SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6178

Filtriranje geoprostornog toka podataka korištenjem platforme Apache Spark

Darko Britvec

Zagreb, lipanj 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 13. ožujka 2019.

ZAVRŠNI ZADATAK br. 6178

Pristupnik: **Darko Britvec (0036498411)**
Studij: Računarstvo
Modul: Programsko inženjerstvo i informacijski sustavi

Zadatak: **Filtriranje geoprostornog toka podataka korištenjem platforme Apache Spark**

Opis zadatka:

Vaš zadatak je osmisliti, izvesti u programskim jezicima Scala i/ili Java te testirati aplikaciju za računalni grozd za filtriranje geoprostornog toka podataka. Elementi toka podataka i trajni upiti sadrže geoprostorni objekt (npr. točku, liniju, poligon). U toku podataka se nalaze elementi koji se sastoje od geoprostornog objekta koji predstavlja lokaciju te dodatnog sadržaja. Trajni upit se sastoji od geoprostornog objekta koji predstavlja geografsko područje interesa te dodatnih uvjeta na sadržaj elementa na osnovu kojih se vrši kontinuirana usporedba s nadolazećim elementima toka podataka.
Prilikom izrade sustava iskoristite Javine programske knjižnice JTS (Java Topology Suite), GeoTools i GeoSpark. Istražite geoprostorne indekse koji su podržani u navedenim Javinim programskim knjižnicama, opišite njihove karakteristike te implementirajte raspodijeljeni indeks u računalnom grozdu korištenjem platforme Apache Spark.
Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 14. lipnja 2019.

Mentor:
Izv. prof. dr. sc. Krešimir Pripužić

Djelovođa:
Doc. dr. sc. Mirjana Domazet-Lošo

Predsjednik odbora za završni rad modula:
Izv. prof. dr. sc. Ivica Botički

Zahvaljujem svom mentoru, izv. prof. dr. sc. Krešimiru Pripužiću, na ukazanoj pomoći prilikom izrade ovog završnog rada.

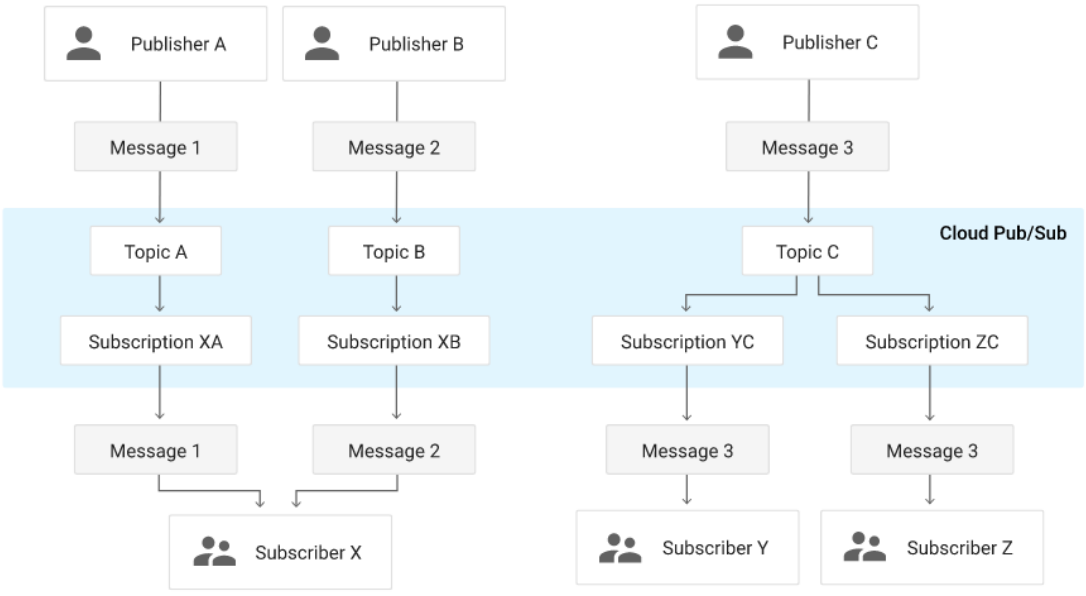
Posebne zahvale i Zavodu za telekomunikacije koji je ustupio svoje resurse za eksperimentalnu evaluaciju programskog rješenja.

Sadržaj

Uvod

Filtriranje geoprostornih podataka čest je zadatak mnogih informacijskih sustava. Razlog tome najčešće je pronalazak odgovarajuće skupine entiteta (pretplatnika) koje bi mogla zanimati objava vezana za određenu informaciju.

Informacijski sustavi koji se bave tom problematikom najčešće koriste arhitekturu "objavi-pretplati" (engl. *publish subscribe pattern*). Prednost korištenja takve arhitekture je izbjegavanje redundantnog prometa te samim time povećanje skalabilnosti. Takvi sustavi često su opsežni te zahtijevaju poslužiteljska računala velikih performansi, zbog toga što je broj pretplatnika i pretplata velik. Primjer jednog takvog sustava je Googleov sustav poruka pod nazivom Cloud Pub/Sub Service koji je poznat po svojoj pouzdanosti i skalabilnosti. Podaci objavljeni na službenim Googleovim stranicama[1] govore da taj sustav u prosjeku šalje preko 500 milijuna poruka u sekundi te mnogi Googleovi proizvodi kao što su Gmail, Google Ads i Google Search ovise o njemu.



Slika 1 *Google Pub/Sub sustav - princip rada*

[1] „Cloud Pub/Sub: A Google-Scale Messaging Service“, https://cloud.google.com/pubsub/architecture

Kada govorimo o podacima čiji je broj reda veličine nekoliko stotina tisuća, pa sve do stotina milijuna podataka u sekundi, možemo pričati o izazovima domene obrade velike količine podataka (engl. *big data*). Zbog takvih zahtjeva treba pribjeći pametnijem i skalabilnijem načinu obrade od klasičnog centraliziranog načina, a to je raspodijeljena obrada podataka.

Cilj ovog rada je implementirati jedan dio raspodijeljenog sustava zaduženog za filtriranje geoprostornog toka podataka koji predstavlja dolazne objave vezane za određeni lokalitet - koordinatu, liniju ili poligon. Ulazni tok koristi format GeoJSON koji sadrži prostornu informaciju u standardnom formatu te dodatne informacije koje se prenose (npr. identifikator objave). Mehanizam filtriranja koristi platformu Apache Spark koji omogućava razvoj raspodijeljenih programa. Dolazne objave grupira se s pripadnim pretplatama predstavljenim poligonima u prostoru. Optimalno pretraživanje skupa pretplata po geoprostornim informacijama obavlja se korištenjem prikladnih struktura podataka (npr. *Quad Tree, R-Tree*, itd.). Izlaz sustava novi je tok podataka koji sadrži identifikator objave i njoj pripadne pretplate. Za tok podataka u implementaciji korištena je platforma Apache Kafka koja se također temelji na arhitekturi "objavi-pretplati". Rezultat rada programski je modul koji bi se mogao iskoristiti unutar bilo kojeg većeg programskog sustava koji zahtjeva sličnu funkcionalnost.

1. Raspodijeljeni sustavi

Raspodijeljeni sustav obilježava pojam raspodijeljenosti ili distribuiranosti. Riječ je o sustavu koji nije cjelovit – monolitan, već je sastavljen od više međusobno povezanih, fizički i logički raspodijeljenih dijelova koji zajedno ostvaruju zadaću kojoj su namijenjeni [2].

1.1. Obilježja raspodijeljenih sustava

Raspodijeljeni sustav najčešće je sastavljen od mreže autonomnih računala spojenih pomoću distribucijskog posrednika koji pomaže u razmjeni dijeljenih resursa. Pogled iz perspektive vanjskog klijenta na takav sustav najčešće je jednak kao i prema običnom centraliziranom sustavu zato što se uslugama pristupa preko definiranog sučelja. Komponente raspodijeljenog sustava su konkurentne te koriste zajedničke resurse istovremeno, no po prirodi su nezavisne jedna o drugoj. Iz tog razloga, sustav ne zahtjeva apsolutnu sinkronizaciju među komponentama, to jest komponente nemaju zajedničko stanje niti zajednički vremenski takt. Također, važno je naglasiti da takav sustav ima veću otpornost na kvarove te dobar odnos cijene i performansi. Da bi se takve performanse ostvarile, potreban je mehanizam koji će odabrati koja će komponenta izvršiti zadatak ovisno o lokaciji te opterećenosti komponenti. Također, ako su komponente heterogene (neistovrsne), potrebno je definirati jedinstveni način komunikacije neovisan o izvedbi komponente (primjerice operacijskog sustava) [1].

Potrebe za korištenjem raspodijeljenog sustava su:

- prostorna raspodijeljenost korisnika, podataka, informacija i sredstava,
- veličina sustava,
- olakšanje nadogradnje sustava,
- raspodjela opterećenosti sustava,
- cijena sklopovlja,
- troškovi rada sustava.

Osnovni zahtjevi koji se postavljaju pri oblikovanju raspodijeljenih sustava su:

- Otvorenost - izvedba sustava čija je usluga izvedena sukladno normiranim pravilima (standardima).
- Transparentnost - sposobnost prikrivanja odabranih značajki raspodijeljenog sustava (transparentnost pristupa, lokacijska, migracijska, relokacijska, replikacijska, konkurencijska transparentnost te transparentnost na kvar).
- Skalabilnost - sposobnost razmjerne prilagodbe sustava broju korisnika i njihovoj rasprostranjenosti. Pojam također podrazumijeva sposobnost uvođenja novih funkcija u postojeći sustav i novih načina upravljanja u jednoj ili više administrativnih domena.
- Kvaliteta usluge - zajednički učinak performansi koji određuje stupanj zadovoljstva korisnika.

Izazovi koji se javljaju prilikom izgradnje raspodijeljenog sustava najčešće su sigurnost na granici sustava i okoliša, otpornost na greške u slučaju nepouzdanih komponenti te koordinacija prilikom zajedničkog korištenja resursa [1].

1.2. Obilježja velikih podataka

Veliki podaci su podaci čiji je sadržaj raznolik te dolazi u velikom obujmu i još većom brzinom. U odnosu na uobičajene skupove podataka (šifrarnici, strukturirane tablice, itd.), velike podatke nije moguće obrađivati korištenjem klasične obrade podataka. Svrha obrade velikih podataka često je rješavanje kompleksnih poslovnih i financijskih problema koji se bez toga ne mogu riješiti [3].

Osnovna obilježja velikih podataka (engl. *the three Vs*) su:

- Obujam (engl*. velocity*) - najčešće izražen u gigabajtima pa sve do nekoliko stotina petabajta.
- Raznolikost (engl. *variety*) - nestrukturirani podaci sadrže razne tipove podataka te formate koje je potrebno prilagoditi za obradu.
- Brzina (engl. *velocity*) - upravljivost u stvarnom vremenu.

1.3. Arhitekture raspodijeljenih sustava

Arhitektura raspodijeljenog sustava opisuje njegove sastavne dijelove, njihova međudjelovanja te funkcionalnosti za koje su namijenjeni [1].

Kada se govori o arhitekturi bilo kojeg sustava, razlikuju se dva pojma:

1.) Programska arhitektura – Logička struktura sustava, njegove programske komponente, njihova organizacija i međudjelovanje.
2.) Sustavska arhitektura – Smještaj programskih komponenti na jedno ili više računala.

1.3.1. Slojeviti arhitekturni model

Razlamanje velikog skupa funkcionalnosti na slojeve koji međusobno komuniciraju putem čvrsto definiranog sučelja česti je pristup modeliranja programskog sustava. Zbog toga je slojeviti arhitekturni model jedan od najraširenijih kada govorimo o raspodijeljenim sustavima. Najpoznatija primjena slojevite arhitekture zasigurno je referentni TCP/IP model na kojem se zasniva današnji Internet [1].

TCP/IP model sastoji se od 4 sloja:

1) Sloj podatkovne poveznice i fizički sloj – odgovoran za fizički pristup mreži.
2) Mrežni (internetski) sloj – dodatni protokoli za usmjeravanje, kontrolu komunikacije i komunikaciju u skupini.
3) Transportni sloj – prijenos bez pogreške, isporuka informacije u nepromijenjenom redoslijedu.
4) Aplikacijski sloj – različite usluge i primjene (npr. HTTP, SMTP, FTP…).

