Univerzitet u Nišu,
Elektronski fakultet u Nišu,
Katedra za računarstvo

Seminarski rad iz predmeta Sistemi za obradu i analizu velike količine podataka
i Naprednih operativnih sistema:

Analiza velike količine podataka iz saobraćaja

Matija Mitić 448

Igor Đorđević 449

Dejan Dimčić 450

Sadržaj

1. Uvod 3
2. Komponente sistema 4
   2.1. Simulation of Urban Mobility (SUMO) 4
   2.2. SUMO TRAffic Control Interface (TraCI) 4
   2.3. SUMO Netcovert 4
   2.4. CrowdNav 4
   2.5. Apache Zookeeper 5
   2.6. Apache Kafka 5
   2.7. Apache Spark 5
      2.7.1. Batch model obrade 6
      2.7.2. Streaming model obrade 6
      2.7.3. Spark MLlib 6
   2.8. Vert.x 7
   2.9. Docker 7
3. Opis sistema 8
   3.1. Priprema podataka 8
   3.2. Offilne podsistem 9
   3.3. Online podsistem 10
4. Deployment sistema 16
   4.1. Instrukcije za pokretanje sistema 16
5. Zaključak 20
6. Reference 21

1. Uvod

Saobraćajna gužva je jedan od najčešćih problema u gradu. Smanjenje zastoja doprinosi većoj mobilnosti ljudi koja očigledno dovodi do poboljšanja kvaliteta života i ekonomskih sposobnosti građana. Postoje mnogo rešenja koja su predložena kako bi se saobraćajne gužve smanjile kao što su smanjenje broja privatnih automobila u najvećoj meri menjanjem politike (povećanjem poreza na gorivo, luksuz, kubikažu ili finansijski podsticaj za upotrebu javnog prevoza), optimizacijom signalizacije glavnih raskrsnica ili proširenjem prometnih saobraćajnica novim trakama. Koliko god se problemi manifestovali istovetno, saobraćajnim gužvama i zastojima, svaki grad ima drugačiji uzrok koji do toga dovodi, a da bi se problem rešio na pravi način, uzroke je potrebno prepoznati, obeležiti i na njih preventivno delovati. Na primer, gradu Nišu nedostaju saobraćajnice koje bi povezivale susadne delove grada i time rasteretile glavne tačke, uska grla, u kojima se najčeše javljaju gužve. Međutim, prirodne ili namerne katastrofe jesu neke od sitacija koje deluju istovetno na svaki grad pa tako i saobraćaj unutar njega, tako da je i njihova kontrola veoma bitna.

Uzroke saobraćajnih gužvi, u vanrednim uslovima, je ponekad teško uočiti. Međutim, simuliranjem saobraćaja unutar saobraćajne mreže jednog grada moguće je uočiti problematična mesta u kojima su zastoji česti, a broj vozila veliki. Praćem ruta vozila koja su se našla u zastoju, može se uočiti da pojedini delovi grada nisu adekvatno povezani sa drugim i da to dovodi do velikih saobraćajnih gužvi i to gotovo uvek na istim mestima. Radi poboljšavanja metoda za uočavanje ovakvih problema, ovaj rad se bavi njihovom implementacijom i proširenjem postojećih mogućnosti, kako bi iznete analize bile podjednako primenljive i u realnim uslovima sa realnim vozilima.

Takođe, kao još jedan izvor podataka za istraživanje uzroka zastoja mogu poslužiti poruke objavljene na Twitter mreži. U slučaju nekog zastoja moguće je analizirati „tweet“ poruke koje su nastale na mestu zastoja koje možda mogu biti pokazatelj zbog čega postoji zastoj baš na tom mesti. Može biti neka nesreća na putu, odron, čekanje voza, ili pak neka od kulturno-umetničkih manifestacija baš u blizini tog mesta.

U drugom poglavlju ovog rada su ukratko opisane sve komponente i gotova rešenja korišćena u sistemu za analizu velike količine podataka iz saobraćaja. Takođe su i najznačajnije funkcionalnosti važne za sam projekat opširnije obrazložene. U trećem poglavlju detaljno je opisan sistem za analizu podataka iz saobraćaja, kako u realnom vremenu, tako i u offline režimu rada gde su korišćeni unapred pripremljeni podaci. Četvrto poglavlje opisuje deployment sistema dok u poslednjem, petom poglavlju su izneti zaključci i rešenja dobijena prethodnom analizom i predlozi za unapređenje postojećeg sistema.

2. Komponente sistema

U nastavku je dat pregled komponenti sistema za analizu velike količine podataka iz saobraćaja.

2.1. Simulation of Urban Mobility (SUMO)

SUMO je besplatan, portabilan i open source paket za simulaciju saobraćaja dizajniran tako da može da opsluži velike mreže. Omogućava modelovanje saobraćajnih sistema uključujući obična vozila, javni prevoz i pešačke zone. SUMO paket uključuje i mnoge alate koji pomažu u modelovanju, rešavajući probleme kao što su rutiranje vozila, vizuelizacija, importovanje mreže ili kalkulacija emisije podataka. Funkcionalnosti SUMO paketa mogu biti proširene uz pomoć različitih API-a koji omogućavaju kontrolisanje simulacije. [1][2]

Tokom razvoja projekta, u razmatranje su uzeti i isprobani i drugi sistemi za simulaciju saobraćaja, poput MATSim-a [3] ili PTV Vissim-a [4], međutim, zbog pomoćnih paketa i funkcionalnog API-a, SUMO simulator se pokazao kao najjednostavnije i funkcionalnije rešenje.

2.2. SUMO TRAffic Control Interface (TraCI)

SUMO TraCI je API koji omogućava automatsko upravljanje simulacijom, pribavljanje informacija o simuliranim objektima i manipulisanje ponašanjem tih objekata. TraCI je zasnovan na TCP klijent/server protokolu gde se SUMO ponaša kao server kome je moguće pristupati. API je moguće koristiti uz pomoć sledećih programskih jezika: Python, Java, .NET, C++. [5]

Projekat je zasnovan na Python API-u koji ima najveću podršku kroz dokumentaciju, primere i slučajeve korišćenja.

2.3. SUMO Netcovert

Sumo Netconvert je alat iz SUMO paketa koji predstavlja konzolnu aplikaciju koja digitalnu mrežu puteva sa različitih izvora transformiše u mrežu odgovarajućeg formata koju će koristiti SUMO simulator saobraćaja [6]. Netconvert mora da ima bar jedan ulazni parametar, ulazni fajl koji sadrži mrežu puteva za konvertovanje. Takođe, postoji i veliki skup opcionih parametra koji podešavaju i poboljšavaju konvertovanje mreže.

Konkretno u projektu, ulazni fajl bila je mreža puteva skinuta sa OpenStreetMap-a u .osm formatu. Netconvert je upotrebljen uz dosta parametara da bi se dobila odgovarajuća mreža, a ceo postupak i parametri su opisani u poglavlju 3.2.

2.4. CrowdNav

CrowdNav je projekat koji vrši simulaciju zasnovanu na SUMO i TraCI-ju i koji poseduje podesiv ruter koji zadaje putanje vozilima, a koji dobija instrukcije putem Kafke i to u toku trajanja simulacije. Takođe, tokom trajanja simulacije, prikupljeni podaci se šalju putem Kafke na obradu ili se lokalno čuvaju u CSV fajlu. [7]

CrowdNav je razvijan u Python-u i koristi Python verziju TraCI-a. Srž same simulacije jeste petlja u kojoj svaka iteracija predstavlja jedan trenutak (tick) u kome se vozila (ili drugi entiteti) pomeraju i menjaju stanje saobraćaja. Sva vozila na početku simulacije dobijaju rute koje treba da izvrše od strane rutera koji može biti „običan“ ili „pametan“. „Običan“ ruter rutira oko 80% (podesiva vrednost) vozila uključena u simulaciju tako što zadaje slučajne putanje. „Pametan“ ruter je podesiv uz pomoć raznih parametara tako da kao output daje putanje u kojima može biti manje zastoja. Kada vozilo završi rutu, ruter generiše novu putanju.

