УНИВЕРЗИТЕТ У НИШУ
ЕЛЕКТРОНСКИ ФАКУЛТЕТ
Катедра за рачунарство

Системи за анализу великих гео-просторних података (Big Spatial Data)

Самостални истраживачки рад 1

Ментор: Проф. др Драган Х. Стојановић

Студент: Теодора Коцић, 1457

Ниш, децембар 2023

Садржај

1. Увод

У ери великих просторних података, огромна количина произведених података о мобилности (подаци генерисани помоћу сензора или уређаја са уграђеним ГПС-ом, мреже за надзор, радари итд.) поставља нове изазове у домену аналитике мобилности. Основа за извршавање аналитике мобилности директно зависи од доступности оквира (framework) за обраду велике количине података (Big Data) и технологија прилагођеним просторним и просторно-временским подацима.

Овај рад пружа преглед оквира за обраду велике количине гео-просторних података, са посебним фокусом на Apache Sedona и Mosaic оквир. Архитектура, својства и перформансе Sedona-е и Mosaic-а биће приказани извршењем апликација чији је домен аналитика мобилности (Mobility Analytics). Апликације, којима се приказују карактеристике два поменута оквира, покрећу се и евалуирају на кластеру Apache Spark контејнера који су покренути на локалном рачунару, као и на кластеру истих контејнера постављених на HPI FSOC серверу. Ради стварања јасније слике како раде Sedona и Mosaic у раду ће пре описа својстава и функционалности ових оквира бити осврта на опис и функционалности самог Spark оквира, као и неких основиних концепата на којима је базирана област Big Data.

Да би резултати које добијамо извршењем апликација на кластерима били што реалистичнији било је потребно радити са правим подацима, те је у складу са тиме у овом истраживању искоришћен симулатор урбане мобилности звани SUMO (Simulator of Urban Mobility) у комбинацији са Geofabrik сервером и доступним подацима прибављених из мапа - OSM (Open Street Map) подаци. У раду ће бити дат детаљни опис генерисаних података, као и средстава коришћених у ту сврху.

У последњем делу рада налази се преглед резултата добијених извршењем апликација на сопственом кластеру Docker контејнера, као и кластеру Docker контејнера лоцираних на FSOC серверу. Поредићемо времена извршења идентичних упита на оба ова кластера што се тиче апликације базиране на Sedona-и. Апликација базирана на Mosaic-у извршава се на Databricks серверу, где је покренут кластер чије је окружење подешено тако да се у позадини извршава Apache Spark оквир као у случају за Sedona апликацију, а укључена је и библиотека за омогућавање коришћења функционалности Mosaic-а.

На крају изведен је генерални закључак о оквирима за обраду велике количине гео-просторних података, који представљају централну тему овог истраживачког рада.

2. Big Data и примена Apache Spark-а у овој области

Концепт Big Data заснован је на подацима, чија структура може доста да се разликује од скупа до скупа података (Variety), при чему је количина података у тим скуповима, који пристижу брзинама (Velocity) чији интензитет непрестано расте, све већа (Volume). Овај концепт познат је под називом *three Vs.* Простије речено, Big Data убраја све комплексне скупове података (овакви скупови података адресирају пословне проблеме који раније нису могли да се поставе због непостојања технологије која би успела да их прецизно представи) који су велики и потичу из неких новијих извора података. Када се каже велики скупови података, говори се о величини коју је немогуће обрадити коришћењем традиционалних софтвера за обраду података.

2.1. Историјски развој Big Data

Иако је концепт Big Data релативно новијег датума, прича о постојању скупова података који садрже огроман број информација јавила се још 60-их и 70-их година прошлог века, када су креирани тек први центри података (Data center) и настао концепт релационих база података. Почетком 21. Века, прецизније 2005. године, развој Facebook-а, YouTube-а и осталих сервиса налик овима, доводи до увиђања да корисници оваквих сервиса генеришу незамисливе количине података за тај период. Исте те године, настао је оквир отвореног кода под називом Hadoop, чија је намена била управо складиштење и анализа великих скупова података, а у паралели и концепт NoSQL (Not only SQL) започиње свој развој.

Са развојем Hadoop-а почињу да се појављују и остали open-source оквири (као што је Spark) чији је основни задатак да олакшају манипулисање великом количином података и пружају јефтиније складиштење.

2.2. Начин рада система који се ослањају на Big Data

Постоје три принципа рада на којима се базира функционалност концепта Big Data и то су [1]:

- Интеграција - Big Data “спаја” различите изворе података и апликације. Овде се углавном избегава коришћење традиционалне интеграције података која редом убраја: извлачење, трансформацију и на крају, учитавање података. Потребно је било да се уведу нови начини интеграције који би подржали рад са терабајтима, или чак петабајтима података. Током интеграције, потребно је преузети податке, извршити процесирање и обезбедити да се тако процесирани подаци памте у формату који је подржан од стране анализатора коме се даље прослеђују процесирани подаци.

- Управљање - Big Data захтева постојање складишта (у cloud-у, у неком физичком облику или комбинација ова два). Подаци у складиштима би требало да могу да се памте у разним форматима зависно од домена проблема за који су ти подаци били генерисани претходно. Људи, најчешће, бирају тип складишта у зависности од тренутног места где им се подаци налазе. У садашњости све је популарније коришћење cloud-a, јер се он прилагођава сваком типу уређаја на коме се подаци генеришу и у складу са потребама користи потребне ресурсе.
- Анализа - Предности коришћења Big Data долазе до изражаја када је потребно одрадити одговарајућу анализу података. Бројне су могућности анализирања података: визуелни приказ, истраживање података тако да се долази до неких нових закључака и открића, подаци се могу делити, од података се могу креирати сопствени модели применом машинског учења и вештачке интелигенције.

2.3. Примери употребе Big Data

Big Data концептом може се адресирати низ пословних активности (business activities), од корисничког искуства, све до анализе података.

Компаније попут Netflix-a и Procter & Gamble-a користе велике количине података како би задовољиле захтеве корисника, што се стручно зове **развој производа** (Product development). Ово се постиже тако што ове компаније креирају моделе предвиђања за нове производе и сервисе на основу класификације кључних атрибута претходних и тренутно актуелних производа и сервиса и праве нове моделе веза између ових атрибута и маркетиншког успеха њихове понуде разних услуга.

Сервиси за безбедност и захтеви сигурности се у данашње време непрекидно развијају. Big Data пружа могућност препознавања образаца код података који се користе за **превару** (fraud) и агрегације бројних информација која повећава ефикасност рада система и алармирања корисника да неко покушава да наштети његовим подацима.

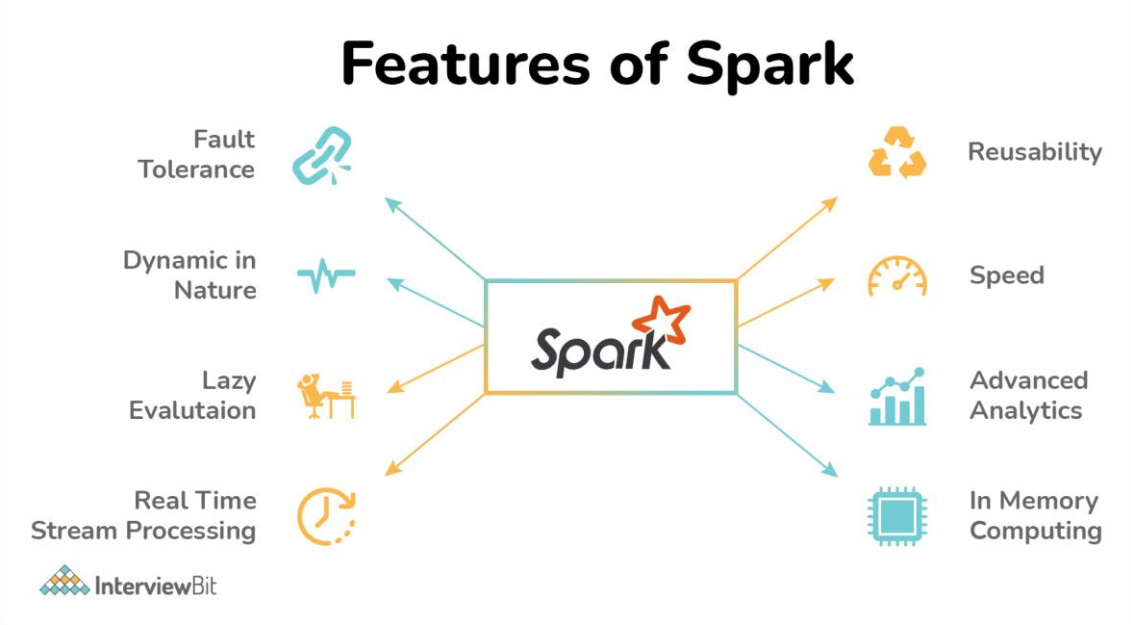
Област **машинског учења** је данас врло популарна, а један од најзаслужнијих фактора је управо BIg Data. Управо су велике количине података оно што омогућава тренирање модела.

Оперативна ефикасност (веза између организационих улазних и излазних параметара) је област у којој Big Data налази највећу примену. Коришћењем концепта Big Data лако се може вршити анализа: производње, корисничког feedback-a, као и осталих фактора који имају улогу у креирању будућих захтева.

Big Data, такође, има значајну улогу у **процесу напредовања** организација. Напредак се може остварити посматрањем и праћењем односа и веза међу људима, институцијама, ентитетима и процесима, након чега се доносе одлуке како ће се изведени закључци на основу посматрања користити у будућности. Концизним праћењем података долази се до могућности напретка код доношења одлука у домену финансија и осталог планирања.

2.4. Основни појмови и еволуција Apache Spark-а

Spark представља кластер engine-а за израчунавање под лиценцом Apache-а и наменски је дизајниран за процесе бржег израчунавања код проблема у домену Big Data. Spark-ова основа се угледа на Hadoop-ов engine ефикасног израчунавања, тако да као и Hadoop, Spark пружа неке додатке (слика 2.1) налик интерактивним упитима, обрадама токова података итд. Spark пружа и могућност извршења написаних апликација у самој меморији кластера, што знатно убрзава саме апликације [2].



Слика 2.1 Apache Spark кључне карактеристике

Apache Spark поседује велики простор за извршавање "послова" (workolad) који укључује batch обраду апликација, интерактивно извршење упита и итеративне алгоритме.