Funkcionalnost raspodijeljenog sustava najčešće određuje najviši (aplikacijski) sloj, dok niži slojevi pružaju uslugu višim slojevima te se pomoću njih ostvaruje povezivanje, razmjena podataka i suradnja. Točka međudjelovanja između dva susjedna sloja naziva se točkom pristupa (engl. *SAP – Service Access Point*), dok se međudjelovanje opisuje uslužnim primitivima (engl. *SP – Service Primitive*) [1].

Povezanost dva procesa u aplikacijskom sloju naziva se asocijacija (združivanje), kako bi se naglasila slaba povezanost naspram čvrste veze (konekcije). Razlog tome

je što primatelj prije razmjene ne mora znati tko je pošiljatelj te je raspoloživost i pouzdanost asocijacije proizvoljna [1].

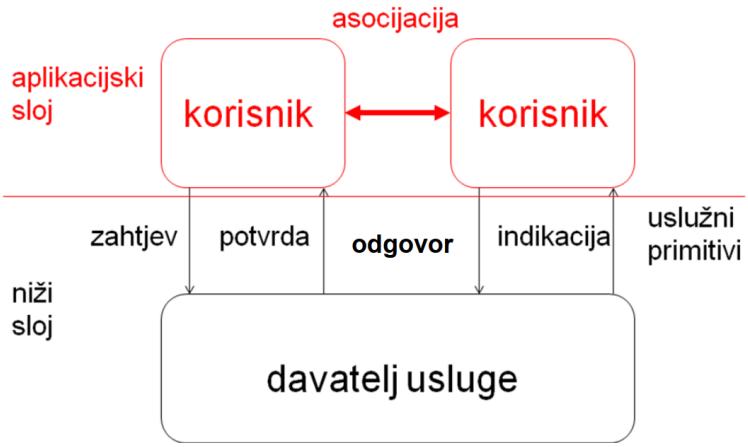
Prilikom komunikacije među slojevima najčešće korišteni uslužni primitivi su:

- Zahtjev – upućuje ga korisnik kako bi od davatelja usluga zatražio izvršenje funkcije.
- Indikacija – upućuje ga davatelj kako bi od korisnika zatražio izvršenje funkcije.
- Odziv – upućuje ga korisnik kao odgovor na indikaciju.
- Potvrda – upućuje ga davatelj kao odgovor na zahtjev.

Redoslijed kojim se primitivi izvršavaju je sljedeći (slika 2):

1) Korisnik pokreće asocijaciju postavlja nižem sloju **zahtjev** za spajanje na drugi proces.
2) Niži sloj prosljeđuje zahtjev pomoću **indikacije** kojom traži spajanje na drugi proces.
3) Drugi proces šalje **odziv** na indikaciju kako bi niži sloj izvijestio pokretača spajanja.
4) Niži sloj šalje **potvrdu** pokretaču spajanja da je asocijacija uspostavljena te razmjena podataka može započeti.

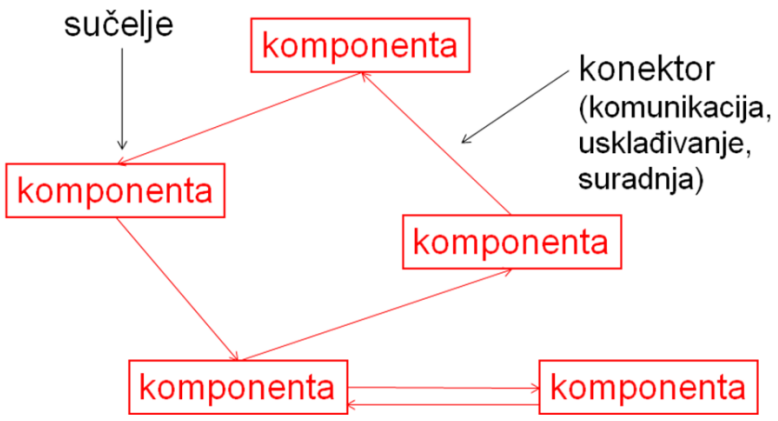
Prednost ove arhitekture je razdvajanje funkcionalnosti te visoka razina apstrakcije između različitih slojeva u sustavu [1].



Slika 2 *Skica asocijacije u slojevitoj arhitekturi (Podnar Žarko, Pripužić, Lovrek, Kušek, 2013.) [1]*

1.3.2. Objektni arhitekturni model

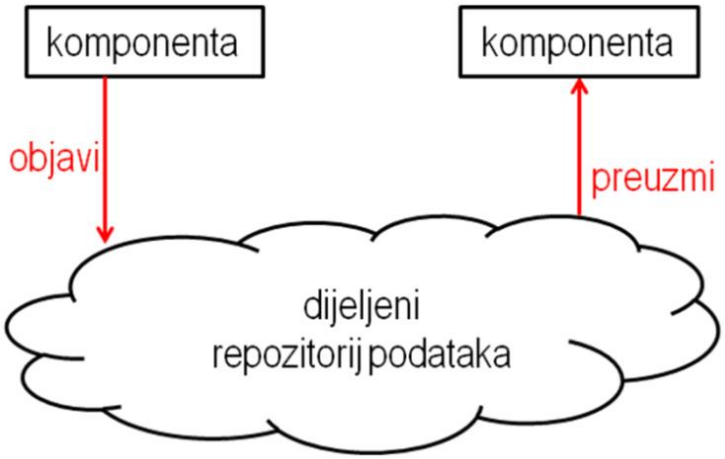
Za razliku od slojevitog modela čiji je primarni cilj funkcijska podjela sustava, kod objektnog modela su u fokusu gradbene jedinice od kojih se sustav sastoji. Objekti su osnovne gradbene jedinice koje enkapsuliraju određenu (jedinstvenu) odgovornost. Komponente su skup objekata koji zajedno surađuju u izvršenju neke funkcionalnosti. Objekti i komponente imaju čvrsto definirana sučelja te se povezuju konektorima kako bi se omogućila sinkronizacija i suradnja [1].



Slika 3 *Skica objektnog arhitekturnog modela (Podnar Žarko, Pripužić, Lovrek, Kušek, 2013.) [1]*

1.3.3. Arhitektura zasnovana na podacima

Ključni dio sustava zasnovanog na podacima je dijeljeni repozitorij podataka putem kojega procesi komuniciraju. Komponenta koja raspolaže podatkom upisuje ga (objavljuje) u repozitorij podataka koji omogućuje čitanje (preuzimanje) podataka drugim komponentama [1].

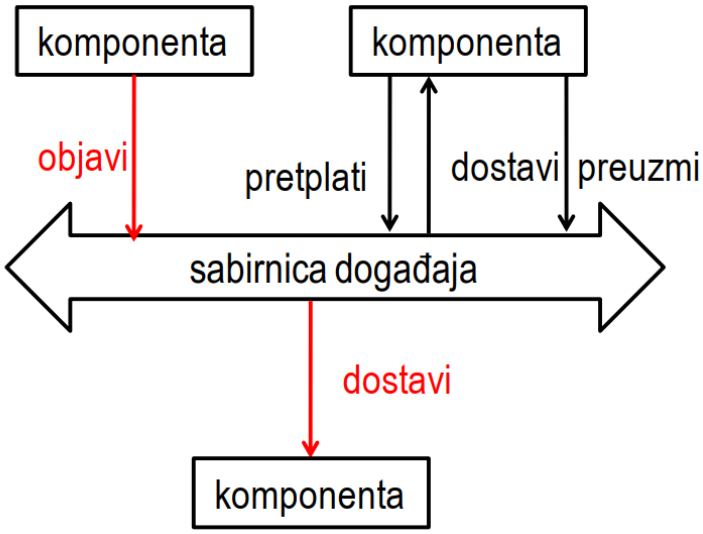


Slika 4 *Skica arhitekture zasnovane na podacima (Podnar Žarko, Pripužić, Lovrek, Kušek, 2013.) [1]*

1.3.4. Arhitektura zasnovana na događajima

Osnovni način komunikacije među procesima u ovoj arhitekturi je razmjena događaja (engl. *event*) kojima se oglašava raspoloživost podataka. Komponente se mogu pretplatiti na podatak ili grupu podataka sa sličnom značajkom kako bi mogle promijeniti stanje prilikom promjene ili objave novog podatka [1].

Jedna od izvedbi ovog sustava je i već spomenuti arhitekturni obrazac „objavi-pretplati“. Arhitektura korištena za programsku izvedbu ovog završnog rada zasniva se na objavama i pretplatama vezanim za neko geografsko područje [1].



Slika 5 *Skica arhitekture zasnovane na događajima (Podnar Žarko, Pripužić, Lovrek, Kušek, 2013.) [1]*

2. Korišteni alati

2.1. Apache Spark

Apache Spark je programski okvir otvorenog koda čija je osnovna namjena izrada raspodijeljenih programa i programskih sustava. Spark pruža sučelje za programiranje čitavog sustava sa implicitnom paralelizacijom podataka i obradom pogrešaka [4].

Spark se zasniva na obradi raspodijeljenog skupa podataka (engl. RDD - *resilient distributed dataset*), koje je moguće isključivo čitati (engl*. read-only*). Taj skup podataka smješten je na grozdu uređaja (engl. *cluster*) na kojem se obrađuje na način koji je otporan na pogreške. Spark 2.x donio je promjenu uvođenjem aplikacijskog sučelja Dataframe koje u pozadini koristi aplikacijsko sučelje RDD [4].

Spark je nastao 2012. godine kao odgovor na ograničenost programske paradigme „*map-reduce“* koja je do tad bila najzastupljenija kod raspodijeljene obrade podataka. Način na koji su se podaci obrađivali tada bio je linearan – čitanje s diska, mapiranje podataka, reduciranje rezultata mapiranja i pohrana na disk. Aplikacijsko sučelje RDD omogućilo je korištenje dijeljene memorije koja smanjuje čitanje i pisanje na disk prilikom pohrane međurezultata i na taj način ubrzava obradu. Spark podržava funkcionalnost iterativnih algoritama kao i interaktivnih pretraživačkih upita nad RDD-jem [4].

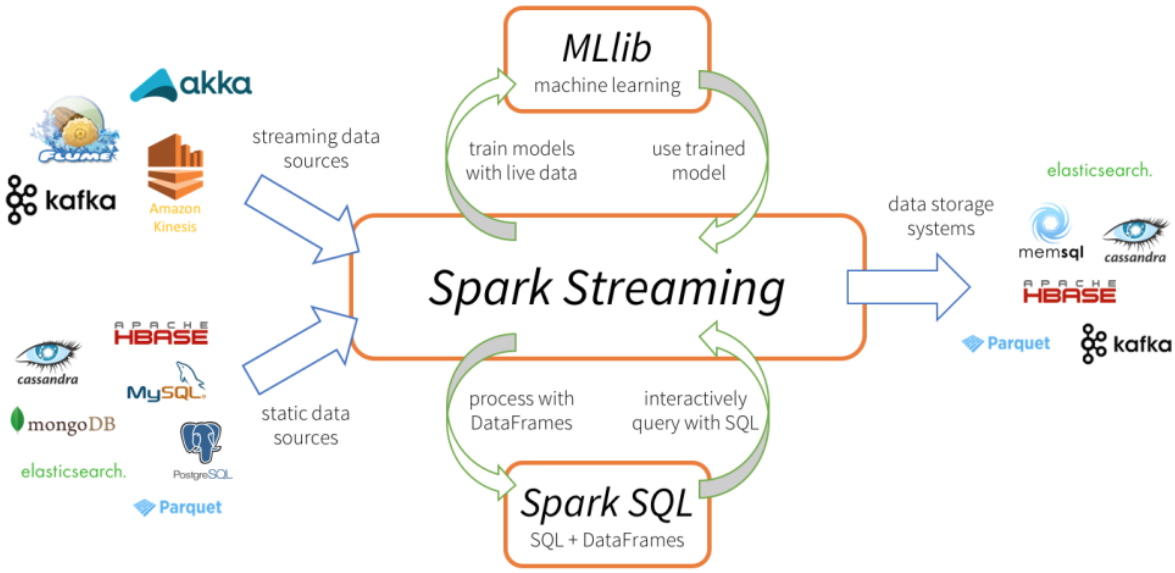
Za korištenje radnog okvira Apache Spark, uz Javin virtualni stroj, potreban je upravitelj grozda (engl. *cluster manager*) te raspodijeljeni sustav za pohranu (engl. *distributed storage system*). Prednost korištenja Apache Sparka je ta što u testnom načinu ne zahtjeva višeračunalno okruženje te se može ispravljati (engl. *debug*) kao i svaki drugi program pisan u Javi ili Scali [4].