Inicijalno, CrowdNav je dizajniran za rad sa RTX komponentom [8] i putem TraCI-a je pretplaćen na vozila koja završavaju svoju putanju. Informacije o ovim vozilima se čitaju svakog trenutka (tick-a) u glavnoj petlji. Oba rutera („običan“ i „pametan“) po dodeljivanju nove putanje vozilu, računaju i koliko vremena (tick-ova) je potrebno vozilu da stigne na odredište (ukupan put / dozvoljena brzina). Sa druge strane, za svako vozilo se pamti trenutak polaska ka cilju kao i trenutak dolaska na cilj, i tako dobija ukupno vreme provedeno u putu. Deljenjem ove dve vrednosti (stvarnog vremena i predviđenog) dobija se vrednost koja ukazuje o stepenu zagušenja saobraćajne mreže. Za svako vozilo koje završi svoju rutu, ova vrednost se izračunava i šalje putem kafke na topic definisan u konfiguracionom fajlu. RTX komponenta čita ove vrednosti sa topic-a i vrši analizu, odnosno određuje stepen dobrote saobraćaja. Na osnovu ove analize, RTX šalje podatke putem Kafke nazad do CrowdNav-a koji predstavljaju vrednosti parametra „pametnog“ rutera. Na ovaj način, ruter se dodatno podešava u toku same simulacije i utiče na saobraćaj (npr. povećava se stepen slobode pri izboru ruta, tako da je moguće izabrati i rute koje bi inicijalno bile okarakterisane kao neefektivne).

U projektu su iskorišćene sve pogodnosti CrowdNav simulatora poput rutera koji vozilu dodeljuje novu rutu po završetku prethodne ili implementiranog Kafka konektora ili CSV Logger-a. Glavne izmene su definisane u konfiguracionom fajlu (imena Kafka topic-a, putanje do mreže) i implementirane u glavnoj petlji koja simulira saobraćaj, tako da se preko Kafke šalju ili u CSV upisuju podaci od značaja za sam projekat.

2.5. Apache Zookeeper

Zookeeper je centralizovani servis za održavanje konfiguracionih informacija i imena (key-value parovi) pružajući distribuiranu sinhronizaciju i grupne servise koji se u nekom obliku koriste u distribuiranim aplikacijama. Arhitektura servisa podržava visoku dostupnost (high avaiability) u redundantnim servisima koji ga koriste i omogućava developer-ima da se fokusiraju na razvoj aplikacione logike i ne brinu o distribuiranoj prirodi same aplikacije. [10]

Mnogi distribuirani sistemi u samom jezgru koriste prednosti koje nudi Zookeeper. Konkretno, Kafka koristi Zookeeper za praćenje statusa čvorova, topic-a, particija…

2.6. Apache Kafka

Apache Kafka je open source platforma za obradu streaming podataka razvijena od strane Apache Software fondacije, napisana u Scala i Java programskim jezicima. Kafka obezbeđuje jedinstvenu platformu sa visokim protokom i niskom latencijom pri obradi podataka u realnom vremenu. Sloj za skladištenje podataka može se opisati kao masovni skalabilni red poruka projektovan kao niz transakcija što ga čini pouzdanim u enterprise distribuiranim sistemima. Kafka ima mogućnost povezivanja sa eksternim sistemima (za čitanje/upis podataka) pomoću Kafka Connect i Kafka Streams biblioteka. [9]

CrowdNav sistem koristi kafka-python modul pomoću koga se kreira Kafka Producer i Consumer koji čitaju i upisuju podatka iz i u red poruka, dok su sva podešavanja postavljena u konfiguracionom fajlu. Sa druge strane, u Spark (streaming) projektima se koristi Kafka Stream biblioteka pomoću koje se takođe kreiraju Kafka Consumer i Producer koji čitaju poruke sa topica i prosleđuju nove na naredni topic.

2.7. Apache Spark

Apache Spark je open source framework za procesiranje podataka na klasteru koji pruža interfejs za programiranje celokupnih klastera sa ugrađenom paralelizacijom i obradom grešaka.

Spark je sistem za batch obradu sa mogućnostima stream obrade. Sagrađen je korišćenjem mnogih istih principa kao i Hadoop-ova MapReduce komponenta, Spark se fokusira prvenstveno na ubrzavanje batch

obrade, time što omogućava izračunavanje u memoriji i optimizaciju obrade. Spark se može koristiti na samostalnom klasteru (ako je uparen sa dobrim slojem za skladištenje) ili može da se poveže sa Hadoop-om kao alternativa za MapReduce.

Offline podsistem sistema za analizu podataka iz saobraćaja koristi Spark batch model obrade, dok se online podsistem oslanja na Spark streaming model. U online sistemu koristi se i Spark MLlib (Machine learning library) biblioteka ukratko opisana u odeljku 2.7.3.

2.7.1. Batch model obrade

Za razliku od MapReduce-a, Spark obrađuje sve podatke u memoriji, komunicira sa slojem za skladištenje samo da bi inicijalno učitavao podatke u memoriju i da bi na kraju sačuvao konačne rezultate. Svim međurezultatima se upravlja u memoriji. Dok procesiranje u memoriji značajno doprinosi na brzini, Spark je brži i na zadacima vezanim za upis na disk zbog optimizacije koja se može postići analizom kompletno svih taskova unapred. Ovo se postiže stvaranjem Direktnih Acikličnih Grafova ili DAG-ova koji predstavljaju sve operacije koje moraju biti izvršene, podatke nad kojima će se raditi, kao i odnose između njih, dajući time procesoru obrade veću sposobnost da inteligentno koordiniše obradu. Za implementaciju batch obrade podataka u memoriji, Spark koristi model pod nazivom Resilient Distributed Datasets ili RDDs. Ovo su nepromenjive strukture koje postoje unutar memorije koje predstavljaju skupove podataka. Operacije nad RDD-om proizvode nove RDD-ove. Svaki RDD ima reference na svoje roditeljske RDD-ove i može se ispratiti povezanost sve do podataka na disku. U suštini, RDD su način za Spark da održava toleranciju na greške bez potrebe za upisivanjem na disk nakon svake operacije.

2.7.2. Streaming model obrade

Spark sam po sebi nema mogućnost čiste obrade tokova podataka, zato je kreiran Spark Streaming. Da bi se ta dva koncepta složila, Spark implementira koncept koji se zove micro-batches. Ova strategija je dizajnirana da tretira tok podataka kao seriju vrlo malih batch-eva jer se na malim batch-evima može primeniti logika samog nativnog Spark batch modela. Spark Streaming funkcioniše tako što baferuje stream podataka u delovima sekunde inkrementalno. Oni se dalje šalju kao mali fiksni skupovi podataka za batch obradu. U praksi, ovo funkcioniše prilično dobro, ali to vodi do drugačijeg profila od pravih sistema za obradu streama. Spark može obrađivati iste skupove podataka značajno brže zbog svoje strategije izračunavanja u memoriji i naprednog DAG planiranja. Još jedna od glavnih prednosti Sparka je njegova svestranost. Može biti postavljen kao samostalni klaster ili integrisan sa postojećim Hadoop klasterom i može da obavlja i batch i stream obradu. Spark takođe ima ekosistem biblioteka koji se može koristiti za mašinsko učenje, interaktivne upite itd. Spark taskovi su skoro univerzalno prihvaćeni da budu lakši za pisanje nego MapReduce, što može biti od značaja za produktivnost. Spark je odlična opcija za one sa različitim radnim opterećenjima obrade. Što se tiče tolerancije na greške, Spark Streaming obrađuje mikro-batcheve na različitim radnim čvorovima. Svaki mikro batch operacija može biti uspešna ili ne. Sa neuspehom, mikro-bacth se jednostavno može ponovno obraditi, jer su microbatchevi neizmenjivi i perzistovani i ovo predstavlja exactly once (samo jednom) model garancije isporuke.