У сржи Spark представља пројекат под окриљем великог пројекта Hadoop. Пројекат је иницијално развио Matei Zaharia 2009. године у AMPLab-у. 2010. године Spark је проглашен платформом отвореног кода. Пар година касније, 2013. године, пројекат присваја фондација Apache Software и од почетка 2014. године Spark постаје један од најпопуларнијих пројеката ове фондације. Популарност овог пројекта проистиче директно из три ствари [3]:

1. Брзина - ово је водећа карактеристика због које је Spark стекао врло брзо популарност. Он пружа чак 100 пута бржу обраду података у поређењу са Hadoop-ом. Такође, пројекат је добар и у погледу уштеде ефективности, како користи само неколико ресурса.
2. Компатибилност - Spark је компатибилан са менаџером ресурса и покреће се са Hadoop-ом на исти начин као и MapReduce. О карактеристикама и начину рада MapReduce-а биће више речи у даљем тексту. И остали менаџери ресурса као што су YARN и Mesos, такође су компатибилни са Spark-ом.

3. Обрада у реалном времену - код Spark-а обрада у реалном времену врши се над подацима који су “груписани” (batch mode). Обрада података врши се у меморијском простору кластера, а не на диску.

2.5. MapReduce

Поменули смо раније у овом поглављу менаџере ресурса, а од интереса за овај рад је начин рада MapReduce-а. MapReduce представља оквир који служи писању апликација које могу да обраде велике количине података, који могу бити структуирани или не прате никакву предефинисану структуру [4]. Подаци који се обрађују у овом оквиру складиште се на HDFS-у (Hadoop Distributed File System). Како се овај фајл систем користи у апликацији која ће бити описана касније у раду, неки детаљи око архитектуре самог HDFS-а биће кратко поменути и објашњени касније у раду. Што се тиче обраде података код MapReduce-а она се базира на batch обради података величине терабајта или петабајта. MapReduce представља основу самог Hadoop-а, јер обезбеђује паралелну и дистрибуирану обраду огромне количине података. Следеће компоненте представљају структуру оквира [5]:

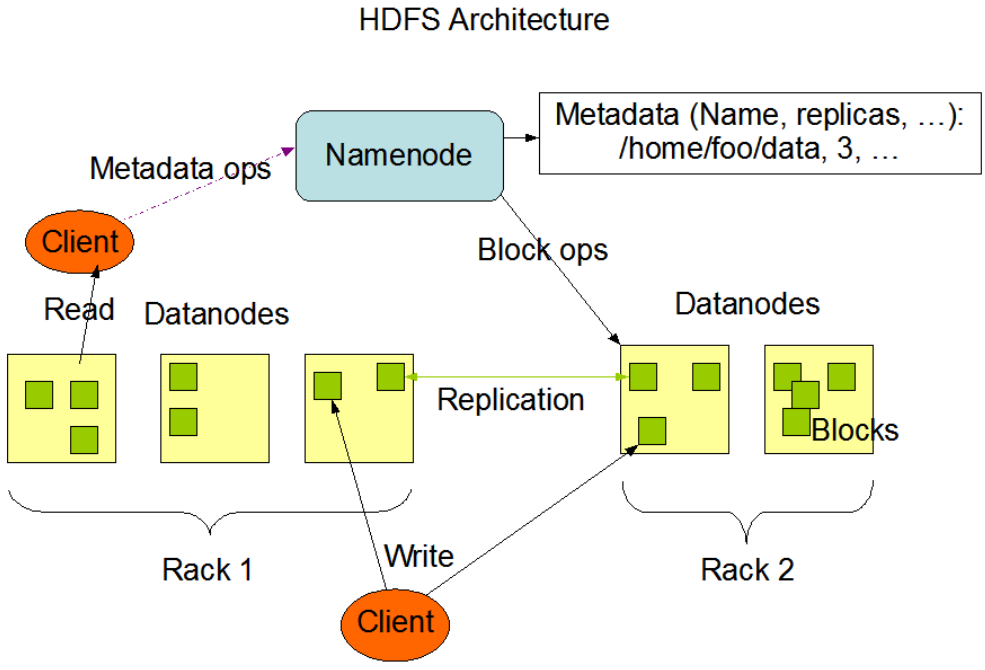
- Два задатка (tasks), улога првог задатка је да одради мапирање (Map), а другог да изврши филтрирање (Reduce)
- Други задатак (Reduce task) може кренути са извршењем тек пошто се комплетира мапирање
- У процесу мапирања блокови података се најпре читају, а потом се врши њихова обрада, а као излаз из овог процеса добијају се кључ-вредност (key-value) парови
- Излаз из процеса мапирања представља истовремено улазне податке за Reduce процес
- Reducer једновремено може да прима податке са више од једног извора
- Reducer на основу генерисаних парова кључ-вредност у оквиру Mapper-а генерише коначни излаз поново као пар кључ-вредност

Једна од предности MapReduce-а је **паралелна обрада** - читав посао је подељен међу више чворова и обраду података сви чворови врше симултано. Обрада се практично обавља по принципу “завади па владај”, где се подаци у паралели обрађују на већем броју машина. Пошто се обрада врши паралелно, укупно време обраде је на овај начин драстично смањено. Друга предност је то што су **подаци локално распоређени**- уместо да се подаци преносе, обрада се премешта на чвор. Ово представља велику уштеду ефективности, јер уколико се ради о подацима чија је количина огромна, што је данас скоро увек случај, пренос података са једног места на друго може да буде изузетно изазовна.

2.6. Hadoop Distributet File System (HDFS)

Да бисмо имали комплетну слику како функционише оквир MapReduce, пожељно је објаснити и принципе који чине Hadoop-ов дистрибуирани систем фајлова. HDFS има сличности са постојећим дистрибуираним фајл системима, међутим и разлике у односу на већину тих система су значајне. HDFS је веома отпоран на грешке и може се покретати на различитим типовима хардвера, укључујући и оне који су у ниском ценовном рангу на тржишту. Представља одличну подлогу за апликације које раде са великим скуповима података. Дизајниран је тако да се лако може прилагодити раду на различитим типовима платформи [6]. Архитектура фајл система, приказана на слици 2.2, је карактеристична и представља концепт власник/роб (master/slave). Кластер HDFS-а састоји се из једне инстанце чвора NameNode, који представља мастер сервер који управља системским простором имена (namespace) и регулише права приступа фајловима за клијенте. Кластер додатно садржи и већи број чворова DataNodes, углавном важи да сваки чвор у кластеру има по један DataNode, који управљају складиштем података за чворове на којима су покренути.

HDFS омогућава складиштење података у фајловима. Интерно посматрано, фајлови су подељени на блокове (један или више њих, зависно од величине блокова, односно самих фајлова), а блокови се памте у оквиру скупова DataNode-ова на датом кластеру. Улога NameNode-а је да извршава операцију отварања, затварања, или промену имена фајлова и фолдера у оквиру namespace-а. Такође, NameNode одређује које мапирање блокова ће бити извршавано. DataNode-ови обрађују захтеве за читање и упис који стижу са клијентског фајл система. DataNode-ови се користе још и за креирања блокова, брисања и репликације истих, све то на захтев NameNode-а.



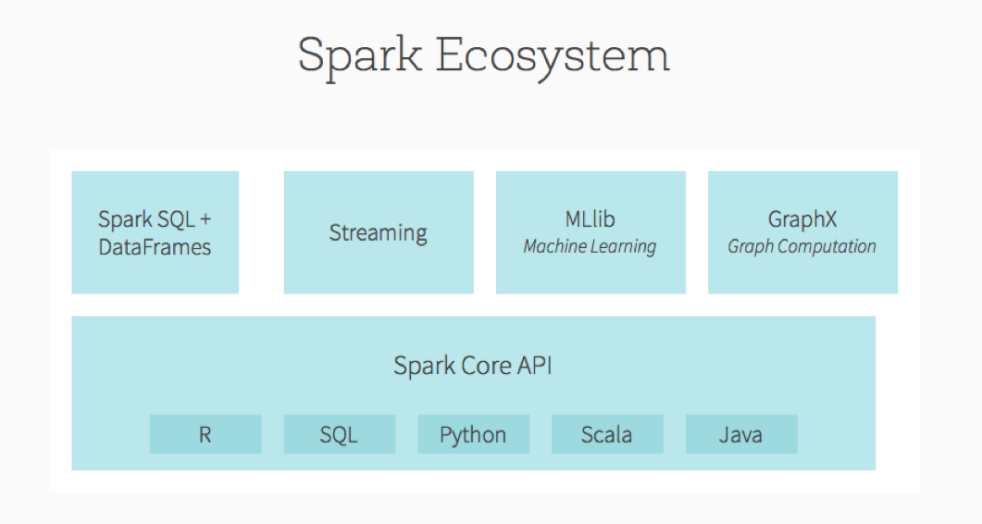
Слика 2.2 Архитектура HDFS-а

NameNode и DataNode представљају софтвере који се покрећу на машинама. Ове машине углавном имају GNU/Linux оперативне системе. HDFS развијен је у програмском језику Java, тако да било која машина која подржава Java-у може да покрене NameNode и DataNode. У пракси веома често се среће ситуација да се посебна машина користи само за покретање NameNode-а. Свака друга машина би требало да покреће по један DataNode.

Архитектура је таква да не искључује да се више инстанци DataNode-а покреће на једној машини, међутим то је у реалним случајевима реткост. Пошто се у оквиру једног кластера може наћи само један NameNode, то значајно упрошћава архитектуру фајл система. NameNode представља део система који извршава задатке и уједно је и репозиторијум за све HDFS мета-податке. Систем је дизајниран тако да кориснички подаци никада не стварају ток кроз NameNode.

2.7. Spark-ов екосистем и карактеристике

Spark пружа могућност бржег израчунавања и лакшег развоја захваљујући постојању одговарајућих компоненти.



Слика 2.3 Spark-ов екосистем

Компоненте које чине Spark-ов екосистем приказане су на слици 2.3:

1. Језгро - све функционалности Spark-а управо су направљене на овој компоненти. Ово је слој који служи за покретање извршења и обраде било којих података. Може да комуницира са екстерним системским складиштима података и обезбеђује израчунавања у меморији (израчунавања се не врше на диску).
2. SQL (Structured query language) - језгро Spark-а пружа још један вид апстракције података која је позната под називом Schema RDD (Resilient Distributed Dataset). SQL

у овом случају подржава и податке који прате одређену структуру, као и оне који немају дефинисану структуру.

3. Токови података - обрада података у реалном времену могућа је искључиво због постојања Spark токова (Spark Streaming). Анализе токова података су подржане у овој компоненти. Обрада података је и овде batch, само што се batch-еви података деле на доста мање групе података (mini-batches). Овде се уводи нова апстракција података која је уско везана за токове података, представља серију RDD-ова и познатија је под називом DStreams.
4. Библиотека која пружа подршку машинског учења - оквир за машинско учење на Spark-у носи назив MLib, а поседује објекте и алгоритме машинског учења. Библиотека садржи разне функционалности као што су кластеризовање, регресија, класификација и доста других.
5. GraphX - овај оквир за дистрибуирану обраду заснованој на графовима знатно убрзава обраду велике количине података.
6. SparkR - ово је комбинација Spark-а и R-а која пружа истраживање различитих техника. Функционалности које поседује Spark се унапређују мешањем R-ових функционалности и Spark-ових скалабилних карактеристика.

