UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

MODEL UPRAVLJANJA VELIKIM SERIJAMA GEOPROSTORNIH PODATAKA

mr Vladimir Pajić

DOKTORSKA DISERTACIJA

Novi Sad, jun 2021

Mentor

prof. dr Miro Govedarica

УНИВЕРЗИТЕТ У НОВОМ САДУ **ОБРАЗАЦ – 5а**
НАВЕСТИ НАЗИВ ФАКУЛТЕТА ИЛИ ЦЕНТРА

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА[1]

Врста рада:	Докторска дисертација
Име и презиме аутора:	Владимир Пајић
Ментор (титула, име, презиме, звање, институција)	Проф. др Миро Говедарица
Наслов рада:	Модел управљања великим серијама геопросторних података
Језик публикације (писмо):	Српски (____латиница_____) или (навести ћирилица или латиница) _____
Физички опис рада:	Унети број: Страница____100_____ Поглавља_____7_____ Референци____77_____ Табела_____9_____ Слика_____43_____ Графикона_____0_____ Прилога_____1_____
Научна област:	Електротехничко и рачунарско инжењерство
Ужа научна област (научна дисциплина):	Геоинформатика
Кључне речи / предметна одредница:	Big Data, геопросторни подаци, облак тачака, криве за попуњавање простора, Apache Spark
Резиме на језику рада:	У оквиру дисертације је представљен је модел за управљање великим серијама геопросторних података базиран на Spark платформи. Модел је фокусиран на облак тачака, с обзиром на његове специфичне карактеристике. Посебно је разматрано индексирање геопросторних података у дистрибуираном окружењу базирано на кривама за попуњавање простора. Модел подржава основне фукционалности попут, структурирања и складиштења података, визуализације, и геопросторних анализа. Извршена је верификација кроз низ експеримената, укључујући поређење са релационим базама података, и испитивање скалабилности.
Датум прихватања теме од стране надлежног већа:	28.05.2014.
Датум одбране: (Попуњава одговарајућа служба)	
Чланови комисије: (титула, име, презиме, звање, институција)	Председник: др Александар Ристић, ред. проф. Члан: др Драган Стојановић, ред. проф. Члан:др Душан Јовановић, ванр. проф. Члан: др Дину Драган, ванр. Проф. Члан, ментор: др Миро Говедарица, ред. проф.
Напомена:	

[1] Аутор докторске дисертације потписао је и приложио следеће Обрасце:
5б – Изјава о ауторству;
5в – Изјава о истоветности штампане и електронске верзије и о личним подацима;
5г – Изјава о коришћењу.
Ове Изјаве се чувају на факултету у штампаном и електронском облику и не кориче се са тезом.

UNIVERSITY OF NOVI SAD
FACULTY OR CENTER

KEY WORD DOCUMENTATION[2]

Document type:	Doctoral dissertation
Author:	Vladimir Pajić
Supervisor (title, first name, last name, position, institution)	Prof. Miro Govedarica, PhD
Thesis title:	A large series of geospatial data management model
Language of text (script):	Serbian language (_____latin_____) or (cyrillic or latin script) _____ language
Physical description:	Number of: Pages_____100_____ Chapters____7_____ References__77_____ Tables_____9_____ Illustrations____43_____ Graphs_____0_____ Appendices_____1_____
Scientific field:	Electrical and computing engineering
Scientific subfield (scientific discipline):	Geoinformatics
Subject, Key words:	Big Data, Geospatial data, point cloud, space-filling curves, Apache Spark
Abstract in English language:	The thesis presents a large series of geospatial data management model based on Spark framework. Model is focused on the point cloud geospatial data type due to it's specific characteristics. Indexing of geospatial data in distributed environment based on space-filling curves is described in detail. The proposed model provides complete set of functionalities like structuring and storing the data, visualization and geospatial analyses. The model is verified through the set of experiments, including comparison to relational databases and scalability assesment.
Accepted on Scientific Board on:	28.05.2014.
Defended: (Filled by the faculty service)	
Thesis Defend Board: (title, first name, last name, position, institution)	President: Aleksandar Ristic, PhD, full professor Member: Dragan Stojanović, PhD, full professor Member: Dušan Jovanović, PhD, associate professor Member: Dinu Dragan, PhD, associate professor Member: Miro Govedarica, PhD, full professor
Note:	

[2] The author of doctoral dissertation has signed the following Statements:
5б – Statement on the authority,
5в – Statement that the printed and e-version of doctoral dissertation are identical and about personal data,
5г – Statement on copyright licenses.
The paper and e-versions of Statements are held at he faculty and are not included into the printed thesis.

Rezime

Razvoj satelitske daljinske detekcije, globanih navigacionih satelitskih sistema, aerofotogrametrijskih kamera, geosenzorskih mreža, radarske daljinske detekcije i laserskog skeniranja je doprineo eksponencijalnom porastu količine prikupljenih geoprostornih podataka. Količina prikupljenih podataka već u velikoj meri prevazilazi mogućnosti njihovog skladištenja na pojedinačnim računarima i iziskuje njihovo skladištenje na klasteru računara. Rast količine prikupljenih podatka, kao i brzine pristizanja velikih serija podataka, osim problema skladištenja, doveo je i do problema njihove obrade koja prevazilazi kapacitete pojedinačnih računara. Da bi se navedeni problemi prevazišli, razvijene su nove metode za distribuirano upravljanje podacima, iz kojih je zatim nastala potpuno nova oblast, Big Data paradigma. Tri ključna elementa za razvoj distribuiranog upravljanja velikim serijama i količinama podataka su Map-Reduce programski model, distribuirani fajl sistem, i Big Data skladišta. Iako, prvobitno, nastali kao vlasničko rešenje kompanije Google, njihovo široko prihvatanje su omogućile njhove open-source verzije Hadoop, HDFS, i Hbase. Spark predstavlja evoluciju Map-Reduce programskog modela zasnovanu na unutar memorijsku obradu, što je za rezultat imalo višestruko uvećanje performansi u odnosu na Hadoop. U okviru disertacije je predstavljena primena Spark platforme, kao osnove za upravljanje velikim serijama geoprostornih podataka. Definisani model je fokusiran na oblak tačaka, kao zasebnog tipa geoprostornih podataka, s obzirom na njegove specifične karakteristike. Osnovu efikasne obrade podataka u okviru definisanog modela čini mogućnost indeksiranja u distribuiranom okruženju. Zbog specifičnosti takvog okruženja, neophodno je bazirati indekse na krivama za popunjavanje prostora. Indeksi predstavljaju osnovu transformacija geoprostornih podataka na jednu dimenziju uz očuvanje prostorne bliskosti tačaka. Prilikom definisanja arhitekture modela uzimaju se u obzir osnovne fukcionalnosti koje on treba da omogući, poput, strukturiranja i skladištenja podataka, vizualizacije, i geoprostornih analiza. Na osnovu toga se vrši implementacija zasnovana na Spark-u i posebno SparkSQL proširenju koje je omogućilo upotrebu konstrukata iz samog jezika domena. Primenom SparkSQL koncepata definisani su odgovarajući korisnički tipovi podataka, kao i korisničke funkcije, a same operacije nad podacima su definisane u obliku SQL upita. Takvim pristupom je omogućeno da se sistem koristi bez poznavanja detalja Spark platforme. Uspostavljeni model je verifikovan kroz niz eksperimenata, u okviru kojih je vršeno poređenje sa relacionim bazama podataka, kao i skalabilnost rešenja u zavisnosti od povećanja količine podataka. Verifikacija je pokazala odličnu skalabilnost modela i robusne performanse sa povećanjem količine podataka, dok je, pri tom, rešenje zasnovano na tradicionalnim tehnologijama koje se baziraju na relacionoj bazi podataka, pokazalo značajno slabiju skalabilnost.

Summary

The development of satellite remote sensing, global navigation satellite systems, photogrammetric cameras, geosensor networks, radar remote sensing and laser scanning led to exponential growth of volume of recorded geospatial data. The amount of collected data already largely surpasses the possibility of storing them on the separate work stations and requires a cluster based storage. The increase of the data volume and the speed of arrival of the data streams, lead to problems related to their processing, beside already mentioned storage problems. In order to solve those problems, new methods for distributed management of data were developed, which led to development of Big Data paradigm. There are three key elements for development of distributed management of huge data streams and volumes, Map-Reduce programming model, distributed file system, and Big Data storage. Although, originated in Google, their open-source versions Hadoop, HDFS, and HBase, were responsible for their wide adoption. Spark represent an evolution of Map-Reduce data model based on in-memory processing, which resulted in several times better performance related to Hadoop. This thesis presents model for management of big streams of geospatial data based on Spark as a platform. Defined model focuses on point cloud, as a geospatial data type, due to its specific characteristics. Efficient processing of the geospatial data is based on indexing in distributed environment. Due to the nature of the distributed environment the indexing have to be based on spatial-filling curves. They provide transformation of multidimensional geospatial data to one dimension while preserving spatial proximity. The architecture of the model includes all functionalities that it needs to provide, like data structuring and storing, visualization, and geospatial analyses. The model is implemented in Spark, mostly using SparkSQL module, which provided the ability to use domain specific language. User defined types and user defined functions were implemented using SparkSQL concepts and the operations on data were implemented as SQL queries. In that way the system became available to the users without knowledge of Spark internals. The model was verified through the experiments, where it was compared to relational database system, and its scalability related to amount of data was tested. It demonstrated excellent scalability with robust performance when large data volumes were applied, where at the same time traditional solution based on relational databases demonstrated performance decrease.

Sadržaj

Spisak slika

Spisak tabela

1. Uvodna razmatranja i hipoteze

1.1. Problem i predmet istraživanja

Razvoj satelitske daljinske detekcije, globanih navigacionih satelitskih sistema, aerofotogrametrijskih kamera, senzorskih mreža, radarske daljinske detekcije i laserskog skeniranja je doprineo eksponencijalnom porastu količine prikupljenih geoprostornih podataka [21]. Količina prikupljenih podataka već u velikoj meri prevazilazi mogućnosti njihovog skladištenja na pojedinačnim računarima i iziskuje njihovo skladištenje na klasteru računara. Rast količine prikupljenih podatka, osim problema skladištenja, dovodi do problema njihove obrade koja takođe prevazilazi kapacitete pojedinačnih računara [69].

Da bi se prevazišli navedeni problemi neophodno je bilo definisati i razviti nove tehnike za rad sa velikim količinama podataka, velikom brzinom prikupljanja ili pristizanja podataka, kao i raznovrsnom prirodom podataka. Big Data paradigma obuhvata skup takvih tehnika, dok se sam pojam Big Data najčešće definiše na osnovu takozvanih 3V karakteristika koje se odnose na količinu, brzinu i raznovrsnost podataka, odnosno *volume-velocity-variety*.

Prema definiciji navedenoj u [69] oblak tačaka predstavlja skup tačaka u nekom koordinatnom sistemu. U oblasti geoinformatike se pod oblakom tačaka, obično, smatra skup tačaka u trodimenzionalnom koordinatnom sistemu. Oblak tačaka se zbog svojih specifičnosti smatra posebnom vrstom geoprostornih podataka, pored rastera i vektora. Zbog toga, uglavnom nije moguće nad oblakom tačaka direkno primeniti rešenja koja su odgovarajuća za rasterske ili vektorske podatke. Moderne tehnologije za prikupljanje i obradu velikih količina geoprostornih podataka, kao što su lasersko skeniranje i fotogrametrija, mogu da generišu oblake tačaka koji se sastoje od milijardi tačaka. Do sada se relativno mali broj istraživanja bavio obradom velikih oblaka tačaka u okviru distribuiranih računarskih sistema.

Za obradu velikih oblaka tačaka u distribuiranom okruženju izabran je Apache Spark koji predstavlja de fakto platformu za obradu na klasteru računara koja se izvršava na nekom od klaster operativnih sistema kao što su Mesos ili YARN [69]. Spark je osmišljen da zadrži skalabilnost i toleranciju na greške MapReduce programskog modela, ali i da podrži aplikacije za koje MapReduce nije bio efikasan. To je omogućeno kroz funkcionalnost *in-memory* izvršavanja.

Spark opisuje distribuirane podatke kroz Resilient Distributed Dataset (RDD) strukturu koja predstavlja read-only kolekciju objekata podeljenih i distribuiranih na klasteru računara. Osnovna osobina RDD skupa podataka je da, ne može doći do gubitka podataka, jer su podaci redundantni i

mogu biti rekonstruisani u slučaju greške ili gubitka pa zbog toga obrada podaka može biti nastavljena u slučaju kvara pojedinačnih radnih stanica. Pridruživanjem šeme podacima skladištenim u okviru RDD nastaje struktura koja se naziva DataFrame koja omogućuje optimizaciju upita nad skladištenim podacima. Dodatna pogodnost DataFrame strukture je mogućnost opis operacija nad podacima u formi SQL upita, kao i definisanje programskih konstrukta vezanih za domen primene, kroz upotrebu korisnički definisanih tipova i funkcija. Na taj način je moguće predstaviti koncepte na visokom nivou apstrakcije i prepustiti Sparku da izvrši optimizaciju obrade.

1.2. Hipoteze sa obrazloženjem

U skladu sa navedenim problemom i predmetom istraživanja definisana su sledeće hipoteze.

Osnovna hipoteza: Tradicionalne metode za upravljanje velikim serijama geoprostornih podataka zasnovane na desktop rešenjima i relacionim bazama podataka ne mogu da obezbede efikasno upravljanje, skladištenje, i obradu. Rešenja zasnovana Big Data paradigmi omogućuju prevazilaženje ograničenja tradicionalnih metoda.

Posebna hipoteza 1: Upravljanje oblakom tačaka zasnovano na Big Data paradigmi omogućuje skladištenje velikih oblaka tačaka uz očuvanje prostorne bliskosti između tačaka. Delovi oblaka tačaka uskladišteni na jednoj lokaciji u okviru distribuiranog sistema sadrže tačke koje su međusobno prostorno bliske.

Posebna hipoteza 2: Distribuirana, odnosno paralelna, obrada podataka omogućuje brže izvršavanje prosotrnih upita nad oblakom tačaka u odnosu na tradicionalne metode.

Posebna hipoteza 3: Upravljanje oblakom tačaka zasnovano na Big Data paradigmi je sklabilno, povećanjem broja računara u distribuiranom sistemu i/ili računarskih resursa povećava se brzina obrade.

Da bi se odgovorilno na postavljenje hipoteze neophodno je definisati i implementirati trasnformaciju, odnosno mapiranje, višedimenzionalnih podataka na jednodimenzionalne memorijske lokacije i distirbuciju podataka. Sledeći korak je definisanje arhitekture distribuiranog sistema za upravljanje velikim oblacima tačaka sa Spark platformom kao okvirom za obradu podataka. U okviru Sparka je neophodno definisti tipove podataka i funkcija specifičnih za domen oblaka tačaka kako bi se podaci skladišteni na disku učitali u Spark aplikaciju.

Navedene hipoteze će biti ispitane na osnovu eksperimentalnog istraživanja u okviru kojeg će biti implemetirano predloženo rešenje. Radi ispitivanja uticaja količine podataka na efikasnost obrade

iskoristiće se tri skupa podataka različitih veličina iz različitih izvora podataka, avionskog i terestričkog laserskog skeniranja. Takođe, da bi se utvrdila skalabilnost sistema koristiće se tri različite hardverske konfiguracije sa 3, 5 i 9 računara.

1.3. Ciljevi i zadaci istraživanja

Kada je reč o oblaku tačaka kao posebnom tipu geoprostornih podataka ne postoje standardizovane metode za upravljanje. Trenutno preovladavajući pristup upravljanju oblacima tačaka je zasnovan na fajl sistemu što sa sobom nosi probleme kao što su konzistentnost podataka, redudantnost, i ograničene pretrage. Drugi pristup je zasnovan na upotrebi standardnih Sistema za upravljanje bazama podataka, primarno, relacionih baza podataka sa prostornim proširenjima, u okviru kojih su definisani odgovarajući tipovi podataka za skladištenje oblaka tačaka. Takav pristup rešava navedene probleme skladištenja u fajl sistemu, ali zahteva dodatni korak transformacije podataka kako bi se oni mogli koristiti u desktop aplikacijama. Sa povećanjem količine prikupljenih podataka pomenuti problemi dovode do ograničene primene pomenutih pristupa.