2.7.3. Spark MLlib

Spark Mllib je Spark biblioteka koja pruža podršku za korišćenje algoritama mašinskog učenja. Cilj biblioteke je jednostavna, praktična ali i skalabilna upotreba ovih algoritama. Biblioteka pruža alate kao što su:

- ML Algoritmi: opšti algoritmi učenja: klasifikacija, regresija, klasterizacija
- Izdvajanje karakteristika (eng. Featurization)
- Povezivanje (eng. Pipelines)

- Perzistentnost: čuvanje i učitavanje modela, poveznica, algoritama
- Druge usluge: linearna algebra, statistika, obrada podataka [16]

2.8. Vert.x

Vert.x je framework zasnovan na događajima koje distribuira putem zajedničke magistrale do servisa koji te događaje obrađuju.Vert.x, poput Node.js-a, koristi neblokirajući single thread model za obavljanje poslova.

Vert.x pored drugih komponenti, koristi SockJS magistralu događaja koja omogućava Web aplikacijama da bidirekciono komuniciraju sa Vert.x magistralom putem Web soketa, koji omogućavaju kreiranje Web aplikacije u realnom vremenu sa push funkcionalnošću. Na ovaj način, Web server komponente straming sistema komuniciraju sa Web aplikacijom koja dobijene podatke vizualizuje. [12]

Upravo opisana funkcionalnost Vert.x framework-a je bila od velikog značaja za sistem za analizu podataka iz saobraćaja, jer podaci prethodno obraćeni Spark job-ovima, a potom primljeni preko Kafke, bivaju automatski dovedeni to Web klijenta koji ih vizualizira.

2.9. Docker

Docker [13] je softverska tehnologija koja olakšava kreiranje, deploy i pokretanje aplikacija korišćenjem kontejnera. Kontejneri olakšavaju da se aplikacija spakuje sa svim potrebnim komponentama i zavisnim bibliotekama. Tako spakovana aplikacija se distribuira kao jedan paket. Na taj način autor aplikacije je siguran da će se njegova aplikacija uspešno izvršavati u izolovanom okruženju na bilo kojoj Linux mašini. Bez obzira na to šta je već instalirano na mašini ili kakve su postavke sistema, spremljena aplikacija će se izvršavati na identičan način, kako je programer postavio.

Docker dosta nalikuje virtuelnoj mašini, međutim ključna razlika je da ne postoji poseban operativni sistem za izvršenje aplikacije. Izvršenje se oslanja na već postojeći Linux Kernel koji korisnik poseduje na računaru. Tako da postoji velika prednost u odnosu na virtuelne mašine u pogledu povećanja performansi i u redukciji veličine aplikacije.

Još je važno napomenuti da je Docker open-source tehnologija i da se u poslednje vreme ubrzano razvija. Takođe pored klasičnih Linux Docker kontejnera od nedavno postoje i Windows kontejneri sa sličnim principom rada.

3. Opis sistema

Sistem za analizu saobraćaja ima za cilj lokalizaciju, analizu i vizuelizaciju zagušenja u saobraćaju u realnom vremenu. Sistem se sastoji iz dva podsistema, offline i online. Rezultat online sistema jeste grafički prikaz zagušenja saobraćaja na mapi, odnosno tačaka nagomilavanja vozila koja su u zastoju duže od jednog minutra. Takođe, sistem za svaku tačku prikazuje i informacije o tome, kada je zastoj započet, kada se završio i koliko maksimalno vozila se nalazili u zastoju. Rezultat offline sistema jeste CSV dokument koji sadrži prosečno vreme koje su vozila provela u segmentu mreže tokom jednog minuta.

Kao dodatak realizovan je analiza poruka sa društvene mreže Twitter, tj. filtriranje poruka nastalih u blizini zastoja kako bi poslužili kao moguć izvor podataka za utvrđivanje uzroka zastoja. (Odeljak 3.4)

3.1. Priprema podataka

Kao što je već naglašeno, CrowdNav projekat nudi izuzetne pogodnosti i sličnosti sa traženim funkcionalnostima, tako da je njegovo razumevanje i modifikacija bila od ključne važnosti. CrowdNav je inicijalno podešen za rad nad mrežom (mapom) grada Eichstätt iz Nemačke. Da bi se rezultati analiza uspešno evaluirali, CrowdNav je podignut nad mrežom grada Niša.

Mapa Srbije preuzeta je sa sledećeg linka: http://download.geofabrik.de/europe/serbia.html

Mapa je dalje obrađivana (crop-ovana) uz pomoć JOSM editora. [14]

Da bi se kreirala mreža odgovarajuća za rad SUMO simulatora, iskorišćen je alat Netconvert opisan u poglavlju 2.3. Uz komandu **`netconvert`**, a da bi mreža zadovoljavajuće funkcionisala u simulatoru, morali su biti uključeni i sledeći parametri:

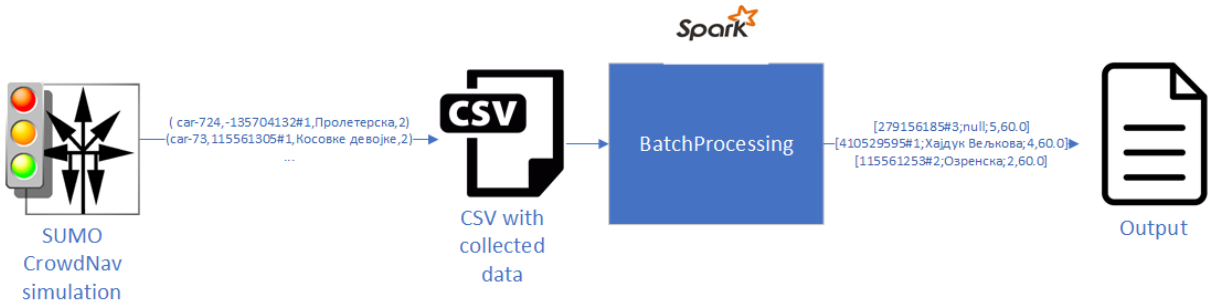
- `--osm` – putanja do .osm fajla koji se konvertuje (npr. map.osm)
- `-o` – putanja na kojoj će se generisati izlazni fajl, mreža koju će SUMO koristiti (npr. map.net.xml)
- `--geometry.remove` – briše čvorove koji nemaju poveznice
- `--roundabouts.guess` – omogućava zaokruživanje loših spojeva
- `--ramp.guess` – predviđa postojanje rampi ukoliko nisu definisane u .osm fajlu
- `--junctions.join` – omogućava spajanje poveznica u raskrsinicu (.osm fajl može imati neprecizne spojeve
- `--tls.guess-signals` – pretvara čvorove na većim raskrsnicama u semafore
- `--tls.discard-simple` – ne pretvara čvorove na svim raskrsnicama u semafore
- `--tls.join` – pokušava klasterovanje semafora
- `--no-internal-links` – isključuje interne segmente
- `--keep-edges.by-vclass` – čuva segmente zadatog tpa (npr. passenger)
- `--remove-edges.by-type` – isključuje segmente zadatog tipa (npr. highway.track, highway.services, highway.unsurfaced)
- `--output.street-names` – uključuje imena segmenata pročitana iz .osm fajla

Izlaz instrukcije je mreža odgovarajućeg formata i čija putanja se mora uneti u konfiguracioni fajl CrowdNav projekta.

3.2. Offilne podsistem

Zadatak offline podsistema jeste analiza prikupljenih podataka o vozilima tako da se za svaki segment putne mreže odredi dinamičko vreme puta po minutima kao prosečno vreme prolaska svih vozila tim segmentom u datom minutu.

Arhitektura offline podsistema predstavljena je na slici 3.2.1. Podsistem je jednostavan i sastoji se od modifikovanog CrowdNab SUMO simulatora i jednog Spark job-a.



