Airport', 'chargers_available': 5, 'status': 'active'}
Received message from p-zg-2-b1: {'station_id': '11P1S1001', 'location': 'Downtown', 'chargers_available': 4, 'status': 'active'}
Received message from p-zg-2-b2: {'station_id': '22P1S01', 'location': 'Downtown', 'chargers_available': 3, 'status': 'active'}
Received message from p-zg-2-b2: {'station_id': '22P1S01', 'location': 'Airport', 'chargers_available': 6, 'status': 'active'}
Received message from p-zg-2-b1: {'station_id': '11P1S1002', 'location': 'Airport', 'chargers_available': 7, 'status': 'active'}
```

Sl. 17 Pretplatnik S1 posrednika 2 prima obavijesti regije 2

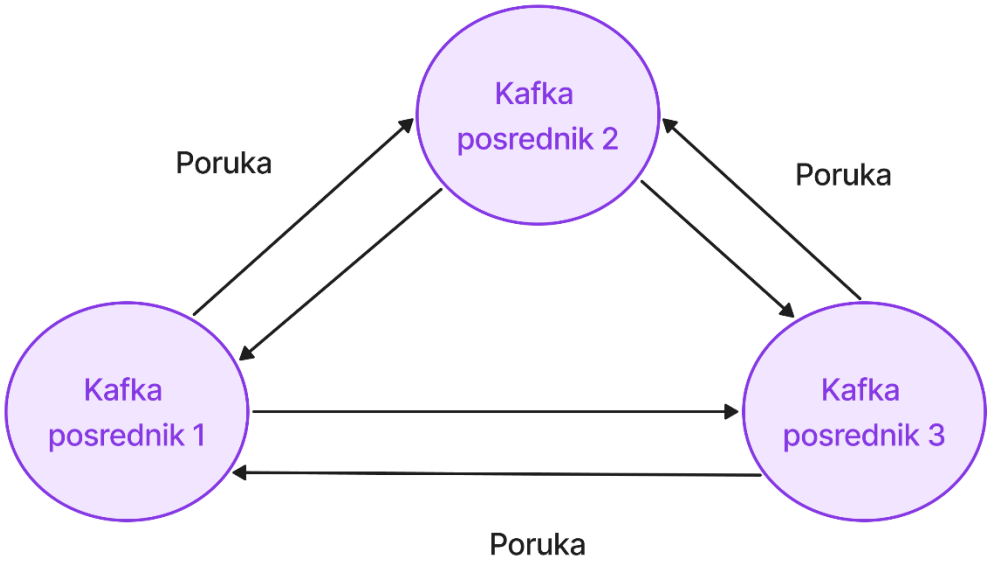
Sl. 18 Pretplatnik S1 posrednika 1 prima obavijesti regije 2

4. Usporedba razvijenog rješenja i grozda posrednika

Implementirano rješenje uspoređuje se s uobičajenom postavom Kafkinih posrednika u grozdu gdje služe za raspodjelu opterećenja i replikaciju. Takav sustav prikazuje slika Sl. 19. U takvom sustavu svi posrednici dijele iste teme jer služe za njihovu replikaciju. Svi posrednici dakle međusobno razmjenjuju sve poruke i svaki je posrednik preslika cijelog sustava jer sadrži sve poruke i sve teme sustava.

Mjerit će se razlika u potrebnim resursima na posrednicima i agentskim aplikacijama. Pratit će se zauzeće memorije i potrošnja CPU-a te mrežni promet.

4.1. Opis testiranja



Sl. 19 Kafkin grozd s tri posrednika

Za testiranje resursnog opterećenja u slučaju korištenja Kafkinog grozda, pokrenuta je Kafka u *KRaft* načinu rada (zamjena za servis *Zookeeper*) na istim virtualnim instancama kao i u

slučaju testiranja vlastitog rješenja. Virtualne instance na kojima su pokrenuti posrednici 1 i 2 imaju procesor s četiri jezgre, 4 GB radne memorije i 10 GB prostora na tvrdom disku, a virtualna instanca posrednika 3 ima 1 GB radne memorije, 18 GB prostora na tvrdom disku i također procesor s četiri jezgre. Dostupni resursi su za oba slučaja testiranja, grozda i samostalnih posrednika jednaka. Pretplatnici i objavljivači su implementirani Python skriptom. Objavljivači šalju poruke veličine 100 B u oba scenarija, 500 poruka/s, što generira promet od 0.045 MB/s.