U skladu sa tim započela su istraživanja na temu upravljanja geoprostornim podacima u Big Data paradigmi, ali su ona malobrojna i fokusirana na vrlo specifične oblasti primene. Takođe, određeni broj tih istraživanja se bavi oblakom tačaka kao posebnim tipom geoprostonih podataka i ona će biti predstavljena u poglavlju 2.

Na osnovu navedenog se može izvesti zaključak da ne postoji istraživanje koje ima za cilj definisanje standardizovanog modela za upravljanje oblakom tačaka u Big Data paradigmi. Zbog toga su definisani sledeći ciljevi ovog istraživanja:

1. Istražiti, analizirati i kritički valorizovati postojeće naučne rezultate i ostvarene doprinose iz područja proučavanja čime će se opisati dosadašnji teorijska i empirijska iskustva iz oblasti upravljanja velikim serijama i količinama geoprostornih podataka, sa posebnim osvrtom na upravljanje velikim oblacima tačaka.

2. Sistematizovati, klasifikovati i raspraviti pojmove i koncepte iz područja proučavanja, potrebnih za sprovođenje ovog istraživanja, kao i s njima povezanih pojmova.

3. Definisati model za upravljanje velikim serijama geoprostornih podataka zasnovan na Big Data paradigmi, sa fokusom na tip podatka oblak tačaka. U okviru modela definisati softversko-hardversku platformu, strukture podataka, i operacije nad njima.

4. Ispitati efikasnost modela poređenjem sa tradicionalnim metodama za upravljanje velikim serijama geoprostornih podataka tipa oblaka tačaka.

1.4 Struktura disertacije

Disertacija se sastoji iz 7 poglavlja.

U uvodnom poglavlju je upravljanje i obrada velikim skupovima oblaka tačaka definisano kao problem i predmet istraživanja. Zatim su definisane hipoteze istraživanja i kao osnovna hipoteza postavljeno da se ograničenja tradicionalnih tehnika za upravljanje oblacima tačaka mogu prevazići korićenjem Big Data metoda. Pored osnovne definisane su i tri posebne hipoteze. Na kraju, definisani su ciljevi i zadaci istraživanja za definisanje standardizovanog modela za upravljanje oblakom tačaka u Big Data paradigmi.

Drugo poglavlje daje pregled stanja u oblasti. Najpre su opisana opšta dostignuća u oblasti geoprostornog Big Data, gde su prikazana dosadašnja istraživanja i softverska rešenja za upravljanje velikim količinama rasterskih i vektorskih geoprostornih podataka. Zatim su opisane tradicionalne tehnike za upravljanje oblakom tačaka, zasnovane na skladištenju podataka u okviru fajl sistema ili relacione baze podataka. Na kraju, prikazana su dosadašnja istraživanja za skladištenje i obradu oblaka tačaka korišćenjem Big Data tehnika.

U okviru drugog poglavlja predstavljena su i Big Data tehnološka rešenja na kojima se zasniva ova disertacija, prvenstveno Apache Spark i HBase. Spark je odabran za platformu nad kojom je izgrađen model za upravljenje velikim serijama geoprostornih podataka tipa oblaka tačaka. Prikazan je uvod u osnovne koncepte Spark platforme, strukturu aplikacije, postupak izvršavanja, distribuciju podataka. Nakon toga su definisane osnovne strukture podataka koje omogućuju distribuiranu obradu, Resilient Distributed Dataset i DataFrame. Takođe je dat osvrt na Big Data skladišta zasnovana na NoSQL konceptima, i Hbase kao rešenje na kojem se zasniva definisani model.

U okviru trećeg poglavlja je najpre predstavljen oblak tačaka kao poseban tip geoprostornih podataka. Zatim su detaljno opisane krive za popunjavanje prostora (space-filling curves) koje predstavljaju osnovnu tehniku za mapiranje višedimenzionalnih podataka na jednodimenzionalne uz zadržavanje prostorne bliskosti. Nakon toga su opisane tehnike za prostorno indeksiranje koje se zasnivaju na krivama za popunjavanje prostora i njihovoj pomenutoj karakteristici. Prikazani su načini primene prostornog indeksiranja zasnovanog na krivima za popunjavanje prostora, za karakterističe tipove prostornih upita, kao što su izdvajanje prostornog opsega ili pronalaženje najbližih susteda.

U četvrtom poglavlju je opisan konceptualni model sistema za upravljanje velikim količinama podataka tipa oblaka tačaka. Prvo je definisana arhitektura modela, koja se sastoji od modula za

struktuiranje, skladištenje, geoprostorne analize, vizualizaciju, distribuciju i kontrolu kvaliteta podataka. Zatim je prikazan logički model podataka za predstavljanje oblaka tačaka u distribuiranom sistemu koji je zasnovan na indeksiranju primenom Z-krive za popunjavanje prostora i Morton kodu. Nakon toga je definisan fizički model podataka, odnosno preslikavanje logičkog modela podataka na ključ-vrednost zapis pogodan za skladištenje u Big Data skladište poput Hbase. U nastavku su definisani korisnički tipovi podataka kao način za predstavljanje logičkog modela podataka u okviru Sparka, kao i korisničke funkcije koje definišu transformaciju podataka iz izvornog oblika u oblik pogodan za distribuiranu obradu (funkcija za generisanje Morton koda). Na kraju poglavlja su opisane operacije, za pretragu prostornog opsega i najbližih suseda, nad definisanim strukturama podataka u distribuiranom okruženju.

U petom poglavlju je opisan implementacioni model zasnovan na prethodno opisanom konceptualnom modelu, kao i verifikacija modela putem izvršenih eksperimenata. Dat je opis skupova podataka koji su korišćeni za verifikaciju modela. Najpre je izvršeno modelovanje softverskog rešenja primenom UML koncepata, dok je kod softverskog rešenja je dat u prilogu teze. Prikazana je postavka eksperimenta koji je obuhvatio poređenje implementiranog modela i rešenja zasnovanog na relacionoj bazi podataka (PostgreSQL) i odgovarajući rezultati.

U okviru šestog poglavlja su diskutovani dobijeni rezultati i analizirani različiti pristupi definisanju modela u zavisnosti od zahteva koji se postavljaju pred sistem.

Sedmo poglavlje obuhvata zaključke, kao i smernice budućeg istraživanja.

2. Stanje u oblasti

Velike količine geoprostornih podataka donose nove mogućnosti, ali i nove izazove za tradicionalne tehnike na kojima se bazira upravljanje geoprostornim podacima. Radi prevazilaženja izazova i maksimalnog iskorišćenja dostupnih informacija protebna je promena pristupa njihovom upravljanju, pre svega, strukturiranju, skladištenju i obradi. Big Data tehnike, koje su omogućile upravljanje velikim količinama alfanumeričkih podataka, predstavljaju osnovu nad kojima se razvijaju tehnike za geoprostorni Big Data. U nastavku poglavlja će biti prikazano stanje u oblasti geoprostornog Big Data i oblasti istraživanja kojim se bavi ova disertacija.

2.1 Uvod u Big Data

2.1.1. Karakteristike podataka

U početku su podaci koji zadovoljavaju jedan ili više od 3V (volume-velocity-variety) parametara smatrani Big Data [13]:

- Volume, odnosno količina podataka, se odnosi na veličinu pojedinačnih skupova podataka.
- Velocity, se odnosi na brzinu prikupljanja ili pristizanja podataka.
- Variety, odnosno raznovrsnost, se odnosi na kompleksnost podataka.

Razvojem oblasti ustanovljeni su i drugi parametri koji utiču na to da se podaci smatraju Big Data [13]:

- Veracity, odnosno pouzdanost, kao i tačnost podataka.
- Variability, predstavlja promenljivost strukture, formata ili izvora podataka.
- Value, predstavlja vrednost informacija koje se mogu dobiti analizom podataka.

2.1.2. Hardverski preduslovi

Hardverski preduslovi koji su omogućili razvoj Big Data rešenja mogu biti grupisani u sledeće četiri kategorije:

1 Sistemi za prikupljanje podatka – omogućeno je prikupljanje podataka reda veličine nekoliko desetina GB u sekundi.

2 Sistemi za skladištenje - moguće je zadržati podatke koji su ranije odbacivani zbog malog kapaciteta.

3 Mreže, brzina prenosa – računari na kojima se izvršava obrada mogu da razmenjuju podatke velikom brzinom.

4 Unapređenje centralnih i grafičkih procesorskih jedinica – moguće je izvršti paralelizaciju na veliki broj procesorskih jezgara i samim tim dramatično ubrzati obradu.

2.1.3. Funkcionalno programiranje

Razvoj programskih jezika, proširenje funkcionalnim konceptima (Java, Python) ili dizajnirani kao fukncionalni jezici (Scala, Haskel) je omogućio razvoj paralelnih algoritama za obradu podataka u okviru distribuiranih sistema.

Koncepti na kojima se zasniva funkcionalno programiranje su dati u [34]:

- Funkcije su koncepti prvog reda, što nije slučaj u imperativnim programskim jezicima.
- Funkcije višeg reda su funkcije koje za parametre imaju druge funkcije, npr. funkcija map koja prima dva argumenta niz, i funkciju koja se izvršava nad pojedinačnim elementima niza. *map(niz_brojeva, dupliraj(broj) { return broj * 2 })*
- Čiste funkcije.
- Nepromenljivost podataka.

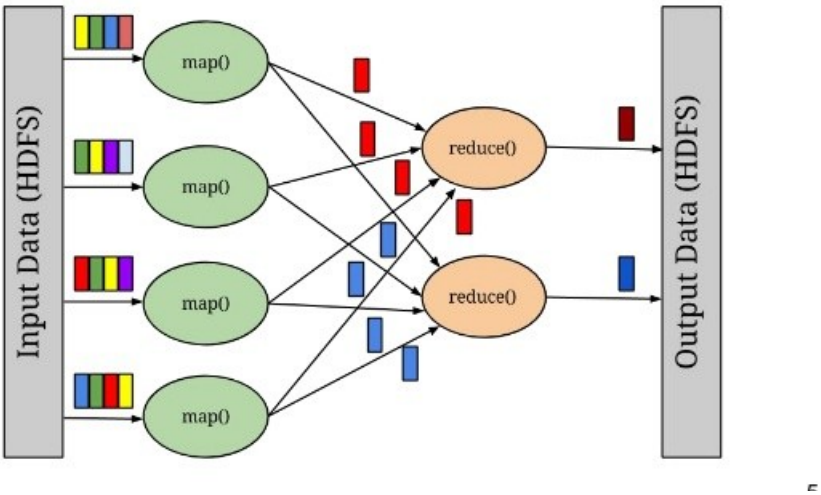
Primenom koncepata funkcionalnog programiranja je moguće distribuirati kod svim izvršiocima u okviru distribuiranog sistema sa sigurnošću da će izvršavanje u svakom trenutku proizvesti korektan rezultat.

2.1.4. MapReduce programski model

MapReduce programski model omogućuje paralelnu obradu podataka u distribuiranom računarskom sistemu. Osmišljen je u kompaniji Google kako bi se prevazišao problem obrade velikih količina podataka (tzv. BigData) [22]. Ujedno, razvijen je i Google File System (GFS) [38] koji je omogućio skladištenje takvih podataka. I druge kompanije koje prikupljaju i obrađuju velike količine podataka, poput Yahoo!, Facebook i New York Times, su ubrzo usvojile takav pristup.

MapReduce programski model je inspirisan *map/reduce* operacijama koje potiču iz funkcionalnog programiranja, ali sa drugačijom semantikom. Kompletna obrada podataka se opisuje primenom te dve operacije. *Map* operacija učitava podatke sa diska i emituje podatke u obliku key-value parova. *Reduce* operacija prima podatke u obliku parova (ključ, [vrednost]) i na osnovu toga izračunava konačne vrednosti. Između *map* i *reduce* operacije postoji *shuffle* faza u kojoj sistem kombinuje

izlaze iz svih map operacija na osnovu *key* vrednosti i distribuira do *reducer*-a. Sistem određuje kojim *reducer*-ima se prosledjuju koji podaci kako bi se minimizovao mrežni saobraćaj i time postigla maksimalna brzina obrade. Na kraju, rezultat *reduce* operacije se skladišti u fajl sistemu. Na slici 2.1 je prikazana šema izvršavanja *Map-Reduce* programskog modela.



Slika 2.1 Map-reduce programski model

Apache Hadoop predstavlja open-source softverski sistem koji omogućuje distribuiranu obradu podataka na klasteru računara korišćenjem MapReduce programskog modela [75]. Pored programskog modela, Apache Hadoop poseduje i sistem za distribuirano skladištenje podataka Hadoop Distributed File System (HDFS). Iako se MapReduce pokazao veoma uspešnim u implementiranju distribuiranih aplikacija nad velikim količinama podataka, on poseduje nedostatke koji ograničavaju njegovu primenu. MapReduce je osmišljen za paralelno izvršavanje sekvencijalnih algoritama nad velikim skupovima podataka. Da bi se rezultat jedne MapReduce operacije mogao koristi kao ulazni podatak za drugu MapReduce operaciju, što je neophodno kod iterativnih algoritama, potrebno je uskladištiti podatke na disk. Skladištenje podataka na disk kao i startovanje svake MapReduce operacije traju dugo i dovode do značajnog usporavanja obrade. Dve osnovne vrste aplikacija kod kojih je primena MapReduce neefikasna su: iterativni algoritmi i interaktivne analize [69].

2.1.5 NoSQL baze podataka

Baze podataka koje po svojoj prirodi nisu relacione se, kolokvijalno nazivaju, NoSQL, odnosno ne samo SQL (*Not only SQL*). NoSQL baze podataka prema dizajnu odstupaju od ACID (Atomic, Consistent, Isolated, Durable) principa [4] na kojima počivaju relacione baze podataka.

Prema [57] osnovni razlozi koji su doveli do napuštanja relacionog modela su:

1. potrebe za skladištenjem veoma velikih količina podataka,
2. potreba za pristupom podacima u realnom vremenu,
3. potreba za fleksibilnom strukturom podataka, koja se može brzo menjati,
4. skladištenje nestrukturiranih podataka, i
5. skalabilnost.

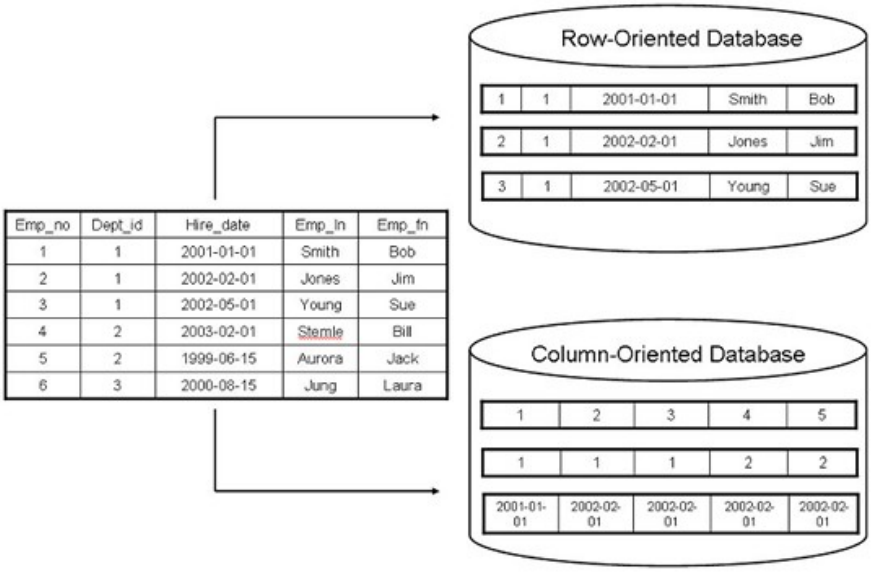
Mogu se razlikovati četiri tipa NoSQL baza podataka [57]:

1. Ključ-vrednost skladišta, kod kojih se za svaki zapis kreira jedinstveni ključ, kojem se pridružuje skup vrednosti. Takva struktura omogućuje brzu pretragu prema ključu nad velikim skupovima podataka.
2. Dokument skladišta, gde su podaci predstavljeni u obliku dokumenata, koji su najčešće u obliku JSON struktura.
3. Graf bazirane baze podataka, gde se podaci skladište u čvorovima grafa.
4. Kolumnarne baze podataka, kod kojih se podaci skladište po kolonama, za razliku od relacionih baza podataka kod koih se oni skladište po redovima.