Slika 3.2.1 Arhitektura offline podsistema

U CrowdNav projektu, osim mreže (mape), izmenjena je i glavna petlja simulatora tako da se u svakoj iteraciji (tick-u) za svako vozilo uneto u mrežu, putem TraCI API-ja, pribavlja segment mreže na kome se to vozilo nalazi. Zatim se, iz pomoćne strukture, za svaki Id segmenta mreže pribavlja i ime ulice u kojoj se taj segment nalazi. Uz pomoć `CSVLogger` klase CrowdNav projekta, u CSV fajl se upisuju Id vozila, Id segmenta mreže, ime ulice, i tick.

Kao test primer, generisan je fajl veličine 16 MB koji sadrži pomenute informacije u periodu od oko 300 tick-a i čiji je red prikazan:

```
car-727,-115561179#11,Драгише Цветковића,2
```

Ovaj fajl predstavlja ulaz za Spark job koji će procesirati ulazne informacije i kao izlaz dati odgovor na traženi zadatak. Da bi se odredilo prosečno zadržavanje vozila po segmentu mreže u toku jednog minuta, potrebno je odrediti ukupna zadržavanje kao i broj svih vozila u tom minutu. Zatim na osnovu ovih vrednosti za svaki segment mreže izračunati prosečno zadržavanje vozila i sortirati po opadajućem redosledu.

`BatchProcessing` je klasa čija se glavna metoda bavi procesiranjem offline podataka. Najpre se ulazni fajl učitava i kreira JavaRDD objekat, da bi se od njega kreirala dva JavaPairRDD objekta. Pri ovoj konverziji, izvršena je i jedna transformacija kolone **tick**. Prostom *div* operacijom, svaki tick je transformisan u minut u kome se tick desio (Npr. tick 39 se dogodio u 0-tom minutu jer je 39 / 60 = 0). Pored ove transformacije, svako pojavljivanje vozila u jednom segmentu mreže u toku jednog minuta je sumirano i kao konačna vrednost dobijen ukupan broj pojavljivanja tog vozila. Kako se jedno vozilo oglasi samo jednom u CrowdNav projektu u toku jednog ticka, ova vrednost zapravo predstavlja broj tick-ova koje je vozilo provelo u jednom segmentu mreže. JavaPairRDD objekat koristi strukturu ključ-vrednost i stoga je prirodniji za korišćenje u zadatom problemu od običnog JavaRDD objekta. Oba objekta će imati isti ključ, a to je kombinacija id-a segmenta, imena ulice i broja minuta, dok će vrednosti u prvom objektu biti suma ukupnog zadržavanja vozila, a u drugom, ukupan broj pojavljivanja vozila po svakom ključu. Kako oba JavaPairRDD objekta imaju isti ključ, pogodno je izvršiti operaciju *join* i napraviti jedinstveni JavaPairRDD ojekat sa obe vrednosti pod istim ključem. Prostom *map* operacijom izvršava se računanje prosečne vrednosti za svaku vrednost kombinovanog JavaPairRDD objekta, a kreiranjem Dataset-a i SQL konteksta, putem SQL upita lako se sortiraju vrednosti u opadajućem redu.

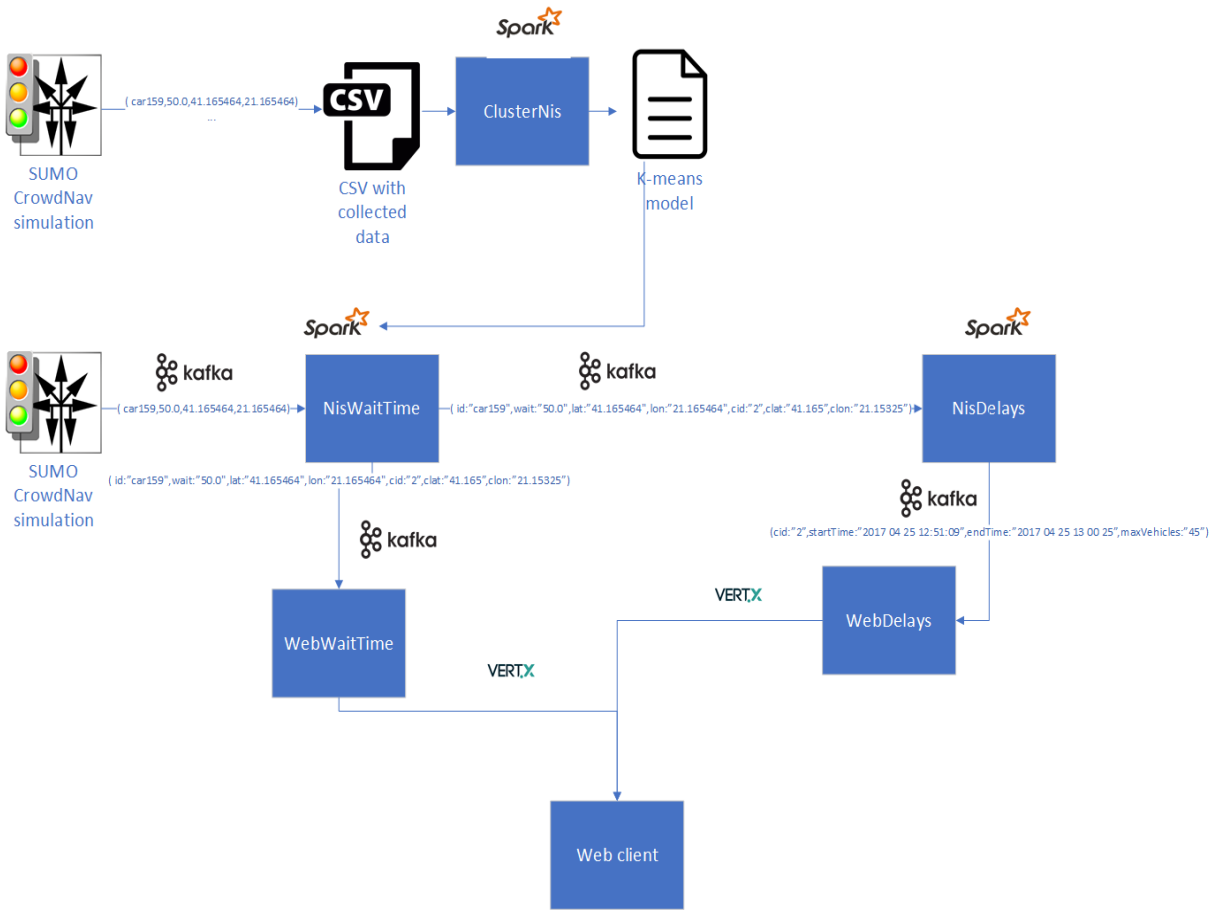
Konačan rezultat je prikazan u nastavku:

```
[-417228513#1;Књажевачка;5,57.8125]
[221857697#0;Војислава Илића;1,57.0]
[266143652#2;Синђелићев трг;3,56.592594]
[512824050#10;Књажевачка;2,55.833332]
[-38375827#0;Краља Стефана Првовенчаног;3,55.695652]
[115561094#1;Школска;4,55.0]
[-38393860#0;Мије Петровића;3,54.68182]
[266143652#2;Синђелићев трг;4,54.32143]
[-459456559#1;Станоја Главаша;0,54.0]
[-38375827#0;Краља Стефана Првовенчаног;5,53.708332]
[140789744#4;Књажевачка;2,53.47619]
[266143652#2;Синђелићев трг;5,52.310345]
[-417309446#0;Булевар 12. фебруара;5,52.0]
[-115561279#0;Гламочка;1,51.166668]
[-103426225#4;Зетска;2,51.0]
[-87494509#0;Хајдук Вељкова;3,50.894737]
[-417228513#4;Књажевачка;4,50.36]
[-38393860#0;Мије Петровића;2,50.217392]
[-109749483#2;Габровачки пут;3,50.0]
[512824050#10;Књажевачка;5,49.866665]
```

3.3. Online podsistem

Zadatak online podsistema bio je analiza podataka o vozilima u realnom vremenu, odnosno detekcija vozila koja su u zastoju kao i klasterovanje tih vozila. Takođe, za svaki klaster potrebno je u realnom vremenu određivati kada je zastoj detektovan, kada je završen i broj vozila koja su u zastoju učestvovala. Dobijene rezultate, potrebno je vizualizovati.