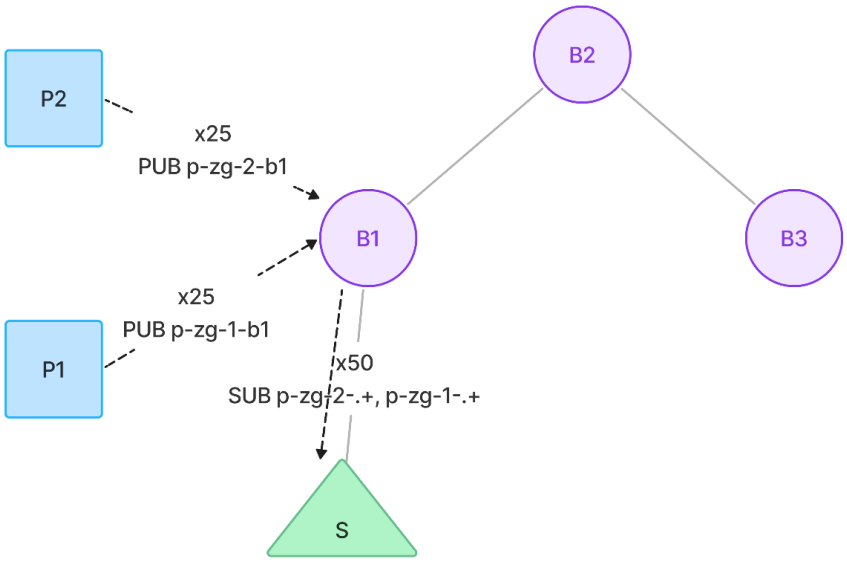
Dva su glavna scenarija za usporedbu dvaju rješenja, a to su sljedeća:

1. Prvi scenarij

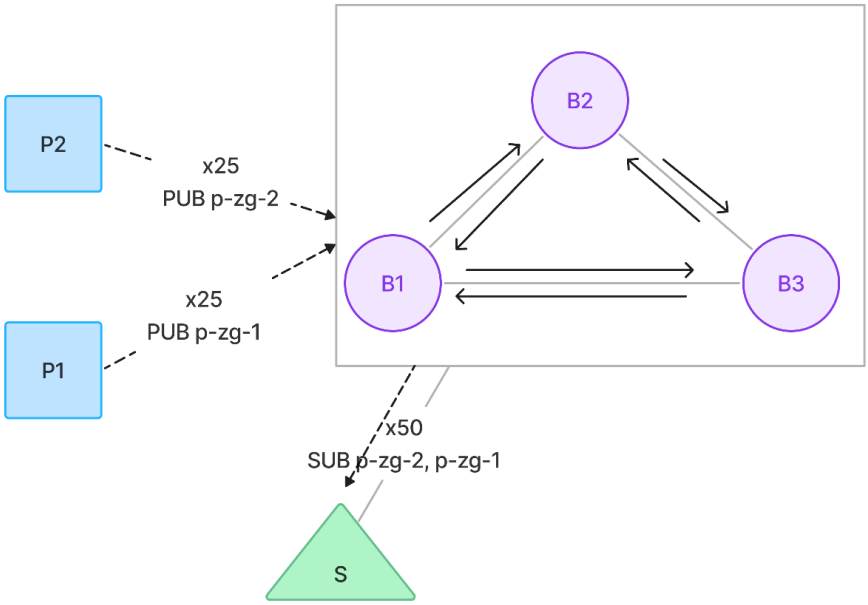
U slučaju grozda, koriste se tri posrednika. Kao što je prikazano na slici Sl. 21, poruke se prosljeđuju ostalim posrednicima iako nemaju pretplaćenih klijenata. Za povezivanje s grozdom objavljivaču su dovoljni podaci barem o jednom posredniku u grozdu, ali za bolju raspodjelu opterećenja navode se IP adrese svih posrednika u grozdu prilikom objavljivanja poruka. Prilikom takvog spajanja na grozd, dohvaćaju se metapodaci o vodećim poslužiteljima particija, nakon čega se šalju poruke vodećem poslužitelju za svaku particiju. Vodeći poslužitelj/posrednik može biti bilo koji od tri posrednika. Vodećeg posrednika za svaku particiju određuje Kafkin interni mehanizam dodjeljivanja particija, a ako dođe do ispada takvog posrednika ili kratkotrajne nedostupnosti, mijenja se vodeći posrednik particije. Zbog toga što je postavljen replikacijski faktor 3, svaka poruka poslana vodećem posredniku replicira se na ostala dva posrednika. Pretplatnici se također spajaju na grozd preko liste posrednika i poruke se zatim dohvaćaju od strane vodećeg posrednika. Za temu Poruke objavljivača i pretplatnika se raspoređuju na sve posrednike, a promet ovisi o tome koji je posrednik vodeći za koju particiju. Kafkini objavljivači i pretplatnici ne znaju unaprijed od kojeg posrednika će dobiti poruke, već o tome odlučuje Kafkin grozd koji prema klijentima djeluje kao „jedan“ posrednik. Zbog toga je razmjena poruka u slučaju grozda jednaka i u prvom i u drugom scenariju testiranja te je dovoljno obaviti samo jednu vrstu mjerenja.