2.1.5.1. Kolumnarne baze podataka

Zbog značaja kolumnarnih baza podataka u Big Data paradigmi one će biti detaljnije opisane. Skladištenje po kolonama pruža niz pogodnosti koje omogućuju podršku velikim količinama podataka, kao, velikim brzinama pristizanja podataka, i različitim strukturama podataka. Naravno, takav pristup nije bez nedostataka, odnosno gubi se mogućnost za obezbeđivanje apsolutne konzistentnosti podataka. Kolumnarne baze podataka pružaju prednost kada je reč o analitičkim upitima, na primer za različite vrste agregacija sumiranje, prosek, minimalne i maksimalne vrednosti. Druga značajna prednost je mogućnost visoke kompresije podataka s obzirom na to da su sve vrednosti u jednoj koloni istog tipa. Primenom tehnika za kompresiju se postižu značajno bolji rezultati nego kod relacionih skladišta sa raznorodnim tipovima podataka u okviru jednog zapisa. Slika 2.2 prikazuje raziličitost skladištenja podataka između kolumnarnih i relacionih baza podataka.

U [52] je prikazan metod za kompresiju podataka o oblaku tačaka primenom kolumnarne baze podataka SAP-HANA.



Slika 2.2 Poređenje relacionih i kolumnarnih baza podataka (preuzeto sa https://d1x2i8adp1v94i.cloudfront.net/wp-content/uploads/2012/07/column-oriented-database-2.jpg.webp)

2.1.5.2. Dizajniranje upita nad NoSQL bazama podataka

Prilikom definisanja modela podataka kod NoSQL baza, neophodno imati na umu koje vrste upita će biti izvršavane. Kod relacionih baza podataka na efikasnost upita direktno utiče poznavanje strukture podataka, koju zatim koriste alati za optimizaciju izvršavanja upita. Suprotno, NoSQL baze prema svojoj prirodi (zbog nedostatka strukture) ne mogu posedovati takve alate nego se optimizacija postiže na nivou modela podataka.

Treba navesti da je Spark kao alat za obradu Big Data u jednom trenutku uveo podršku za strukturirane podatke kroz SparkSQL i nastavio da je razvija kroz DataFrame koncept. Više detalja će biti dato u nastavku poglavlja.

2.1.6. ISO standardi za Big Data

Zbog sve većeg značaja Big Data, Međunarodna organizacija za standardizaciju (ISO), je donela mere za regulisanje ovog aspekta. ISO i IEC (Međunarodna elektrotehnička komisija) su zajednički radili na stvaranju serije standarda o Big Data kako bi definisali njihovo ispravno korišćenje. Odbor stvoren za ovu funkciju je ISO / IEC JTC 1 / SC 32, koji je do danas već objavio 77 standarda i projekata s još 29 koji su u pripremi.

ISO / IEC serija standarda o Big Data obuhvata sledeće standarde [12]:

- ISO / IEC DIS 20546 Informacione tehnologije – Big Data – sadržaj i rečnik.

- ISO / IEC AWI TR 20547-1 Informacione tehnologije - Referentna arhitektura velikih podataka - Deo 1: Okvir i postupak prijave.
- ISO / IEC TR 20547-2: 2018 Informacione tehnologije - Referentna arhitektura velikih podataka - Deo 2: Slučajevi upotrebe i izvedeni zahtevi.
- ISO / IEC DIS 20547-3 Informacione tehnologije - Referentna arhitektura velikih podataka - Deo 3: Referentna arhitektura.
- ISO / IEC AWI 20547-4 Informacione tehnologije - Referentna arhitektura velikih podataka - Deo 4: Okvir sigurnosti i privatnosti.
- ISO / IEC TR 20547-5: 2018 Informacione tehnologije - Referentna arhitektura velikih podataka - Deo 5: Plan standarda.

2.2. Geoprostorni podaci u Big Data paradigmi

Geoprostorni podaci opisuju fizičke lokacije geoprostornih objekata, njihove međusobne relacije kao i ostale njihove karakteristike. Oni se mogu predstaviti preko različitih formata ili medijuma kao što su planovi, karte, satelitski snimci ili aero-fotografije.

Geoprostorni podaci predstavljaju osnovu za donošenje odluka, bilo da se radi o državnim organima ili komercijalnom sektoru. Njihova upotreba eksponencijalno raste u oblastima kao što su e-komerc i poslovna inteligencija, za donošenje pravovremenih i preciznih odluka. Mnoge organizacije su istovremeno korisnici i proizvođači velikih količina geoprostornih podataka. Prema tome, pristup odgovarajućim podacima u određenom vremensom periodu je ključno za proces donošenja odluka. Zbog toga je neophodno da vladajuće strukture, poslovne kompanije, i javnost ima pristup potpuno ispravnim podacima za potrebe planiranja, analize, navigacije i vizualizacije.

S obzirom da količina, raznolikost i pristupačnost geoprostornim podacima neprekidno raste, nameće se potreba za postojanjem organizovanih skupova podataka. Inicijativa Ujedinjenih Nacija za globalno upravljanje geoprostornim podacima (United Nations Global Geospatial Information Management – UN-GGIM) je pokrenuta sa namerom da se razvije okvir za izgradnju organizovanih skupova podataka, pod nazivom “Integrisani okvir za geoprostorne podatke” (Integrated Geospatial Information Framework – IGIF). On pruža osnovu i smernice a razvoj, integraciju i unapređivanje upravljanja geoprostornim podacima i povezanim resursima u svim državama [42].

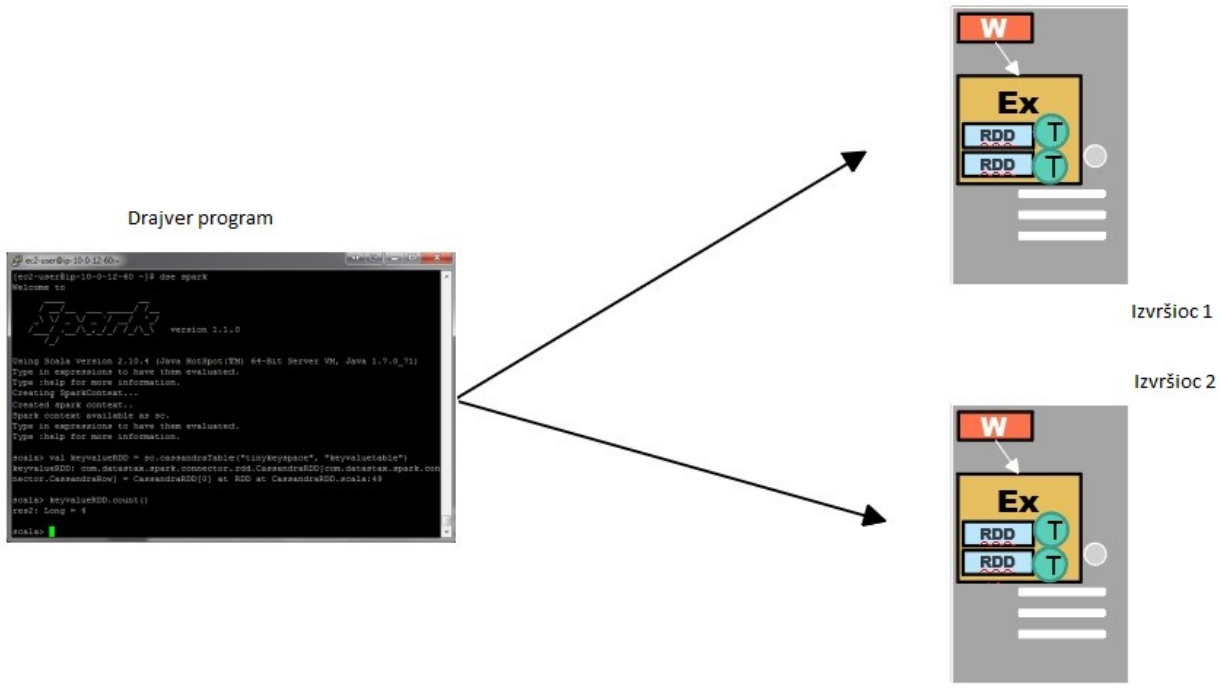
2.3. Tehnologije i tehnološka rešenja

2.3.1. Apache Spark

Apache Spark predstavlja softversku platformu za distribuiranu obradu velikih količina podataka [47]. Osnovne karakteristike Sparka su opštost, odnosno širok domen primene, i brzina izvršavanja. Brzinu izvršavanja Spark postiže keširanjem podataka u radnoj memoriji između dva izvršavanja. Na slici 2.3 je prikazana razilika u načinu izvršavanja i skladištenja podataka između Hadoop-a, kod kojeg se podaci nakon svake operacije skladište na disk, i Spark-a kod kojeg se podaci čuvaju u memoriji između dve operacije.

Slika 2.3 Razlika između Hadoop i Spark modela obrade

Spark aplikacija se sastoji od upravljačkog - drajver programa i izvršioca (Slika 2.4). Drajver program je proces koji izvršava korisnički kod u okviru kojeg se kreira Spark kontekst kreiraju RDD-ovi, izvršavaju transformacije i akcije. Prva funkcija drajver programa je konverzija korisničkog programa u jedinice fizičkog izvršavanja koje se nazivaju zadaci. Spark program implicitno kreira logički direktni aciklični graf operacija. Kada se drajver program izvršava, on konvertuje takav logički graf u fizički plan izvršavanja. Spark vrši nekoliko optimizacija, kao što je spajanje više transformacija, i konvertuje graf izvršavanja u niz faza. Svaka od faza se sastoji od više zadataka. Zadaci su namanje jedinice izvršavanja u Spark-u i tipičan korisnički program može pokrenuti stotine ili hiljade pojedinačnih zadataka [69].



Slika 2.4 Arhitektura Spark aplikacije

Druga funkcija drajver programa je koordinacija rasporeda individualnih zadataka na izvršiocima prema fizičkom planu izvršavanja. Drajver u svakom trenutku poseduje kompletnu sliku o izvršiocima jer se izvršioci, prilikom njihovog pokretanja, prijavljuju drajveru. Prilikom rasporeda zadataka drajver pokušava da na osnovu trenutnog stanja izvršioca rasporedi svaki zadatak na odgovarajuću lokaciju na osnovu distribucije podataka. Tokom izvršavanja zadataka može doći do keširanja podataka. Drajver takođe prati lokacije keširanih podataka i koristi ih da rasporedi buduće zadatke koji pristupaju tim podacima [69].

Spark izvršioci su procesi odgovorni za izvršavanja pojedinačnih zadataka. Izvršioci se pokreću na početku Spark aplikacije i obično se izvršavaju tokom kompletnog njenog trajanja. Može se dogoditi da dođe do prekida izvršenja izvršioca, ali to ne utiče na izvršavanje kompletne aplikacije. Izvršioci imaju dve uloge. Prva je izvršavanje zadataka od kojih se aplikacija sastoji i vraćanje rezultata drajveru, a druga, obezbeđivanje memorijskog skladišta za RDD-ove koji su keširani od strane korisničkog programa [69].

2.3.2. Resilient Distributed Datasets (RDD)

RDD-s su osnovne logičke jedinice u okviru Apache Spark. Oni predstavljaju distribuirane kolekcije objekata uskladištenih u memoriji ili na disku na različitim računarima u klasteru. RDD je podeljen na logičke celine koje se skladište i obrađuju lokalno. Prema tome, programer aplikacije za distribuiranu obradu podataka posmatra RDD kao nedeljivu kolekciju i nad kojom izvršava operacije, ostavljajući Sparku da optimizuje i paralelizuje obradu.

Sledeća važna karakteristika RDD-ova je njihova nepromenljivost. Nakon kreiranja RDD-a podaci u njemu se ne mogu menjati. Jedini način za izmenu podataka je transformacija RDD-a u novi RDD. Na taj način je omogućena distribuirana obrada podataka, jer se fukcije koje obrađuju podatke proselđuju na računare na kojima se vrši obrada, kojom prilikom se podaci transformišu kreirajući novi RDD. U nastavku su navedene osnovne karakteristike RDD-ova.

- Kao što ime kaže, osnovna karakteristika RDD-ova, je *resilience*, odnosno u prevodu, elastičnost, izdržljivost. Ova osobina ozačava mogućnost RDD da se oporavi od problema koji mogu dovesti do gubitka delova podataka. Na primer, usled hardverskog kvara, jedan računar u klasteru može prestati da radi. Samim tim podaci koji su bili uskladišteni na njemu su izgubljeni. Zbog toga, RDD sadrži log u kojem se nalaze informacije neophodne za rekonstrukciju izgubljenih podataka. Navedena karakteristika RDD-a se drugačije naziva otpornost na greške.
- Distribuiranost podataka u okviru klastera.
- Lenja evaluacija (lazy evaluation), odnosno izvršavanje. Nakon definisanja RDD-a u okviru Spark aplikacije, on i dalje ne sardži nikakve podatke. Čak ni korišćenje određenih operacija, treansformacija, nad RDD-om, ne dovodi do učitavanja podataka. Podaci se učitavaju u RDD tek kad se zatraži rezultat obrade, bilo da je u pitanju generisanje rezultata ili skladištenje transformisanih podataka u okviru fajl sistema.
- Nempromenljivost podataka u okviru RDD-a.
- Spark koristi princip izračunavanja u okviru memorije, što znači da se podaci dokle god je to moguće čuvaju i obrađuju u memoriji. Tek ukoliko nije moguće čuvati sve podatke u memoriji oni se skladište na disku.
- RDD je izdeljen na logičke delove koji se nazivaju particije.

Dve osnovne grupe operacija nad RDD-ovima su transformacije i akcije. Transformacije predstavljaju preslikavanje ulaznog RDD u izlazni RDD. Pri tom se podaci u ulaznom RDD ne menjaju, i takođe ove operacije ne pokreću izvršavanje obrade nad podacima (lazy evaluation). U tabeli 2.1 su prikazane osnovne RDD transformacije.

Tabela 2.1 RDD transformacije

Funkcija	**Opis**
map()	Generiše izlazni RDD tako što primenjuje zadatu funkciju na svaki element
filter()	Generiše izlazni RDD tako što zadržava samo one elemente za koje zadata funkcija vraća tačno
reduceByKey()	Agregira podatke za svaki ključ primenom zadate funkcije
groupByKey()	Grupiše elemente prema ključu, odnosno prevodi (ključ, vrednost) u formu (ključ, niz vrednosti)
union()	Spaja dva RDD-a u jedan
intersection()	Vraća RDD koji sadrži identične lemente iz dva ulazna RDD-a

Akcije predstavljaju funkcije koje na osnovu podataka iz ulaznog RDD-a izračunavaju rezultat (Tabela 2.2). Kao što je već pomenuto, Spark pokreće proces izračunanja tek kada dođe do izvršavanja akcije. Prateći graf izvršavanja, sekvencijalno se izvršavaju transformacije i na kraju se izvršava akcija koja vraća rezultat u drajver program.

Tabela 2.2 RDD akcije

Funkcija	**Opis**
count()	Izračunava ukupan broj elemenata u RDD
collect()	Kreira niz koji sadrži sve elemente RDD-a
reduce()	Izračunava agregirani rezultat na osnovu elemenata RDD-a primenom zadate funkcije za agregaciju.
take(n)	Kreira niz koji sadrži prvih n elementata RDD-a
foreach(operation)	Izvršava zadatu operaciju nad svakim elementom RDD-a
first()	Vraća prvi element RDD-a

RDD se može generisati na tri načina.

- Učitavanjem podataka iz spoljašnjeg skupa podataka. Ne primer, učitavanjem podataka iz fajlova ili ključ-vrednost skladišta.
- Paralelizacijom kolekcije objekata.
- Transformacijom postojećeg RDD-a.