Arhitektura online podsistema prikazana je na slici 3.3.1.



Slika 3.3.1 Arhitektura online podsistema

Kao i kod online sistema, neophodno je bilo uneti određene modifikacije u CrowdNav projekat. Glavna petlja CrowdNav simulatora je izmenjena tako da se u svakoj iteraciji (tick-u) za sva vozila koja su uneta u mrežu, pribavljaju koordinate. Dobijene koordinate se zatim, uz pomoć TraCI API-a, konvertuju u geo koordinate i zajedno sa Id-om vozila šalju na Kafku. Da bi se bolje ispratio tok podatka i ispitale nelogičnosti u toku razvoja, uz ove informacije, šalje se i *waiting time* podatak o svakom vozilu. Ova vrednost se izražava u tick-ovima i predstavlja vreme koliko je vozilo provelo ne krećući se brzinom vecom od 0.1 m/s. Primer poruke izgleda ovako:

```
car-727,52,43.323908,21.909074
```

Da bi se tačke na mapi, u kojima zastoj postoji prepoznale, potrebno ih je na neki način grupisati. Za grupisanje je izabran metod klasterizacije koji će prepoznati tačke u kojima ima najviše zastoja i proglasiti ih za centroide klastera. U ovom projektu za metod klasterizacije izabran je metod K-srednjih vrednosti koji je sastavni deo Apache Spark ML biblioteke.

Klasterizacija metodom k-srednjih vrednosti (engl. k-means clustering) metod je za analizu grupisanja čiji je cilj particionisanje n opservacija u k klastera u kojem svaka opservacija pripada klasteru sa najsličnijim značenjem. Iako je problem NP kompleksan, postoje efikasni heuristički algoritmi koji se često primenjuju i koji dovode brzo do lokalne optimalnosti. [16]

Da bi K-means algoritam mogao uspešno da se koristi, najpre je potrebno kreirati model koji će kasnije za svaku ulaznu geo lokaciju vozila, kao izlaz dati broj klastera u kome se to vozilo nalazi. Za kreiranje modela, potreban je trening i test set podataka, a to je značilo opet modifikovati CrowdNav projekat ali u mnogo manjem obimu. Potrebno je bilo promeniti da se podaci o lokacijama vozila ne šalju na Kafku, već uz pomoć CSVLogger-a upisuju u fajl, i to samo ona vozila čiji je *waiting time* parametar bio veći

od 60s. Time su u izlaznom fajlu bile samo lokacije vozila koja su u zastoju. Eksperimentalnim posmatranjem simulacije i markiranjem vozila koja su u zastoju, pokazalo se da u Nišu se mogu 6 mesta okarakterisati kao mesta sigurnog zastoja u slučaju gužve, tako da je za K uzet broj 6. Za veći broj klastera, s obzirom na uličnu mrežu i raspored saobraćaja, desilo bi se da se pojave dva ili više bliska centroida.

`ClusterNis` je klasa čija glavna metoda učitava pomenuti CSV fajl, definiše koje kolone su od značaja za K-means algoritam (eng. feature colums), i deli učitane podatke u odgovarajućoj razmeri (7:3) u dva Dataset-a, trening i test Dataset. K-means model se kreira korišćenjem ugrađene funkcije Spark ML biblioteke prosleđujući proste parametre kao što su broj iteracija i broj klastera, a model se potom čuva na disku. Za dalji rad, važno je napomenuti da se iz modela mogu uvek pročitati centri klastera i da je model spreman da za ulazni niz tačaka (geo lokacija), kao izlaz, da klaster kome te tačke pripadaju.

Izvršiti klasterizaciju geo lokacija svih vozila koja se kreću u simulatoru i to u svakoj sekundi, ne bi bilo od velikog značaja. Iz tog razloga je potrebno informacije filtrirati i obraditi samo ona vozila koja su u zastoju već duži vremenski period. Filtraciju ovog tipa je mogla biti rešena još u samom simulatoru, slanjem samo podataka o vozilima čiji je **waiting time** parametar veći od 60s. Međutim, da bi se problem približio realnom, ova mogućnost je odbačena i rešeno je da se takvo procesiranje obradi uz pomoć Spark-a. I u ovom slučaju, uočena su dva rešenja:

1. Kreirati Spark streaming job sa batch intervalom dužine 1s i implementirati **`mapToPair`** i **`reduceByKeyAndWindow`** operacije, koje će mapirati vrednosti po ključu koji će činiti id vozila, latitude i longlitude i prosleđivati vrednost 1, a zatim ih redukovati i po ključu ali i po vremenskom prozoru dužine 60s, sa sliding prozorom dužine 1s. Nakon izvršenja ovih transformacija, dobijeni objekti se mogu filtrirati i izvući samo oni čija je vrednost veća ili jednaka 60 jedinica. Ti objekti predstavljaju vozila čija se geo lokacija u celom intervalu nije promenila ni jednom.
2. Kreirati Spark streaming job koji će koristiti **`updateStateByKey`** statefull transformaciju nad dobijenim podacima o vozilima. Ovom metodom, Spark bi u memoriji čuvao id-eve svih vozila o kojima su podaci primljeni i ažurirao njihovo stanje. Npr. ako bi vozilo bilo registrovano, brojač bi se inkrementirao i sačuvala informacija o vremenu kada se promena desila, a ukoliko je prošlo više od jednog tick-a između dva ažuriranja, to bi značilo da se desilo kretanje i da se brojač najpre mora resetovati, a tek zatim inkrementirati.

Analizom uočenih rešenja, dogovorena je implementacija prvog zbog svoje jednostavnosti i lake evaluacije validnosti podataka. Ipak, potrebno je naglasiti da prvo rešenje ima i određene loše strane. Kako simulator zahteva određene računarske resurse, može se dogoditi da simulacija ima kašnjenje veće nego predviđeno i da podaci o svim vozilima ne stižu na Kafku svake sekunde. Međutim, ovakvi problemi se retko javljaju i mogu biti zanemareni ili prevaziđeni u opštem slučaju.

`NisWaitTime` je klasa koja modeluje Spark job čiji je glavni zadatak prethodno opisana filtracija podataka o vozilima i detektovanje klastera kome ta vozila pripadaju. Ovaj Spark job čita podatke sa Kafke, ali i kao izlaz upisuje nove podatke na drugi Kafka topic uz pomoć Kafka Spark konektora i kreiranih Consumer i Producer objekata sa svim neophodnim parametrima. Takođe, pri inicijalizaciji svih pomenutih komponenti, i kreiranju samog Spark Streaming konteksta, u memoriju se učitava i K-means model koji je prethodno generisan. Svaka Spark Streaming obrada nad batch-om intervala 1s, filtrira podatke o vozilima kako bi se pronašla samo vozila u zastoju, a zatim nad tako generisanim JavaPairDStream objektom vrši transformaciju i kreira Dataset koji osim već postojećih kolona, dodaje i novu, koja sadrži Id klastera. Kako su u memoriji već učitane informacije o svakom klasteru (id, centroid_latitude, centroid_longitude), moguće je ove informacije pridružiti već postojećim u Dataset-u, koristeći **join** operaciju po Id-u klastera. Primer izlaza, odnosno jedne poruke koja se šalje putem Kafke na novi topic je dat u nastavku:

```
{id:"car-159",wait:"59",lat:"43.323908",lon:"21.909074",
cid:"0",clat:" 43.323",clon:" 21.909"}
```