Osnovni dijelovi programskog okvira Apache Spark su:

1) Spark Core - temeljni dio zadužen za izvođenje zadataka (engl. *task execution*), zakazivanje zadataka (engl. *task scheduling*), funkcionalnosti čitanja i pisanja (engl. *input-output operations*) i ostale ključne funkcionalnosti potrebne za

izvršavanje raspodijeljenog programa. Programski jezici pomoću kojih se mogu pisati programi za Apache Spark su Java, Python, Scala i R.

2) Spark SQL – nadgradnja koja omogućuje manipulaciju nad apstrakcijom zvanom DataFrames, koja pruža potporu za obradu strukturiranih i polustrukturiranih skupova podataka.
3) Spark Streaming – nadgradnja koja služi za obradu toka podataka, podaci se prikupljaju iz toka kroz definirani period (npr. 1 sekunda) te se zatim obrađuju kao i svaki RDD. Spark Streaming dolazi sa ugrađenom podrškom za Kafku, Flume, Twitter, ZeroMQ, Kinesis i obične TCP/IP priključke.
4) MLib – radni okvir za raspodijeljeno strojno učenje.
5) GraphX – radni okvir za raspodijeljenu obradu grafova.



Slika 6 *Apache Spark "ekosistem" [4]*

2.2. Apache Kafka

Apache Kafka je raspodijeljena platforma za prijenos toka podataka (engl. *distributed streaming platform*) koja objavljuje podatke svim pretplaćenim klijentima, slično kao i red (engl. *message queue*) ili sustav poruka (engl. *enterprise messaging system*). Također, pomoću alata Zookeeper, Kafka pohranjuje tokove podataka na održiv način otporan na greške te ih obrađuje redoslijedom kojim su primljeni [5].

Kafka se generalno koristi za izgradnju pouzdanih tokova podataka između sistema ili aplikacija koje rade u stvarnom vremenu. Pokreće se na jednom ili više poslužiteljskih računala koji mogu posluživati više pretplaćenih klijenata. Kafka pohranjuje tokove podataka u kategorijama koje se nazivaju „teme“ (engl*. topic*) te se svaki zapis u toku podataka sastoji od ključa (engl*. key*), vrijednosti (engl*. value*) i oznake vremena (engl. *timestamp*) [5].

Kafka ima četiri glavna aplikacijska programska sučelja (engl. *application programming interface*, skraćeno API):

- Producer API omogućava objavu toka podataka na jednu ili više „tema“.
- Consumer API omogućava pretplatu na jednu ili više „tema“ te obradu primljenog toka.
- Streams API dopušta aplikaciji da se ponaša kao obrađivač toka podataka, transformirajući ulazni u izlazni tok podataka.
- Connector API dopušta izgradnju i pokretanje proizvođača (engl. *producers*) koje je moguće ponovno koristiti ili potrošača (engl. *consumers*) koji povezuju Kafka „teme“ s postojećim aplikacijama ili bazama podataka. Na primjer, proizvođač koji može registrirati svaku promjenu u tablici relacijske baze podataka.

Za komunikacija između klijenata i poslužitelja koristi se *language agnostic TCP (Transmission control protocol)* protokol visokih performansi [5].

2.3. Programska knjižnica GeoSpark

GeoSpark je programska knjižnica (engl. *library*) otvorenog koda koja služi kao nadgradnja za Apache Spark. Njezina glavna funkcionalnost je obrada velikih geoprostornih podataka. Iako je još vrlo mlad projekt, GeoSpark je od svog začetka dobio mnoge pozitivne kritike te je 2018. godine u članku[2] objavljenom na stranicama publikacije PVLDB proglašen kao jedan od najkompletnijih sustava za prostornu analizu podataka [6].

Knjižnica GeoSpark sastoji se od nekoliko modula:

1) GeoSpark Core – za manipulaciju nad prostornim RDD-jevima.
2) GeoSpark SQL – SQL sučelje za GeoSpark Core.
3) GeoSpark Viz – ekstenzija za vizualizaciju podataka.
4) GeoSpark Zeppelin – proširenje programa Apache Zeppelin za vizualizaciju podataka.

Glavne mogućnosti knjižnice GeoSpark su:

- prostorni RDD,
- prostorni SQL upiti,
- kompleksni geometrijski i trajektorijski objekti – točka, poligon, linija, multi-točka, multi-poligon, multi-linija, geometrijske kolekcije,
- razni podržani formati ulaznih podataka - .csv, .tsv, .wkt, .wkb, .geojson, .shp, itd.,
- prostorni upiti – *range query, range join query, distance join query,*
- prostorni indeksi – R-tree i Quad-Tree,
- prostorno particioniranje – KDB-Tree, Quad-Tree, R-Tree, Voronoiov dijagram, Hilbertova krivulja, uniformna raspodjela,
- različiti koordinatni sustavi,
- vizualizacija mapa visoke rezolucije.

[2] „How Good Are Modern Spatial Analytics Systems“, http://www.vldb.org/pvldb/vol11/p1661-pandey.pdf

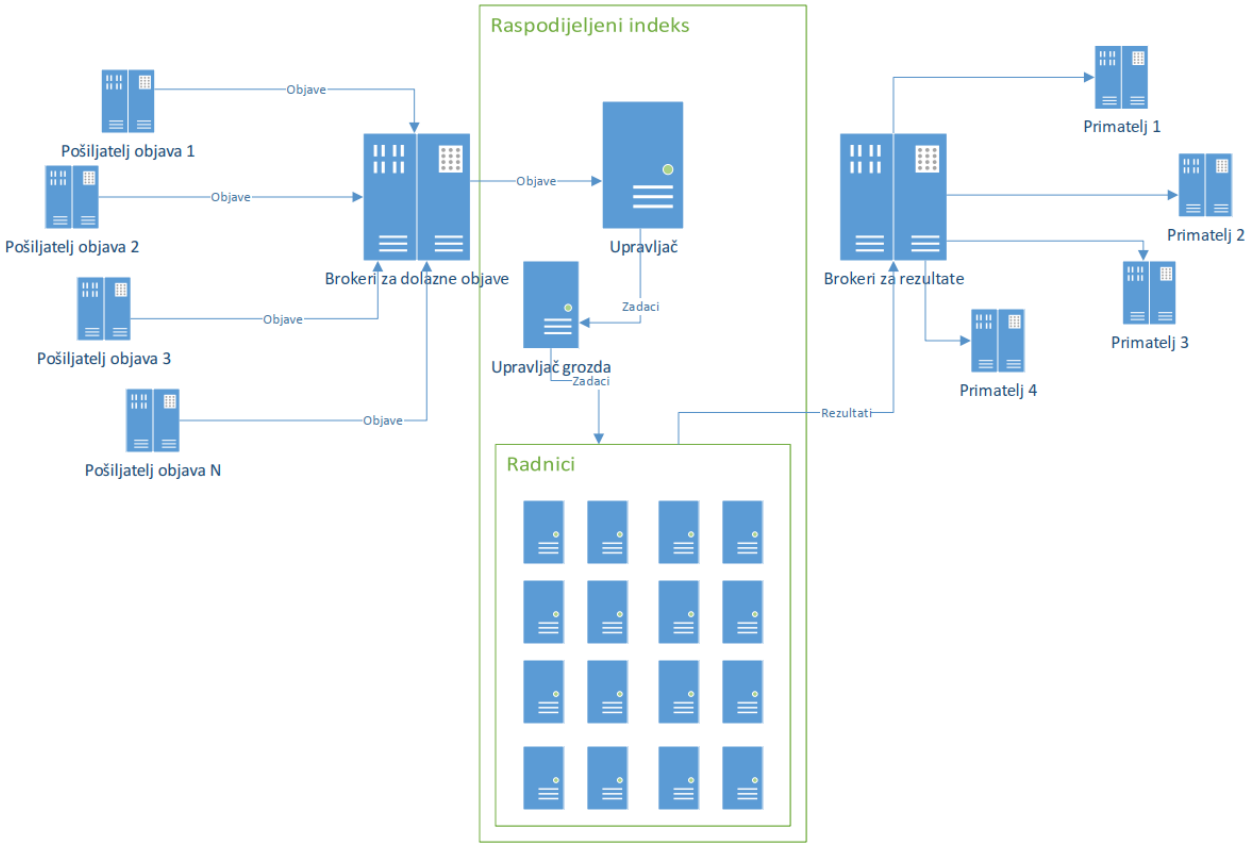
3. Arhitektura sustava

3.1. Dizajn sustava

Funkcionalnost sustava je obrada toka podataka kojim se šalju objave koje sadrže geoprostornu informaciju na temelju koje se pridružuju pripadnim pretplatama. Pretplate su predstavljene kao poligoni u prostoru. Budući da geoprostorna informacija ne mora nužno biti točka, objave se mogu grupirati s više različitih pretplata.

Sustav se sastoji od:

- pošiljatelja objava,
- brokera za poruke,
- raspodijeljenog indeksa pretplata,
- primatelja rezultata.



Slika 7 *Dizajn sustava*

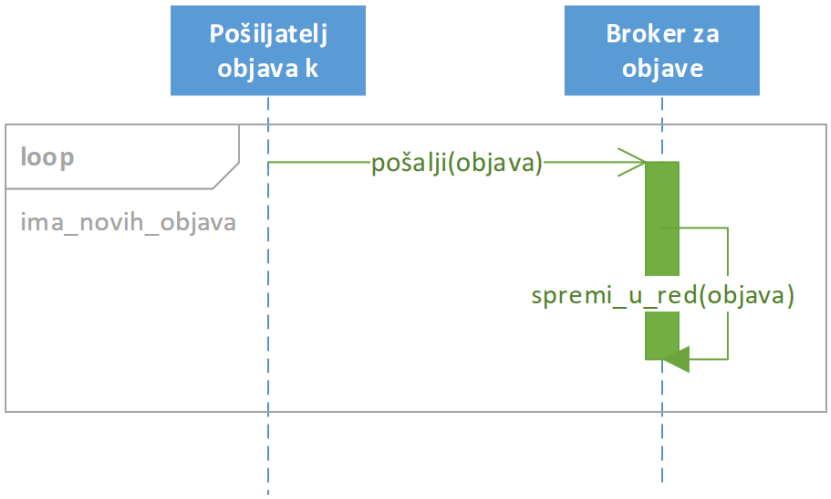
3.2. Pošiljatelji objava

Pošiljatelji objava predstavljaju zasebne sustave koji funkcioniraju kao izvori podataka. Svaki pošiljatelj može imati različitu implementaciju, no mora znati komunicirati s brokerima za dolazne objave. Također, pošiljatelj mora poštovati format koji se prima kao ulaz za raspodijeljeni indeks. Pošiljatelji šalju objave na ulaznu „temu“. „Tema“ je imenski prostor koji grupira poruke koje nose istu vrstu informacije.

Primjeri pošiljatelja mogu biti razni, a za testiranje implementacije raspodijeljenog indeksa korišten je jednostavan program koji čita linije iz datoteke koje su formata GeoJSON (opisano u poglavlju 4.1.1.) te ih šalje na broker (opisano u poglavlju 3.3).

3.3. Brokeri za poruke

Brokeri služe za skladištenje poruka koje se naknadno prosljeđuju do raspodijeljenog indeksa. Način prosljeđivanja poruka do raspodijeljenog indeksa funkcionira na principu prozivke (engl. *poll*); upravljački proces u određenom vremenskom razmaku čita poruke vezane za određenu „temu“ koje su do tada poslane na brokere te se zatim poruke serijaliziraju u objekte (opisano u poglavlju 4.1.3.) nad kojim se radi daljnja manipulacija. U sustavu se brokeri također koriste i za prosljeđivanje rezultata zainteresiranim primateljima.