У наредном делу поглавља говориће се о карактеристикама Spark-а:

1. Брзина - Spark извршава обраду чак 100 пута брже од MapReduce-а, што је од велике важности када се говори о проблемима у домену Big Data. Овако велике брзине извршавања Spark може да постигне због контролисаног партиционисања. Захваљујући овом партиционисању Spark управља дистрибуираним подацима и обрађује их паралелно, при чему је пренос података (traffic) минималан.
2. Подршка за различите формате - извори података у форматима: JSON, Cassandra и Hive који нису у форми текстуалног фајла, табеле свих релационих база података и CSV су сви формати који су подржани од стране Spark-а. Такође, Spark SQL омогућава да портабилни механизми приступају различитим структуираним подацима, тако да се широк дијапазон извора података може наћи у Spark-овој бази.
3. Израчунавање у реалном времену - како се израчунавања код Spark-а не изводе на диску већ у самој меморији, кашњења код обраде података у реалном времену су јако мала. Spark је направљен тако да подржава масивну скалабилност и корисничку продукцију на кластерима који поседују на хиљаде чворова и већи број модула за израчунавање.
4. Спорија процена - процена је углавном таква да постоји извесно кашњење, а извршава се тек онда када је неопходно. Овакав начин процењивања управо је разлог зашто је брзина извршења код Spark-а велика.
5. Интеграција са Hadoop-ом - Spark је компатибилан са Hadoop-ом, чак се сматра да је он потенцијална замена за MapReduce.
6. Машинско учење - скоро поменута библиотека MLib је прилично корисна када говоримо о обради података. Компоненте Spark-а користе већи број алата, углавном један алат се користи за процесирање података, док се други, посебан, користи за извођење алгоритама машинског учења.

У апликацијама, чији ће домен и имплементације бити детаљно описане у овом раду, често се користила структура података звана оквир података (Data Frame). Data Frame представља Apache-ову колекцију дистрибуираних података. Подаци су у оквирима распоређени по колонама и оптимизовани у табелама. Оквири података у Spark-у могу да се креирају из различитих извора података, као што су: различити формати фајлова (Avro, CSV, elastic search) у којима су смештени подаци, екстерне базе података или већ постојећи оквири података у Spark-у.

3. Оквири за обраду велике количине гео-просторних података

У овом поглављу фокус је постављен на оквире за обраду гео-просторних података, а када кажемо подаци мислимо пре свега на огромне количине података, при чему се посебна пажња посвећује Sedona-и и Mosaic-у, који јесу екстензија самог Spark-а, јер су и апликације које ће бити детаљније обрађиване у наредном поглављу имплементиране тако да користе функције и предности Sedona-е и Mosaic-а.

Због потребе развоја технологија које се примењују у системима базираним на концептима Big Data и употребе истих у области информатике и рачунарства, у последње време неколико истраживачких пројеката је проширено - популарне платформе за паралелну обраду података, као нпр. Hadoop или поменути Spark, прилагођене су тако да имају могућност обраде велике количине просторних или просторно-временских података. Неки од најпознатијих прототипова и система у овој области су: GeoMesa, Beast, AsterixDB, Sedona, Mosaic [7].

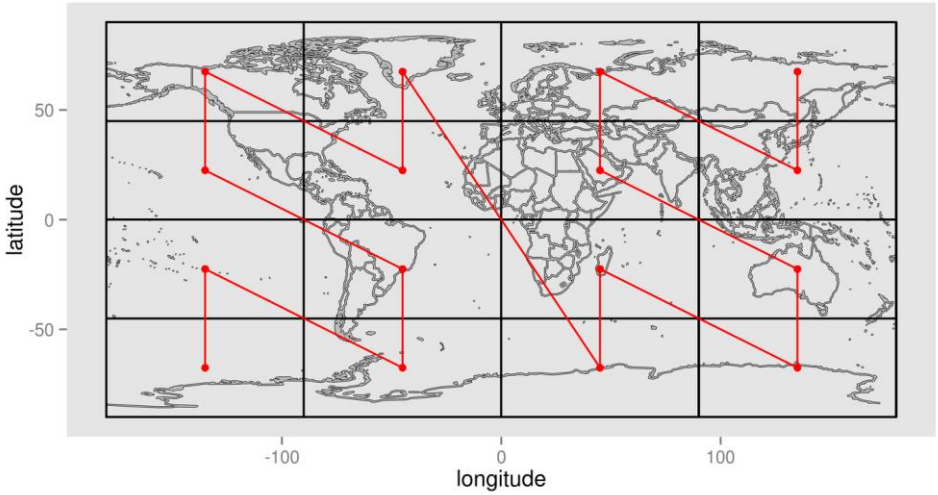
3.1. GeoMesa

GeoMesa је пакет алата отвореног кода (open-source) који се налази под лиценцом Apache-а и користи се за гео-просторну анализу великих размера на дистрибуираним рачунарским системима и на тај начин обезбеђује могућност управљања и анализирања великих скупова гео-просторних података, који се данас у области ИоТ-а (IoT – Internet of Things) све учесталије користе - различити типови мобилних апликација, апликације за праћење, друштвени медији [8]. GeoMesa то чини тако што обезбеђује постојаност просторно-временских података на врху популарних дистрибуираних база података за масовно складиштење података о тачкама, линијама и полигонима. Омогућава брз приступ овим подацима путем упита који у потпуности искоришћавају географска својства за одређивање удаљености и површине дефинисане области. GeoMesa поред овога, пружа и подршку за обраду токова (streams) просторно-временских података у реалном времену постављањем слоја просторне семантике на врху протокола за размену порука система Apache Kafka.

Преко сервера који садржи географске информације као што је GeoServer, GeoMesa олакшава интеграцију са широким спектром постојећих клијената за мапирање и тако добија приступ базама података и могућности стриминга преко стандардних OGC (Open Geospatial Consorcium) АПИ-ја (API – Application Programming Interface). Ови интерфејси такође омогућавају GeoMesa-и да управља корисничким интерфејсима за мапирање и одлучује над којим подацима ће се анализа извршавати. Под анализом подразумевају се: различити упити, хистограми, топлотне мапе и анализе временских серија.

GeoMesa има додатак који омогућава несметано коришћење GeoServer-а, тако да GeoServer може да користи податке директно из GeoMesa-иних табела. Ово даље омогућава мапирање својстава GeoServer-а која су у директној вези са форматираним табелама у оквиру GeoMesa-е. GeoMesa, такође имплементира GeoTools интерфејс и на тај начин обезбеђује HTTP приступ OGC стандардима (Web Feature Service – WFS, Web Mapping Service – WMS, Web Processing Service – WPS и Web Coverage Service - WCS).

Што се тиче памћења података, GeoMesa користи складишта података која раде на принципу кључ-вредност, што су заправо NoSQL базе података које сваки појединачни података уписују и прибављају преко идентификатора, који се у овом контексту користи као кључ. База података HBase овакве кључеве може да памти на великом броју чворова (сервера), а кључеви би требало да буду креирани тако да претрага која је резултат извршења корисничких упита треба да буде што бржа и тиме рад апликације што ефикаснији. Основни принципи рада код GeoMesa-е управо су засновани на коришћењу структуре кључ и вредност. Да би се складиштила просторна термална мапа, најпре је потребно креирати просторно-временски положај податка (record) који би био кључ. GeoMesa користи овакав систем како би се локације складиштиле као тачке, а поред тога креирала и линија која пролази кроз све секторе мапе (црвена линија на слици испод).



Слика 3.1 З-линија

На слици 3.1 приказана је З-линија (Z-line) која је у теорији позната као линија која испуњава простор. Ова линија пролази кроз сваку ћелију мапе тачно једном и тако се одређује јединствени поредак ћелија. Просторне линије могу да раде и са мапама веће резолуције у односу на мапу са слике 3.1. Свакој превојној тачки З-линије може бити додељена секвенцијална вредност, на основу чега би GeoMesa могла да представи пар вредности лонгитуде и латитуде као јединствену целобројну вредност. На овај начин се омогућава да се дводимензионалне вредности представе у једној димензији, при чему би се та вредност могла да искористи као кључ у key-value базама података. Сличан поступак

могао би да се уместо на две, примени и на n димензија, па да се одради линеаризација таквог простора и да се као излаз добије тачно једна димензија.

Основни принцип индексирања које користи GeoMesa је да представља три димензије (лонгитуда, латитуда и време) помоћу тродимензионалне просторне линије, при чему се вредности тачака са ове линије користе као кључ који се даље може памтити у базама података.

3.2. Beast

Оквир Beast се састоји из пет главних компоненти [9]. Поседује компоненту за читање и писање података које раде у паралели, а подржавају различите формате фајлова (Shapefile, CSV, GeoJSON, GeoTIFF). Ова компонента поседује и скалабилне генераторе просторних података. Друга компонента Beast оквира је компонента која служи за просторно партиционисање и контролу количине података којом се тренутно манипулише (load balancer). Задатак ове компоненте је да групише податке који су просторно слични у партиције, користећи притом различите технике партиционисања тако да се води рачуна о оптерећењу чворова на којима се извршава партиционисање. Партиције које се добијају као излази могу бити уписане у било ком формату на диск, што би значило да се ти подаци могу даље користити како од стране самог оквира, тако и од стране других система. Трећа компонента је интерактивни процесор упита који смањује сложеност извршења упита као што су хистограм тачака, Ојлеров хистограм (Euler histogram) и Блумов филтер (Bloom filter). Ова компонента се користи још и код креирања алгоритама селекције и кластеровања. Наредна компонента овог оквира се користи у сврху скалабилног повезивања (scalable join) података различитог типа и тиме омогућава да се једновремено користе подаци из већег броја скупова података. За ефикаснију обраду ових података користе се дистрибуирани алгоритми за оптимизацију у оквиру ове компоненте, који у позадини раде на основу написаних правила која би требало да буду написана у маниру да се аутоматски бира најбољи алгоритам за дати случај. Коначно, пета компонента је интерфејс мапе који корисницима приказује улазне податке, као и резултате добијене извршењем упита у визуелној форми на интерфејсу који је у суштини нека мапа.

Корисници користе Beast најчешће преко интерфејса који је прилично разноврсан. Многе функционалности Beast-а се лако могу бирати и примењивати коришћењем графичког интерфејса. Свим функционалностима корисник може приступити и из интерфејса командне линије (CLI), или употребом интерактивног Spark Scala shell-а. За интеграцију Beast-а у неки Java или Scala пројекат користе се SQL и RDD API, направљени по угледу на Spark-ove API-је.