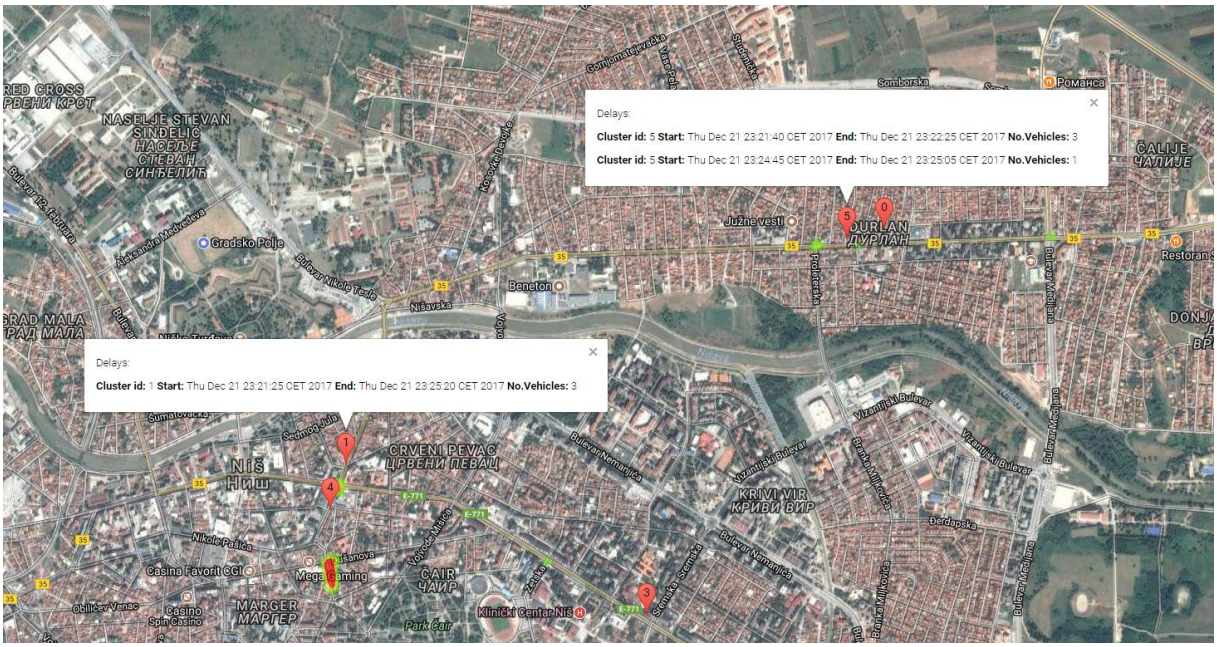
Ovom obradom, prvi deo zadatka online sistema je uspešno završen i predstavljeni podaci se šalju putem Kafke na Web server koji će ih putem Vert.x-a push mehanizmom slati klijentu. Međutim, da bi se za svaki klaster odredilo vreme početka i kraja zastoja i koliko vozila je u tom zastoju bilo, arhitektura je duplirana, odnosno kreiran je još jedan Spark Streaming (`NisDelays`) job i Web server (`WebDelays`).

`NisDelays`, kao i `NisWaitTime`, ima inicijalizovan Kafka konektor sa jednim Consumer-om koji čita podatke sa istog topic-a kao i WebServer, i jednim Producer-om koji obrađene podatke upisuje na novi topic (topic koji čita `WebDelays`). Spark Streaming kontekst je inicijalizovan na batch interval od 5s i ima podešene Spark checkpoint-e kako bi se statefull operacije uspešno obavljale. `NisDelays` koristi pomenutu **`UpdateStateByKey`** transformaciju kako bi se podaci o zastojima čuvali i neophodna logika uspešno implementirala, ali je pre toga neophodno obaviti određene stateless transformacije. Za početak, potrebno je prepoznati i eliminisati uzajamno pojavljivanje zastoja istih vozila. Npr. jedno vozilo može biti u zastoju 65s, a kako ova komponenta prima batch intervale na svakih 5s, može se dogoditi da se jedan isti podatak pojavi 5 puta. Ovaj problem se rešava jednostavnim mapiranjem po ključu svih podatka za istu kombinaciju Id-a vozila i Id-a klastera. Nakon ove transformacije, potrebno je sve podatke grupisati po Id-u klastera, a prateće vrednosti sačuvati i preneti dalje, za šta je iskorišćen viševrednosni Tuple objekat. Ova transformacija je odrađena uz pomoć **`mapToPair`** operacije, dok je uz pomoć **`reduceByKey`** operacije sračunat broj vozila koja su trenutno u zastoju u svakom klasteru. Unutar tela UpdateStateByKey operacije implementirane je logika koja, uz pomoć dodatih fleg-ova, detektuje početak zastoja, kraj zastoja i čuva i ažurira vrednost koja predstavlja najveći broj vozila u okviru jednog zastoja. Za svaki klaster, ukoliko se desi završetak zastoja, podaci se šalju na novi Kafka topic. Primer poruke je dat u nastavku:

```
{cid:"0",startTime: "2017-04-12 13:56",endTime: "2017-04-12 14:59",maxVehicles"26"}
```

Oba Web servera, **`WebWaitTime`** i **`WebDelays`** imaju identičnu arhitekturu i skup objekata koje je potrebno inicijalizovati (Vertx, Router, KafkaConsumer...). Na njima se ne vrši nikakva obrada već se podaci čitaju sa određenih Kafka topic-a i prosleđuju putem Vert.x-a do Web klijenta.

Web klijent je jednostavna HTML stranica povezana sa neophodnim JS bibliotekama CSS skriptama koje joj omogućuje glavne funkcionalnosti i neophodni dizajn. Ugrađena skripta glavne HTML stranice, učitava Google mapu i inicijalizuje Array objekat koji će predstavljati podatke **heatmap**-e. Takođe, unutar skripte definisana su i dva Event bus-a, odnosno dva kanala povezana sa Vert.x komponentom oba Web servera. Pomoću ovih kanala, podaci stižu push metodom do Web klijenta. Na osnovu dobijenih podataka, generišu se markeri koji predstavljaju centroide klastera, a heatmap-a puni podacima koji predstavljaju lokacije vozila u zastoju. Klikom na marker, otvara se poup-up poruka koja sadrži sve informacije o zastojima unutar tog klastera. Primer Web klijenta je predstavljen na slici 3.3.2.



Slika 3.3.2 Krajnji izgled rezultata na mapi

3.4. Streaming Twitter poruka

Za stream poruka sa Twitter-a iskorišćena je ekstenzija za Spark pod imenom Apache Bahir[1]. Apache Bahir pruža dodatne mogućnosti za korišćenje drugih platformi kao izvora podataka tako što implementira konektore za povezivanje sa tim platformama. Platforma može biti Akka, MQTT, ZeroMQ i još neki, a ovde je iskorišćen konektor za Twitter.

Korišćenje ove ekstenzije je vrlo jednostavno. Metoda koja kao vraća objekat tipa **`JavaReceiverInputDStream`** je **`TwitterUtils.createStream`**. Na dalje se koristi kao i svaki drugi Spark Stream. Neophodno je još i postaviti četiri ključa (`consumerKey, consumerSecret, accessToken i accessTokenSecret`) koji se dobiju od Twitter-a kada se registruje nalog i aplikacija na njihovom sajtu.

Većina tweet-ova koji se objave nema povezanu lokaciju na kojoj su objavljeni (geo-tag). Odluka je kompanije, iz razloga bezbednosti, da ova opcija podrazumevano bude isključena. Znači sami korisnici je moraju eksplicitno uključiti ako žele da njihovi tweet-ovi budu povezani sa nekom lokacijom na svetu. Prema jednoj analizi[2] iz 2012. godine, jedva 1% tweet-ova ima tačnu lokaciju. S toga je u ovom radu pribegnuto veštačkom nadovezivanju tweet-ova lokacijom. Naime, svaki tweet je dobio lokaciju negde na prostoru Niša. Ta lokacija je rezultat random operacije nad koordinatama pravougaonika koji pokriva teritoriju grada Niša, tzv. Bounding Box. Koordinate su određene preko Internet stranice www.mapdevelopers.com/geocode_bounding_box.php koja za unetu lokaciju (grad, država ili neki drugi površinski objekat) pruža koordinate Bouding Box-a.