2.3.3. Dataframe

Osnovna apstrakcija u Spark SQL aplikacionom programerskom interfejsu (API) je DataFrame. Predstavlja distribuiranu kolekcija redova sa homogenom šemom. DataFrame se može posmatrati kao tabela u relacionoj bazi podataka, jer pored samih podataka sadrži i šemu prema kojoj su oni organizovani. Najlakši način za generisanje DataFrame-a je pridruživanje šeme postojećem RDD-u. Naravno, razvijeni su i mehanizmi kojima se DataFrame automatski generiše prilikom učitavanja iz nekog skladišta podataka.

DataFrame je uveden u Apache Spark od verzije 1.3, kada su autori su ustanovili da je moguće višestruko ubrzati izvršavanje aplikacija ukoliko je šema podataka unapred definisana. U nastavku su navedene osnovne karakteristike DataFrame strukture podataka.

- Omogućuje korišćenje engine-a za optimizaciju obrade, poput Catalyst optimizatora, koji pruža mogućnost optimizacije zasnovane na pravilima i troškovima. Optimizacija zasnovana na pravilima određuje način izvršavanja upita na osnovu skupa definisanih pravila, dok optimizacija zasnovna na troškovima generiše više planova izvršavanja, poredi ih, i određuje onaj koji ima najniže troškove [63].

- Obrada strukturiranih podataka, odnosno podataka kojima je pridružena semantika.

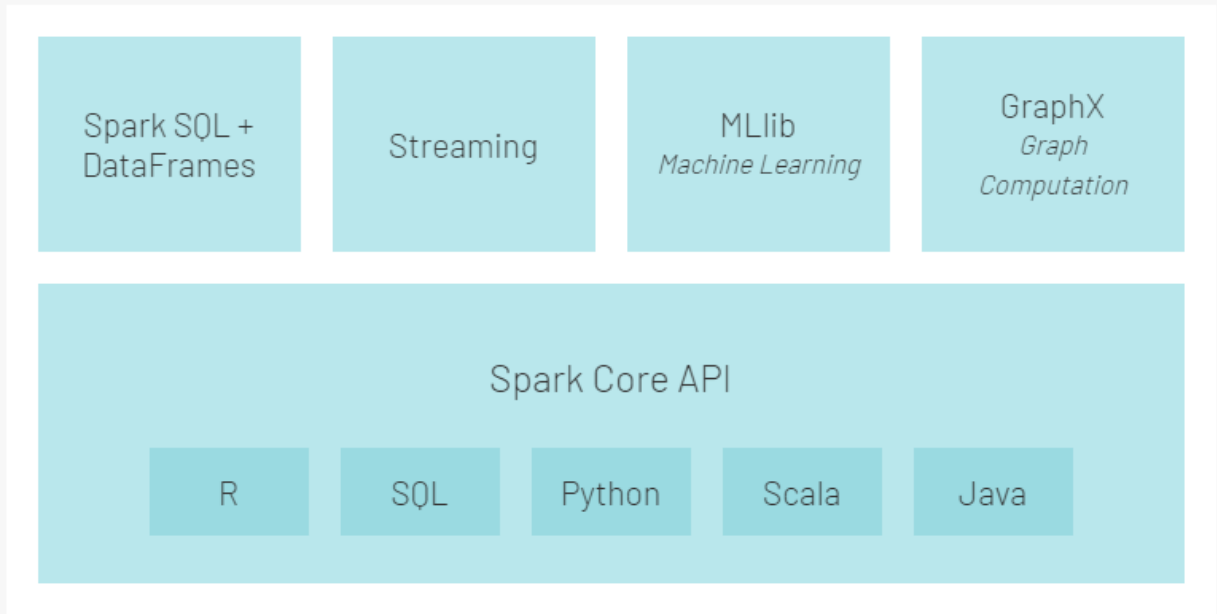
- Proizvoljno upravljanje memorijom. Za razliku od RDD gde se podaci skladište u memoriji izvršioca, kod DataFrame, podaci se skladište u RAM-u, ali van memorijskog prostora izvršioca, što omogućuje manji uticaj *garbage collection* na performanse.
- Poput RDD, i DataFrame podržava veliki broj različitih formata podataka.
- DataFrame je veoma skalabilan i omogućuje obradu podataka reda veličine, od megabajta do petabajta.
- DataFrame se lako integriše sa drugim BigData alatima poput Hive [9] ili Hbase [8], kao i sa drugim elementima spark ekosistema.

Rad sa DataFrame u okviru SparkSQL omogućuje korišćenje svih važnijih SQL tipova podataka, uključujući osnovne *boolean*, *integer*, *double*, *decimal*, *string*, *date*, i *timestamp*, kao i kompleksne (npr. ne-atomske) tipove podataka: *structs*, *arrays*, *maps* i *unions*. Pored toga, kompleksni tipovi

podataka mogu biti ugnježdeni, te na taj način omogućiti modelovanje kompleksnih strukura podataka. Takođe, značajna je DataFrame podrška korisnički definisanim tipovima. Na taj način je moguće, kreirati i koristiti specifične jezičke konstrukte u SparkSQL upitima, što omogućuje rešavanje problema na visokim nivoima apstrakcije. Na taj način je moguće da se precizno i detaljno modeluju podaci iz raznovrsnih izvora, i različitih formata, poput Hive-a, relacionih baza podataka, JSON, Parquet, itd.

2.3.4. Apache Spark ekosistem

Prema [47] Spark platforma se sastoji od nekoliko usko povezanih komponenti. U osnovi Spark predstavlja platformu za upravljanje, distribuciju i nadzor aplikacija koje se sastoje od mnogo zadataka, nad nizom izvršioca u okviru računarskog klastera. Zbog toga što je osnovni *engine* brz i opšte namene, on omogućuje izvršavanje različitih komponenti višeg nivoa kao što su SQL ili mašinsko učenje. Pomenute komponente su osmišljene da imaju mogućnost bliske saradnje poput biblioteka u okviru softverske aplikacije. Slika 2.5 prikazuje komponente Spark platforme kao i programske jezike koji se mogu koristiti za interakciju sa njom.



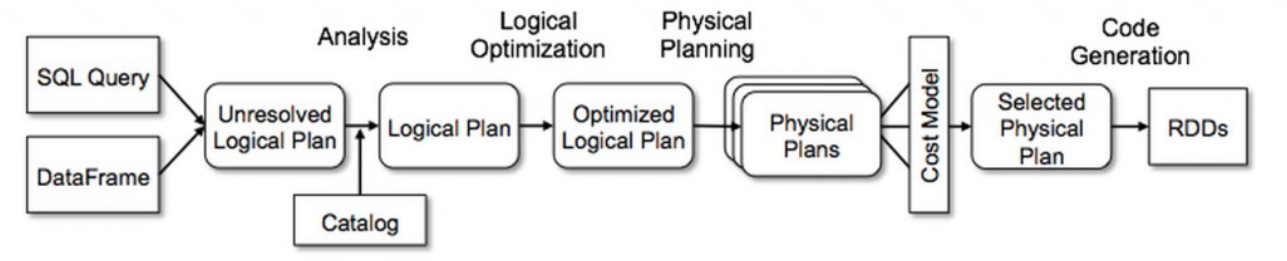
Slika 2.5 Apache Spark ekosistem (preuzeto sa https://databricks.com/spark/about)

2.3.4.1. Spark SQL

Spark SQL je biblioteka koja omogućuje efikasnu distribuiranu obradu podataka tako što osnovnom engine-u pruža podatke o strukturi podataka i same obrade. Na osnovu tih informacija Spark vrši dodatnu optimizaciju procesa obrade. Definisanje i manipulacija strukturiranim i polustrukturiranm podacima se vrši kroz SQL izraze ili DataFrame API.

Spark SQL omogućuje učitavanje podataka iz velikog broja različitih izvora te na taj način podržava različite obalsti primene. Neki od izvora podržanih od Spark SQL su Hive, Avro, Parquet, ORC, JSON, i JDBC. Posebno treba istaći potpunu kompatibilnost sa Hive podacima jer Spark SQL pretenduje da nasledi Hive-a na mestu najčešće korišćenog alata za distribuiranu obradu strukturiranih podataka. Druga važna karakteristika Spark SQL-a je ugrađena tolerancija na greške u sistemu u toku izvršavanja. Prema tome, u koliko dođe do prekida u radu bilo kog elementa u distribiranom sistemu Spark će to detektovati i preusmeriti izvršavanje na drugi element u okviru sistema.

Ipak, najznačajnija karakteristika Spark SQL je optimizacija upita zasnovana na Catalyst optimizatoru koji se nalazi u jezgru Spark platforme. Catalyst optimizator podržava optimizaciju zasnovanu na pravilima i optimizaciju zasnovanu na troškovima. Prilikom optimizacije zasnovane na pravilima koristi se skup pravila da bi se odredio plan izvršavanja SQL upita. U slučaju optimizacije zasnovane na troškovima definiše se više planova izvršavanja primenom pravila, a zatim se za svaki od njih određuju troškovi izvršavanja [23].



Slika 2.6 Postupak optimizacije Spark SQL upita (preuzeto sa https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html)

2.3.4.2. Spark Streaming

Spark Streaming predstavlja proširenje Spark-a koje omogućuje skalabilnu i visoko-propusnu obradu serija podataka u realnom vremenu koja je pri tom otporna na greške koje se mogu dogoditi u sistemu. Serije podatka mogu poticati iz izvora kao što su Kafka, Flume, Kinesis ili TCP soket. Nakon obrade podaci se mogu skladištiti u fajl sistem, bazu podataka ili se vizualizovati u realnom vremenu. Spark koristi micro-batching tehniku za obradu serija podataka u realnom vremenu. Na taj način serija podataka se deli na male delove koji sadrže određeni broj zapisa i zatim se obrada vrši nad tim delovima primenom osnovnih Spark funkcija.

Postoje tri faze obrade podataka u okviru Spark Streaming-a: prikupljanje, obrada i skladištenje. Podaci se mogu preuzimati iz osnovnih izvora za koje je podrška ugrađena u API ili napredni koje je potrebno uključiti dodavanjem odgovarajućih biblioteka. U osnovne izvore spadaju fajl sistem i soket konekcije, a u napredne Kafka, Flume, i Kinesis. Faza obrade predstavlja primenu funkcija visokog nivoa, kao što su map, reduce, join i window, nad učitanim podacima. Na kraju se rezultat obrade može skladištiti ili prezentovati korisniku primenom tehnika vizualizacije.

Serije podataka u okviru Spark Streaming se mogu predstaviti apstrakcijom visokog nivoa koja se naziva DStream. Dstream može nastati u postupku učitavanja podataka ili se može kreirati od drugih Dstream-ova u postupku obrade. Interno Dstream predstavlja sekvencu RDD-ova.

2.3.4.3. Mllib

Spark dolazi sa ugrađenom podrškom za osnovne funkcionalnosti mašinskog učenja, u okviru biblioteke koja se naziva Mllib [47]. MLlib pruža više tipova algoritama za mašinsko učenje, kao što su klasifikacija, regresija, klasterovanje, zajedničko filtriranje, kao i funkcionalnosti poput evaluacije modela i učitavanja podataka. Takođe, pruža i ML primitive nižeg nivoa uključujući, generički algoritam optimizacije opadanja gradijenta. Sve pomenute metode su dizajnirane tako da budu skalabilne.

2.3.4.4. GraphX

GraphX je biblioteka za rad sa grafovima i izvršavanje graf baziranih paralelnih izračunavanja [47]. Poput Spark Streaming i Spark SQL, GraphX proširuje Spark RDD API, omogućujući kreiranje usmerenog grafa sa proizvoljnim atributima čvorova i ivica. GraphX takođe poseduje različite operatore za rad sa grafovima (naprimer, subgraph i mapVertices) i biblioteku uobičajnih algoritama (na primer, PageRank i brojanje trouglova).

2.3.5. HBase

HBase predstavlja open-source implementaciju Google Bigtable skladišta podataka [31]. Prema tome, definicija koja se odnosi na Bigtable, važi i za Hbase. Da bi izbegli gubitak značenja prilikom prevoda, definicija će biti prikazana u originalnom obliku, „*A Bigtable is a sparse, distributed, persistent multidimensional sorted map*“. Jednostavnije rečeno, Hbase predstavlja ključ-vrednost distribuirano višedimenzionalno skladiše podataka. Ključ čine identifikator reda, identifikator kolone, i vreme, a vrednost predstavlja bilo kakav niz bajtova, bez pridružene semantike.

2.3.5.1. *HBase terminologija*

Prema [8] Hbase terminologija uključuje sledeće pojmove:

- Tabela
  - HBase tabela se sastoji iz velikog broja redova.
- Red
  - Red u okviru Hbase se sastoji iz identifikatora reda i jedne ili više kolona sa pridruženim vrednostima. Redovi su uskladišteni prema alfabetskom redosledu. Zbog toga je veoma značajno pravilno dizajnirati identifikator reda, odnosno tako da povezani redovi budu uskladišteni jedan blizu drugog.
- Kolona
  - Kolona u Hbase se sastoji od familije kolona i identifikatora kolone koji su povezani : karakterom.
- Familija kolona
  - Familije kolona fizički povezuju skupove kolona i njihovih vrednosti, najčešće iz razloga perfomanse. Za svaku kolonu se mogu postaviti parametri skladištenja, na primer da li se verdnosti trebaju keširati u memoriji, kako su podaci kompresovani, i drugi. Svi redovi moraju imati iste familije kolona, ali sadržaj istih se može razlikovati.
- Identifikator kolone
  - Identifikator kolone se dodaje familiji kolona kako bi se definisao indeks za dati podatak. Familije kolona moraju biti fiksirane u trenutku kreiranja tabele, a identifikatori kolona se mogu značajno razlikovati od reda do reda. T
- Ćelija
  - Ćelija predstavlja kombinaciju reda, familije kolona i identifikatora kolone i sadrži vrednost i vremenski trenutak (timestamp) kojim se definiše verzija podatka.
- Timestamp
  - Timestamp se pridružuje svakoj vrednosti i definiše njenu verziju.

2.3.6. Integracija HBase – Spark

U [8] su opisana četiri osnovna načina za interakciju između Sparka i Hbase, to su:

- Osnovni način – obezbeđivanje konekcije na Hbase u bilo kojoj tački Spark izvršavanja.
- Spark Streaming - obezbeđivanje konekcije na Hbase u bilo kojoj tački Spark Streaming izvršavanja.

- Spark Bulk Load – mogućnost direktnog zapisa u HBase Hfile struturu za masovni import podataka.
- SparkSQL/DataFrames – mogućnost korišćenjenja SparkSQL nad Hbase tabelama.

2.4. Primena Big Data tehnoloških rešenja nad geoprostornim podacima

2.4.1. GeoSpark

Biblioteka otvorenog koda GeoSpark je jedan od projekata DataSys Lab sa Arizona State University [45]. Ona proširuje Apache Spark sa skupom prostornih RDD-ova (Spatial Resilient Distributed Datasets - SRDD) koji omogućuju efikasno učitavanje, obradu i analizu velikih količina prostornih podataka u distribuiranom okruženju. GeoSpark omogućuje efikasnu distribuciju SRDD prostornih elemenata u okviru računarskog klastera i definiše paralelizovane prostorne operacije, prema Open Geosptial Consortium (OGC) standardima, koje omogućuju korisnicima razvoj analitičkih programa za prostorne podatke. SRDD struktura omogućuje izvođenje prostornih upita (npr. okvirni upit, KNN upit i Join upit) nad velikim količinama prostornih podataka. Operacije prostornih upita su implementirane u Spatial Query Processing Layer-u koji definiše kako su prostorne objektno-relacione n-torke indeksirane i skladištene u okviru SRDD-a. Eksperimenti su pokazali da GeoSpark ima mnogo veću brzinu od Hadoop baziranih sistema u aplikacijama prostornih analiza kao što je prostorno spajanje, agregacija i prostorna ko-lokacija.