Што се тиче индексирања, Beast користи R*-стабло (R*-tree) како би индексирао сваку од партиција [10]. R-tree је структура стабла која групише блиске објекте и представља ту групу као минимални правоугаоник (minimum bounding rectangle) у наредном нивоу стабла [11]. Како се сви објекти из групе налазе унутар дефинисаног правоугаоника, уколико се

извршењем упита не добије ни један део правоугаоника то значи да ни један објекат из посматране групе не задовољава услове упита. Листови стабла могу бити правоугаоници који садрже само један објекат. Структура подсећа на Б-стабло (B-tree) које је балансирано стабло претраге. Сви чворови листови се налазе на истој дубини. Алгоритми тражења код P-стабла исти су стандардни, као и за остале типове стабала (пресецање (intersection), претрага најближих суседа, садржање (containment)). Идеја којом се воде сви ови алгоритми јесте да се врши испитивање по дефинисаним правоугаоницима, па се тако долази до закључка да ли је потребно извршити претрагу за дато подстабло или не. Просторно повезивање (spatial join) суседа се извршава врло ефикасно када се ради над чворовима (суседима) на задатој удаљености r и за k најближих суседа, што је врло значајно када се примењује техника кластеризације у оквиру.

R*-tree структура је врло слична претходно описаној, R-tree структури, само што је потребно више меморије за складиштење података (неки подаци морају више пута да се учитавају), али као последица тога добија се стабло код којег је извршење упита краће у односу на извршење код P-стабла.

За визуелизацију гео-просторних података код Beast-а се користи AID* (A Spatial Index for Visual Exploration of Geo-Spatial Data) [12]. Овим индексом се огромни скупови података обрађених за визуелну претрагу чувају на једној машини. Индекс дефинише више могућих резолуција плочица (tiles) димензија таквих да када се једном дефинишу након тога нема промене. Дефинисање индекса врши се над улазним подацима и класификује их као слику, податак, shallow или празну листу зависно од ресурса које је потребно уложити да би се улазни податак обрадио. Класификација се даље користи да се креира индекс са најмањом величином и да је при томе време уложено за креирање индекса минимално а приказивање података на интерактивном интерфејсу остаје да ради у реалном времену. Индекс се креира коришћењем Spark-а или Hadoop-а, тако да крајњи корисници имају приступ преко веб интерфејса налик Google мапама. Због мање величине индекса могуће је да се врши манипулација над подацима из више различитих скупова података на једној машини.

3.3. AsterixDB

AsterixDB је систем за управљање базама података који ради са високим нивоом паралелизације и изузетно је скалабилан. Поседује сопствени полу-структуирани модел података који подсећа на JSON/XML, као и посебан интерфејс за писање упита (SQL++) који у себи садржи низ оригиналних функција из SQL-а, али и доста нових које олакшавају писање упита [13]. У позадини користи хеширање (hashing) и сопствени engine за извршавање упита (Apache Hyracks) како би се осигурао да планирање упита максимално и на најбољи начин искористи могућности партиционисања и паралелизације.

AsterixDB је продукт заједничког учинка студената, факултета и професора на Универзитетима у Риверсајду (Riverside) и Арвајну (Irvine). У време развоја овог система, није постојао ни један менаџер база података отвореног кода који подржава паралелно

програмирање. Главни концепти AsterixDB-а заснивају се на паралелном програмирању, полу-структуираним подацима и Hadoop-у. Прва верзија система постала је доступна јавности 2013. године, а од 2015. године је овај пројекат својство фондације Apache Software.

Код AsterixDB-а постоје два типа компресије: подаци остају некомпресовани, или се користи Google-ов Snappy алгоритам компресије, који користи максималне брзине, а однос компресије задржава неке стандардне вредности. Тренутно, може се вршити компресија једино примарних индекса.

Систем закључавања (locks) функционише са примарним индексима; за секундарне индексе приступ не захтева употребу система за закључавање, али интегритет претраге (lookup) код секундарних индекса је очуван и проверен захваљујући интегритету примарних индекса. Закључавање се може применити над упитима за неку претрагу података, додавање или брисање - остале операције налик убацивању података на диск не захтевају коришћење поменутог механизма. AsterixDB подржава само трансакције у једном нивоу, уколико се извршава трансакција у више нивоа (комплекснији SQL++ упити), она се посматра као више јединствених објеката и сваки се решава засебно.

Што се тиче модела података, AsterixDB има сопствену репрезентацију Asterix Data Model (ADM), која доста подсећа на JSON објекат. Подржава стандардне примитивне типове података, геометријске податке (тачка, линија, правоугаоник, кружница, полигон), као и податке који носе неку информацију о времену. Подржава и изведене типове података, као и null или вредност која недостаје (missing value).

Као примарни индекс AsterixDB користи Log-Structured Merge trees (LSM стабла), а за секундарне индексе подржава коришћење B+ стабала, R стабала или инвертовани индекс. Међутим над секундарним индексима врши се нека обрада тако да добију облик који може да подржи архитектуру LSM стабала, како би се неке LSM операције (претрага, додавање, спајање делова индекса) могле да врше над њима. Обрада секундарних индекса подразумева да се њихова репрезентација у меморији "меша" са меморијском репрезентацијом B+ стабла и тако се добија B+ стабло са обрисаним кључем. Нове промене и додавање индекса виде се у меморији оригиналног индекса, операција брисања се памти у виду складиштења кључева обрисаних ентитета у структуру B+ стабла.

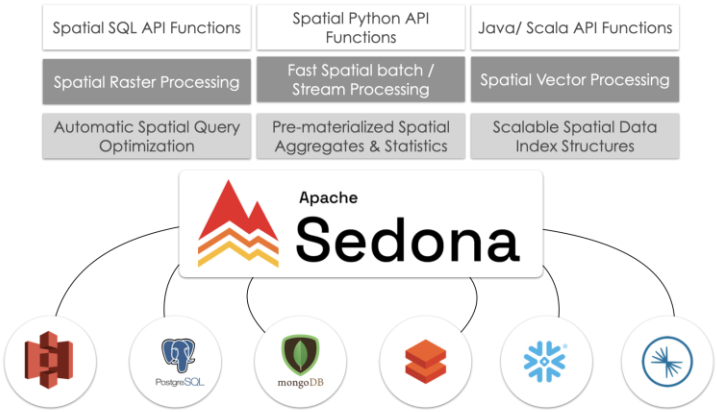
Више типова операције повезивања (join) се користе у AsterixDB-у и то су хеш (hash), угњеждена петља и пренос информације свима (broadcast). Ако се не нагласи другачије користи се хеш повезивање и ова врста повезивања се користи када је потребно да се за резултат упита добије вредност која је идентична некој константи (тачна вредност је задата). Повезивање коришћењем концепта угњеждене петље се користи код "like" оператора који се примењује над текстуалним подацима.

AsterixDB-ов примарни LSM индекс има компоненту у меморији и на диску, а подаци се пребацују на диск када компонента у меморији постане велика. AsterixDB је DBMS чије компоненте немају заједнички простор и користи партиционисање базирано на хеширању у циљу расподеле података међу чворовима. Упити над подацима се прослеђују одређеном контролеру кластера. Потом се упит проследи до свих потребних контролера чворова и

контролера мета-података тих чворова, што значи да је крајњи резултат извршење упита на одговарајућим чворовима у систему.

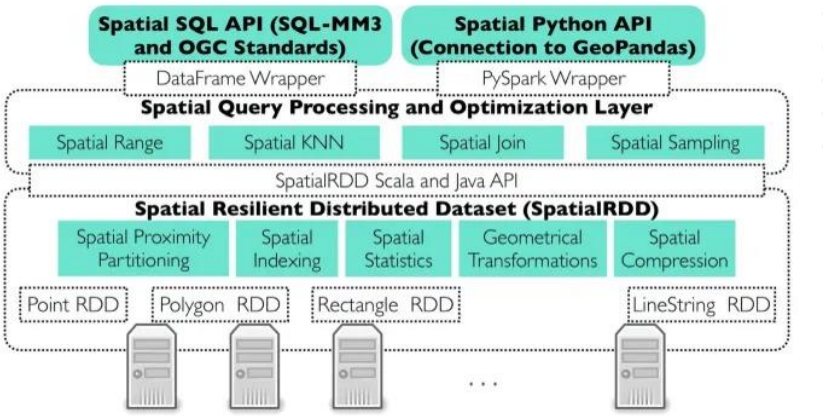
3.4. Sedona

Apache Sedona је дистрибуирани систем који омогућава учитавање, обраду, трансформацију и анализу огромне количине гео-просторних података распоређених на различитим машинама, чији је екосистем приказан на слици 3.2. Овај оквир је заправо екстензија Spark-а за дистрибуиране скупове података (структура је Spatial RRD - SRDD) и користи у основи просторне SQL упите у сврху решавања комплексних проблема [14]. Sedona има API-је за програмске језике Java, Scala, Pyton и R, посебно поседује и API за SQL ради писања просторних упита на што једноставнији начин.



Слика 3.2 Екосистем Sedona-е

Комплексни упити, који би се извршавали дуже време над гео-просторним подацима су захваљујући постојању три концепта код Sedona-е прилично оптимизовани. Реч је о просторном партиционисању, просторним индексима и просторној серијализацији података.



Слика 3.3 Структура концепата за оптимизацију

Apache Sedona уводи нове механизме повезивања (join) у односу на већ постојеће у оквиру Apache Spark-a како би се дистрибуирана обрада података вршила ефикасније. Изглед структуре која директно утиче на ефикасност обраде података, тј. Структура која се користи у сврху оптимизације рада Sedona-е приказана је на слици 3.3 Sedona по аутоматизму извршава опсег, повезивање, стандардне упите и упите који зависе од повезивања на основу удаљености (distance join queries). Уколико је реч о broadcast повезивању, сам корисник мора да нагласи у коду да ће користити broadcast функције над просторно повезаним оквирима података. У даљем тексту биће мало више описа за сваки од join-ова који се користе у Sedona-и:

- Range join - овај вид оптимизације примењује се када се позивају операције *ST_Contains, ST_Within, ST_Intersects* када Spark извршава повезивање
- Distance join - користи се када је потребно пронаћи објекте који се налазе у простору задатог радијуса
- Broadcast join - корисно је у случају када су подаци са десне стране join-а довољно мали да се копирају на свим машинама. У том случају, мешање података и просторно партиционисање може да се занемари. Иначе је опција овог join-а искључена, а може се користити тек у случајевима када се комбинује са broadcast функцијама доступних у Spark-у
- Predicate Pushdown - пре него да се одради просторно повезивање, подаци се морају филтрирати по опсегу. Sedona ће најпре филтрирати по области, а потом повезивати просторно оквире података

Да би подаци могли да буду дистрибуирани, тј. распоређени по машинама, Sedona додељује свакој геометрији партицију на којој би требало да буде обрађивана [15]. Правило важи да што је већи број тачака у области, мања је величина партиције. Apache Sedona тренутно подржава два типа просторног партиционисања и то су KDBTree и QuadTree. KDBTree је у суштини бинарно стабло у којем сваки чвор представља k-димензионалну тачку [16]. Сваки чвор који није лист може се научити подели простора на два дела, креирајући на тај начин два полу-простора. Чворови са леве стране одговарају левом подстаблу које полази из тренутног чвора, а исти случај важи и за чворове са десне стране. QuadTree је структура података типа стабло у којој сваки интерни чвор садржи тачно 4 потомка [17]. Ово су дводимензионални простори октета и најчешће се користе за партиционисање дводимензионалног простора рекурзивно делећи га на четири квадранта или региона. Чворови листови у оваквим стаблима носе јединице недељивих просторних података. QuadTrees деле простор на ћелије (регионе) који су адаптивни, тако да свака ћелија буде максималног капацитета. Када се пређе потенцијално тај максимални капацитет ћелија се дели.