Podaci o zastojima vozila se preuzimaju sa Kafke i to su isti podaci koji generiše **`NisDelays`** posao. Oba ova stream-a (sa Twitter-a i Kafke) prethodno označena sa kog izvora dolaze se stapaju u jedan operacijom **`union`** i transformacijom **`window`**. Vremenski perioda „kliženja“ window operacija je 2 sekunde sa širinom prozora 6 sekundi. Ova vremena su promenljiva i mogu se korigovati u zavisnosti od potrebnih rezultata. U okviru svakog vremenskog okvira (6 sekundi) posebno se filtriraju Twitter poruke a posebno poruke o zastoju. Zatim se uradi Dekartov proizvod nad njima i time se dobija da jes

[1] http://bahir.apache.org

[2] https://www.quora.com/What-percentage-of-tweets-are-geotagged-What-percentage-of-geotagged-tweets-are-ascribed-to-a-venue

svaka Tweet poruka povezana sa svakim zastojem. Pošto je nemoguće utvrditi kom vozilu pripada koji Tweet (ako uopšte potiče od vozila) smatra se da se Tweet može pridružiti zastoju ako se nalazi u neposrednoj blizini. Blizina se definiše kao parametar aplikacije i predstavlja minimalno rastojanje da bi se moglo reći da Tweet pripada nekom zastoju. Na ovaj način je moguće da se veći broj Tweet-ova duplira pa se vrši još jedno dodatno filtriranje duplikata.

Ovako dobijeni podaci o Tweet porukama, njihovom sadržaju i lokaciju mogu se iskoristiti za neku dalju analizu sadržaja poruke. Analize može pokazati pravi uzrok zastoja na putu. Možda je u pitanju neka saobraćajna nezgoda, odron ili pak se vozila nalaze u blizini održavanja velike manifestacije pa je i saobraćaj zbog toga usporen.

4. Deployment sistema

S obzirom na kompleksnost sistema u pogledu veza više različitih komponenti kao rešenje koje se nameće samo po sebi jeste smeštanje komponenti u Docker[13] kontejnere. Na taj način svaka komponenta radi u svom posebno prilagođenom okruženju.

Spark Streaming aplikacija se može pokrenuti na jednom čvoru (računaru) ili klasteru (više računara) za koji je prvenstveno i namenjena. Međutim konfiguracija Docker kontejnera za rad u klaster režimu nije trivijalna pa je u ovom radu nije obrađena. Predviđeno je izvršavanje na jednom računaru ili klasteru više računara na kome je i uspešno testirana.

U Git repozitorijumu koji je sastavni deo ovog projekta je postavljen je Docker Compose fajl (`docker-compose.yml`). U njemu je opis dva povezana kontejnera. U jednom kontejneru se nalazi i Kafka i Zookepeer. Naravno, moguće ih je razdvojiti u posebne kontejnere, ali se pokazalo kao praktično ta budu zajedno, jer se smanjuje mogućnost za grešku u konfigurisanju. Za ovu priliku je iskorišćena gotova slika kontejnera `spotify/kafka` koja se može naći na oficijelnom sajtu[3]. Korišćena verzija Kafke je 0.10.1.

U drugom kontejneru se nalazi SUMO simulator zajedno sa CrowdNav-om i TraCI-jem. I ovde je kao osnova za kontejner iskorišćena slika `starofall/crowdnav` sa zvaničnog Internet sajta[4]. Međutim, s obzirom da je osnovni kod CrowdNav-a izmenjen, trebalo je napraviti novu sliku. To je učinjeno korišćenjem recepta za kreiranje slike (`Dockerfile`) i novog izvornog koda. U Git repozitorijumu izmenjeni kod se nalazi u `app` folderu, a sam `Dockerfile` u korenu repozitorijuma.

Ostale četiri komponente (2 Spark Streaming aplikacije i 2 Web Servera) su realizovane u jednom zajedničkom Maven Java projektu. On se nalazi u `bigdata-java` folderu. Da bi se dobilo na prenosivosti aplikacije, ona je izbuild-ovana u jedan veliki JAR fajl kao paket svih potrebnih biblioteka, tzv. Uber JAR fajl. Biblioteke koje su korišćene su navedene u `pom.xml` fajlu projekta.

4.1. Instrukcije za pokretanje sistema

Za uspešno pokretanje sistema neophodno je ručno mapirati ime hosta `kafka` u lokalnu adresu računara. Ovo se postiže dodavanjem novog reda u `Windows/System32/drivers/etc/hosts` fajlu.

```
127.0.0.1 kafka
```

Docker kontejneri se pokreću izvršenjem naredbe

```
docker-compose up –d
```

u folderu u kome se nalazi docker-compose fajl. Atribut –d je opcioni i znači da se kontejneri pokreću u detached modu, odnosno izlaz njihovih konzola se ne prosleđuje direktno na izlaz konzole domaćina. Može se dogoditi da se CrowdNav kontejner ne pokrene jer ne može da se konektuje na Kafku. Ovo se dešava jer ponekad je potrebno malo vremena da se Kafka sa Zookeeper-om inicijalizuje a CrowdNav je već pokušao konekciju. U tom slučaju je samo potrebno restartovati taj kontejner recimo ponovnim izvršenjem `docker-compose` komande.

Već pomenuti kompajlirani JAR fajl zauzima oko 162 MB prostora na disku, pa nije distriburan kroz Git. Zato jer najpre potrebno iskompajlirati projekat koji se nalauzi u `bigdata-java` folderu. Nakon toga za pokretanje je dovoljno iskoristiti BAT skripte koje se nalaze u `scripts` folderu. Trebalo bi ih pokrenuti sve četiri za punu funkcionalnost sistema.

[3] https://hub.docker.com/r/spotify/kafka/
[4] https://hub.docker.com/r/starofall/crowdnav/

U slučaju da se Spark aplikacija izvršava na klasteru, samo pokretanje je nešto složenije i opisano u poglavlju 4.2.

Ispravnost rada se može nadgledati praćenjem odgovarajućih Kafka topica, tj. izvršavanjem sledeće naredbe:

```
kafka-console-consumer.bat --zookeeper kafka:2181 --topic <topic-name>
```

gde su nazivi topica redom: `nis-out`, `nis-waittime` i `nis-delays`.

Kao krajnja karika u nizu jeste pokretanje Web stranice `index.html` kako bi se videli rezultati simulacije i obrade podataka. Testirano je u otvaranje stranice u Chrome-u gde nije bilo nikakvih problema sa korišćenjem Vert.x JavaScript biblioteka.

4.2. Deploy na klaster

Puna moć Spark aplikacije se postiže tek izvršenjem na klasteru. Spark podržava izvršavanje na četiri tipa klastera: Standalone, Apache Mesos, Hadoop YARN i Kubernetes. Najjednostavniji za korišćenje je Standalone, pa je on i korišćen u ovom radu. Osnovne karakteristike su da izvršava poslove po redu pristizanja (FIFO) i aplikacija pokušava da zauzme sve čvorove radnike (worker nodes). Postoji mogućnost konfigurisanja koliko maksimalno jezgara se može zauzeti na radniku kao i maksimalna količina memorije.

Klaster koji je iskorišćen sastoji se od 9 Linux virtuelnih mašina. Od toga jedna je glavna sa malo boljom konfiguracijom od ostalih osam. Glavna mašina poseduje 8 jezgara, 8 GB RAM-a i 80 GB HDD-a, dok svaka od ostalih osam poseduje po 4 jezgra, 4 GB RAM-a i 40 GB HDD-a. Sve ukupno daje klaster koji se sastoji od 40 jezgara i 40 GB radne memorije.