2.4.2. Magellan

Biblioteka otvorenog koda Magellan, kompanije Geospatial Analytics, koristi Spark kao platformu za izvršavanje geoprostornih upita [49]. Zaniva se na Spark Catalyst optimizatoru za efikasno izvršavanje prostornih JOIN operacija. Takođe kroz SparkSQL konstrukte za definisanje upite u izvornom domen jeziku, kao i Pyspark radi integracije sa Python. Biblioteka podržava Esri formate fajlova. Cilj je podrška kompletnom skupu OpenGIS primitiva za SQL prostorne funkcije i operatore zajedno sa dodatnim topološkim funkcijama. Biblioteka podržava Scala i Python aplikacije.

2.4.3. GeoMesa

Geomesa predstavlja proširenje Apache Accumulo baze podataka za skladištenje velikih količina prostorno-vremenskih vektorskih podataka. Svaki vektorski podatak u okviru Geomesa je opisan sa dve prostorne koordinate, geografske širine i dužine, i vremenskom koordinatom. Podaci se skladište u obliku dve vrste ključ-vrednost parova, indeksa i podataka. Indeks ključ-vrednost parovi

omogućuju prostorno vremensko indeksiranje, dok ključvrednost parovi podataka sadrže podatke vezane za posmatrane prostorno-vremenske koordinate [32]. Na osnovu prostornih koordinata se formira Geohash [35] string od 7 karaktera koji odgovara prostornoj rezoluciji od 150 metara. Vremenska koordinata je opisana stringom formata godina-mesec-dan-sat što obezbeđuje vremesku rezoluciju od 1 sata. Indeks ključ se formira preplitanjem dva prethodno opisana stringa. Šema indeksiranja se može prilagoditi strukturi i rezoluciji korisnikovih podataka.

2.4.4. GeoWave

Biblioteka GeoWave služi za skladištenje, indeksiranje i pretragu više-dimenzionalnih podataka u okviru Big Data skladišta [37]. Uključuje implementaciju OGC prostornih tipova podatka (do tri dimenzije), kao i implementaciju ograničenih i neograničenih vremenskih vrednosti.

GeoWave indeks služi kao šablon koji se koristi za efikasan pristup podaima u okviru Big Data skladišta u okviru zadatog skupa dimenzija. Svaki indeks može imati podatke iz više adaptera, dokle god ti adapteri podržavaju dimenzije koje indeks koristi. Prema tome, prostorno-vremenski indeks nije u mogućnosti da indeksira podatke bez vremenske komponente, ali samo prostorni indeks može da indeksira prostorno-vremenske podatke e uzimajući u obzir vremensku kompoenentu.

GeoWave koristi krive za popunjavanje prostora (SFC) za indeksiranje podataka u okviru Big Data skadišta. Vrednost indeksa je uskladištena u okviru ključa i može biti korišćena u toku obrade podataka. Takva arhitektura omogućuje da vremena izvršavanja upita, obrade, i renderovanja budu višestruko umanjena.

2.4.5. GeoTrellis

GeoTrellis je biblioteka za efikasnu obradu geoprostornih rasterskih podataka [36]. U saradnji sa Digital Globe, kompanijom za prikupljanje i distribuciju satelitskih snimaka, implementirana je podrška za distribuirano procesiranje velikih geoprostornih rasterskih podataka bazirano na Spark-u. Geoprostorni rasteri se prostorno dele na blokove i skladište u okviru HDFS ili u Apache Accumulo bazu podataka u obliku ključ-vrednost parova. Ključ predstavlja jedinstveni identifikator bloka, a vrednost raster koji predstavlja deo originalnog fajla. Prilikom izvršavanja svaki izvršioc obrađuje jedan ili više blokova.

2.5. Tradicionalne tehnike za upravljanje oblacima tačaka

Ekspanzijom tehnologije laserskog skeniranja na početku ovog veka javio se problem upravljanja oblacima tačaka. Jedan od odgovora na taj problem bi bilo skladištenje oblaka tačaka u okviru

sistema za upravljanje bazama podataka (SUBP). Međutim jedino Oracle i PostgreSQL trenutno poseduju podršku za tu vrstu podataka.

Oracle je u paket Oracle Spatial uveo podršku rad sa oblacima kroz implementaciju SDO_PC tipa podataka [60]. Tačke iz oblaka tačaka su fizički uskladištene po blokovima tako da svaki blok sadrži prostorno bliske tačke. Geometrije blokova se kreiraju na način da svaki blok sadrži približno jednak broj tačaka i one su indeksirane primenom R-stabla. Upiti nad oblakom tačaka se izvršavaju u dva koraka. U prvom koraku se određuju blokovi koji su relevatni za dati upit. U drugom koraku se izdvajaju tačke iz blokova i određuju one koje odgovaraju datom upitu. Tačke unutar bloka nisu indeksirane i obrađuju se sekvencijalno. To ne utiče mnogo na performansu jer svaki blok sadrži relativno mali broj tačaka, nekoliko hiljada ili nekoliko desetina hiljada.

U okviru [58] je opisano proširenje PostgreSQL tipovima podataka i funkcijama koje omogućuju skladištenje i manipulaciju oblacima tačaka. Pristup je sličan kao i kod Oracle-a. Prostorno bliske tačke se skladište po blokovima, nazvanim "patches", od kojih svaki sadrži po nekoliko stotina tačaka. Preporuka je da maksimalan broj tačaka u bloku iznosi 600. Blokovi su pravougaonog oblika i konstruisani su tako da svaki sadrži približno isti broj tačaka.

Prethodno opisana rešenja su prvenstveno orijentisana na skladištenje 2.5D oblaka tačaka. U [61] je opisano indeksiranje 3D oblaka tačaka primenom oktalnog stabla u okviru Oracle Spatial-a. Oktalno stablo je hijerarhijska struktura podataka koja se formira rekurzivnom dekompozicijom 3D regiona. Jedan čvor oktalnog stabla predstavlja kocku u tom regionu. Svaki od osam podčvorova jednog čvora predstavlja oktant u okviru nadređenog čvora, a podaci se skladište u okviru listova. Jedan od najčešće korišćenih načina za realizaciju oktalnog stabla je primenom heš tabele. U tom slučaju se svakom čvoru pridružuje ključ koji ga identifikuje i služi za izračunavanje njegove adrese u okviru heš tabele. Ključ se može izračunati na osnovu pozicije čvora unutar hijerarhije oktalnog stabla kroz sistematsku orijentaciju oktanata u čvoru. Jedan od najčešće korišćenih mehanizama za generisanje ključeva su Mortonovi kodovi [17][61]. Oni se jednostavno izračunavaju i pružaju dobru prostornu bliskost.

2.6. Oblak tačaka u Big Data paradigmi

Kada je reč o primeni Apache Spark platforme nad velikim oblacima tačaka, postoji relativno mali broj istraživanja. Najveći deo objavljenih radova je rezultat istraživanja u okviru projekta Iqmulus [43].

Liu i Boehm [15] su sproveli jedno od prvih istraživanja koje se bavilo skladištenjem velikih oblaka tačaka u okviru NoSQL baza podataka, konkretno MongoDB. Ulazni oblaci tačaka su skladišteni u okviru LAS fajlova. Oblaci tačaka su skladišteni u tri kolekcije dokumenata. Prva kolekcija je sadržala dokumente u okviru kojih su skladišteni metapodaci izvedeni analizom LAS fajlova. Druga kolekcija je sadržala atribute fajlova, a treća podatke iz fajlova. Eksperimentalno su pokazali da je rešenje zasnovano na NoSQL bazi podataka, skalabilno, otporno na greške i obezbeđuje *high-availability.* Takođe, MongoDB poseduje internu implementaciju *map-reduce* paradigme koja predstavlja osnovu obrade velikih količina podataka.

Autori su u okviru [16] istraživali mogućnosti primene postojećih softverskih biblioteka za učitavanje oblaka tačaka u Apache Spark. Predstavili su pristup koji primenom map funkcije distribuira učitavanje velikih količina podatka, reda veličine nekoliko milijardi tačaka, na računare u okviru klastera. Upoređivana su dva pristupa za učitavanje podataka, „naivni“ pristup i isecanje. Oba sistema se zasnivaju na pretpostavci da postoji distribuirani fajl sistem koji je *mount*-ovan kao lokalni virtualni fajl sistem na svim računarima u klasteru. Kod „naivnog“ načina se formira niz fajlova koje treba učitati i zatim se taj niz deli na onoliko delova koliko ima računara u klasteru i svakom računaru se prosleđuje jedan podniz. Međutim, takav pristup ima dva osnovna nedostatka. Prvo, ukoliko postoji manje fajlova nego računara u klasteru, deo računara će biti maksimalno opterećen, dok će deo računara biti ne aktivan. Drugi, problem je što se podaci iz jednog fajla kompletno učitavaju u memoriju računara, što u slučaju velikih fajlova dovodi do pada sistema nakon nestanka slobodnog memorijskog prostora. Pristup koji koristi isecanje rešava oba prethodna problema tako što se fajlovi dele u isečke predefinisane veličine. Da bi se to postiglo koriste se dve map funkcije. Prva funkcija emituje za svaki fajl par vrednosti u obliku (naziv fajla, broj tačaka), a druga funkcija zatim emituje tri vrednosti koje definišu isečak (naziv fajla, početak, kraj). Zatim se niz isečaka distribuira na sve računare u klasteru, gde svaki učitava samo deo fajla. Oba pristupa su poređena sa Iqumulus bibliotekom koja izvorno implementira učitavanje podataka u okviru Apach Spark sistema. Pokazano je da metoda isečaka poseduje veoma dobru skalabilnost u pogledu količine podataka i veličine klastera, ali da u određenoj meri ima slabije performanse u odnosu na izvornu implementaciju. Na kraju, iako izvorno implementiran metod za učitavanja oblaka tačaka se pokazuje kao najbolje rešenje, mogućnost korišćenja postojećih alata otvara dodatne mogućnosti za obradu velikih oblaka tačaka na Apache Spark platformi.

U [5] autori su predložili korišćenje Z krive za indeksiranje oblaka tačaka i ispitali mogućnosti njene primene za određivanje K najbližih suseda. U njihovoj implementaciji se za svaku tačku određuje 62 bitni Z indeks, odnosno Mortonov kod. Zatim se iz Morton koda generiše broj particije

koji sadrži parni broj bita (n bita po x i y koordinatama) i podaci se distribuiraju u okviru Apache Spark sistema na osnovu broja particije. Autori su poredili performanse dva algoritma za određivanje najbližih suseda: FLANN, zasnovan na postojećoj softverskoj biblioteci, i WINDOW, zasnovan na DataFrame API. Takođe, poređena su dva pristupa za podelu na particije: sa i bez bafera. Pristup sa baferom pored tačaka koje pripadaju određenoj particiji, uključuje i određeni broj tačaka iz susednih particija, kako bi se obezbedilo da su za svaku tačku, u trenutku obrade, prisutni u memoriji svi njeni mogući susedi. Autori su došli do zaključka da su najbolje performanse ostvarene primenom postojeće softverske FLANN biblioteke, kao i da korišćenje bafera nije značajno doprinelo ni u pogledu tačnosti, ni u pogledu performanse.

Liu i Boehm [48] su istraživali mogućnost upotrebe Apache Spark platforme za klasifikaciju velikih oblaka tačaka. Oni su implementirali *random forest* algoritam [59] primenom pet RDD transformacija. U prvom koraku se niz fajlova paralelizuje u okviru klastera računara. U sledećem koraku se paralelno učitavaju tačke. Treći korak se sastoji od izdvajanja slučajnih objekata, a zatim se u četvrtom koraku vrši klasifikacija na osnovu unapred treniranog modela. U petom koraku se rezultati klasifikacije pridružuju tačkama generisanim u drugom koraku. Autori su ustanovili da je potignuta tačnost klasifikacije zadovoljavajuće, kao i da implementirano rešenje linearno sklira sa povećanjem broja tačaka.

Autori [24] su predstavili istraživanje mogućnosti upotrebe Big Data tehnika, konkretno Apache Spark i Apache Cassandra za distribuiranu obradu veoma velikih LiDAR skupova podataka. Za studiju slučaja je odabran proces izrade digitalnog modela terena na osnovu velike količine LiDAR podataka. Za paralelizaciju i distribuiranje je izabran jednostavan pristup podele LiDAR skupa podataka na dvodimenzionalne kvadratne zone, a zatim su podaci iz svake zone importovani u Cassandru kao jedinstveni binarni objekti. Na taj način je ostavljeno internim mehanizmima Cassandra sistema da zapravo izvrše distribuciju. Iako Big Data tehnike omogućuju paralelnu podelu inicijalnog skupa podataka to nije korišćeno u ovom istraživanju te može predstavljati dodatno unapređenje predloženog metoda. U sledećem koraku je izvršena izrada digitalnog modela terena, za svaku zonu ponaosob, primenom progresivnog morfološkog filtera, gde je Spark iskorišćen kao platforma za distribuiranu obradu. S obzirom da se granice zona ne preklapaju, autori su morali da reše problem graničnih efekta, odnosno odstupanjima ili prazninama na granicama zona. Autori su taj problem rešili tako što su generisali još jedan skup zona, koje su nazvali correction patches, tako da se centralni delovi tih zona poklapaju sa graničnim delovima prvobitno generisanih zona. Takav dodatni skup podataka je iskorišćen za popravku graničnih efekata. Rezultati eksperimenta su pokazali da je primenom distribuirane obrade postignuto

značajno ubrzanje, u najboljem slučaju skoro deset puta, u odnosu na lokalnu obradu. Zaključeno je da je za postizanje optimalnih rezultata neophodno izabrati odgovarajući prostorni opseg zona, odnosno njihovu veličinu. Optimalan prostorni opseg je neophodana kako bi se omogućila najbolja prostorna raspodela zona po Spark izvršiocima, kako bi svaki izvršioc lokalno posedovao sve podatke koji su mu potrebni. Takođe je primećeno da veći broj tačaka po zoni utiče na slabiju performansu distribuirane obrade (samo 4 puta bolje nego lokalno), ali autori nisu dali objašnjenje za uticaj tog parametra.

U okviru [64] je prikazan metod za kodiranje/dekodiranje trodimenzionalnih oblaka tačaka u okviru diskretnog globalnog grid sistema (Discrete Global Grid System, DGGS) koji predstavlja prostor Zemlje kao hijerahijsku sekvencu identičnih površina ili zapremina, poput geohash. Prednost ove metode u odnosu na geohash je što omogućuje rad sa oblacima tačaka visoke rezolucije, odnosno prevazilazi ograničenje base36 formata koji koristi geohash i koji ima ograničenja u pogledu veličine ćelije i nejednakih površina. Autori prevazilaze ograničenje Morton koda od 64 bita tako što koriste dva pristupa. U prvom pristupu koriste stringove, a u drugom preplitanje bita i lookup tabelu.

Istraživanje [71] prikazuje sistem za upravljanje bazom podataka, implementiran u Hbase koji je proširen novom metodom za kodiranje podataka i mehanizmima za indeksiranje tako da podrži oblak tačaka koji beleži kompletne odbitke. Pomenuti sistem je ograničen na podatke dobijene Lidar snimanjem iz vazduha.

U okviru [72] je prikazana istraživanje skladištenja velikih količina podataka (na nivou države) dobijenih Lidar skeniranjem iz vazduha u okviru Hbase. Predstavljena su četiri načina za organizaciju podataka, dva zasnovana na flat modelu, a dva zasnovana na blok modelu. Identifikator reda je definisan u zavisnosti od pristupa, u slučaju blok modela, korišćen je jedan Hilbet kod kao identifikator bloka, a u slučaju flat modela, korišćena su dva Hilbert koda, jedan na nivou bloka, a drugi za identifikaciju tačke u okviru bloka. Takođe su korišćene dve organizacije kolona, jedna gde je svaki atribut predstavljen posebnom kolonom i druga gde su atributi kompresovani u niz bajtova i predstavljeni kao jedna kolona. Rezultati eksperimenta su pokazali da Hbase rešenje mnogo bolje skalira u odnosu na povećanje broja tačaka u odnosnu na PosgreSQL rešenje koje je korišćeno kao referenca. Takođe je pokazano da blok model sa grupisanim atributima pokazuje najbolje performanse kada je u pitanju prostorno filtriranje. Autori, se nisu bavili filtriranjem prema vrednosti atributa koje bi moglo biti problematično kod rešenja sa grupisanim atributima jer bi zahtevalo dodatnu serijalizaciju/deserijalizaciju.