Za razvijeno rješenje to je slučaj s tri samostalna Kafkina posrednika u kojem su aktivni samo klijenti posrednika 1, dakle samo taj posrednik ima objavljivače i pretplatnike. Prvi scenarij će za razvijeno rješenje izgledati kao na slici Sl. 20 (zbog jednostavnosti agentske aplikacije nisu prikazane). U slučaju samostalnih posrednika na koji se spajaju agentske aplikacije, poruke se zadržavaju na posredniku 1 jer nema zainteresiranih klijenata na drugim posrednicima.

U ovom scenariju za oba testna slučaja pokrenuto je 25 objavljivača na teme regije 1 i 25 objavljivača na teme regije 2 te 50 pretplatnika na obje teme.



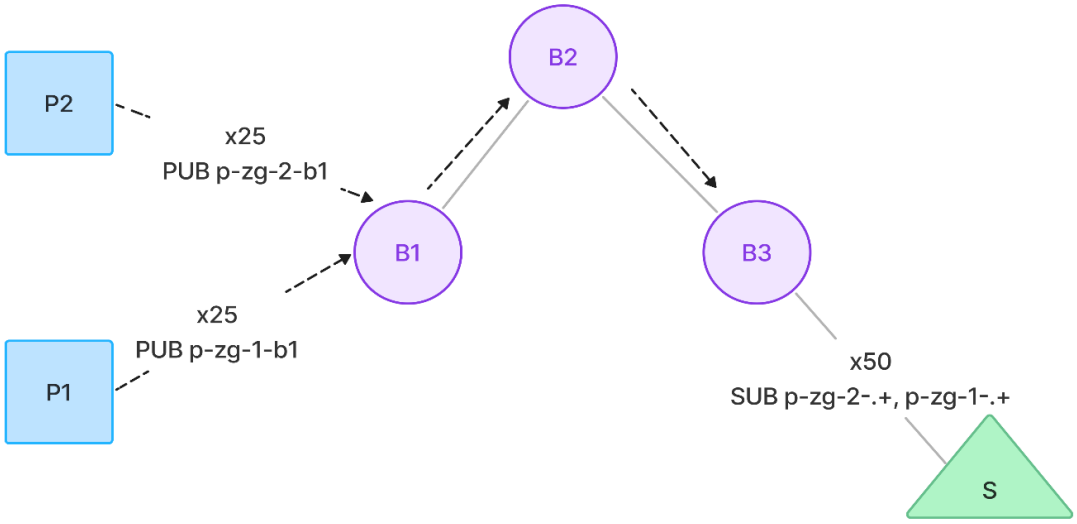
Sl. 20 Scenarij 1 - razvijeno rješenje



Sl. 21 Scenarij 1 i 2 - razmjena poruka u grozdu

2. Drugi scenarij

Testiranje rješenja sa samostalnim posrednicima i agentskim aplikacijama izgleda tako da posrednik 1 ima objavljivače, a posrednik 3 ima pretplatnike. Ovaj scenarij će za implementirano rješenje izgledati kao na slici Sl. 22, a za opciju s grozdom jednako kao i u prethodnom scenariju jer se objavljivači i klijenti spajaju na cijeli grozd, kao na slici Sl. 21. U ovom testnom slučaju bit će također pokrenuto 25 objavljivača na teme regije 1 i 25 objavljivača na teme regije 2 te 50 pretplatnika na obje teme. U oba slučaja poruke se prosljeđuju svim posrednicima u sustavu, a glavna je razlika što poruke u slučaju samostalnih posrednika dolaze od posrednika 1 do posrednika 3 preko posrednika roditelja (posrednika 2), zbog čega nema mrežnog prometa između posrednika djece 1 i 3.