Slika 8 *Sekvencijski dijagram - slanje objave*

Za implementaciju sustava korišteni su brokeri izvedeni u alatu Apache Kafka, stoga pošiljatelji poruka moraju implementirati sučelje `KafkaProducer` ili koristiti neki od već postojećih klijenata koji ga implementiraju. S druge strane, raspodijeljeni indeks je izveden kao Spark aplikacija koja koristi Spark Streaming za obradu toka podataka. Spark Streaming ima ugrađenu potporu za Kafku, ali i za druge vrste tokova (primjerice Twitter, Flume, Kinesis, TCP, itd.). Unutar svakog Spark zadatka uključeno je i slanje poruka na brokere za rezultate te se za to koristi Javina implementacija sučelja Kafka Producer.

3.4. Raspodijeljeni indeks

Raspodijeljeni indeks je program za raspodijeljeni sustav napisan za radni okvir Apache Spark. Kao i većina drugih programa pisanih za Apache Spark, i ovaj se sastoji od 2 dijela:

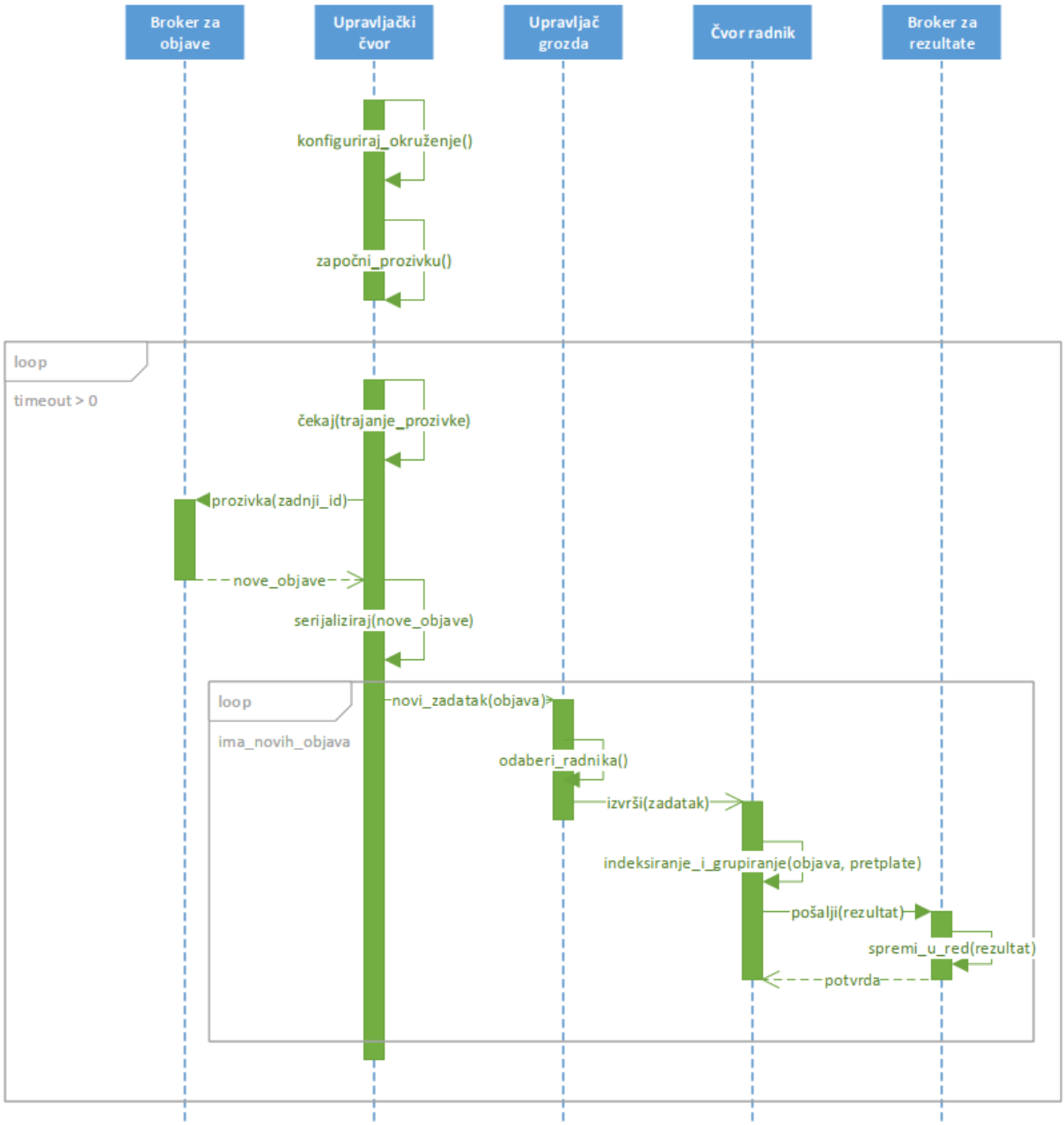
1.) Upravljački program (engl. *driver program*)
2.) Zadaci (engl. *tasks*)

Upravljački program izvodi se na upravljačkom čvoru sustava (engl. *driver node*) te je zadužen za:

- čitanje konfiguracije sustava,
- učitavanje pretplata u memoriju,
- stvaranje Spark Streaming konteksta i otvaranje toka podataka prema brokerima za pretplate,
- konzumiranje objava iz toka podataka - modeliranje i zadavanje upita nad prostornim pretplatama spremljenim u RDD.

Zadaci su dijelovi programa koji se izvode na čvorovima radnika (engl. *worker nodes*). Njihove osnovne funkcionalnosti su:

- izvršavanje upita nad prostornim pretplatama,
- generiranje poruka na temelju dolazne objave i pripadne pretplate,
- slanje poruka na broker za rezultate.



Slika 9 *Sekvencijski dijagram - obrada objava*

Važno je naglasiti da, ovisno o konfiguracijama, program se može pokrenuti i na jednom računalu kako bi se mogao testirati i kako bi se otklonile grube greške prije nego što se pokrene na grozdu. Također, bitan dio sustava je i upravitelj grozdom koji prosljeđuje zadatke od upravljačkog čvora do čvora radnika te je zadužen za balansiranje opterećenja raspodijeljenog sustava.

3.5. Primatelji poruka

Primatelji poruka zasebni su sustavi koji konzumiraju tok podataka (poruka) s brokera zaduženog za rezultate. Primatelji također mogu imati različite implementacije kao i pošiljatelji, no moraju znati čitati podatke s Kafka brokera, što bi značilo da moraju implementirati sučelje `KafkaConsumer` ili koristiti klijente koji implementiraju to sučelje.

U svrhu testiranja, primatelja poruka predstavlja skripta koja čita podatke s „teme“ na koju se šalju rezultati te ih ispisuje na standardni izlaz.

4. Ulazni i izlazni podaci

4.1. Objave

Objave su podaci koje izvori informacija šalju na sustav te su vezane za određenu geografsku značajku (objašnjeno u poglavlju 4.1.2.). Objava sadrži dodatne informacije koje predstavljaju tijelo poruke, dok bi se geografska značajka mogla smatrati zaglavljem poruke budući da govori o odredištu objave. Sve objave formatirane su po specifikaciji GeoJSON (objašnjeno u poglavlju 4.1.1).

4.1.1. Format GeoJSON

GeoJSON je format za razmjenu geoprostornih podataka baziran na formatu JSON (*JavaScript Object Notation*). U specifikaciji definirano je nekoliko različitih vrsta JSON objekata i načina na koji se oni međusobno kombiniraju kako bi predstavili složene geografske podatke. Specifikacija GeoJSON[3] objavljena je 2016. godine. Tome je prethodilo osnivanje radne grupe IETF-a (*Internet Engineering Task Force*) pod nazivom GeoJSON WG koja je radila na standardizaciji formata GeoJSON [7].

Format GeoJSON koristi standardizirane tipove geoprostornih podataka kao što su:

- `Point` – točka u prostoru definirana s koordinatama
- `MultiPoint` – skup točaka u prostoru definiranih poljem koordinata
- `LineString` – skup povezanih točaka u prostoru definiranih poljem koordinata koje čine jednu neprekinutu liniju
- `MultiLineString` – skup neprekinutih linija definiranih poljem polja koordinata od kojih svako polje predstavlja jednu neprekinutu liniju
- `Polygon` – dio prostora omeđen linijama definiran poljem polja koordinata od kojih svako polje predstavlja jednu neprekinutu liniju te je početna koordinata prvog polja (linije) ujedno i završna točka zadnjeg polja (linije).
- `MultiPolygon` – skup više poligona predstavljen kao polje informacija koje opisuju jedan objekt tipa `Polygon`

[3] RFC 7946, „The GeoJSON Format“

- `GeometryCollection` – heterogena komponenta koja predstavlja kolekciju iznad navedenih tipova

Uz standardizirane tipove format GeoJSON definira dva tipa JSON objekata:

1. `Feature` – objekt koji predstavlja stvar ograničenu prostorom.

```
{
      "type": "Feature",
      "geometry": {
            "coordinates": [-2.910034, 54.372822],
            "type": "Point"
      },
       "properties": {
            "id": 2005030000008
      },
      "id": 43243
}
```

Slika 10 *Primjer objekta tipa* `Feature`

   Svaki objekt tipa `Feature` ima 3 obavezna i 1 opcionalan podatkovni član:
   - `type` – definira tip objekta
   - `geometry` – objekt koji sadrži podatkovne članove koji opisuju prostornu značajku, sadrži podatkovne članove `coordinates` i `type` pomoću kojih se definira standardizirani geoprostorni tip podataka
   - `properties` – objekt koji sadrži korisničke informacije koji može biti proizvoljne strukture ovisno o načinu uporabe
   - `id` – opcionalni atribut koji služi za enumeraciju objekata

   Dodatno, specifikacija definira podatkovni član `bbox` koji služi za definiranje pravokutnog okvira oko geoprostorne značajke.

2. `FeatureCollection` – objekt koji definira skup objekata tipa `Feature`, sadrži podatkovni član `features` koji predstavlja polje objekata tipa `Feature` te podatkovni član `type` koji definira tip tog objekta.

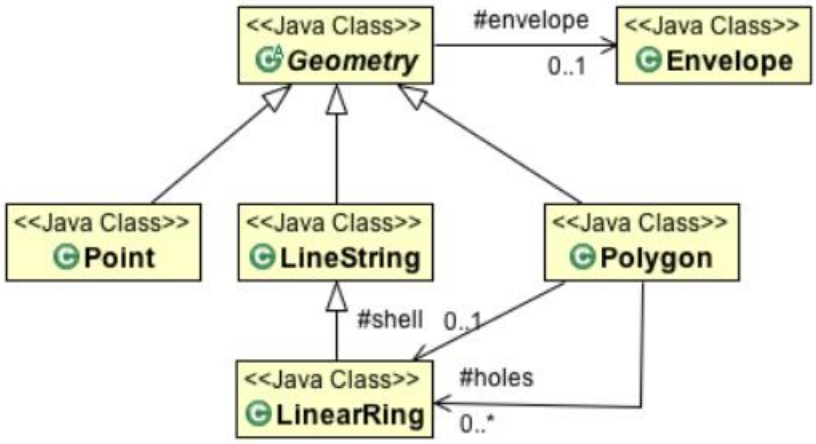
```
{
	"type":"FeatureCollection",
	"features": [{
		"type": "Feature",
		"geometry": {
			"type": "Point",
			"coordinates": [102.0, 0.5]
		},
		"properties": {
			"prop0": "value0"
		}
	}, {
		"type": "Feature",
		"geometry": {
			"type": "LineString",
			"coordinates": [
				[102.0, 0.0],
				[103.0, 1.0],
				[104.0, 0.0],
				[105.0, 1.0]
			]
		},
		"properties": {
			"prop0": "value0",
			"prop1": 0.0
		}
	}]
}
```

Slika 11 *Primjer objekta tipa* `FeatureCollection`

Važno je naglasiti da specifikacija za referentni koordinatni sustav koristi notaciju WGS 84 (*World Geodetic System 1984*) koja koristi geografske duljine i širine u stupnjevima [7].