2.7. Zaključna razmatranja stanja u oblasti

Na osnovu pregleda stanja u oblasti mogu se izvesti elementi koji su zajednički za sva istraživanja. Pre svega, većina rešenja za indeksiranje geoprostornih i prostorno-vremenskih podataka zasnivaju na krivama. To je samo po sebi opravdano jer predstavlja najpogodniji metod indeksiranja u distribuiranom okruženju. Najčešće se koriste Z-kriva ili Hilbertova kriva. Takođe, postojeća rešenja zasnovana na Big Data paragimi za upravljanje rasterskim i vektorskim podacima su razvijena u velikoj meri i omogućuju primenu u okviru produkcionih aplikacija. S druge strane, istraživanja o upravljanju oblakom tačaka u Big Data paradigmi su još uvek relativno malobrojna i najveći broj publikacija su rezultat Iqmulus projekta. Pomenuta istraživanja su razuđena i pokrivaju određene uske oblasti primene. Prema saznanju autora ne postoji istraživanje koje razmatra celokupnu arhitekturu modela za upravljanje oblakom tačaka u Big Data paradigmi. Kada je reč o sistemima za upravljanje relacionim bazama podataka (Oracle i PostgreSQL), oni poseduju veoma razvijenu funkcionalnost za skladištenje i obradu oblaka tačaka, ali pokazuju nedostatak skalabilnosti pri značajnom povećanju količine podataka ili povećanju brzine pristizanja serija podataka.

Na osnovu navedenog, u okviru modela za upravljanje velikim serijama geoprostornih podataka je fokus postavljen na upravljanje oblakom tačaka zbog njegove kompleksnosti i još uvek relativno manje pokrivenosti istraživanjima. Model je zasnovan na novijim Big Data konceptima i platformama poput Sparka te stoga prevazilazi ograničenja sa kojima se suočavaju tradicionalne metode upravljanja. Dalje, njegova prednost je to što je implementiran primenom SparkSQL modula i DataFrame API-ja što omogućuje bolju optimizacija upita zbog poznavanja strukture podataka, kao i korišćenje jezika domena za opis podataka i operacija. Takav pristup omogućuje korišćenje sistema bez potrebe za poznavanjem detalja o njemu. Na primer, korisnik je u mogućnosti da izvršava operacije zadavanjem SQL upita. Na kraju, kao još jednu prednost možemo navesti i mogućnost upravljanja vremenskim serijama podataka i praćenje istorije izmena. Takva funkcionalnost može biti veoma značajna korisnicima sistema, a podrška za to već svakako postoji u okviru Big Data platformi, na primer u okviru Hbase.

3. Prostorno indeksiranje i krive koje popunjavaju prostor

Kao što je prikazano u prethodnom poglavlju indeksiranje geoprostornih podataka nije jednostavan zadatak. On postaje još kompleksniji povećanjem dimenzionalnosti podataka. S obzirom na višedimenzionalnu prirodu oblaka tačaka, počevši od tri dimenzije ukoliko posmatramo samo koordinate, do značajno više dimenzija ukoliko se u obzir uzmu i druge karakteristike. Zbog toga se u ovom poglavlju fokus postavlja na indeksiranje oblaka tačaka. Prostorno-vremensko indeksiranje, je ostavljeno van okvira ove disertacije. Međutim, predstavljene metode se mogu generalizovati i na takve skupove podataka.

3.1. Karateristike oblaka tačaka

Poslednjih godina, oblak tačaka se sve više koristi za modelovanje okruženja. Iz matematičke perspektive, oblak tačaka je skup tačaka u trodimenzionalnom prostoru koji opisuje površinu jednog ili više objekata. S obzirom na način prikupljanja, odnosno generisanja, oni predstavljaju nestrukturirane podatke te otuda termin oblak. Takva njihova priroda ih čini posebnim tipom prostornih podataka. S jedne strane su slični vektorskim podacima jer predstavljaju kolekciju tačaka, dok sa druge strane imaju sličnosti sa rasterskim podacima, jer nastaju neselektivnim prikupljanjem podataka. Iako se deo tehnika za obradu rasterskih podataka može primeniti i na oblak tačaka, oni se ipak posmatraju kao poseban tip podataka jer ne predstavljaju regularnu strukturu.

Tačke koje čine oblak tačaka su opisane prostornim koordinatama i pridruženim atributima. U zavisnosti od metode prikupljanja tačke mogu imati pridružene različite atribute. LAS format [65], koji je de facto standardni način za skladištenje oblaka tačaka, tačkama pridružuje sledeće atribute:

- vreme snimanja tačke,
- klasifikaciju, vrsta objekta kojem tačka pripada, teren, zgrada, vegetacija, itd.
- intenzitet povratnog laserskog zraka,
- boju, u RGB formatu,
- broj odbitka laserskog zraka,
- ugao skeniranja

3.2. Tehnike upravljanja oblakom tačaka

Poslednjih godina je oblak tačaka postao osnovni izvor informacija za mnoge vrste prostornih aplikacija. Međutim, velike količine podataka, kao i kompleksnost njihovih atributa, predstavljaju izazov u upravljanju velikim oblacima tačaka. Dodatnu kompleksnost donose slučajevi gde je potrebno kombinovati oblake tačaka iz različitih izvora, sa različitim rezolucijama, i vremenom akvizicije. Preovlađujći način za skladištenje i upravljanje oblacima tačaka predstavljaju rešenja zasnovana na fajl sistemu. Veliki oblaci tačaka se dele na manje blokove, koji sadrže po nekoliko miliona tačaka, a zatim se svaki blok skladišti kao pojedinačni fajl, koristeći najčešće LAS format zapisa. Iako je korišćenje fajl sistema najjednostavniji način za upravljanje oblacima tačaka, to nosi sa sobom veliki broj nedostataka, kao što su praćenje izmena, konkurentan pristup podacima, bekapovanje podataka, i distribucija podataka. Zbog toga je potrebno da svi koji rade sa tako organizovanim oblacima tačaka prate strogo definisane procedure za pristup i obradu podataka. Protekle decenije se radi na razvoju rešenja za upravljanje oblacima tačaka u oviru sistema za upravljanje bazama podataka. Oni obezbeđuju mehanizme kojima se a priori rešavaju nedostaci koje nosi skladištenje u okviru fajl sistema. Sa druge strane, tu postoje dodatni napori za pripremu podataka i njihovo učitavanje u bazu.

U okviru [70] su predstavljene osnovne funkcionalnosti upravljanja oblacima tačaka bez obzira na način organizovanja podataka.

1 Skladištenje X, Y, i Z koordinata i podrška za referentne koordinatne sistema.

2 Skladištenje pridruženih atributa, kao što su, intenzitet, klasifikacija, boja, ili bilo koja druga kombinacija.

3 Prostorna organizacija tačaka koja uključuje efikasnu podelu na blokove, redukovanje dimezionalnosti, i indeksiranje u višedimenzionalnom prostoru.

4 Vremenska komponenta i njen značaj u organizaciji oblaka tačaka. Pri tom, vreme može biti samo pridruženi atribut ili jedna od zasebnih dimenzija oblaka tačaka. U drugom slučaju se vreme koristi u procesu indeksiranja.

5 Tehnike za kompresiju potrebne za redukovanje prostora potrebnog za skladištenje i distribuciju podataka kroz mrežne sisteme.

6 Generisanje različitih nivoa detaljnosti radi smanjenja količine podataka koji se prosleđuju klijentskim aplikacijama radi podrške efikasnoj obradi i vizualizaciji podataka. U drugom slučaju korisniku se u zavisnosti od kretanja kroz renderovan oblak tačaka prosleđuju

različiti nivoi detalja. Ako korisnik posmatra renderovan oblak tačaka sa velike udaljenosti, renderuje se samo mali broj karakterističnih tačaka. Približavanjem tačke gledišta se učitavaju tačke sa sve većim nivoom detaljnosti. Postoje dva pristupa za generisanje nivoa detalja, predefinisani ili kontinualni pristup. U prvom slučaju se generišu nivoi detalja predeinisanog sadržaja, dok se u drugom slučaju svakoj tački pridružuje atribut, značajnost. Zatim se taj atribut može koristiti u postupku indeksiranja i kasnije prilikom upita za dobavljanje samo tačaka koje imaju odgovarajući značaj.

7 Podrška operacijama za učitavanje, selekciju, i algoritmima nad oblakom tačaka, kao što suestimacija normala, pronalaženje najbližih suseda, itd.

8 Tehnike za paralelnu obradu oblaka tačaka radi potpunog iskorišćavanja mogućnosti modernih računara. Dosadašja praksa se uglavnom zasnivala na obradi podataka na pojedinačnim radnim stanicama. Paralelnim izvršavanjem bi se proces obrade značajno ubrzao.

3.2.1. Upravljanje oblacima tačaka zasnovano na fajl sistemu

Dugi niz godina oblaci tačaka su skladišteni u okviru fajlova, unutar strukture direktorijuma u okviru operativnog sistema. Aplikacije za rad sa oblacima tačaka su prema potrebi pristupali tim fajlovima.

Oblaci tačaka mogu biti skladišteni u originalnom obliku (kao nizovi tačka) ili mogu biti pojednostavljeni i skladišteni u formi rastera [56]. Poslednji slučaj se najčešće koristi kada se podaci prosleđuju krajnjim korisnicima radi analiza. Dva osnovna formata za skladištenje oblaka tačaka u originalnoj formi su tekstualni format ili LAS format [65]. Kod tekstualnog formata podaci se skladište u fajlove gde svaka linija teksta predstavlja jednu tačku. Koordinate tačaka, kao i pridruženi atributi, su zapisani u tekstualnoj formi (jedna cifra predstavlja jedan karakter) i razdvojene delimiter karakterom (obično space, tab, ili zarez). S obzirom na način zapisa, tekstualni format zauzima veliki memorijski prostor. Sa druge strane, LAS je binarni format prilagođen za optimizovano skladištenje oblaka tačaka. Pored samih tačaka, LAS format može da sadrži i veliki broj metapodataka. U zavisnosti od raspoloživih atributa, LAS format može da podrži nekoliko različitih zapisa tačaka, na primer, zapis GNSS vremena ili boje tačaka. Dodatno, u poslednje vreme sve više aplikacija podržava, i komprimovani LAS zapis, odnosno LAZ, koji se dobija primenom LASzip algoritma nad LAS fajlovima. Na taj način se količina podataka umanjuje i preko 10 puta. Prema [56], prednosti korišćenja fajlova za skladištenje oblaka tačaka su to što su podržani su u svim relevantnim aplikacijama, bibliotekama i programskim jezicima, jednostavni za

korišćenje, i postoji mogućnost kompresije. Sa druge strane, prema [56], nedostaci su, da porastom količine oblaka tačaka kojima se upravlja, postaje komplikovano pratiti koji skupovi podataka se gde nalaze i kada se koriste. Zbog toga je neophodno održavati veliku količinu metapodataka, što može biti problematično zbog nedostatka standarda. Radi podjednostavljenja pomenutog procesa korisnici često pribegavaju korišćenju rasterskog zapisa oblaka tačaka.

3.2.2. Upravljanje oblakom tačaka primenom baza podataka

Tokom nekoliko prethodnih godina je primetan intenzivan razvoj rešenja za upravljanje oblakom tačaka u open-source i komercijanlnim sistemima za upravljanje bazama podataka, i to u relacionim, kao i ne-relacionim sistemima [56]. Kao što je već pomenuto, baze podataka pružaju brojne prednosti u odnosu na organizaciju po fajlovima. Današnji sistemi za prikupljanje podataka omogućuju česte akvizicije, što vodi ka brzom uvećanju količina uskladištenih podataka, kao i njihove česte izmene. Takav režim rada bi, u slučaju organizacije podatka po fajlovima, zahtevao neprekidnu dokumentaciju metapodataka. U slučaju baza podataka takva funkcionalnost je integrisana u sam sistem. Ostale prednosti baza podataka su, konkurentan pristup više korisnika istovremeno, skalabilnost, i lakša integracija sa drugim tipovima geoprostornih podataka.

Prema [25] osnovne funkcionalnosti koje sistem za upravljanje bazama podataka treba da poseduje da bi podržao oblak tačaka kao tip podataka su, prostorno i alfanumeričko filtriranje, pretraga po primarnom ključu, pronalaženje najbližih suseda, interaktivna vizualizacija, dodavanje, izmena i brisanje zapisa podržano efikasnim šemama indeksiranja.

3.3. Krive za popunjavanje prostora

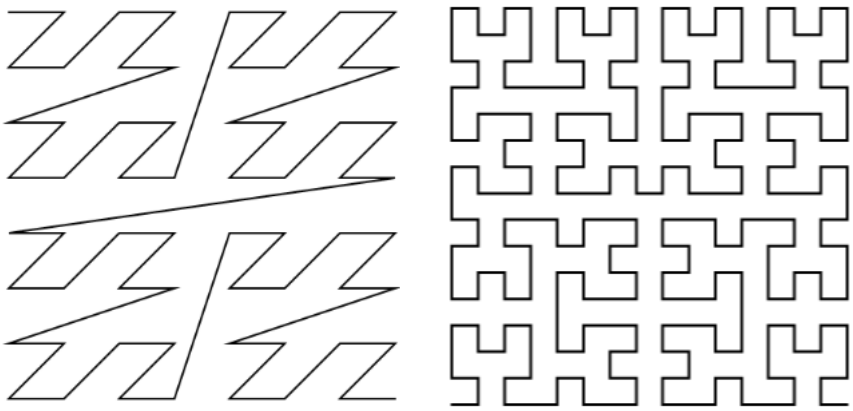
Krive za popunjavanje prostora (engleski Space filling curves, skraćeno SFC) predstavljaju tehniku za transformaciju višedimenzionalnih podataka na jednodimenzionalnu strukturu. S obzirom da računarska memorija predstavlja linearnu strukturu neophodno je primeniti SFC kako bi se oblak tačaka mogao skladištiti u istoj. Veliki broj operacija nad oblakom tačaka koristi tačke koje su prostorno bliske. Zbog toga je osnovi problem koji se javlja pri SFC transformaciji je kako zadržati prostornu bliskost tačaka, odnosno kako da tačke koje su susedne u prostoru budu na susednim (ili bar bliskim) lokacijama u računarskoj memoriji. U distribuiranim računarskim sistemima je to još značajnije jer se teži minimalnoj razmeni podataka između računara, odnosno minimalnoj potrebi za mrežnim saobraćajem jer on predstavlja usko grlo prilikom obrade.

Iako su SFC prvenstveno osmišljene za rad sa regularnim podacima, mogu se primeniti i na oblak tačaka ako se nad oblakom tačaka postavi regularan grid kod kojeg svaka ćelija grida sadrži najviše

jednu tačku. Radi jednostavnijeg opisa, a bez gubitka opštosti, u narednim primerima će oblak tačaka biti posmatran kao dvodimenzionalna struktura. Najjednostavniji način za linearizaciju oblaka tačaka je skladištenje tačaka red po red. Međutim u tom slučaju tačke koje pripadaju istoj koloni, iako prostorno bliske, su veoma udaljene nakon transformacije. Do danas je osmišljen veliki broj SFC koje se razlikuju prema stepenu očuvanja prostorne bliskosti, kao i prema kompleksnosti izračunavanja [11].

3.3.1. Generisanje SFC

Dve najčešće korišćene SFC su Mortonova (koja se drugačije naziva i Z-kriva) [53] i Hilbertova [41] (Slika 3.1). Mortonova kriva jednostavno izračunava preplitanjem bita pojedinačnih koordinata. Da bi se odredio Morton kod za neku tačku, neophodno je koordinate skalirati do celobrojne vrednosti, zatim odrediti binarnu predstavu X i Y koordinate i formirati Mortonov kod preplitanjem bita. Odnosno, za (X, Y) = ($x_i\ x_{i-1} \dots x_0$, $y_i\ y_{i-1} \dots y_0$) odgovarajući Mortonov kod je $x_i y_i x_{i-1} y_{i-1} \dots x_0 y_0$. Isti pristup se može primeniti i na višeimenzionalne podatke.