Sedona креира два индекса када обрађује податке, глобални и локални. Главни циљ глобалног индекса је да одбаци партиције које не садрже податке који су корисни за даље извршавање упита. Овако се убрзава извршење упита, јер неће постојати чвор који би обрађивао “празну” партицију. Локални индекси се за сваку партицију посебно креирају, како би се смањио број поређења геометрија. Ово је од изузетне важности за геометрије које садрже велики број линија, или је реч о великим полигонима.

У сврху смањења утицаја обраде гео-просторних података, Sedona имплементира објекат за серијализацију. У Spark-у већ постоји имплементиран kryo серијализатор, међутим у Sedona-и је ово много боље изведено. Kryo објекат за серијализацију нема добру подршку за серијализовање просторних индекса који су врло често и већи од самих просторних објеката. Sedona и поред серијализације индекса, врши серијализовање објеката како би смањила утрошак меморије и утрошак код израчунавања. Користи се алгоритам DFS (Depth For Search) и wkb (Wentzel–Kramers–Brillouin) методологија за записивање геометрије као низ бајтова.

Sedona може да ради са подацима из различитих извора као што су: shapefile, geojson, wkt, wkb, postgis и spatial parquet. Оквир пружа и могућност: учитавања података коришћењем библиотеке geopandas, рад са објектима директно из библиотеке shapely, конверзије секвенце објеката геометрије директно у структуру податка Geospatial Data Frame. Такође, Sedona је једноставна за покретање. Може бити инсталирана коришћењем PyPI-a уз додатне фајлове за покретање, или уколико се ради у апликацији која је писана у Java-и или Scala-и тада се Sedona укључује као dependency.

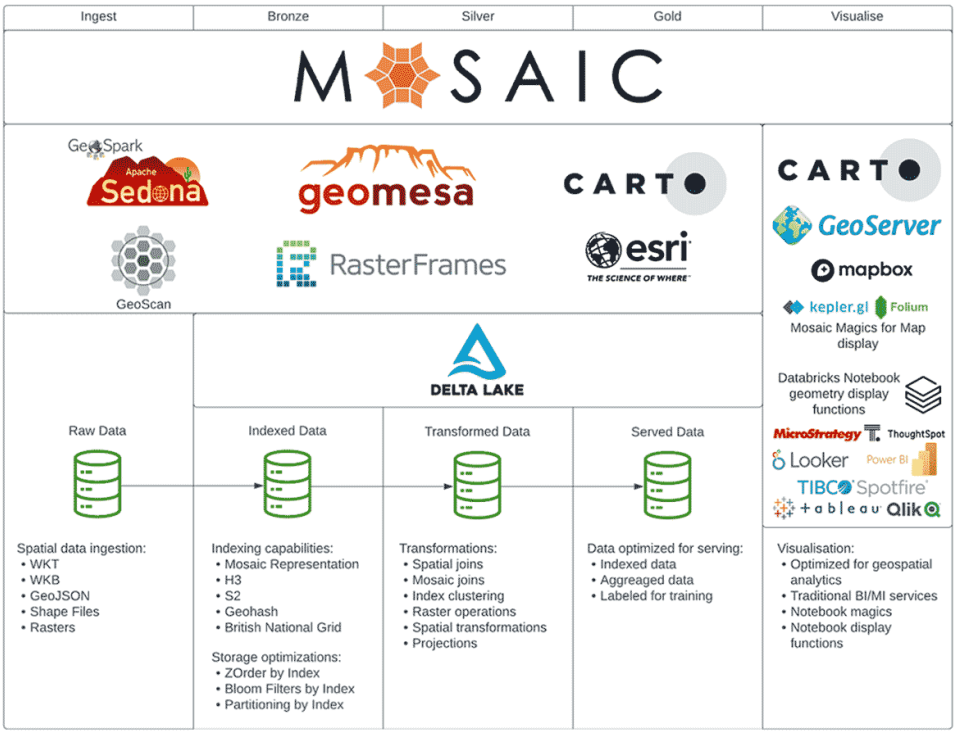
3.5. Mosaic

Mosaic представља екстензију оквира Apache Spark и користи се за лаку и брзу обраду велике количине гео-просторних података. Mosaic обезбеђује [18]:

- Инжењерски приступ гео-просторним подацима тако да се максимално искористе могућности Databricks платформе (специфично DataLake), при чему неометано могу да се користе неке друге екстерне библиотеке
- Високе перформансе тако што се генерише Spark код коришћења и функција које су имплементиране у оквиру Mosaic-a
- Велики број OGC функција за извршавање стандардних SQL просторних упита, које су имплементиране као Spark изрази (Spark Expressions)
- Оптимизацију скалабилног извршења упита за повезивање
- Једноставну конверзију међу стандардним форматима у којима се памте гео-просторни подаци (wkt, wkb, geojson)
- Конструкторе помоћу којих се на једноставан начин креирају геометрије из стандардних типова података које нуди сам Spark и конвертовање у геометријске типове података JTS (JTS Topology Suite) и Esri (Environmental Systems Reasearch Institute)
- Избор између употребе следећих API-ја: Scala, SQL и Python

Mosaic је библиотека која има за циљ да повеже функционалности ефикасног манипулисања гео-просторним подацима са системима који могу да подрже висок ниво паралелизације. Постоји опција да се ова библиотека користи заједно са популарним оквирима као што су Sedona и GeoMesa, али систем који не садржи неке додатне оквире за обраду гео-просторних података, лако може да се пребаци на жељену архитектуру и

извршава све функционалности само употребом алата из Mosaic-а. Као што је представљено на слици 3.4, Mosaic се налази у слоју изнад Lakehouse архитектуре, што даље отвара могућност коришћења AI/ML и напредних анализа платформе на којој се покрећу анализе над гео-просторним подацима.



Слика 3.4 Mosaic и гео-просторни системи

Што се тиче визуелизације, решења типа CARTO, GeoServer, MapBox и други постоје у архитектури заједно у комбинацији са Mosaic-ом. Mosaic је углавном фокусиран на побољшавање перформанси и скалабилности корисничког дизајна апликације и архитектуре, део за визуелизацију и интерактивне мапе би требало да буду покривене неким другим решењима, које више одговарају решавању тих врста проблема у односу на Mosaic.

Код Mosaic-а користи се H3 просторни индекс. H3 представља глобални хијерархијски систем мапирања хексагона у целобројне идентификаторе коришћењем индекса. Хексагони имају доста предности - управљање тачношћу и искоришћење наслеђених индексних структура које би се користиле за израчунавање прецизне удаљености, у поређењу са осталим облицима [19]. H3 долази са API-јем који веродостојно реплицира Mosaic-ов приступ, а поред тога поседује и интеграцију са KeplerGL библиотеком која се користи за представљање (rendering) просторног садржаја у оквиру специјалних развојних окружења (нпр. Databricks notebook). Mosaic је направљен тако да може да се користи у било којем хијерархијском систему просторног индекса којим се добијају савршене просторне партиције.

Стратегије индексирања које се користе овде су З-поредак (Z-Ordering) и Блумов филтер. З-поретком се подаци организују у складишту тако да се највећи могући број података прескочи приликом извршења упита, ако нису од значаја за посматрани упит.

У оквиру Mosaic-а постоје две операције које се воде као оне од највећег значаја: повезивање тачке и полигона (point to polygon join) и join за пресецање полигона. Операција за пресецање два полигона као излаз враћа идентификатор који може да узима вредности “тачно” или “нетачно” (boolean indicator).

3.6. Поређење перформанси оквира за обраду велике количине гео-просторних података

У овом поглављу видели смо основне принципе рада и функционалности за неке, данас, најпознатије оквире за обраду велике количине гео-просторних података. За сваки описивани оквир било је објашњено који вид индексирања се користи, које типове и шта се од алгоритама користи у позадини приликом извршавања просторних упита (најчешће је реч о примени join оператора), као и да ли је подржан неки вид визуелизације података.

У табели дат је сажет преглед перформанси оквира који су били обрађивани у оквиру овог поглавља.

Feature	Beast	Sedona	AsterixDB	GeoMesa	Mosaic
Приступ подацима (I - улазни, O - излазни)					
CSV/WKT	I/O	I/O	I/O	I/O	I/O
Shapefile	I/O	I*	-	I/O	I/O
KML/KMZ	O	-	-	-	-
GeoJson	I/O	I/O*	I/O*	I/O	I/O
GPX	I	-	-	I	-
GeoTIFF	I/O	-	-	I	I/O
HDF	I	I	-	-	-
Generators	да	не	не	не	не
Прикупљање					
Sample	да	да	да	да	да
Histogram	да	да	да	да	да
Bloom Filter	да	не	да	да	да
Партиционисање и индексирање					
Partitioning	да	да	не	не	да
Local index	да	да	да	да	да
Dimensions	n	2	3	3	n
Оптимизација повезивањем (join)					
Filter	да	да	да	да	да
Refinment	да	да	да	да	да
Визуелизација					
Client-side	да	да	да	да	да

Single-level	да	не	не	не	не
Multilevel	да	да	не	не	не

Табела 1 Перформансе оквира за обраду велике количине гео-просторних података

Напомена: * означава да постоје нека ограничења зависно од тога који је формат фајла у питању.

4. Апликације за анализу урбане мобилности (Urban Mobility Analytics)

У овом поглављу рада упознаћемо се са доменом проблема око којег су креиране апликације (једна апликација коришћењем Sedona-е, друга коришћењем библиотеке Mosaic). Даље биће поменуте технологије коришћене у имплементацији апликација, дат детаљан опис скупова података које апликације користе - од тога где и како су подаци били генерисани, до навођења и описа свих атрибута изгенерисаних скупова података. У делу поглавља моћи ће да се погледају делови имплементације апликација који су од значаја.