Sl. 22 Scenarij 2 – razvijeno rješenje

4.2. Rezultati ispitivanja

Za mjerenje zauzeća memorije i postotka iskorištenosti CPU-a na virtualnim instancama Ubuntu korišten je alat naredbenog retka *top*, a za praćenje ulaznog i izlaznog mrežnog prometa alat *nload.* Za mjerenje potrošnje lokalnih resursa Javinih aplikacija na računalu s operacijskim sustavom Windows i procesorom s četiri jezgre, korišten je alat Upravitelj zadataka.

U tablici, *IN* označava prosječni ulazni mrežni promet, *OUT* prosječni izlazni mrežni promet, stupac *CPU* predstavlja postotak iskorištenosti procesorskog vremena, a *MEM* zauzeće radne memorije. *IN+OUT* je ukupan mrežni promet izmjeren na agentskim aplikacijama. Povećanje zauzeća radne memorije, označeno u tablicama znakom „+“, odnosi se na usporedbu sa stanjem kada nema razmjene poruka, već je Kafka samo pokrenuta na posrednicima. Zauzeće memorije na virtualnim instancama izraženo je postotkom, a za Javine aplikacije u apsolutnoj vrijednosti.

Tablica 1 Rezultati ispitivanja scenarija 1 – vlastito rješenje

	Scenarij 1			
	CPU	MEM	IN	OUT
Posrednik 1 (B1)	24%	30.6% (povećanje 15%)	6.6 Mb/s	28.7 Mb/s
Posrednik 2 (B2)	0.3%	15.1% (+0.5%)	16.3 kb/s	18.8 kb/s
Posrednik 3 (B3)	0.7%	58.4 % (+1%)	10.57 kb/s	14.55 kb/s
Agent 1	1.8 %	104 MB	0.1Mb/s (IN+OUT)	/
Agent 2	28%	324 MB	0.1 Mb/s (IN+OUT)	/
Agent 3	0.2 %	89 MB	0.1 Mb/s (IN+OUT)	/

Tablica 2 Rezultati ispitivanja scenarija 2 – vlastito rješenje

	Scenarij 2			
	CPU	MEM	IN	OUT
Posrednik 1 (B1)	8.3%	30.9% (+0%)	417 kb/s	406 kb/s
Posrednik 2 (B2)	10.3%	15.8% (+0.7%)	240 kb/s	264 kb/s
Posrednik 3 (B3)	31.3%	67.8% (+7.5%)	1.95 Mb/s	8.62 Mb/s
Agent 1	0.1%	104 MB	0.1 Mb/s (IN + OUT)	/
Agent 2	42%	390 MB	0.5 Mb/s (IN + OUT)	/

Agent 3	3.4%	105 MB	0.5 Mb/s (IN + OUT)	/

Tablica 3 Rezultati ispitivanja scenarija 1 i 2 - grozd

	Scenarij 1 i 2			
	CPU	MEM	IN	OUT
Posrednik 1 (B1)	51.3%	29.3% (+15.6%)	6.7 Mb/s	25.8Mb/s
Posrednik 2 (B2)	49.7%	25.9% (+12.3%)	5.6 Mb/s	18.7 Mb/s
Posrednik 3 (B3)	4.3%	43.0% (+39.4%)	1.01 Mb/s	360 kb/s

4.3. Analiza učinkovitosti filtriranja i usmjeravanja poruka

U prvom scenariju iz mjerenja navedenih u *Tablica 1* i *Tablica 3*, može se primijetiti da je mrežni promet u grozdu najveći između posrednika 1 i 2 iz čega možemo zaključiti da su to vodeći posrednici za particije na koje se klijenti najčešće pretplaćuju i objavljuju poruke u ovom scenariju. Na posrednika 3 poruke se uglavnom prosljeđuju u svrhu replikacije. Razlika između grozda i samostalnih posrednika najviše je izražena u mrežnom prometu na posredniku 2. Posrednik 2 u grozdu ima ulazni promet od 6.7 Mb/s i izlazni promet od 25.8 Mb/s, dok u razvijenom rješenju posrednik 2 ima značajno manji ulazni i izlazni promet - ulazni promet iznosi samo 16.3 kb/s, a izlazni 18.8 kb/s. Ova razlika u prometu pokazuje da se smanjilo opterećenje na tom posredniku u razvijenom rješenju. Promet je u manjoj mjeri povećan i za posrednika 3 u grozdu, u odnosu na posrednika 3 u razvijenom rješenju. Posrednici 2 i 3 u razvijenom rješenju imaju minimalan promet jer se njima ne prosljeđuju obavijesti. Velika je razlika i u iskorištenosti CPU-a između dva slučaja. Grozd koristi značajno više procesorske snage na posrednicima 1 i 2 zbog visoke propusnosti uzrokovanog