Slika 3.1: Mortonova (levo) i Hilbertova (desno) kriva

Hilbertova kriva transformiše višedimenzionalne podatke na jednu dimenziju primenom rekurzivne procedure. Generisanje u 2D započinje podelom kvadratnog domena na četiri zone. Zatim se svaka od četiri zone deli na isti način, sa tim da se kriva iz svake od zona rotira ili reflektuje tako da se krive iz susednih zona povežu u jednu. U 3D se kocka deli na osam zona gde se zatim svaka zona deli i krive se povezuju ukoliko dele stranicu.

Iz navedenog se može zaključiti da je Mortonova kriva jednostavnija za generisanje, u odnosu na Hilbertovu, ali po cenu lošijeg očuvanja prostorne bliskosti. Na slici 3.1 se može videti da kod Mortonove krive postoje „skokovi“ koji pokazuju da tačke sa susednim kodovima nisu uvek i prostorno bliske. Karakteristika Hilbertove krive je da je rastojanje između tačaka uniformno, te

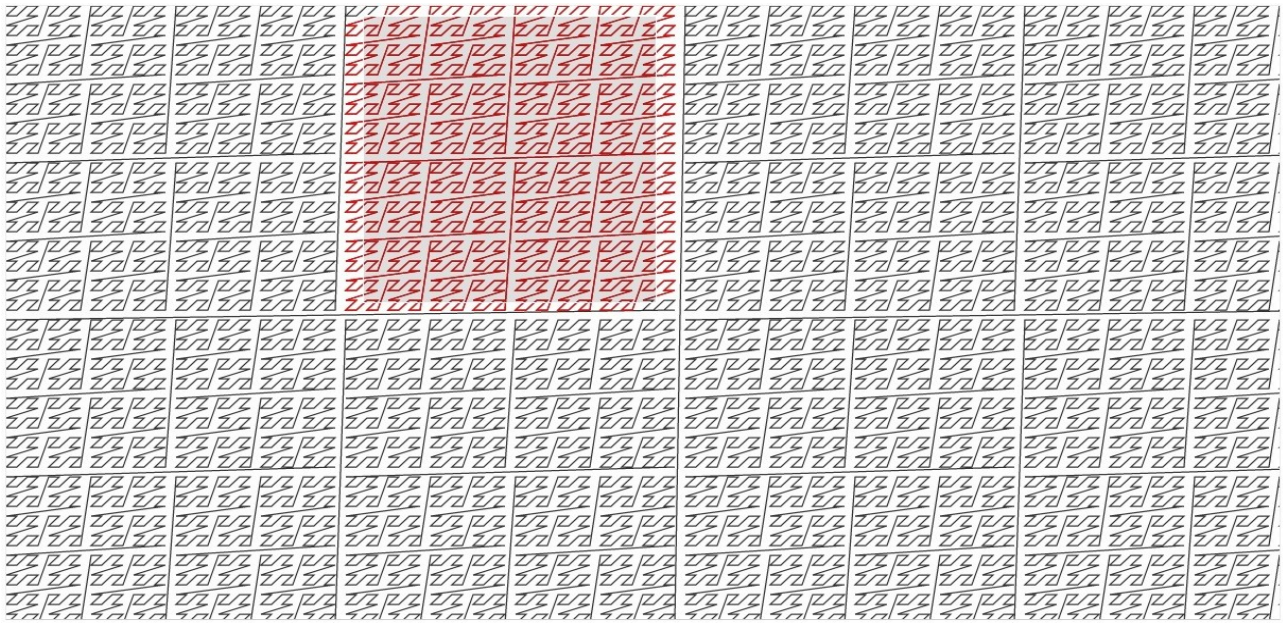
prema tome, ne sadrži skokove. U [30] je pokazano da Hilbertova kriva obezbeđuje veću efikasnost prostornih upita.

3.3.2. Prostorni upiti nad SFC

Pošto primenom SFC nije moguće u potpunosti očuvati prostornu bliskost potrebno je o tome voditi računa prilikom definisanja prostornih upita. Najjednostavniji primer prostornog upita je prostorno filtriranje, odnosno pronalaženje svih tačaka koje se nalaze u zadatom opsegu.

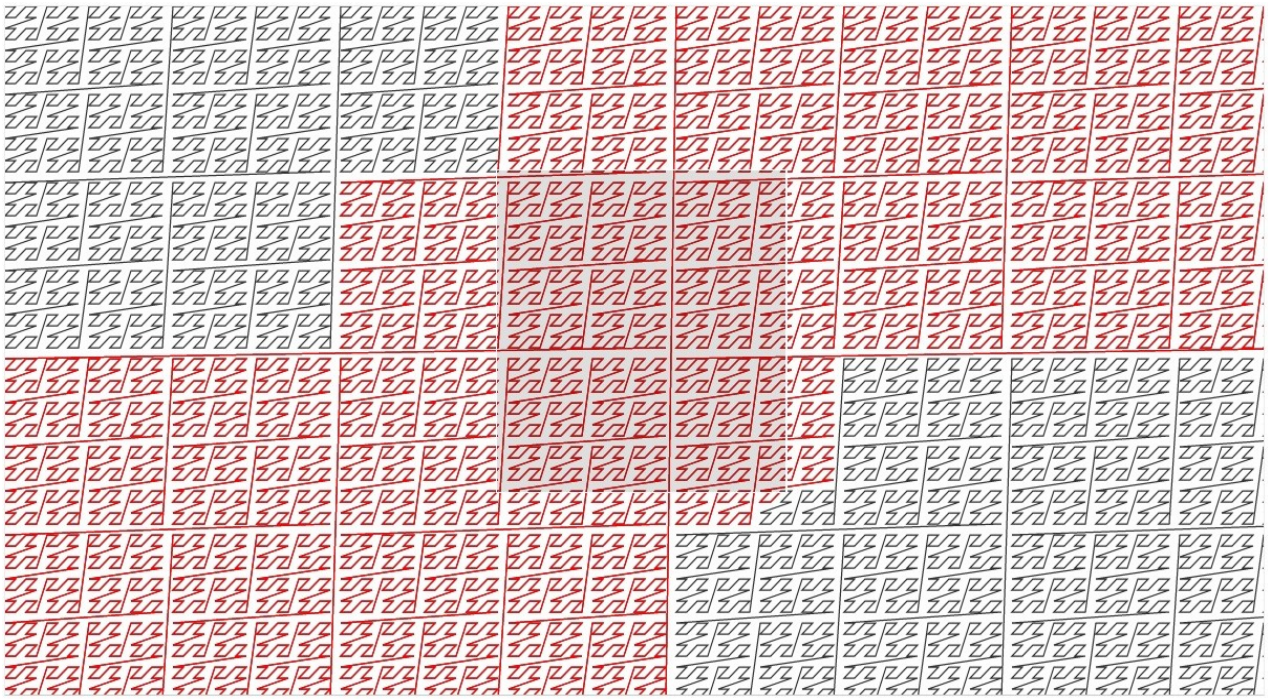
3.3.2.1. Prostorno filtriranje

Prostorno filtriranje je operacija kojom se pronalaze sve tačke u zadatom prostornom opsegu. U 2D koordinatnom sistemu prostorno filtriranje pronalazi sve tačke koje zadovoljavaju uslove $x_{min} \leq x \leq x_{max}$ i $y_{min} \leq y \leq y_{max}$. S obzirom na veliki broj tačaka u oblaku, sekvencijalno prolaženje i poređenje tačaka bi bilo dugotrajno. Uloga SFC se ogleda u tome da se smanji prostor pretrage, odnosno smanji broj tačaka koje su kandidati za rešenje upita (Slika 3.2). Trivijalan pristup bi bio da se na osnovu x_{min}, y_{min} odredi z_{min} (najmanja vrednost koda), a na osnovu x_{max}, y_{max} odredi z_{max} i zatim izvrši poređenje $z_{min} \leq z \leq z_{max}$ kako bi se dobio skup kandidata za rešenje. Zbog prirode SFC takav upit može da da za rezultat veoma veliki broj pogrešnih rešenja (Slika 3.3). Zbog toga je potrebno definisati pristup kojim se što precizije određuju kandidati za konačno rešenje prostornog upita.



Slika 3.2 Prostorno filtriranje primenom Z-krive (generisano sa http://bl.ocks.org/jaredwinick/5073432)

Za zadati prostorni opseg se generiše minimalni skup linearnih intervala Morton kodova koji sadrži sve tačke. Cilj je izbacivanje svih linearnih intervala koji se nalaze izvan zadatog prostornog opsega, radi izbegavanja nepotrebnih upita. Tačke se mogu skladištiti i indeksirati prema njihovim Morton kodovima u bilo kojem skladištu koje omogućuje efikasnu pretragu nad linernim intervalima. To mogu biti relacione baze podataka ili ključ-vrednost Big Data skladišta, kao što su Apache Hbase ili Apache Accumulo.



Slika 3.3 Prostorno filtriranje primenom Z-krive niz opsega (generisano sa http://bl.ocks.org/jaredwinick/5073432)

Predloženi algoritam koristi podeli i osvoji pristup. Ulazni podatak predstavlja niz Morton kodova tačaka, kao i Morton kodovi donje leve i gornje desne tačke prostornog opsega [20].

Osnovna ideja ovog algoritma je provera da li opseg pretrage sadrži jedan kontinualni segment krive. Ako je to tačno, algoritam se zaustavlja i vraća taj segment, u suprotnom opseg, pretrage se deli duž hiper ravni, i kreiraju se dva manja opsega. Zatim se algoritam primenjuje nad njima. Nakon pretrage nad obe strane, vrši se provera da li se segmenti mogu kombinovati u jedan kontinualni segment. U najgorem slučaju, rezultat će sadržati segmente koji obuhvataju po jednu tačku.

Algoritam se može opisati kao rekurzivna funkcija koja obuhvata sledeće korake:

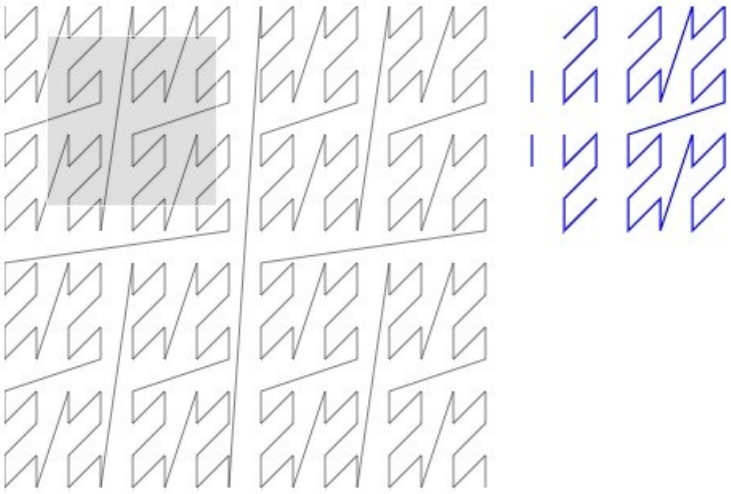
1 Proveri da li opseg pretrage predstavlja tačku. Ako da, vrati tu tačku kao rezultujući segment.

2 Proveri da li opseg pretrage sadrži jedan kontinualni segment. Sledeća činjenica se koristi a implementaciju te provere. Ako niz bita Morton koda donje leve tačke ima isti prefiks (može sadržati 0 bita) kao niz bita Morton koda gornje desne tačke i prvi ima sufiks 00000...00000, a drugi ima sufiks 11111...11111, vrati kompletan segment.

3 Pre svega se odredi indeks prve pozicije gde su niz bita donje leve tačke i niz bita gornje desne tačke različiti, a sufiks ne zadovoljava gornji šablon. Zatim se definiše hiper-ravan za tu poziciju indeksa i odrede se dva nova opsega (levo i desno od hiper-ravni).

4 Zatim se korak 1. ponavlja rekurzivno za oba nova opsega.

5 U poslednjem koraku se opsezi iz oba rekurzivna poziva spajaju u jednu listu. Dodatno se, vrši provera da li se poslednji segment prvog rekurzivnog poziva može spojiti sa prvim segmentom desnog poziva. U tom slučaju se ta dva segmenta spajaju u jedan. Slika 3.4 prikazuje rezultat izvršavanja algoritma.



Slika 3.4 Redukcija opsega Z-krive (generisano sa http://bl.ocks.org/daniarleagk/278709dedf09451b794ff72c4c05cda1)

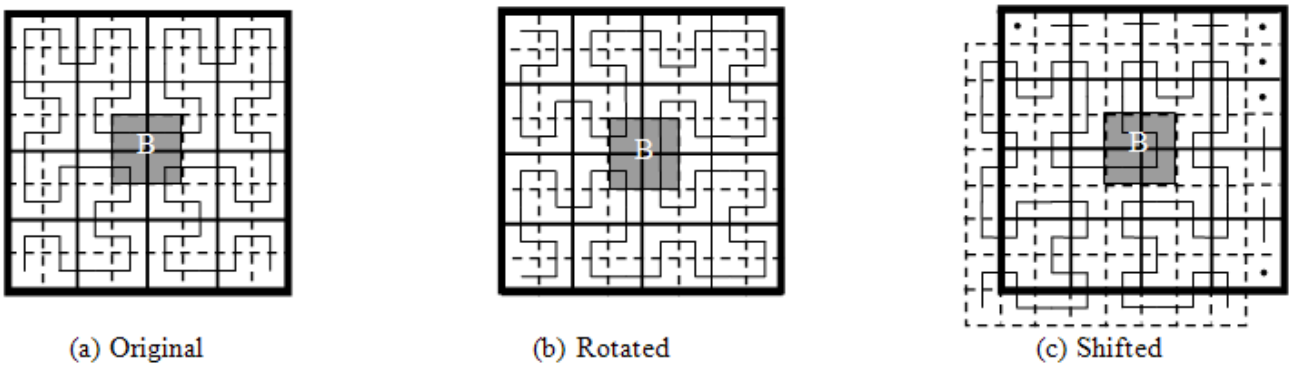
3.3.3. Unapređenje performansi prostornih upita primenom višestrukih SFC

Hilbertova kriva se u velikoj meri koristi za prostorno indeksiranje gde su podaci uskladišteni u linernom nizu memorijskih lokacija, koje mogu biti u RAM memoriji ili disku jednog ili više računara. Takođe se koristi i kao tehnika za indeksiranje višedimenzionalnih podataka, kao i prostorno-vremenskih podatka. Takođe, postoje i istraživanja za primenu Hilbertove krive za paralelno prostorno indeksiranje. Veliki broj istraživanja je pokazao da space filling krive pružaju

bolje rezultate za indeksiranje velikog broja dimenzija u poređenju sa R-stablima. Prethodna istraživanja su pokazala da, u poređenju sa ostalim SFC, Hilbertova kriva obezbeđuje najbolje očuvanje prostorne bliskosti pri mapiranju višedimenzionalnog prostora na jednodimenzionalni. Međutim, ni jedna SFC ne može da apsolutno zadrži prostornu bliskost te zbog toga postoje slučajevi gde se, inače prostorno bliske tačke, mapiraju na udaljene lokacije unutar SFC. Pojedina istraživanja su pokušala da redukuju prostor pretrage za prostorno filtriranje, kao i određivanje k najbližih suseda (engleski k nearest neighbour, kNN), korišćenjem parcijalnih Hilbertovih krivih. Međutim, to nije smanjilo ukupan broj opsega koji se pretražuju, a ni u potpunosti izbacilo sve rezultate izvan opsega pretrage. Drugi pristup za ubrzanja pretrage je korišćenje približnog upita najbližih suseda (engleski nearest neighbour, NN) koji je zasnovan na višestrukim transliranim Hilbertovim krivima [19].