SUMO simulator je ograničen na simulaciju 650 vozila. Ovo se pokazalo kao najoptimalniji broj vozila da simulacija teče glatko i da pri tom stvarno dođe do zagušenja na ulicama grada. Pošto se svake sekunde podaci o pozicijama svih vozila šalje na Kafku, znači da se za jedan minut pošalje skoro 40.000 slogova. Upravo je to količina podataka nad kojom radi NisWaitTime posao i koji obrađuje svake sekunde. Pošto se inače gleda kašnjenje vozila u proteklih 60 sekundi.

NisWaitTime

Naredba za submit job-a na klaster koje je ovde korišćena je:

```
./bin/spark-submit \
  --class com.elfak.bigdata.sample.streaming.NisWaitTime \
  --master spark://vm-20172018-009.fsoc.hpi.uni-potsdam.de:7077 \
  --driver-memory 2g \
  --executor-memory 2g \
  --executor-cores 2 \
  ~/bin/bigdata-traffic/bigdata.jar \
  spark://vm-20172018-009.fsoc.hpi.uni-potsdam.de:7077 \
  hdfs://vm-20172018-009.fsoc.hpi.uni-potsdam.de:9000/checkpoint \
 ~/bin/bigdata-traffic/data/savemodel
```

Dodeljeno je maksimalno 2 GB radne memorija na svakom od čvorova za ovaj posao iz razloga jer je najsloženiji tj. radi sa najvećom količinom podataka. Još 1 GB je dostupan za ostale poslove, a preostali gigabajt za potrebe operativnog sistema.

Sama aplikacija očekuje tri paramtera koji se u naredbi nalaze odmah nakon navođenja JAR fajla:

- Adresa Spark klastera

- Putanja za smeštaj privremenih podataka koja je inače neophodna kod Streaming operacija. Preporučuje se da se ova putanja nalazi na distribuiranom fajl sistemu; konkretno ovde na HDFS-u
- Putanja do modela za KMeans algoritam.

Vreme izvršenja dosta varira u zavisnosti od toga da li postoje zastoji duži od 60 sekundi ili ne. Ovo se objašnjava činjenicom da je potrebno te podatke proslediti na naredni Kafka topic i potrebno je napraviti objekat koji će tu poruku poslati. Takođe se dešava da simulator nije u stanju da svake sekunde generiše podatke za sva vozila pa se i tu javljaju dodatni zastoji. Prosečno vreme obrade Batch-a je 0.6 sekundi, dok se kada postoje zastoji penje i do nekoliko sekundi (3-5s) i onda nastaju zagušenja u obradi podataka.

NisDelays

Za posao koji se bavi određivanjem početka i kraja trajanja zastoja kao i brojem vozila komanda za pokretanje je vrlo slična prethodnoj:

```
./bin/spark-submit \
  --class com.elfak.bigdata.sample.streaming.NisDelays \
  --master spark://vm-20172018-009.fsoc.hpi.uni-potsdam.de:7077 \
  --driver-memory 512m \
  --executor-memory 512m \
  --executor-cores 2 \
  ~/bin/bigdata-traffic/bigdata.jar \
  spark://vm-20172018-009.fsoc.hpi.uni-potsdam.de:7077 \
  hdfs://vm-20172018-009.fsoc.hpi.uni-potsdam.de:9000/checkdelays
```

Takođe se kao parametri aplikacije prosleđuju adresa Spark klastera kao i putanja do Checkpoint direktorijuma. Za memoriju je predviđena nešto manja veličina od 512 MB po radniku jer je i obim posla dosta manji.

Offline Processing

Za Offline processing podataka iskorišćen je CSV fajl sa 75 MB podataka koji sadrži podatke o vremenu zadržavanja pojedinih vozila na nekom segmentu. Podaci su prikupljeni za vreme rada simulatora i postavljeni na HDFS.

Komanda za startovanje posla

```
./bin/spark-submit \
  --class com.elfak.bigdata.sample.offline.BatchProcessing \
  --master spark://vm-20172018-009.fsoc.hpi.uni-potsdam.de:7077 \
  --driver-memory 2g \
  --executor-memory 2g \
  --executor-cores 2 \
  ~/bin/bigdata-traffic/bigdata.jar \
  spark://vm-20172018-009.fsoc.hpi.uni-potsdam.de:7077 \
  hdfs://vm-20172018-009.fsoc.hpi.uni-potsdam.de:9000/offline/offline.csv \
  hdfs://vm-20172018-009.fsoc.hpi.uni-potsdam.de:9000/offline/result
```

Tri parametra potrebna aplikaciji su

- Adresa Spark klastera
- Putanja do fajla u kome se nalaze ulazni podaci

- Putanja gde treba smestiti rezultat

Vreme izvršenja obrade fajla od 75 MB na klasteru se kreće od 37-47 sekundi.

Twitter Streaming

Komanda za startovanje posla:

```
./bin/spark-submit \
  --class com.elfak.bigdata.sample.streaming.NisTwitter \
  --master spark://vm-20172018-009.fsoc.hpi.uni-potsdam.de:7077 \
  --driver-memory 512m \
  --executor-memory 512m \
  --executor-cores 2 \
  ~/bin/bigdata-traffic/bigdata.jar \
  spark://vm-20172018-009.fsoc.hpi.uni-potsdam.de:7077 \
  hdfs://vm-20172018-009.fsoc.hpi.uni-potsdam.de:9000/checktwitter \
  0.01
```

Slično, prva dva argumenta su Spark putanja i mesto za Checkpoint. Poslednji paramtar je broj koji predstavlja minimalno rastojanje za poklapanje Tweet poruke sa zastojem.

5. Zaključak

Na osnovu analize i poređenja dobijenih rezultata sa realnim stanjem saobraćaja u gradu dolazi se do zaključka da se tačke nagomilavanja poklapaju i ukazuju na realne problematične deonice putne mreže. Na osnovu praćenja vremena zastoja i njihovog trajanja moguće je uočiti pravilnosti i zavisnosti između klastera. Npr. zagušenje u jednom klasteru može dovesti do nagomilavanja velikog broja vozila u njegovom okviru, čime se ostatak putne mreže (ostali klasteri) prazni i biva lako prohodan za vozila koja izađu iz pomenutog klastera. Na osnovu korelacija između klastera moguće je utvrditi koji klaster ima najveći rizik za zagušenje izazvano zagušenjem u prethodnom klasteru kao i pratiti broj vozila koja u tom zagušenju učestvuju.

Zamenom simuliranih podataka, realnim vrednostima iz realnog sistema mogle bi se utvrditi i mnoge druge korelacije. Npr. u kom delu dana je koja deonica putne mreže bila pod najvećim opterećenjem. Povezivanjem ovih informacija, za drugim streaming izvorima, poput Twittera moguće je izvesti nove informacije dodatne vrednosti i pokazati razloge nagomilavanja vozila na određenom mestu.

Većim razrađivanjem teme, moguće je realizovati sistem koji bi poput CrowdNav-a vozačima generisao predložene rute, ali na osnovu generalnog stanja saobraćaja u gradu dobijenog na osnovu informacija sa terena (vozila) i tako rasteretio glavne tačke nagomilavanja, a korisnicima olakšao dolazak do odredišta.

6. Reference

[1] http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/

[2] http://sumo.dlr.de/wiki/Simulation_of_Urban_MObility_-_Wiki

[3] http://www.matsim.org/

[4] http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/

[5] http://sumo.dlr.de/wiki/TraCI

[6] http://sumo.dlr.de/wiki/NETCONVERT

[7] https://github.com/Starofall/CrowdNav

[8] https://github.com/Starofall/RTX/blob/master/DETAILS.md

[9] https://en.wikipedia.org/wiki/Apache_Kafka

[10] https://www.tutorialspoint.com/zookeeper/zookeeper_overview.htm

[11] https://www.tutorialspoint.com/apache_spark/apache_spark_introduction.htm

[12] https://mapr.com/blog/monitoring-uber-with-spark-streaming-kafka-and-vertx/

[13] https://www.docker.com/what-docker

[14] https://josm.openstreetmap.de/

[15] https://spark.apache.org/docs/latest/ml-guide.html

[16] https://en.wikipedia.org/wiki/K-means_clustering