dodatnim mrežnim prometom u svrhu replikacije. Kada su posrednici povezani u grozd također je povećano zauzeće radne memorije. Agentske aplikacije ne dodaju veliko mrežno opterećenje jer je zbroj ulaznog i izlaznog prometa na njima 0.1 Mb/s. Veći zahtjev dodaju na radnu memoriju i procesor za slučaj agentske aplikacije posrednika 2.

U drugom scenariju, iz mjerenja navedenim u *Tablica 2* i *Tablica 3*, može se uočiti da je razlika između grozda i samostalnih posrednika najviše izražena u generiranom mrežnom prometu između posrednika. U slučaju samostalnih posrednika veći je promet samo na jednom posredniku (posredniku 3), dok je u grozdu generiran značajniji mrežni promet na dvama posrednicima. Posrednik 1 u grozdu ima ukupni promet (zbroj ulaznog i izlaznog) od 32.5 Mb/s, posrednik 2 ima ukupni promet od 24.3 Mb/s, a posrednik 3 znatno manji promet od 1.37 Mb/s. Ukupni promet na posrednicima 1 i 2 u razvijenom rješenju je minimalan, a na posredniku 3 iznosi 10.57 Mb/s. Ako gledamo ukupni promet u sustavu on je znatno veći u slučaju grozda. Veća je razlika i u iskorištenosti CPU-a, grozd koristi značajno više procesorske snage na posrednicima 1 i 2 (približno 50 % iskorištenosti CPU-a na oba posrednika). U razvijenom rješenju značajnije iskorištenje procesora prisutno je na posredniku 3 (oko 30%). Povećano je i zauzeće radne memorije u grozdu. Agentske aplikacije dvaju posrednika dodaju mrežno opterećenje od 0.5 Mb/s što nije značajna vrijednost. Veći zahtjev dodaju na radnu memoriju i procesor za slučaj agentske aplikacije posrednika 2.

Iz opisanih ispitivanja može se zaključiti da iako se u slučaju samostalnih posrednika za funkcionalnost filtriranja dodatno pokreću Javine aplikacije, one zahtijevaju znatno manje resursa kada se usporedi sa slučajem komunikacije u grozdu, za isti testni slučaj. Rješenje s agentskim aplikacijama dodaje najviše opterećenja na procesor, ali ono je primjetno samo za agentsku aplikaciju posrednika u računalnom oblaku (posrednik 2, u tablici - B2). Ta bi se aplikacija trebala pokretati na poslužitelju s jačim procesorom. Dodatno je opterećenje i mreže za komunikaciju između agentskih aplikacija i posrednika, ali ono nije značajno. Slučaj korištenja grozda u oba scenarija zahtijeva više resursa nego razvijeno rješenje. Generira se znatno veći mrežni promet, razlika dolazi do 20 Mb/s. Također znatno je veći zahtjev za procesorskom snagom u slučaju grozda.