4.1. Домен проблема и коришћени скупови података

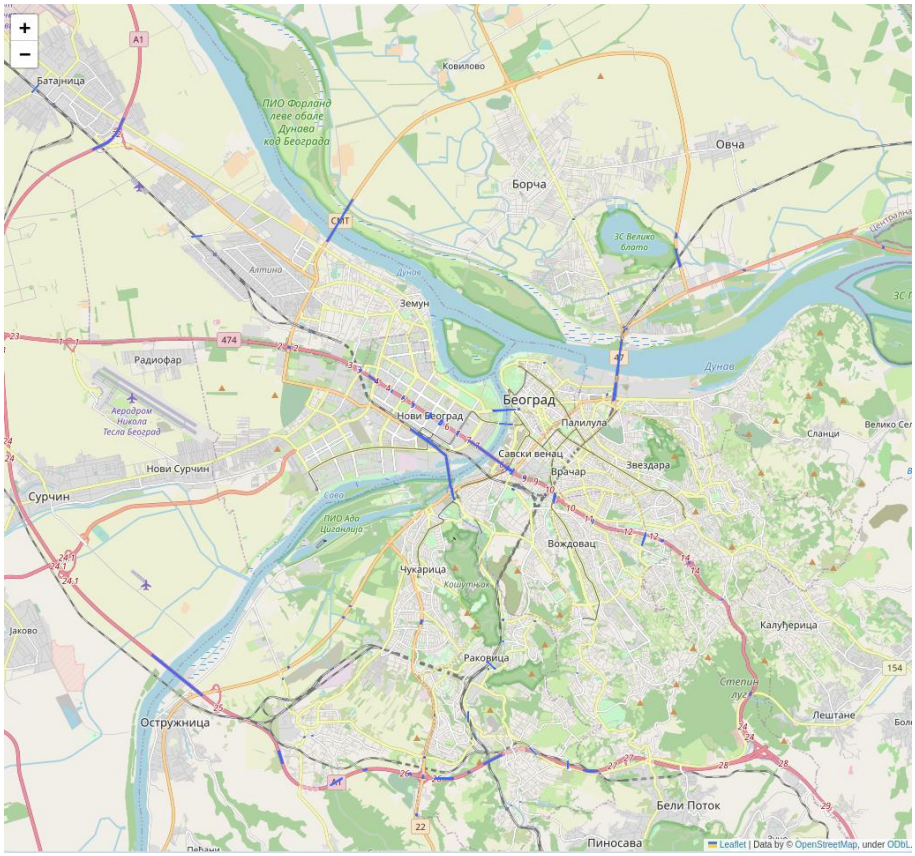
Апликације прате кретање возила у Београду, а просторна и временска анализа којом се обрађују велике количине података које припадају по својој структури групи гео-просторних података, су вршене над генерисаним подацима о улицама, прецизније саобраћајним тракама, којима су се возила кретала и количини загађених материја (емисија штетних гасова) која је произведена у посматраном временском интервалу од стране возила, као и над подацима који носе информације о локацији мостова у граду Београду. Анализе које су рађене над наведеним скуповима података као резултат давале су улицу која је била најзагађенија за посматрани временски интервал, број возила који је прошао кроз област која окружује најзагађенију улицу, дужину сваке саобраћајне траке, интензитет којим се саобраћај одигравао (traffic volume), најдужи мост у Београду и број возила који је у одређеном интервалу времена прешао преко моста и још доста других резултата. Додатно је код апликације у Mosaic-у одрађена и анализа применом специфичног индексирања које је карактеристично за тај оквир како би се показало колико је коришћењем таквог индекса убрзано извршење упита.

Од ова три скупа података прва два скупа изгенерисана су коришћењем симулатора под називом SUMO (Simulation of Urban MObility). SUMO је пакет симулатора за саобраћај, који је отвореног кода, преносив и континуиран [20]. Иако ово није захтеван софтвер, он налази примену у праћењу огромних мрежа. Симулатор саобраћаја на основу улазних података (у случају овог пројекта то су били подаци из Open Street Map-а за област Београда) и предефинисане конфигурације (дефинише временски оквир у којем ће се симулација извршавати, као и мрежу гео-просторних података која се добија конверзијом OSM података у оквиру самог симулатора и додатне параметре чије навођење није неопходно) може да генерише различите типове излазних података. У пројекту је сада коришћен скуп података који носи информације о саобраћајним тракама и скуп података за емисију штетних гасова.

У даљем тексту дат је преглед свих атрибута који се користе за опис информација у поменутим скуповима података:

1. Скуп података за саобраћајне траке - interval_begin [simulation seconds], interval_end [simulation seconds], interval_id [string], edge_id [string], lane_arrived [int], lane_density [flaot], lane_departed [int], lane_entered [int], lane_id [string], lane_laneChangedFrom [int], lane_laneChangedTo [int], lane_laneDensity [float], lane_left [int], lane_occupancy [float], lane_overlapTraveltime [flaot], lane_sampledSeconds [float], lane_speed [float], lane_speedRelative [float], lane_timeloss [float], lane_traveltime [float], lane_waitingTime [float].
2. Скуп података за емисију штетних материја произведене од стране посматраних возила - timestep_time [float], vehicle_CO [float], vehicle_CO2 [float], vehicle_HC [float], vehicle_NOx [float], vehicle_PMx [float], vehicle_angle [float], vehicle_eclass [string], vehicle_electricity [flaot], vehicle_pos [float], vehicle_route [string], vehicle_speed [float], vehicle_type [string], vehicle_waiting [float], vehicle_x [float], vehicle_y [float].

Излазни фајлови ове структуре добијају се иницијално у XML формату, након чега се коришћењем једног од алата симулатора врши конверзија у жељени CSV формат.



Слика 4.1 Визуелни приказ београдских мостова из Geofabrik сервера

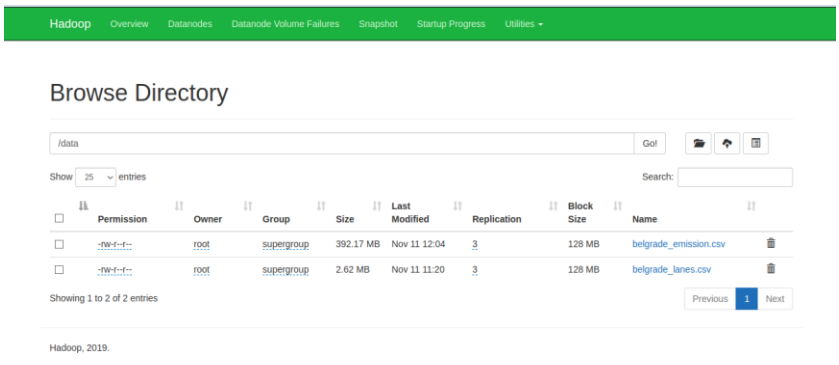
Трећи скуп података добијен је коришћењем функционалности Geofabrik сервера, који садржи актуелне податке из Open Street Map-а за сваки регион света. Дакле подаци који су преузети са сервера, као што се из претходног закључује, имају стандардну структуру података из Open Street Map-а и подаци садрже информације за целу Србију. Написана је Python скрипта специфично за манипулацију овим скупом података, јер се подаци најпре филтрирају да би се из серверских података добили подаци само за област Београда, јер је тај

регион од интереса за апликације. Након примене филтера резултат би требало да буду све тачке које за атрибут city имају вредност name-а (име града) “Београд”, а сада се из тих података издвајају геометрије мостова (претрага по атрибуту tag који задовољава вредност “bridge”) које се коначно уписују у фајл формата geojson. Визуелна репрезентација података који су уписани у поменути geojson фајл приказана је на слици 4.1.

4.2. Архитектура апликација и њихове компоненте

Апликације су написане за извршавање на Spark кластеру контејнера, а коришћени API је за Python. Овај кластер контејнера састоји се из три компоненте: мастер и преостала два контејнера која представљају чворове на којима се извршавају послови које им мастер додели. У оквиру саме апликације доделе се параметри конфигурације (који је драјвер, колико меморије иде чворовима који извршавају послове и остали).

Подаци који се користе у апликацији су раније у тексту били поменути, сада ћемо видети како се подаци користе у оквиру апликације. Дакле, два генерисана фајла у CSV формату се постављају на дистрибуирани систем фајлова Hadoop-а (HDFS). HDFS је покренут заједно са кластером Spark-а и састоји се из неколико контејнера (nodemanager, resourcemanager, datanode и namenode), а подаци се коришћењем корисничке скрипте постављају на HDFS. Креира се нови фодер задатог имена (ако већ не постоји) на namenode-у и онда се копира фајл у CSV формату са локалног рачунара на namenode. Подаци постављени на HDFS приказани су на слици 4.2 која представља кориснички интерфејс namenode компоненте. Касније манипулација подацима прати ток који је описан раније у документу у делу за HDFS. Обе, Sedona и Mosaic апликација читају податке тако што креирају шеме које одговарају заглављима CSV фајлова на HDFS-у, приступају namenode-у и као резултат у апликацији добија се Spark структура Data Frame-а за сваки скуп података претходно прочитаног са HDFS-а. Трећи скуп податак се преузима директно из апликације, тако што се дефинише путања до geojson-а на контејнеру, а оквири за обраду гео-просторних података (Sedona и Mosaic) имају функције којима се врло једноставно извлачи геометрија за сваки објекат у geojson фајлу.



Слика 4.2 Фолдер data са фајловима на namenode-у

Да би се могле користити функционалности оквира за обраду гео-просторних података потребно је у конфигурацији Spark сесије поред доделе ресурса чворовима, специфицирати и који JAR фајлови морају бити укључени како би се функције Sedona-е могле покренути успешно. Слично је и код Mosaic апликације, која се такође извршава на кластеру Spark-а, али се покреће на серверу компаније Databricks, која нуди да се у корисничком интерфејсу одмах инсталирају све неопходне библиотеке за рад апликације, тако да је у случају покретања Mosaic апликације потребно инсталирати ову библиотеку на Spark кластеру.

Апликација се може покренути командом *spark submit* на контејнеру који је мастер чвор. У оквиру команде се може специфицирати још неки део конфигурације Spark кластера. Детаљи о извршењу апликације и праћење извршења кроз интерактивни кориснички интерфејс биће обрађени у наредном поглављу рада.

4.3. Анализа и просторни упити

На почетку овог поглавља укратко су биле описане анализе које су одрађене у оквиру апликација. Овде ће поред објашњења за неке од анализа бити дата и имплементација упита којим се заправо врши процес анализирања гео-просторних података.