4.1.2. Razred `Geometry`

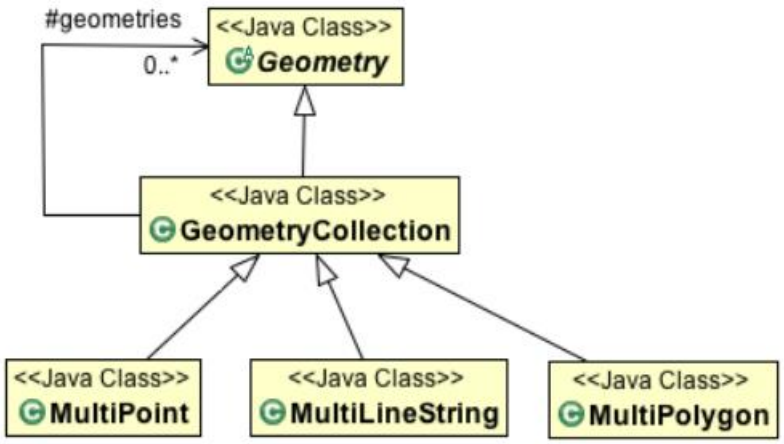
Razred `Geometry` definiran u programskoj knjižnici Java Topology Suite osnovni je razred koji predstavlja apstraktni geoprostorni objekt koji nosi dodatne informacije. Razredi izvedeni iz tog razreda upravo su vrste standardiziranih tipova podataka (`Point`, `LineString`, `Polygon`) [8].



***Slika 12** Dijagram razreda `Geometry` [8]*

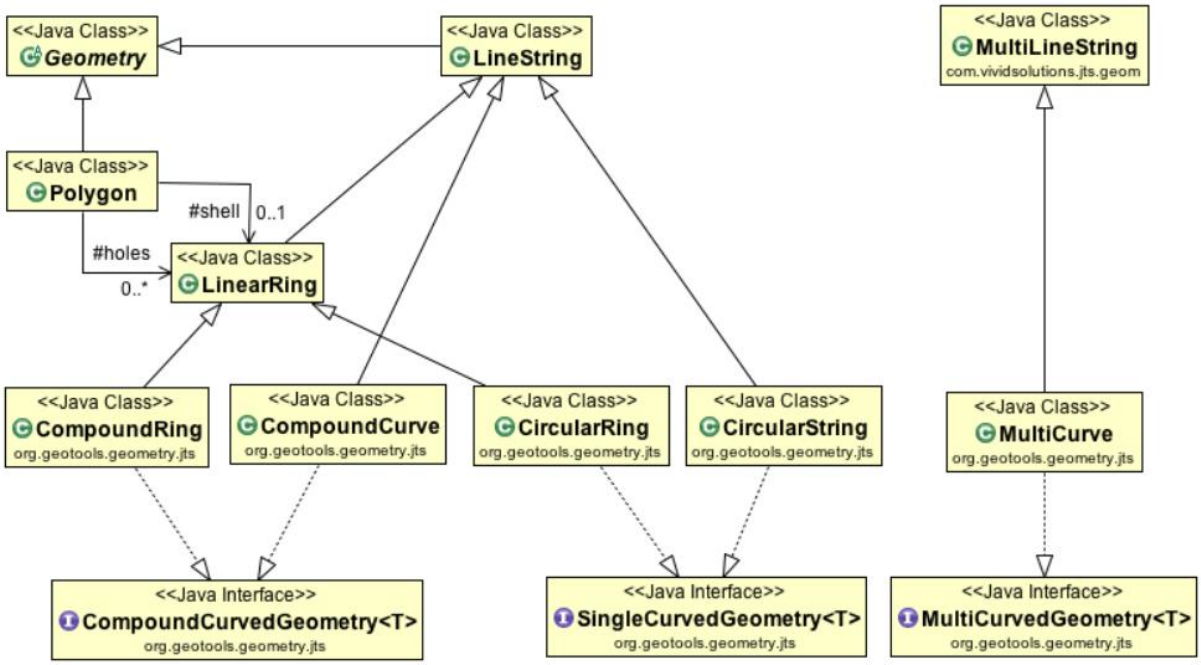
Razred `Geometry` sadrži člansku varijablu tipa `Envelope` koja predstavlja pravokutni okvir (engl. *bounding box*) oko nekog geografskog prostora. Upravo se pomoću te članske varijable radi prostorna pretraga pretplata na način da se traže poligoni koji djelomično ili potpuno prekrivaju taj okvir [8].

Razred `GeometryCollection` predstavlja skup razreda tipa `Geometry` (obrazac kompozit).Takav razred je i sam izveden iz baznog razreda `Geometry` i shodno s time se nad njime može također računati pravokutni okvir za pronalazak presjeka. Izvedeni tipovi razreda `GeometryCollection` su `MultiPoint`, `MultiLineString` i `MultiPoligon` [8].



Slika 13 *Dijagram razreda* `GeometryCollection` *[8]*

Programska knjižnica GeoTools definira razrede koji predstavljaju krivulje u prostoru te su izvedeni iz razreda definiranih u knjižnici JTS.



Slika 14 *Dijagram razreda definiranih u knjižnici GeoTools [8]*

4.1.3. Deserijalizacija objava

Dolaskom do raspodijeljenog indeksa, objave je potrebno serijalizirati iz znakovnog niza u objekt tipa `Geometry` kako bi se pomoću njih mogli izvršavati daljnji upiti nad skupom pretplata. U implementaciji programskog rješenja, za serijalizaciju GeoJSON formata koristi se razred `FormatMapper` definiran u programskoj knjižnici GeoSpark. `FormatMapper` interno koristi razred `GeometryFactory` za stvaranje objekte tipa `Geometry` (obrazac tvornica). Uz to, `FormatMapper` također koristi programsku knjižnicu JTS2geojson za pomoć pri serijalizaciji i deserijalizaciji različitih razreda definiranih u programskoj knjižnici JTS.

4.2. Pretplate

Pretplate u sustavu definirane su kao poligoni u prostoru uz kojih je vezana neka dodatna informacija. Pretplate se prilikom pokretanja programa učitavaju u RDD pomoću metode `readToGeometryRDD` definirane u razredu `GeoJsonReader`. Tekstualna datoteka definirana je na način da svaki redak predstavlja jedan GeoJSON objekt.

4.3. Agregirane poruke

Agregirane poruke predstavljaju rezultat pridruživanja objava i pretplata. Svaka agregirana poruka sadrži identifikator objave i pretplate. Poruku stvaramo iz objekta tipa `SubscriptionPairModel` koji sadrži članove `subscriptionId` i `dataId`. Ti se objekti prilikom slanja serijaliziraju u JSON format.

```
{"dataId:": 2412, "subscriptionId": 25}
```

Slika 15 *Primjer agregirane poruke*

5. Optimizacija prostornih upita

5.1. Geoprostorno particioniranje podataka

Geoprostorni skupovi podataka često su veliki i neorganizirani te je zbog toga teško vršiti ikakve upite i transformacije nad njima. Da bi se uklonila ta negativna pojava, podatke je potrebno particionirati (engl. *partitioning*). Particioniranje označava logičko razdjeljivanje podataka u neke manje spremnike (particije). Na temelju particija stvara se logička struktura koju možemo na lakši način pretraživati i koja rezultira boljim performansama.

Programska knjižnica GeoSpark omogućava particioniranje klasičnog RDD-ja pomoću nekoliko strategija koje stvaraju klasične geoprostorne strukture koje se inače koriste kod rada s geoprostornim podacima. Vrste particioniranja odabiru se na temelju zadane strategije koja kao rezultat generira prostornu mrežu (engl. *grid type*).

5.1.1. `FlatGridPartitioner`

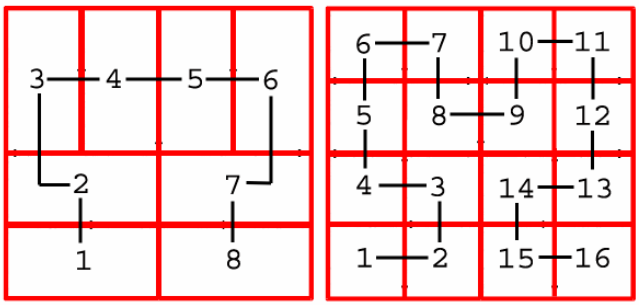
`FlatGridPartitioner` je razred koji implementira najjednostavniju logiku particioniranja. On predstavlja jednostavnu strukturu nalik listi koju je moguće slijedno pretraživati. Podaci će biti sortirani ovisno o strategiji koja se koristila za generiranje prostorne mreže.

Strategije koje koriste ovu vrstu razdjeljivača (engl. *partitioner*) su:

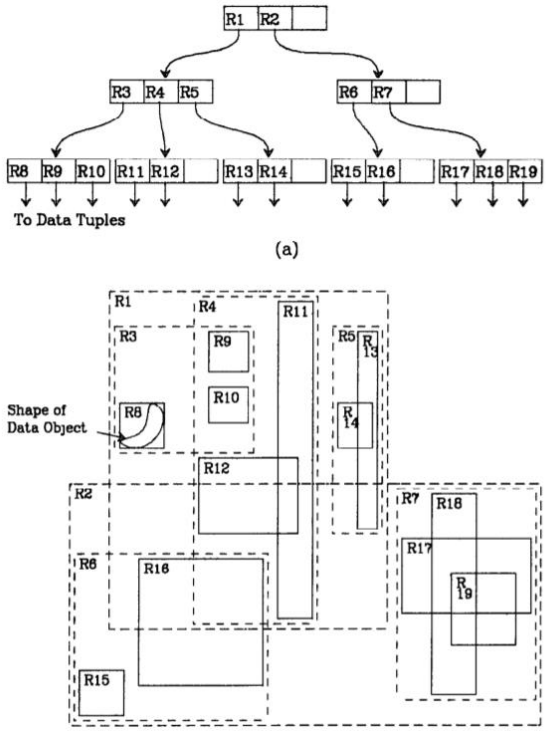
- Jednolika raspodjela- najjednostavnija strategija particioniranja koja dijeli geoprostorne podatke na jednoliku pravokutnu mrežu.
- Raspodjela pomoću Hilbertove krivulje – strategija koja koristi Hilbertovu krivulju za optimalno pretraživanje prostora. Način na koji je prostor razdijeljen prikazan je na slici 16.
- Raspodjela pomoću R-stabla –podatkovna struktura posebno prilagođena za indeksiranje geoprostornih podataka. Ideja pri stvaranju takve strukture je grupiranje objekata u minimalni prostorni okvir (engl. *minimum bounding box*)

rekurzivno u dubinu. Primjer particioniranja pomoću R-stabla prikazan je na slici 17.

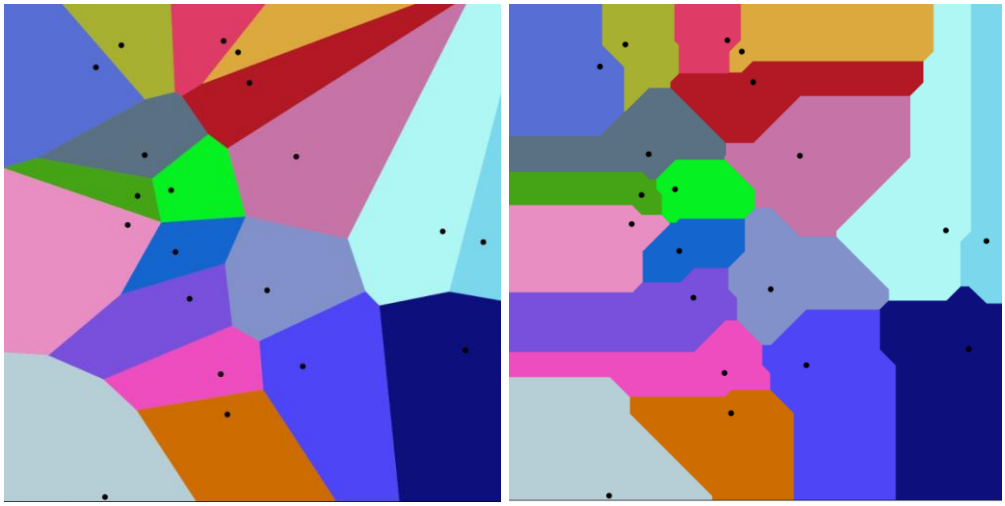
- Raspodjela pomoću Voronoijevog dijagrama – strategija koja koristi načelo raspodjele prostora na dijelove (poligone) koji predstavljaju skup točaka koje su najbliže udaljene nekim inicijalno zadanim točkama. Način računanja najbliže točke može biti pomoću euklidske ili pomoću Manhattan udaljenosti. Primjer particioniranja pomoću Voronoijevog dijagrama prikazan je na slici 18. [10]