Dodatne značajke oba rješenja koje treba uzeti u obzir pri odlučivanju koje rješenje primijeniti u kontekstu računarstva na rubu su sljedeće:

Grozd

- Prednosti:
  - Osigurava dostupnost u slučaju ispada posrednika
  - Smanjuje gubitak poruka
- Nedostaci
  - Zahtijeva više prostora na disku, radne memorije i veću iskorištenost CPU-a
  - Povećava lokalni mrežni promet
  - Može dodati kašnjenje u isporuci poruka zbog replikacije

Razvijeno rješenje

- Prednosti:
  - Bez obzira na dodano resursno opterećenje koje je potrebno zbog agentskih aplikacija, može se reći da je ukupno opterećenje manje za ovaj slučaj za razliku od grozda
  - Smanjen mrežni promet
  - Smanjeno mrežno kašnjenje jer se poruke šalju na onaj posrednik koji je najbliži u mreži
  - Agentske aplikacije omogućuju implementaciju prilagođene logike usmjeravanja i filtriranja za specifične slučajeve upotrebe na rubu mreže i omogućuje veću kontrolu nad tim kada i kako se podaci dijele između čvorova
- Nedostaci:
  - Nedostatak otpornosti na greške jer svi uređaji spojeni na jedan posrednik ovise samo o njemu (ALI u računarstvu na rubu ne zahtijeva se uvijek visoka dostupnost, ako je to slučaj onda se mogu koristiti samostalni posrednici i nema potrebe za grozdom posrednika)
  - Nije opcija za sustave koji zahtijevaju visoku dostupnost Kafkinih posrednika
  - Skaliranje sustava znači da se dodavanjem novog posrednika dodaje i agentska aplikacija čiju funkcionalnosti treba u nekim slučajevima prilagoditi

Zaključak

Apache Kafka pronalazi sve češću primjenu u okruženju računarstvu na rubu mreže gdje služi za obradu podataka s mnogobrojnih IoT-uređaja blizu njihovog izvora. Kako Kafka nije prvenstveno namijenjena za fleksibilno filtriranje poruka, za mnoge slučajeve filtriranja poruka, Kafkine nativne komponente nisu najbolje rješenje jer ne podržavaju specifične slučajeve filtriranja ili imaju druga ograničenja. Postavlja se pitanje koje su druge mogućnosti za fleksibilno usmjeravanja i filtriranje poruka u sustavima koji koriste Kafku. U radu je razvijeno rješenje koje unaprjeđuje komunikaciju u okruženju računarstva na rubu dodavanjem agentskih aplikacija samostalnim Kafka posrednicima koji se nalaze na različitim slojevima u okruženju računarstva na rubu. Razvijeno rješenje omogućilo je fleksibilno usmjeravanje i filtriranje poruka na temelju pretplata. Provedena ispitivanja pokazala su da je predloženo rješenje dovelo do značajnog smanjenja mrežnog prometa i iskorištenosti procesora na posrednicima, posebno pri velikom opterećenju porukama kada se usporedi s klasičnim Kafkinim grozdom bez ikakve logike filtriranja. Ovime je pokazano da se fleksibilno filtriranje može uspješno ostvariti korištenjem prilagođenih agentskih aplikacija u svrhu optimizacije komunikacije i smanjenja resursnog opterećenja u raspodijeljenim sustavima Interneta stvari, osobito kada je potrebno dodati složeniju logiku filtriranja poruka.

Literatura

[1] Amazon, *What is Edge Computing*, (2024.). Poveznica: https://aws.amazon.com/what-is/edge-computing/; pristupljeno 5. ožujak 2024.

[2] Rausch, Dustdat, Ranjan, *Osmotic Message-Oriented Middleware for the Internet of Things*, IEEE Cloud Computing (2018.)