Након што су подаци прочитани са HDFS-а, односно након што је учитан и geojson фајл са OSM подацима, за сваки пар x и y-координате положаја возила се креира геометријски објекат тачке.

```
spark.sql("SELECT *, ST_Area(areaPolution) AS area FROM belgrade_buffer").show()
belgradeBuffer = spark.sql(f"""SELECT lane_id, ST_Buffer(ST_GeomFromText({belgradeLineStringArgument}), 175) AS areaPolution FROM position
belgradeBufferArgument = "\'" + str(belgradeBuffer) + "\'"
```

Слика 4.3 Пример просторног упита за израчунавање површине полигона

Даље је потребно наћи геометрију улице (саобраћајне траке) за коју је у датом временском интервалу одређен највиши степен загађености. Прво се пронађе укупна загађеност за сваку саобраћајну траку и након тога се нађе она трака за коју је укупна загађеност максимална. Положај возила на улици која садржи претходно пронађену саобраћајну траку (положај се односи на удаљеност возила од почетка улице) се израчунава и налазе се минимална и максимална вредност од којих се додатно креирају тачке и од тих тачака добија се коначно геометрија линије за посматрану улицу. Око улице се креира полигон који одговара баферу (buffer) одређеног радијуса око пронађене линије. Израчунава се и површина овог полигона (упит којим се рачуна површина полигона представљен је на слици 4.3), као и број свих возила која су у симулацији прошла кроз овај полигон. Додатно се рачуна и удаљеност сваког возила од најзагађеније улице, број возила на свакој улици, дужина сваке улице, потом интензитет (volume) саобраћаја. Још врши се анализа код које резултат треба да даје најдужи мост у Београду, број возила које је прешло тај мост, укупна количина горива која је утрошена на датом мосту, као и просечна брзина возила приликом преласка моста и максимална вредност чекања на најдужем мосту.

Сличне анализе су одрађене и у оквиру Mosaic апликације, само што је сем класичног приступа за проналажење броја возила која су прешла најдужи мост, додата и анализа чији резултат треба да буде исти само трајање извршења упита краће, јер је коришћено индексирање. Упити писани у Mosaic-у који користе карактеристике индексирања приказани су на слици 4.4.

```
#select area around every bridge from dataset
bridgesAreaDf = bridges.withColumn("bridge_area", mos.st_buffer(mos.st_astext("geometry"), lit(150.)))

#create indexes on every bridge area
indexedBridgesDf = bridgesAreaDf.select(mos.grid_polyfill("bridge_area", lit(10)).alias("ix_set")).drop("geometry")
explodedIndexBridgesDf = indexedBridgesDf.withColumn("ix", explode("ix_set")).drop("ix_set")
explodedIndexesBridgeWithRing = explodedIndexBridgesDf.withColumn("grid_area", mos.grid_cellarea("ix"))

#inner join on indexes to find vehicle's count on every bridge
joinedIndexesDf = indexedVehicleTripsDf.alias("t").join(explodedIndexBridgesDf.alias("b"), on="ix", how="inner").count()
```

Слика 4.4 Коришћење индекса у Mosaic-у

5. Извршење апликација и визуелизација податак

У овом делу рада пажња се посвећује извршењу апликација. Апликације се покрећу на кластерима Spark контејнера како на локалном рачунару, тако и на удаљеним серверима. Одрадићемо поређење перформанси извршења упита и добијене резултате по завршетку просторних анализа у оквиру апликација. На крају приказаћемо и визуелизацију скупова података коришћених у обе апликације.

5.1. Покретање, извршавање апликација и поређење резултата

Sedona апликација се покреће на кластеру Spark контејнера на сопственом рачунару, као и на FSOC серверу. Након што се апликација у оквиру мастер чвора submit-ује извршење апликације се може пратити на интерактивном корисничком интерфејсу приказаном на слици 5.1. Брзина извршења Sedona апликације је мања за случај када је апликација извршавана FSOC серверу.

Слика 5.1 Кориснички интерфејс Spark-а

Са слике 5.1 може се видети да је за извршење комплетне Sedona апликације на кластеру Spark контејнера (мастер и два чвора типа worker) било потребно скоро четири минута.

Да би се ова иста апликација покренула на FSOC серверу потребно је проћи кроз исте кораке процедуре за покретање апликације на кластеру Spark контејнера на локалном рачунару. Дакле под претпоставком да је покренута читава инфраструктура Hadoop (раније поменута четири контејнера) и Spark система - прво се креира *Docker* слика за покретање одговарајуће верзије Spark-ових контејнера и "дозволи" коришћење Sedona-иних функција, потом се покретањем *compose* фајла активира инфраструктура која подржава рад ове апликације. Даље се одговарајући фајлови постављају на HDFS (коришћењем скрипте као

што је описано у претходном поглављу) и апликација се доставља мастер чвору Spark-а где започиње њено извршење.

Како је апликација покренута на Spark кластеру, кориснику се пружа могућност конфигурације система. Spark даје опцију конфигурације система на три начина (три потенцијална места са којих се може извршити конфигурација) - користећи особине Spark-а (Spark properties), где се кроз објекат *SparkConf* може поставити велики број параметара апликације, други начин конфигурације је постављањем вредности променљивих окружења (environment variables) кроз које се постављају параметри машина (као нпр. IP адресе чворова посматраног кластера) и постоји могућност конфигурације логова путем l*og4j2.properties*.

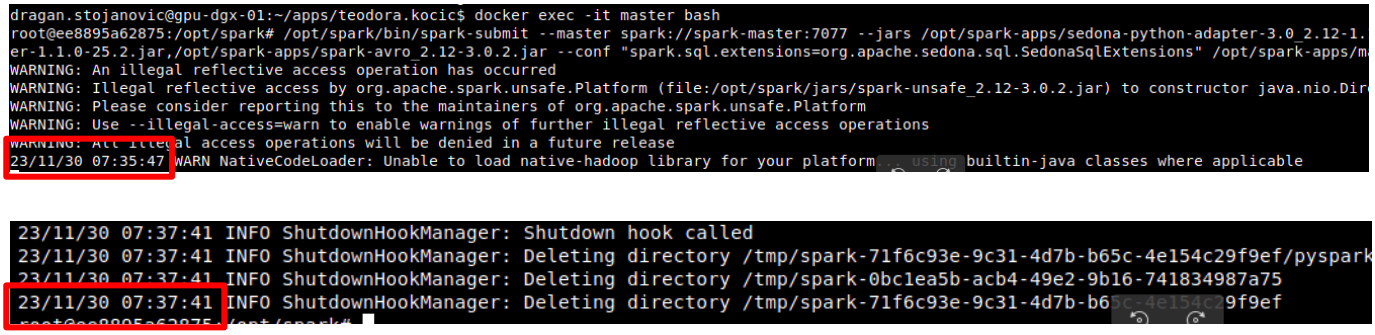
У апликацијама су, директно коришћењем конфигурационог објекта, постављени параметри: име апликације (**spark.app.name**) - име које се појављује и код Spark корисничког интерфејса и у логовима постављено је на вредности "sedona-app", односно "mosaic-app", респективно код Sedona и Mosaic апликације; иницијално време за све мрежне интеракције (s**park.network.timeout**) - вредност која је постављена је 1000 секунди; максимална величина података добијених као резултат извршења операција (s**park.driver.maxResultSize**) - дефинише максималну величину података (у бајтовима) који се добијају као резултат извршења било које Spark операције, у апликацијама ова вредност је постављена на 5 GB, па уколико резултујући подаци буду већи од наведеног максимума, операција се одбацује; Maven координате *jar* пакета који се приликом навођења одвајају запетом (**spark.jars.packages**) - специфицирају карактеристике које ће моћи да користе драјвер и чворови извршиоци; класа за сериајализацију објеката (**spark.serializer**) - објекти који су серијализовани коришћењем ове класе шаљу се у оквиру мреже или могу бити привремено кеширани у датој форми; класа за регистровање корисничких објеката (**spark.kryo.registrator**) - користи се уколико је претходно коришћен и *kryo* серијализатор.

Кроз доделу специфичних вредности променљивима окружења (вредности су додељене у *docker-compose* фајлу) специфицирани су следећи параметри у апликацијама: IP адреса машина (**SPARK_LOCAL_IP**) - за три Spark контејнера (мастер и два контејнера који извршавају послове додељене од стране мастера) вредност ове променљиве окружења одговара називима одговарајућих контејнера (у апликацији то су *master*, *worker-a* и *worker-b*); меморија драјвера (**SPARK_DRIVER_MEMORY**) - у апликацијама је коришћено 10 GB меморије коју ће процеси драјвера моћи да користе (једна од оваквих операција је иницијализација контекста Spark-а); број језгара који ће бити додељен worker чвору (**SPARK_WORKER_CORES**) - оба worker чвора у апликацијама добијају по четири језгра; количина меморије која је додељена worker чворовима (**SPARK_WORKER_MEMORY**) - у Sedona и Mosaic апликацијама додељен је 1 GB простора; количина меморије која се користи у току извршења (**SPARK_EXECUTOR_MEMORY**) - у апликацијама вредност ове променљиве окружења за контејнере је иста као и за претходну променљиву и износи 1 GB.

Вредности променљивих окружења могу се мењати, а од њих директно зависи како ће тећи извршење апликације. На FSOC серверу је Sedona апликација била покренута два пута, при чему су код другог покретања променљиве SPARK_WORKER_MEMORY и

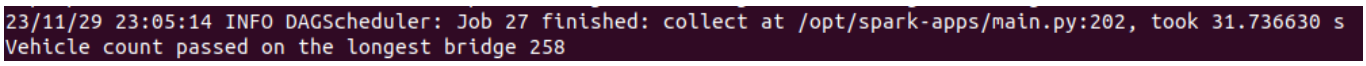
SPARK_EXECUTOR_MEMORY постављене на 2 GB. Убрзање извршења апликације није било велико, али је ипак извршење трајало краће за пар секунди.

Покретање и праћење извршења апликације на удаљеном серверу је омогућено коришћењем софтвера *Remmina*. На слици 5.2 може се видети када је апликација достављена мастеру и када је започело њено извршење, док се на слици 5.3 види временски тренутак када је апликација извршила све анализе. Одавде се види да је извршење апликације трајало нешто испод два минута, што значи да су се анализе дупло брже извршиле на удаљеном серверу у поређењу са извршавањем идентичних анализа на локалном кластеру.



Слика 5.2, 5.3 Почетни и крајњи временски тренутак извршења апликације на FSOC серверу

Mosaic апликација се покреће, такође, на кластеру Spark контејнера али на серверу Databricks-a. Део анализа који је идентични у овој апликацији као код Sedona апликације захтева скоро исто време за извршење анализа, док се значајна разлика у ефикасности извршења упита види код анализа за одређивање броја возила која прећу преко најдужег моста у Београду, јер се тада у Mosaic апликацији користи индексирање које доприноси бржем извршењу просторних упита.



Слика 5.4 Време потребно за одређивање броја возила која су прешла најдужи мост у Sedona апликацији

Време које је на локалном кластеру Spark контејнера било потребно да се изврше упити како би се пронашао број возила који је у временском интервалу за који је симулација одрађена прешао најдужи мост износи преко 30 секунди, као што се види на слици 5.4. За извршење исте те анализе и идентичних просторних упита на FSOC серверу износи 18 секунди, док за те исте просторне упите Mosaic апликација утроши 2 секунде (слика 5.5).