Slika 16 *Particioniranje prostora pomoću Hilbertove krivulje (Buisson, 2007.) [9]*



Slika 17 *Particioniranje prostora pomoću R-stabla (Guttman, 1984.) [13]*

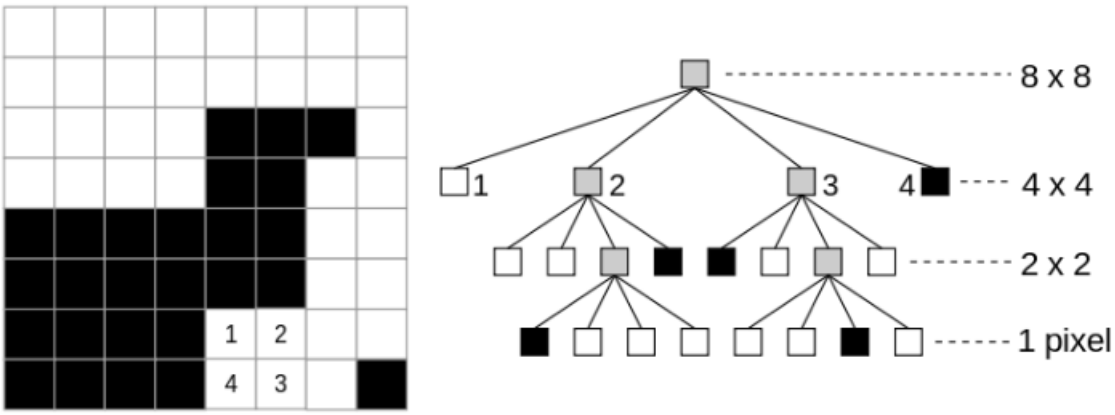


Slika 18 *Particioniranje prostora pomoću Voronoijevog dijagrama korištenjem euklidske (slika lijevo) i Manhattan udaljenosti (slika desno) (Ertl, 2012.) [10]*

5.1.2. `QuadTreePartitioner`

`QuadTreePartitioner` je razred koji implementira particioniranje pomoću podatkovne strukture naziva Q-stablo (engl. *Q-tree, quad tree*).

Q-stablo je podatkovna struktura koja se često koristi u analizi i obradi podataka. U suštini, to je stablo čiji su čvorovi stupnja 4, odnosno svaki čvor (osim završnog) ima četvero podčvorova (čvorova djece, engl. *child nodes*). Način na koji se gradi najčešće započinje s običnom kvadratnom mrežom koja se rekurzivno dijeli u podmreže sve dok se u svakoj ćeliji mreže ne nalazi maksimalno jedan podatak (ili jedna vrsta podataka). Primjer izgradnje jednog Q-stabla vidimo na slici 19. [12]



Slika 19 *Izgradnja Q-stabla na primjeru grupiranja piksela (Muła, 2015.) [12]*

Ovakva struktura podataka omogućuje optimalno pretraživanje složenosti $O(\log_4 n)$ i zbog toga se često koristi za particioniranje velikih količina podataka koje je potrebno često pretraživati. Ovisno o skupu podataka, ovaj način particioniranja generira strukturu određene složenosti – najbolje funkcionira za jednoliko raspršene podatke, dok se lošije performanse dobivaju za gusto grupirane podatke (točke, piksele) [12].

5.1.3. `KDBTreePartitioner`

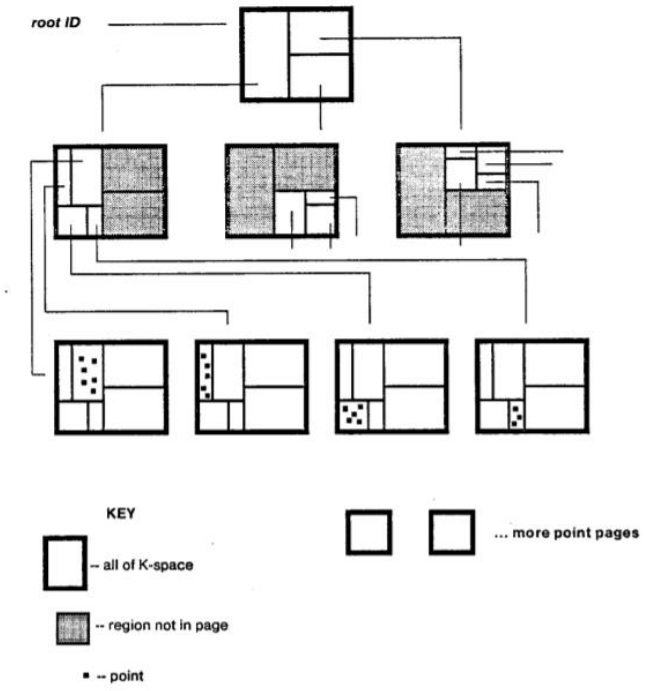
`KDBTreePartitioner` je razred koji implementira particioniranje pomoću K-D-B stabla (engl. *k-dimensional B-tree*).

K-D-B stablo služi za particioniranje prostora u k-dimenzionalnom prostoru u svrhu optimizacije prostornih upita (engl. *range query*). Takva struktura ima svojstvo „savršene“ balansiranosti koja označava jednoliku udaljenost svakog lista od korijena stabla. Svaki čvor stabla je zapravo stranica (engl. *page*) stoga se pristup dječjim čvorovima može obaviti učinkovito tehnikom straničenja (engl. *paging*) [13].

Postoje dva tipa stranica:

1. Stranica regije – unutarnji čvorovi stabla, sadrži pokazivače na druge stranice.
2. Stranice točaka – listovi stabla, sadrži pokazivače na podatke.

K-D-B stablo ima svojstva slična B-stablu koje se najčešće koristi za indeksiranje u relacijskim sustavima, no za razliku od B-stabla, K-D-B stablo ne osigurava 50% iskorištenosti svakog čvora u stablu [13].



Slika 20 *Primjer K-D-B stabla (Robinson, 1981.) [13]*

5.2. Geoprostorno indeksiranje podataka

Programska knjižnica GeoSpark uz već navedenu podršku za geoprostorno particioniranje nudi i opciju stvaranja prostornih indeksa. Te dvije funkcionalnosti moguće je i kombinirati na način da se prvo napravi particioniranje, a zatim indeksiranje particija. Podržane implementacije prostornih indeksa zasnivaju se na Q-stablu i R-stablu koji su opisani u prethodnom poglavlju (5.1).

6. Opis implementacije

Raspodijeljeni indeks izveden je kao aplikacija za radni okvir Apache Spark. Zbog interoperabilnosti, dio programskog rješenja napisan je u programskom jeziku Java, a dio u programskom jeziku Scala. Skripte za testiranje i evaluaciju rješenja napisane su u programskom jeziku Python.

Glavna metoda (engl. *main method*) koja označava početnu točku programa nalazi se u razredu `GeospatialIndexDistributed`. Metoda kao argument prima putanju do konfiguracijske datoteke koja sadrži potrebne detalje za konfiguraciju raspodijeljenog indeksa podataka. Ta datoteka mora biti formatirana u formatu „ključ=vrijednost“ (standardni Javin format za konfiguracijske datoteke, ekstenzija *.properties*). Primjer konfiguracije nalazi se u prilogu 1. Popis konfiguracijskih vrijednosti te njihov opis nalazi se u tablici 1.

Tablica 1 *Popis konfiguracijskih vrijednosti*

Naziv	Opis	Vrijednosti
`subscriptions.location`	Putanja do datoteke koja sadrži popis svih pretplata u formatu GeoJSON	Putanja u standardnom formatu (relativna ili apsolutna)
`stream.kafka.brokers.in`	Popis brokera za ulazne poruke	Lista parova `host:port` koji su odijeljenih zarezom
`stream.kafka.brokers.out`	Popis brokera za izlazne poruke	Lista parova `host:port` koji su odijeljenih zarezom
`stream.kafka.groupId`	Identifikator grupe za konzumaciju toka	Neprazan niz znakova
`stream.kafka.topics.in`	Popis naziva tema s kojih se konzumiraju poruke	Lista nepraznih nizova
`stream.kafka.topic.out`	Naziv teme na koju se šalju izlazne poruke	Neprazan niz znakova
`spark.appName`	Naziv aplikacije	Neprazan niz znakova

`spark.master.url`	URL (*Uniform Resource Locator*) grozda	URL grozda (`local[*]` za lokalno testiranje, `yarn` za pokretanje pomoću upravitelja Yarn)
`spark.cache`	Drže li se pretplate u memoriji ili ne	Logička vrijednost `true/false`
`geospark.spatialPartitioning`	Strategija particioniranja podatka	• `QUADTREE` • `RTREE` • `VORNOI` • `KDBTREE` • `HILBERT` • `EQUAL`
`Geospark.spatialIndex`	Strategija indeksiranja podataka	• `QUADTREE` • `RTREE`
`logging.level`	Razina zapisivanja u dnevnik događaja	• `DEBUG` • `INFO` • `WARN` • `ERROR`
`spark.streaming.batchTime`	Duljina intervala za tzv. *micro-batch interva* u mikrosekundama	Pozitivni cijeli broj
`spark.streaming.timeout`	Vrijeme nakon kojeg prestaje prozivka (engl. *poll duration*) u sekundama	Pozitivni cijeli broj

Nakon učitavanja konfiguracije, konfigurira se i stvara objekt tipa `SparkContext` koji služi za komunikaciju sa instancom Sparka. Pomoću tog objekta stvara se i `StreamingContext` koji je zaslužan za komunikaciju s različitim tipovima toka podataka. Tom objektu dodaje se promatrač za spremanje testnih podataka (više u poglavlju 6.2). Zatim slijedi učitavanje pretplata u tzv. `SpatialRDD` nad kojim će se obavljati prostorni upiti. Nakon toga konfiguriramo tok podataka i zadajemo posao koji se obavlja prilikom dolaska nove objave. Za slanje podataka na Kafka broker stvaramo dijeljenu (engl. *broadcast*) varijablu koja je tipa `KafkaProducerWrapper`. U njoj je implementirana logika za lijeno (engl. *lazy*) instanciranje koje je nužno za *broadcast*

varijable. Razlog korištenja *broadcast* varijabli je izbjegavanje stvaranja objekta `KafkaProducer` pri obradi svake dolazne objave. Na kraju se pokrećemo Spark Streaming i dodajemo tzv. *shutdown hook* čija je funkcionalnost spremanje testnih podataka u formatu TSV (*Tab Separated Value*).

7. Eksperimentalna evaluacija

7.1. Testni podaci

Za eksperimentalnu evaluaciju rješenja odabrani su skupovi geoprostornih podataka vezanih za Veliku Britaniju. Skup pretplata predstavljat će prostorne regije Velike Britanije[4], dok će skup objava predstavljati podatci o automobilskim nesrećama unutar Ujedinjenog Kraljevstva u razdoblju od 2005. do 2015. godine[5] te skup podataka koji sadrži popis svih prometnica na prostoru Ujedinjenog Kraljevstva[6]. Skupovi podataka su pretvoreni iz formata .csv u format GeoJSON pomoću jednostavne skripte napisane u Pythonu (prilog 2.). Ukupan broj pretplata (prostornih regija) je 191. Broj objava o automobilskim nesrećama korišten pri evaluaciji je jedan milijun iako se koristi samo dio od tih podataka. Broj objava o prometnicama korišten pri evaluaciji je 5026. Razlog korištenja dva različita skupa objava je taj što su objave o automobilskim nesrećama tipa `Point` dok su objave o prometnicama tipovi `LineString` i `MultiLineString`. S druge strane, pretplate su tipa `Polygon`.

7.2. Pokretanje primjera

Programsko rješenje evaluirano je empirijski na način da je pokretano za 24 različite konfiguracije. Testiranje je izvršeno automatski uz pomoć jednostavne skripte napisane u programskom jeziku Python (prilog 3.) koja pokreće izvršavanje skripte `spark-submit` kojoj se predaju argumenti pomoću kojih se konfigurira okruženje za izvođenje programa, puno ime razreda (engl. *fully qualified class name*), putanja do jar arhive te argumenti komadne linije koji se predaju programu prilikom pokretanja. Primjer jedne takve naredbe prikazan je na slici 21.