[3] LinkedIn, *Message Oriented Middleware Market Size, Potential*, (2024, ožujak). Poveznica: https://www.linkedin.com/pulse/message-oriented-middleware-market-size-potential-l9xgf/; pristupljeno 1. travanj 2024.

[4] Jeyaraman, *Kafka in Edge Computing*, (2023. rujan). Poveznica: https://www.linkedin.com/pulse/kafka-edge-computing-brindha-jeyaraman; pristupljeno 20. ožujak 2024.

[5] K. Waehner, *Use Cases and Architectures for Kafka at the Edge*, (2020. listopad). Poveznica: https://www.kai-waehner.de/blog/2020/10/14/use-cases-architectures-apache-kafka-edge-computing-industrial-iot-retail-store-cell-tower-train-factory/ pristupljeno 20. ožujak 2024.

[6] J. Myers, *Hybrid Data Collection from the IoT Edge with MQTT and Kafka*, (2022. prosinac). Poveznica: https://thenewstack.io/hybrid-data-collection-from-the-iot-edge-with-mqtt-and-kafka/; pristupljeno 20. ožujak 2024.

[7] J. Rao, *How to Choose the Number of Topics/Partitions in a Kafka Cluster*, (2015. ožujak). Poveznica: https://www.confluent.io/blog/how-choose-number-topics-partitions-kafka-cluster/; pristupljeno 29. ožujak 2024.

[8] Podnar Žarko, Pripužić, Lovrek, Kušek, *Raspodijeljeni sustavi*, inačica v.1.3 (2013.)

[9] Maison, Stanley, *Kafka Connect*, O'Reilly Media (2023.)

[10] Gwen Shapira, Todd Palino, Rajini Sivaram and Krit Petty, *Kafka: The Definitive Guide, 2nd Edition,* O'Reilly Media (2020.)

[11] Kafka Streams, Poveznica: https://kafka.apache.org/documentation/streams/

[12] ksqlDB, Poveznica: https://ksqldb.io/

[13] Kafka Connect, Poveznica: https://docs.confluent.io/platform/current/connect/index.html

Sažetak

Međuoprema za komunikaciju porukama u okolinama računarstva na rubu

U raspodijeljenom okruženju Interneta stvari obrada podataka seli se na rub mreže, bliže izvoru podataka. Uređaji na rubu mreže, iako resursno ograničeni u odnosu na računalni oblak, donose mnoge prednosti poput smanjenog mrežnog prometa i kraćeg vremena obrade podataka s IoT-uređaja. Kao međuoprema za komunikaciju porukama u okruženju računarstva na rubu sve se češće koristi sustav Apache Kafka. U radu je razvijeno rješenje koje dodaje mogućnosti filtriranja i usmjeravanja poruka između IoT-uređaja koji objavljuju podatke i mogu se na njih pretplaćivati te samostalnih Kafkinih posrednika korištenjem prilagođenih agentskih aplikacija na svakom posredniku. Aplikacije služe kao posrednici u komunikaciji između Kafkinih posrednika i imaju ulogu i potrošača i proizvođača u sustavu. Provedena su također mjerenja korištenih resursa razvijenog sustava za razliku od klasičnog sustava koji nema logiku filtriranja podataka. Rezultati su pokazali kako je dodavanje agentskih aplikacija dodalo zanemarivu potrošnju resursa u usporedbi sa značajno većim zahtjevima u slučaju korištenja grozda bez logike filtriranja. Predloženo rješenje značajno je smanjilo mrežni promet i iskorištenost procesora na Kafkinim posrednicima, što se najviše primjećuje pri velikom opterećenju porukama.