Слика 5.5 Време потребно за одређивање броја возила која су прешла најдужи мост у Mosaic апликацији

5.2. Визуелизација података

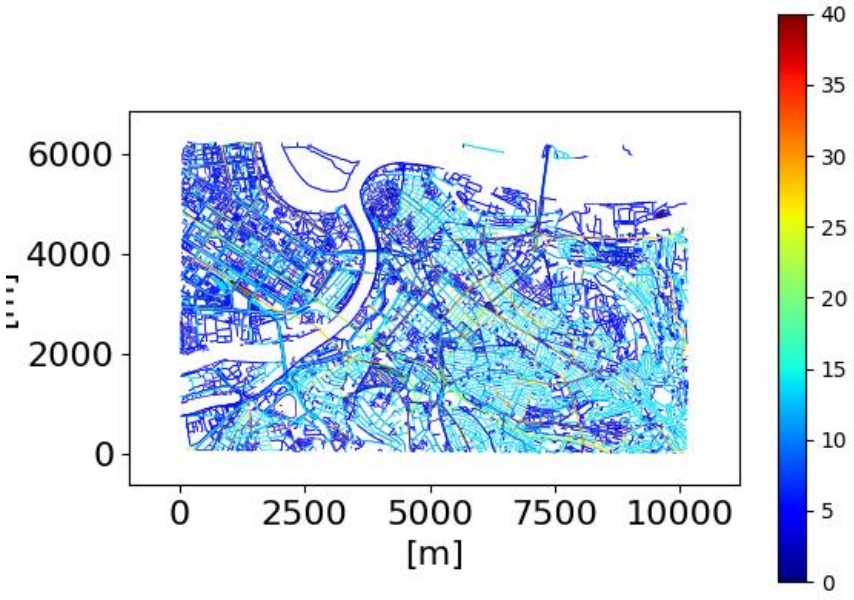
У претходном поглављу видели смо како се на интерактивној мапи виде мостови који су пронађени на територији града Београда на основу OSM података преузетих са Geofabrik сервера, а то је било омогућено коришћењем библиотеке у Python-у за визуелизацију OSM података у оквиру скрипте за генерисање geojson фајла из обрађених података са Geofabrik-а. Преостала два скупа података генерисана су била коришћењем симулатора за саобраћај и искоришћена је била скрипта за конвертовање генерисаних SUMO фајлова у XML формату у CSV формат.

SUMO поред ове функционалности има низ других којима се може манипулисати улазним подацима и генерисаним скуповима излазних података. Од значаја овде јесте део за визуелизацију скупова података добијених након одрађене симулације. У сврху додатних података за визуелни приказ генерисан је још један скуп података и он садржи опште информације о кретању возила (floating car data output). На сликама испод биће приказане неке зависности параметара који су директно преузети из поменутих скупова података - скуп података о општим информацијама о кретању возила, скуп података о емисији која је настала као последица кретања возила и скуп података о информацијама о свим улицама, прецизније свим саобраћајним тракама.

Слика 5.6 Трајекторије свих возила

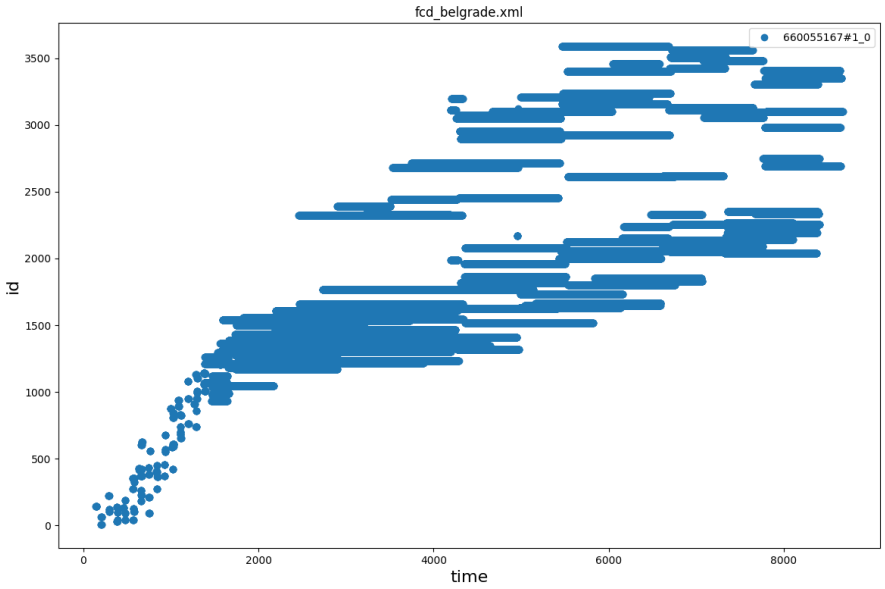
На слици 5.6 дат је приказ свих путања (трајекторија) возила које су биле забележене за време симулације. Слично, изгенерисан је и приказ свих улица којима су се возила кретала

за време симулације. На слици 5.7 дат је приказ расподеле интензитета просечне брзине кретања сваког од возила.



Слика 5.7 Скала расподеле интензитета просечне брзине возила

Коначно, слика 5.8 пружа могућност праћења кретања возила у току времена на улици са максималном количином емитованих штетних материја. За визуелни приказ овде је коришћена мрежа (.net екстензија SUMO фајла) у комбинацији са скуповима података који носе информације о кретању возила и информације о карактеристикама саобраћајних трака.



Слика 5.8 Кретање возила у времену на улици са максималним загађењем

6. Закључак

Да би данас технолошка подршка за проблеме урбане мобилности успела да задовољи потребне ресурсе за складиштење и обраду података који се генеришу у наведеном домену проблема, потребно је користити концепте Big Data. Количина податак која се генерише у апликацијама које прате кретање објеката у времену је огромна. Пре свега ти подаци морају да се складиште у базама података које припадају покрету NoSQL.

Такође, да би анализа овако велике количине података уопште била могућа потребно је да се користе решења новијег датума, као што је оквир Apache Spark који је био описан на почетку овог рада. Овај оквир пружа могућност ефикасног извршења анализа, чак и анализа у реалном времену, над дистрибуираним скуповима података (подаци могу да долазе из различитих извора).

Подаци који се генеришу код проблема урбане мобилности углавном имају гео-просторну структуру, јер се скоро увек прати положај праћених објеката у времену. Такви подаци се не могу обрађивати на Spark-у применом стандардних функција код позива за извршење упита, већ је потребно искористити екстензије Spark оквира које имају подршку да манипулишу и врше обраду гео-просторних података. У раду смо видели неколико оваквих оквира и могли да испратимо начин рада истих и поредимо перформансе и њихове карактеристике.

У другом делу рада описане су апликације, које су за циљ имале да покажу како раде Sedona и Mosaic, као једни од представника оквира за обраду великих гео-просторних података. Кроз покретање апликација на Spark кластеру локалног рачунара и удаљених сервера стекли смо увид у то како се понашају поменути оквири и од колико су значаја примена просторног партиционисања и индексирања у погледу ефикасности извршења просторне анализе.

Литература

[1] Oracle, “What is big data?,” *www.oracle.com*, 2021. https://www.oracle.com/big-data/what-is-big-data/

[2] JanbaskTraining, “What Is Spark? Apache Spark Tutorials Guide & Ecosystem Components, Features,” *JanbaskTraining*, Apr. 13, 2018. https://www.janbasktraining.com/blog/what-is-spark/#1 (accessed Nov. 30, 2023).

[3] B. Chambers and M. Zaharia, *Spark : the definitive guide : big data processing made simple*. Sebastopol, CA: O’Reilly Media, 2018.

[4] T. E. White, *Hadoop : the Definitive Guide (4th Edition)*. O’reilly Media, 2015.

[5] D. Miner and A. Shook, *MapReduce Design Patterns*. “O’Reilly Media, Inc.,” 2012.

[6] “Apache Hadoop 3.3.1 – HDFS Architecture,” *hadoop.apache.org*. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[7] D. Jonitez, M. Sester, K. Stewart, S. Winter, M. Tomko, and Y. Xin, “Urban Mobility Analytics,” Dagstuhl Seminar, Apr. 2022. Available: http://www.dagstuhl.de/22162

[8] “GeoMesa Documentation — GeoMesa 4.0.1 Manuals,” *www.geomesa.org*. https://www.geomesa.org/documentation/stable/index.html (accessed Nov. 30, 2023).

[9] “Bitbucket,” *bitbucket.org*. https://bitbucket.org/bdlabucr/beast/src/master/doc/ (accessed Nov. 30, 2023).

[10] A. Eldawy *et al.*, “Beast: Scalable Exploratory Analytics on Spatio-temporal Data,” University of California, Riverside, Riverside, CA, USA, Nov. 2021. Available: https://www.cs.ucr.edu/~eldawy/publications/21-CIKM-Beast.pdf

[11] “R-tree,” *Wikipedia*, Aug. 18, 2021. https://en.wikipedia.org/wiki/R-tree

[12] “AID*: A Spatial Index for Visual Exploration of Geo-Spatial Data | IEEE Journals & Magazine | IEEE Xplore,” *ieeexplore.ieee.org*. https://ieeexplore.ieee.org/document/9207856 (accessed Nov. 30, 2023).

[13] “Apache AsterixDB,” *asterixdb.apache.org*. https://asterixdb.apache.org (accessed Nov. 30, 2023).

[14] “Apache Sedona™,” *sedona.apache.org*. https://sedona.apache.org/1.5.0/ (accessed Nov. 30, 2023).

[15] G. | P. of X. TechTeam, “Introduction to Apache Sedona (incubating),” *Getindata Blog*, Dec. 08, 2022. https://medium.com/getindata-blog/introduction-to-apache-sedona-incubating-d819cbc0886 (accessed Nov. 30, 2023).

[16] “k-d tree,” *Wikipedia*, Apr. 02, 2022. https://en.wikipedia.org/wiki/K-d_tree

[17] Wikipedia Contributors, “Quadtree,” *Wikipedia*, Nov. 06, 2019. https://en.wikipedia.org/wiki/Quadtree

[18] “High Scale Geospatial Processing With Mosaic,” *Databricks*, May 02, 2022. https://www.databricks.com/blog/2022/05/02/high-scale-geospatial-processing-with-mosaic.html (accessed Nov. 30, 2023).

[19] “H3 | H3,” *h3geo.org*. https://h3geo.org (accessed Nov. 30, 2023).

[20] “SUMO Documentation,” *sumo.dlr.de*. https://sumo.dlr.de/docs/index.html#output (accessed Nov. 30, 2023).