[4] www.opendoorlogistics.com/downloads/
[5] www.kaggle.com/silicon99/dft-accident-data
[6] www.ordnancesurvey.co.uk/business-and-government/products/os-open-roads.html

```
spark-submit \
--master yarn \
--deploy-mode cluster \
--driver-cores 4 \
--num-executors 8 \
--executor-cores 2 \
--executor-memory 2G \
-files ~/configs/test.properties \
--class hr.fer.ztel.geospatial.GeospatialIndexDistributed \
~/jars/final_v3/geospatial-index-distributed-assembly-0.1.jar test.properties
```

Slika 21 *Primjer pokretanja aplikacije pomoću naredbe spark-submit*

Nakon pokretanja aplikacije te s odmakom od 20 sekundi pokreće se program napisan u programskom jeziku Python (prilog 4.) koji predstavlja testnog kreatora objava. Odmak od 20 sekundi je potreban kako bi se glavni program uspješno pokrenuo i pretplatio se na temu na brokeru zaduženom za objave.

7.3. Rezultati

Kao što je rečeno, za evaluaciju programa korištena su dva skupa objava. Prvi skup predstavlja objave vezane uz točku, dok drugi predstavlja objave vezane uz linijski podatak u prostoru. U svakom od testova mjereno je prosječno vrijeme potrebno za obradu jedne objave.

7.3.1. Objave vezane za točke u prostoru

Prilikom evaluacije pomoću točkastih objava korištena je konfiguracija koja je opisana u tablici 2.

Tablica 2 *Konfiguracija okruženja za prvo testiranje*

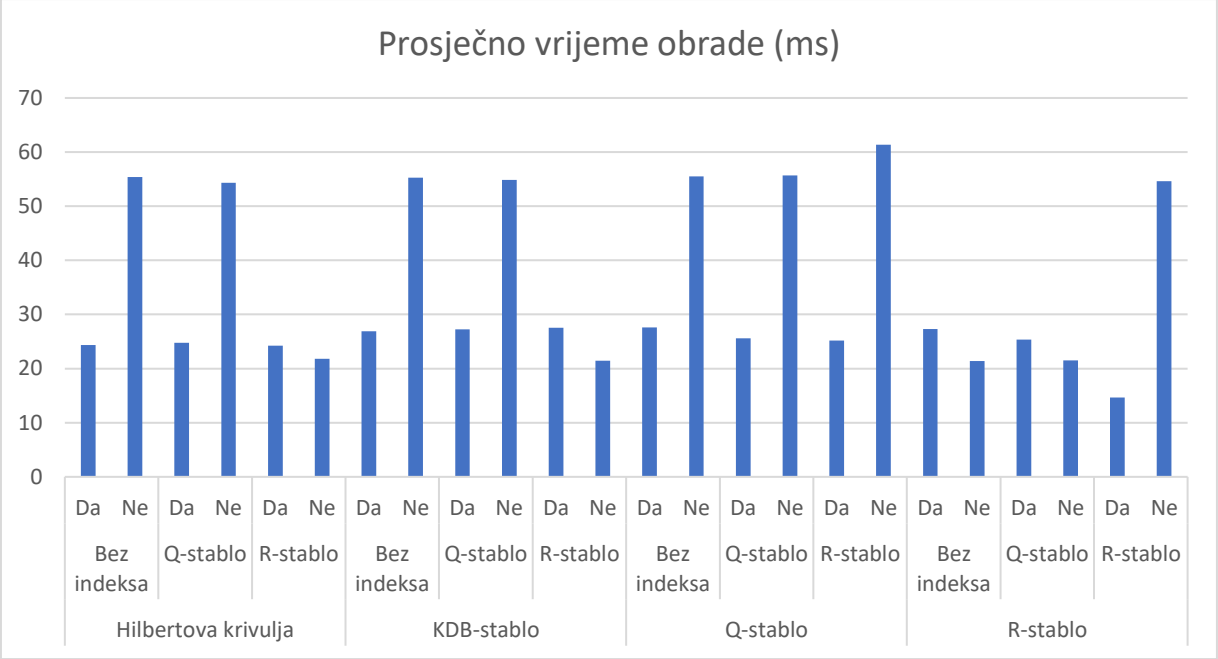
Broj jezgri upravljačkog računala	4
Broj računala radnika	4
Broj jezgri računala radnika	2
Veličina pojedine memorije dodijeljena pojedinom radniku	2 GB

Za evaluaciju korišteno je 10000 ulaznih objava uz interval prozivke od 2 sekunde kako bi izbjegli utjecaj tzv. *micro-batcheva* na vrijeme izvođenja. Rezultati su prikazani u tablici 3.

Tablica 3 *Tablični prikaz rezultata prvog testiranja*

Korištenje *cachea*	**Strategija particioniranja**	**Vrsta prostornog indeksa**	**Vrijeme obrade po objavi (ms)**
Da	Hilbertova krivulja	Bez indeksa	24.37113711
Da	KDB-stablo	Bez indeksa	26.91730827
Da	R-stablo	Bez indeksa	27.3304
Da	Q-stablo	Bez indeksa	27.6116
Da	Hilbertova krivulja	R-stablo	24.2321
Da	KDB-stablo	R-stablo	27.52814719
Da	R-stablo	R-stablo	14.66726673
Da	Q-stablo	R-stablo	25.17558244
Da	Hilbertova krivulja	Q-stablo	24.7849
Da	KDB-stablo	Q-stablo	27.24887421
Da	R-stablo	Q-stablo	25.33390017
Da	Q-stablo	Q-stablo	25.56636991
Ne	Hilbertova krivulja	Bez indeksa	55.37556244
Ne	KDB-stablo	Bez indeksa	55.26547345
Ne	R-stablo	Bez indeksa	21.40385961
Ne	Q-stablo	Bez indeksa	55.4826
Ne	Hilbertova krivulja	R-stablo	21.8377
Ne	KDB-stablo	R-stablo	21.46615338
Ne	R-stablo	R-stablo	54.62593741
Ne	Q-stablo	R-stablo	61.34243971
Ne	Hilbertova krivulja	Q-stablo	54.31276872
Ne	KDB-stablo	Q-stablo	54.86868687
Ne	R-stablo	Q-stablo	21.50880352
Ne	Q-stablo	Q-stablo	55.6752

Grafički prikaz rezultata nalazi se na slici 22. Os x ima 3 razine; najniža razina predstavlja korištenu strategiju particioniranja, iznad nje se nalazi struktura korištena za indeks i na najvišoj razini navedeno je koristi li se pričuvna memorija ili ne.



Slika 22 *Grafički prikaz rezultata za prvo testiranje*

Iz rezultata možemo vidjeti da se kod particioniranja pomoću Hilbertove krivulje, KDB-stabla i Q-stabla osjeti zavidno poboljšanje performansi u slučaju da se za indeks koristi Q-stablo ili da se ne koristi ni jedno stablo. S druge strane, particioniranje pomoću R-stabla odskače od tog ponašanja te za iste konfiguracije daje bolje rezultate za primjere koji ne koriste pričuvnu memoriju.

U primjerima koji koriste R-stablo kao strukturu na temelju koje je izgrađen indeks, bolje rezultate prilikom korištenja pričuvne memorije daju strategije particioniranja pomoću Q-stabla i R-stabla. S druge strane, strategije koje se temelje na Hilbertovoj krivulji te KDB-stablu daju bolje rezultate bez korištenja pričuvne memorije.

Treba naglasiti da ova rješenja uvelike ovise i o podacima koji su korišteni kao skup pretplata te o podacima korištenim za skup dolaznih objava. Ispostavlja se da se dosta dolaznih objava mapiralo u istu (najveću) regiju, te je najvjerojatnije zbog toga najbolji rezultat dalo baš particioniranje pomoću R-stabla.

Također, u obzir treba uzeti i dodatni trošak (engl. *overhead*) budući da se program izvodi na raspodijeljenom sustavu, no više o tome u zaključku.

7.3.2. *Objave vezane uz linije u prostoru*

Prilikom evaluacije pomoću objava vezanih za linije u prostoru korištena je konfiguracija koja je opisana u tablici 4. Budući da je mapiranje linijskih objekata zahtjevnije, povećan je broj računala radnika i njihovi resursi.

Tablica 4 *Konfiguracija rješenja za drugo testiranje*

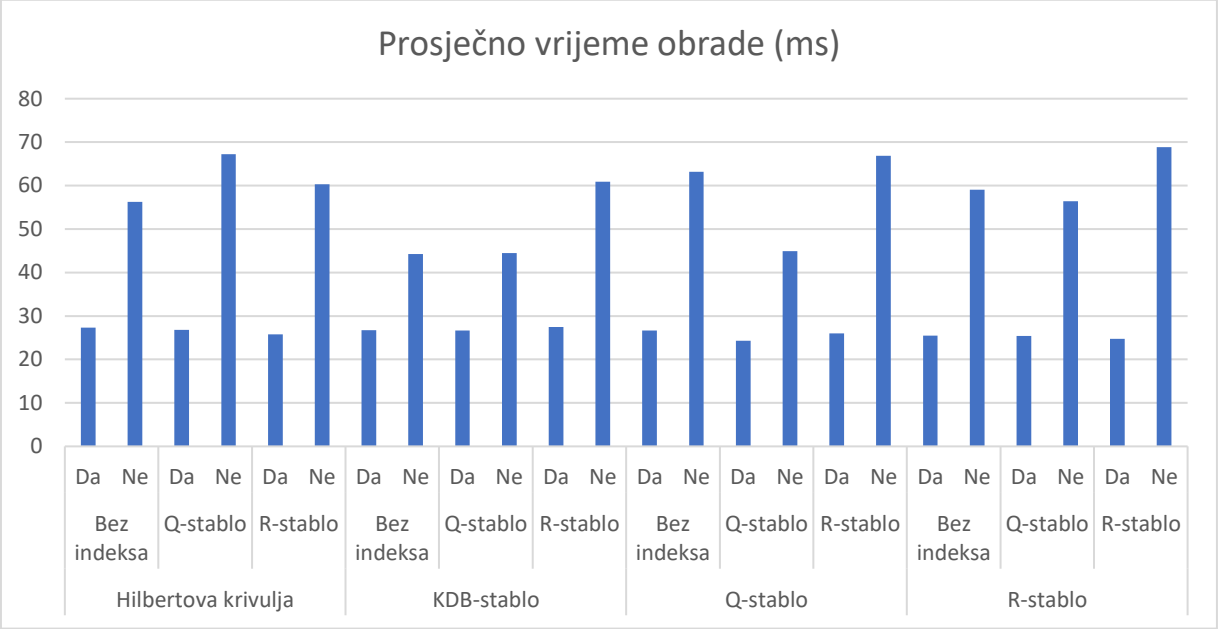
Broj jezgri upravljačkog računala	4
Broj računala radnika	6
Broj jezgri računala radnika	4
Veličina pojedine memorije dodijeljena pojedinom radniku	2 GB

Za evaluaciju korišten je skup od 10000 objava vezanih za linijske objekte u prostoru uz interval prozivke od 2 sekunde. Rezultati su prikazani u tablici 5.

Tablica 5 *Tablični prikaz rezultata drugog testiranja*

Korištenje *cachea*	Strategija particioniranja	Vrsta prostornog indeksa	Vrijeme obrade po objavi (ms)
Da	Hilbertova krivulja	Bez indeksa	27.29422886
Da	KDB-stablo	Bez indeksa	26.76278607
Da	R-stablo	Bez indeksa	25.49363184
Da	Q-stablo	Bez indeksa	26.68567164
Da	Hilbertova krivulja	R-stablo	25.78636816
Da	KDB-stablo	R-stablo	27.49014925
Da	R-stablo	R-stablo	24.73751244
Da	Q-stablo	R-stablo	26.01920398
Da	Hilbertova krivulja	Q-stablo	26.81532338
Da	KDB-stablo	Q-stablo	26.68627451
Da	R-stablo	Q-stablo	25.43054726
Da	Q-stablo	Q-stablo	24.27751244
Ne	Hilbertova krivulja	Bez indeksa	56.27412935
Ne	KDB-stablo	Bez indeksa	44.22913143
Ne	R-stablo	Bez indeksa	59.02806529
Ne	Q-stablo	Bez indeksa	63.18179104
Ne	Hilbertova krivulja	R-stablo	60.29333333
Ne	KDB-stablo	R-stablo	60.92926077
Ne	R-stablo	R-stablo	68.8551532
Ne	Q-stablo	R-stablo	66.82965174
Ne	Hilbertova krivulja	Q-stablo	67.23611388
Ne	KDB-stablo	Q-stablo	44.50596896
Ne	R-stablo	Q-stablo	56.43761194
Ne	Q-stablo	Q-stablo	44.8958209

Grafički prikaz rezultata nalazi se na slici 23. Forma grafa jednaka je kao i kod prvog testiranja.