Ključne riječi:

međuoprema za komunikaciju porukama, računarstvo na rubu, Internet stvari, Apache Kafka

Summary

Message-Oriented Middleware in Edge Computing Environments

In a distributed environment of Internet of Things, message processing and storage are moving closer to the data source, at the "edge". Edge computing devices, though resource-constrained compared to the cloud, offer several advantages such as reduced network traffic and decreased IoT data processing time. Apache Kafka, as a message-oriented middleware in edge computing environments, is being increasingly used. This thesis presents a solution that demonstrates message filtering and routing capabilities between IoT devices, which publish and subscribe to data, and standalone Kafka brokers, using custom agent applications on each broker. These applications act as middleware facilitating communication between brokers, while functioning as both Kafka consumers and producers. Measurements were conducted to investigate resource usage on the developed system and compare it to a conventional Kafka system that is lacking data filtering logic. The results show that adding agent applications caused minimal resource consumption compared to the significantly higher demands of using a cluster without filtering logic. The proposed solution notably reduced network traffic and CPU usage on Kafka brokers, especially under heavy message loads.

Key words:

message-oriented middleware, edge computing, Internet of Things, Apache Kafka

Skraćenice

IoT	*Internet of Things*	Internet stvari
MOM	*message-oriented middleware*	međuoprema za komunikaciju porukama
CPU	*central processing unit*	procesor

Privitak

Upute za postavljanje Kafkinih posrednika

Apache Kafka postavljena je na računalnim resursima Zavoda za telekomunikaciju. Za postavljanje tri samostalna Kafkina posrednika koja se koriste u razvijenom rješenju, potrebne su tri virtualne instance s operacijskim sustavom Ubuntu. Sve virtualne instance imaju procesor s četiri jezgre, dvije instance imaju 4 GB radne memorije i 10 GB prostora, a treća instanca ima 1 GB radne memorije i 18 GB prostora na tvrdom disku. Na virtualne instance spaja se pomoću alata *Putty* za udaljeni način rada: punjaci-2 (161.53.19.19:55161), punjaci-3 (161.53.19.19:55162) i punjaci-6 (161.53.19.19:55998). Na svakoj instanci treba instalirati Javu verzije 8 ili više. Potrebno je instalirati i postaviti Kafku preuzimanjem Kafka binarnog paketa sa službene stranice Apache Kafke (https://kafka.apache.org/downloads) i raspakirati u neki direktorij (*/usr/local/kafka*). U konfiguracijskoj datoteci *server.properties* koja se nalazi u direktoriju */usr/local/kafka/config*, potrebno je postaviti iduće vrijednosti (primjer za punjaci-2, posrednik 1):

```
listeners=PLAINTEXT://0.0.0.0:9092
advertised.listeners=PLAINTEXT://161.53.19.19:55961
```

Za punjaci-3 (posrednik 2):

```
advertised.listeners=PLAINTEXT://161.53.19.19:55962
```

Za punjci-6 (posrednik 3):

```
advertised.listeners=PLAINTEXT://161.53.19.19:55998
```

Prije pokretanja Kafke na svakoj instance prvo se pokreće servis Zookeepr naredbom:

```
/usr/local/kafka$ sudo bin/zookeeper-server-start.sh
config/zookeeper.properties
```

Nakon toga može se pokrenuti Kafka naredbom:

```
/usr/local/kafka$ sudo bin/kafka-server-start.sh
config/server.properties
```

Upute za korištenje programske podrške

Za pokretanje Javinih aplikacija potrebno je imati instaliran alat Maven. Za pokretanje aplikacija potrebno je pozicionirati se u direktorij u kojem se nalazi datoteka *pom.xml*. Zatim je potrebno iz naredbenog retka pokrenuti naredbu koja će prevesti kod i stvoriti direktorij *target/classes* :

```
mvn compile
```

Svaka aplikacija se zatim pokreće zasebno naredbama:

```
java -cp target/classes agents.KafkaAgent1
java -cp target/classes agents.KafkaAgent2
java -cp target/classes agents.KafkaAgent3
```

Za pokretanje Python skripti koje simuliraju Kafkine klijente potrebno je imati instaliran Python (verziju 3.x). Potrebno je instalirati biblioteku *confluent_kafka* naredbom:

```
pip install confluent-kafka
```

Nakon pozicioniranja u direktorij u kojem se nalaze skripte (*.py), pokreću se naredbom (*client_script.py* zamijeniti nazivom skripte koja se želi pokrenuti):

```
python client_script.py
python clientS1B1.py
```