Slika 23 *Grafički prikaz rezultata drugog testiranja*

Iz rezultata sasvim je jasno da korištenje priručne memorije u svim slučajevima ubrzava obradu u prosijeku za 50%. Razlog zbog kojeg su rezultati drugačiji od prvog testiranja je taj što je odabrani skup podataka raspršeniji u odnosu na skup korišten kod prvog testiranja i zbog toga daje općenitiju sliku. Iz rezultata je vidljivo da strategija particioniranja pomoću Q-stabla daje najbolje rezultate uz korištenje priručne memorije dok se particioniranje pomoću KDB-stabla pokazalo optimalnim u slučaju kad ne koristimo pričuvnu memoriju. Također, zaključujemo da korištenje indeksa u svim slučajevima uvjetuje konačnom rješenju te je iz rezultata moguće očitati da se određene strategije particioniranja slažu s pojedinom vrstom indeksa, npr:

- particioniranje pomoću KDB-stabla i indeks pomoću Q-stabla,
- particioniranje pomoću Q-stabla i indeks pomoću Q-stabla,
- particioniranje pomoću R-stabla i indeks pomoću R-stabla (uz korištenje priručne memorije).

Zaključak

Iz eksperimentalne evaluacije programskog rješenja moguće je zaključiti da je povoljno koristiti pričuvnu memoriju za pohranu podataka nad kojima se stalno vrše upiti uz uvjet da je taj skup podataka dovoljno malen kako ne bi prepunio radnu memoriju računala.

Također, rezultati testiranja pokazali su da su strategije particioniranja pomoću struktura R-stabla i Q-stabla učinkovite u slučaju kad se koristi pričuvna memorija, dok je strategija pomoću KDB-stabla učinkovita u slučaju kad ju ne koristimo.

Tijekom pisanja programskog rješenja, evaluacija se izvodila na prijenosnom računalu na četiri dretve (1 za upravljački proces, 3 za procese radnika). Rezultati tih testiranja bili su slični kao i oni dobiveni testiranjem na raspodijeljenom sustavu. Razlog tome je dodatni posao potreban za pokretanje procesa radnika te prijenosa podataka s računala na računalo. Također, prilikom testiranja na prijenosnom računalu Kafka brokeri su također bili procesi pokrenuti na istom računalu te slanje poruka nije zahtijevalo pristup mreži.

Stoga možemo zaključiti da bi se utjecaj korištenja raspodijeljenog sustava osjetio tek za veliki skup geoprostornih pretplata, dok bi u ovom slučaju (skup od 191 geoprostorne pretplate) bolje rezultate davala centralizirana obrada.

Literatura

[1] Podnar Žarko, I; Pripužić, K; Lovrek, I; Kušek, M. Raspodijeljeni sustavi: Radna inačica udžbenika. v.1.3. FER, 2017.

[2] „Definition – What does Distributed System mean?“, *Techopedia* – dostupno na: https://www.techopedia.com/definition/18909/distributed-system [02.05.2019.]

[3] „What is Big Data?“, *Oracle Big Data* – dostupno na: https://w ww.oracle.com/big-data/guide/what-is-big-data.html [10.05.2019.]

[4] „Definition – What does Apache Spark mean?“, *Techopedia* – dostupno na: https://www.techopedia.com/definition/30113/apache-spark [21.05.2019.]

[5] „Apache Kafka Introduction“, *Apache Kafka Documentation* – dostupno na: https://kafka.apache.org/intro [21.05.2019.]

[6] „GeoSpark Introduction“, *DataSystemsLab* – dostupno na: http://datasystemslab.github.io/GeoSpark/ [21.05.2019.]

[7] Format. H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub „RFC7946 The GeoJSON“, Kolovoz, 2016. – dostupno na: https://tools.ietf.org/html/rfc7946 [01.06.2019.]

[8] „GeoTools Geometry“, GeoTools User Guide – dostupno na: http://docs.geotools.org/stable/userguide/library/jts/geometry.html [01.06.2019.]

[9] Weisstein, Eric W. „Hilbert Curve.", MathWorld - *A Wolfram Web Resource* – dostupno na: http://mathworld.wolfram.com/HilbertCurve.html [02.06.2019.]

[10] Weisstein, Eric W. „Voronoi Diagram.", *MathWorld - A Wolfram Web Resource* – dostupno na http://mathworld.wolfram.com/VoronoiDiagram.html [02.06.2019.]

[11] Guttman, A. „R-Trees. A dynamic index structure for spatial searching“, Berkly University of California, 1984. – dostupno na: http://www-db.deis.unibo.it/courses/SI-LS/papers/Gut84.pdf [02.06.2019.]

[12] Weisstein, Eric W. „Quadtree.", MathWorld - *A Wolfram Web Resource* – dostupno na: http://mathworld.wolfram.com/Quadtree.html [02.06.2019.]

[13] John T. Robinson, „The K-D-B-Tree : a search structure for large multidimensional dynamic indexes“, Carnegie Mellon University, 1981. dostupno na: https://klevas.mif.vu.lt/~algis/dsax/K-D-B-Tree.pdf [02.06.2019.]

Sažetak

Filtriranje geoprostornog toka podataka korištenjem platforme Apache Spark

Cilj ovog rada je stvaranje učinkovitog raspodijeljenog sustava za filtriranje geoprostornog toka podataka korištenjem platforme Apache Spark. Arhitektura korištena u implementaciji programskog rješenja naziva se „objavi-pretplati“. Objave u sustavu predstavljaju poruke formata GeoJSON koje sadrže geoprostornu značajku (točku, liniju, poligon) te dodatne informacije. Pretplate u sustavu predstavljaju poligoni u prostoru uz koje vežemo dodatne značajke. Tok podataka implementiran je pomoću alata Apache Kafka. Za konzumaciju toka podataka koristi se Spark Streaming koji tok obrađuje u dijelovima koji se mogu primiti u određenom vremenskom intervalu (engl. *micro-batch*). Za filtriranje geoprostornih podataka korištena je programska knjižnica GeoSpark koja dodatno koristi knjižnice Java Topology Suite i Geotools. Učinkovito filtriranje prostornih podataka ostvareno je pomoću particioniranja skupa pretplata pomoću različitih strategija particioniranja. Izlaz sustava je poruka u formatu JSON koja sadrži identifikator objave i pripadne pretplate. U evaluaciji rješenja isprobane su strategije particioniranja pomoću Hilbertove krivulje, KDB-stabla, Q-stabla i R-stabla. Dodatno, za poboljšanje rezultata ispitano je i korištenje geoprostornih indeksa ostvarenih pomoću R-stabla i Q-stabla. U evaluaciji je također ispitan utjecaj korištenja pričuvne memorije za spremanje skupa pretplata.

Ključne riječi: raspodijeljeni sustav, Apache Spark, objavi-pretplati, geoprostorni tok, GeoJSON, Apache Kafka, Spark Streaming, GeoSpark, Java Topology Suite, Geotools, particioniranje, Hilbertova krivulja, Voronoijev dijagram, R-stablo, Q-stablo, KDB-stablo, prostorni indeks

Summary

Filtering of Geospatial Data Streams using Apache Spark Platform

The goal of this paper is to create an efficient distributed system for filtering of geospatial dana streams using the Apache Spark platform. Architecture used in the implementation is based on publish-subscribe pattern. Incoming messages in system carry a geospatial feature (eg. point, line or polygon on Earth) and some additional data formatted with GeoJSON specification. Geospatial subscriptions represent a set of polygons on Earth with some additional data. Geospatial data stream is implemented with Apache Kafka tool. Spark Streaming platform is used for consuming the incoming messages. It handles messages in portions (micro-batches) consumed within a time interval. GeoSpark library along with Java Topology Suite and GeoTools are used for processing the geospatial data. The effectiveness of geospatial filtering is improved with partitioning of the subscriptions dataset by using a different partitioning strategies. The system output consists of the identifiers of both messages corresponded subscriptions. Partitioning strategies used in the experiment are ones based on Hilbert curve, KDB-tree, Q-tree and R-tree. Additionally, evaluations examined the impact of using geospatial indexes based on Q-tree and R-tree and also the impact of using the cache functionality for persisting the subscription dataset in the memory of worker processes.

Keywords: distributed system, Apache Spark, publish-subscribe, geospatial data stream, GeoJSON, Apache Kafka, Spark Streaming, GeoSpark, Java Topology Suite, Geotools, partitioning, Hilbert curve, Voronoi diagram, R-tree, Q-tree, KDB-tree, geospatial index

Prilozi

Prilog 1. Konfiguracijska datoteka `conf.properties`

```
subscriptions.location=data/subscriptions/subscriptions.geojson
stream.kafka.brokers.in=broker01.streamslab.fer.hr:9092,broker02.strea
mslab.fer.hr:9092,broker03.streamslab.fer.hr:9092
stream.kafka.brokers.out=broker04.streamslab.fer.hr:9092
stream.kafka.groupId=geospatial_index
stream.kafka.topics.in=geospatial_index_in
stream.kafka.topic.out=geospatial_index_out
spark.appName=GeospatialIndexDistributed
spark.master.url=yarn
spark.cache=false
geospark.spatialIndex=QUADTREE
logging.level=WARN
spark.streaming.batchTime=500
spark.streaming.timeout=60
```

Prilog 2. Konverzija CSV formata u GeoJSON format

```
import csv, json
from geojson import Feature, FeatureCollection, Point

features = []
with open('input_csv.csv', newline='') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')

    counter = 0
    for Accident_Index, Longitude, Latitude in reader:
        latitude, longitude, id = 0.0, 0.0, 0

        try:
            latitude, longitude = map(float, (Latitude, Longitude))
            id = int(Accident_Index)
        except:
            continue

        features.append(
            Feature(
                geometry = Point((longitude, latitude)),
                properties = {
                    'id': id
                }
            )
        )

        counter += 1
        if counter > 1000000:
            break

collection = FeatureCollection(features)
with open("input1000000.geojson", "w") as f:
    f.write('%s' % collection)
```

Prilog 3. Automatizirano testiranje

```
import os
from time import sleep


producerScript = """ python ~/scripts/testProducer.py
\"data/input/input10000.geojson\" -b
\"broker01.streamslab.fer.hr:9092\" -t \"geospatial_index_in\" """

sparkSubmitScript = """ spark-submit --class
hr.fer.ztel.geospatial.GeospatialIndexDistributed --master yarn --
deploy-mode cluster --driver-cores 4 --num-executors 4 --executor-
cores 2 --executor-memory 2G --files ~/configs/{}
~/jars/final_v3/geospatial-index-distributed-assembly-0.1.jar {}"""

for file in os.listdir("configs"):
    if not file.endswith(".properties"):
        print "Skipping file: " + file

        continue


    conf = file.split(".")[0]
    pid = os.fork()

    if pid == 0:
        sleep(20)
        print "Test producer started"

        if os.system(producerScript) != 0:

            raise Exception("Failed to run testProducer.py")

        exit()


    print "Spark job started"

    if os.system(sparkSubmitScript.format(file, file)) != 0:

        raise Exception("Failed to submit a job.")


    finished = os.waitpid(0, 0)
```

Prilog 4. Kreator testnih objava

```
from argparse import ArgumentParser, FileType
from kafka import KafkaProducer
from time import time, sleep


parser = ArgumentParser(prog="kafka-test-producer",
description="Simple Kafka producer which reads data from file and
pushes it on the topic")

parser.add_argument("input_file", type=FileType("r"), help="Input file
- one message per line.")

parser.add_argument("-b", "--bootstrap_servers", nargs='*',
default=["localhost:9092"], help="List of Kafka bootstrap servers.")

parser.add_argument("-t", "--topics", nargs='*', default=[],
help="List of Kafka topics.")


args = parser.parse_args()

producer = KafkaProducer(bootstrap_servers=args.bootstrap_servers,
value_serializer=lambda x: x)


for message in args.input_file:

    for topic in args.topics:

        producer.send(topic, value=message)


args.input_file.close()
```