U [19] je prikazano kako se primenom višestrukih SFC može ubrzati pretraga uz povećanje preciznosti rezultata. Osnovna ideja je da se za ulazni skup podataka na osnovu originalne SFC, generiše još nekoliko SFC (Hilbert) koje se dobijaju rotacijom i/ili translacijom originalne SFC. Prostorna pretraga u okviru SFC (Hilbert) se sastoji od tri koraka, mapiranja, filtriranja i poboljšanja. U prvom koraku se prostor pretrage mapira na niz jednodimenzionalnih opsega prateći smer prostiranja krive. Skup međusobno kontinualnih ćelija krive predstavlja jedan linearni opseg pretrage. U drugom koraku se prethodno određeni niz opsega primenjuje za određivanje svih tačaka koje pripadaju prostornom opsegu pretrage. S obzirom da je se pretraga vrši nad jednodimenzionalnim podacima, koristi se binarno stablo [14]. Treći korak je neophodan ukoliko se tačke skladište po blokovima umesto pojedinačno, a sami blokovi su indeksirani primenom SFC. Tada je neophodno za svaku tačku iz bloka, koji pripada skupu rešenja, proveriti da li se nalazi u opsegu pretrage.

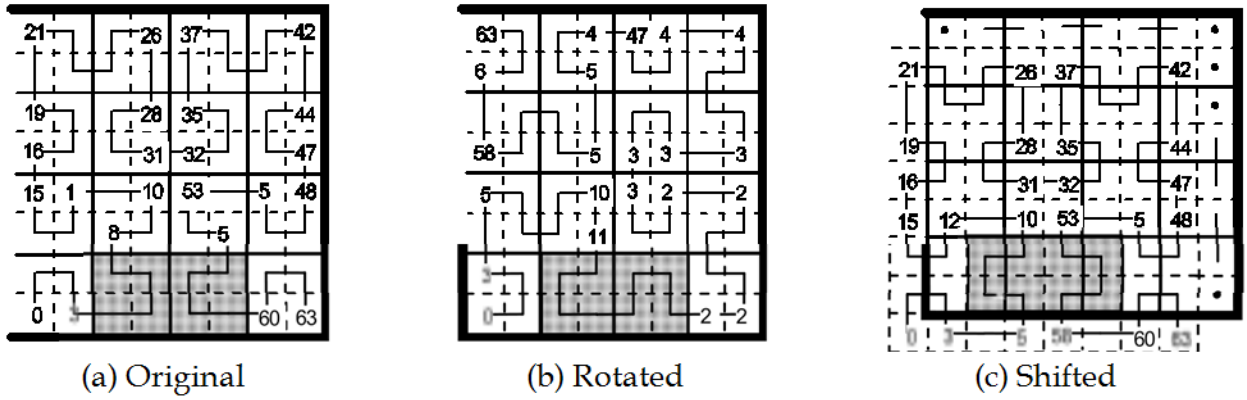
Zbog već opisanih osobina SFC, odnosno gubitka prostorne bliskosti, u opštem slučaju broj lineranih opsega pretrage može biti veoma velik i samim tim uticati na performansu upita. Međutim, primenom rotacije, translacije, ili kombinacijom nekoliko transformacija, skup lineranih opsega može značajno redukovati.



Slika 3.5 Transformacije SFC (preuzeto iz Dai, Jing. (2011). Efficient Range Query Using Multiple Hilbert Curves. 10.5772/22413.)

Hilbertove krive sa različitim orijentacijama rezultuju različitim brojem linearnih opsega za isti prostorni opseg pretrage. Hilbertova kriva je rekurzivna, pa se prema tome u 2D ravni kriva i-tog reda se dobija tako što se svaki kvadrant zameni krivom i-1 reda, od kojih dve rotirane 90^o u smeru kazaljke na satu, a dve obrnuto.

Na slici 3.6 se može videti da se rotacijom Hilbertove krive za 90^o u smeru kazaljke na satu prostor pretrage redukuje sa 2 opsega na 1.



Slika 3.6 Redukcija prostora pretrage (preuzeto iz Dai, Jing. (2011). Efficient Range Query Using Multiple Hilbert Curves. 10.5772/22413.)

U nekim slučajevima se primenom različitih rotacija ne može redukovati broj opsega. Na primer, ako se prostor pretrage nalazi u centru krive, broj opsega će ostajati isti bez obzira na primenjenu rotaciju. Međutim, ako se kriva translira dijagonalno, broj opsega se svodi na 1.

Iz navedenog se vidi uticaj rotacija i translacije na broj opsega u određenim slučajevima, iz čega se može izvesti zaključak da se primenom nekoliko krivih (originalna + modifikovane) može redukovati broj opsega pretrage. Kao što je već navedeno, primena samo rotacija ne može da reši

problem oko centralnog regiona, dok sam translacija takođe ne daje optimalan rezultat u opštem slučaju. Zbog toga, najbolje rešenje je kombinacija transformacija.

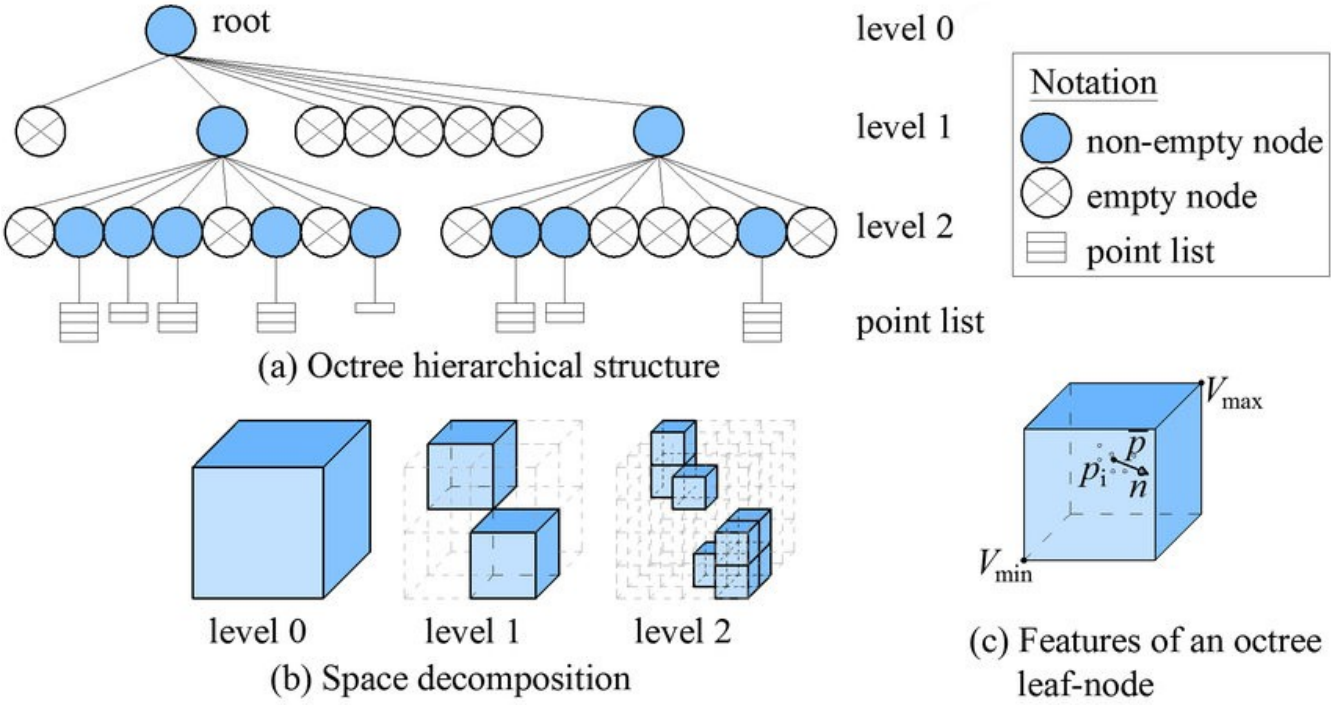
Istraživanje [19] je pokazalo da se dodavanjem još jednog indeksa (rotirane ili translirane krive) značajno uvećava performansa upita. Zatim, dodavanjem još jednog indeksa suprotnog tipa od prethodnog se još dramatičnije unapređuje brzina upita. Međutim, primećeno je da se daljim dodavanjem novih indeksa ne postižu značajna unapređenja.

Autori u [26] predstavljaju primenu više Hibertovih krivih dobijenih primenom slučajno generisane translacije radi povećanja tačnosti pretrage najbližih suseda. Utvrdili su da se na taj način pojava grešaka u velikoj meri redukuje.

S ozirom, na prirodnu redudantnost Big Data skladišta i paralelnu obradu, korićenje više od jednog indeksa ne predstavlja značajano opterećenje. Kreiranje originalne krive, translacija i rotacija su operacije koje se mogu izvršavati paralelno nad distribuiranim skupom podataka, a jedan od osnovnih postulata ne-relacionih skladišta podataka je da podaci nisu normalizovani i da se njihova struktura prilagođava potrebama upita.

3.4. Korišćenje SFC za predstavljanje kvaternalnog i oktalnog stabla

Struktura podataka tipa stabla je široko korišćena kao metod za indeksiranje podataka. Najjednostavniji oblik stabla su binarna stabla [14], koja se koriste za indeksiranje jednodimenzionalne podatke. Kod binarnih stabala se pojam pretrage poredi sa čvorovima stabla i u zavisnosti od rezultata poređenja pretraga se nastavlja na nižem nivou. Pretraga se izvršava kad se pronađe traženi element ili se dođe do čvora koji predstavlja list.



Slika 3.7 Generisanje oktalnog stabla (preuzeto sa https://www.researchgate.net/publication/274645446_Octree-based_region_growing_for_point_cloud_segmentation/figures?lo=1)

Kvaternalno stablo predstavlja generalizaciju binarnog stabla u dve, a oktalno u tri dimenzije. S obzirom da je oblak tačaka, u osnovi, trodimenzionalna struktura oktalno stablo se, logično, može koristi kao metod za indeksiranje. Primer indeksiranja oblaka tačaka primenom oktalnog stabla je prikazana u [28] . Segmentacija oblaka tačaka zasnovana na oktalnom je prikazana u [73].

Sličnost između SFC i strukture stabla se zasniva na tome što obe strukture služe za indeksiranje podataka. Obe strukture imaju prednosti jedna u odnosu na drugu. Oktalna stabla omogućuju brzu pretragu u računarskoj memoriji, dok su SFC fleksibilnije i pogodnije za upotrebu gde je neophodno linearno skladištenje podataka. Pod određenim uslovima je moguće transformisati podatke iz jedne strukture u drugu. Na primer, kao što je prikazano u [77], Z-kriva se može koristiti za efikasno generisaje kvaternalnih i oktalnih stabala. U osnovi algoritma je sortiranje podataka prema Morton kodu, nakon čega se može generisati željeni tip stabla, bilo da je reč o binarnom stablu zasnovanom na Morton kodu ili višedimenzionalnom stablu zasnovanom na pokazivačima.

3.5. Zaključna razmatranja o prostornom indeksiranju

U okviru ovog poglavlja su razmatrane tehnike za indeksiranje, geoprostornih podataka, prvenstveno oblaka tačaka. S obzirom na njihov značaj, prikazana je primena krivih za popunjavanje prostora u okviru podataka uskladištenih u okviru relacionih baza podataka i

razmatrana je njihova primena u okviru Big Data paradigme. Detaljno su razmatrane dve najčešće korišćene krive, Z-kriva i Hilbertova. Iako Hilbertova kriva bolje reflektuje prostornu bliskost tačaka, značajno je kompleksnija za implementaciju i manje intuitivna od Z-krive, koja se jednostavno može transformisati u strukturu oktalnog stabla veoma često korišćenu u različitim algoritmima za rad sa oblakom tačaka. Prema autorovim saznanjima ne postoje istraživanja koja se bave korišćenjem višestrukih krivih za popunjavanje prostora kao mehanizma za unapređenje performansi obrade oblaka tačaka u Big Data okruženju.

4. Model sistema za upravljanje velikim serijama geoprostornih podataka tipa oblak tačaka

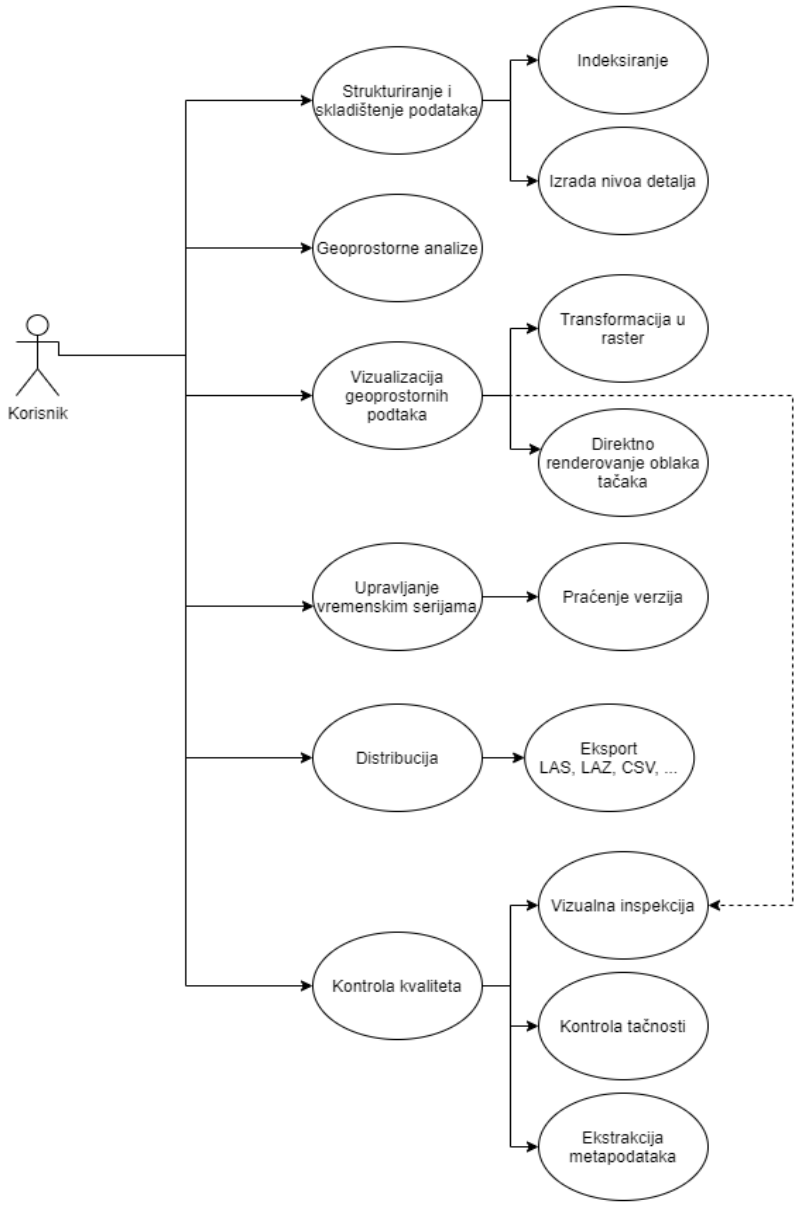
U poglavljima 2. i 3. su detaljno objašnjeni koncepti na kojima se zasniva model za upravljanje velikim serijama geoprostornih podataka. Poseban značaj za distribuirano upravljanje podacima imaju krive za popunjavanje prostora na kojima se zasniva prostorno indeksiranje i distribuirano skladištenje podataka. Model razmatran u ovom poglavlju ima za cilj obuhvat svih relevantnih aspekata upravljanja geoprostornim podacima u Big Data okruženju, gde je oblak tačaka izabran kao reprezentativan tip podataka zbog količine, kompleksnosti i raznovrsnosti. Prvi deo poglavlja daje prikaz elemenata arhitekture modela sa detaljnim opisom pojedinalnih modula. U drugom delu poglavlja su dati logički i fizički model podataka, gde prvi opisuje strukturu podataka tipa oblak tačaka, a drugi strukturu zapisa podataka u okviru Hbase skladišta. Zatim je dat prikaz primene Spark SQL korisnčki definisanih tipova i funkcija kojim su elementi logičkog modela preslikani na odgovarajuće strukture u okviru Spark platforme. U poslednjem segmentu poglavlja se razmatraju operacije prostornog filtriranja i kNN nad oblakom tačaka u distribuiranom okruženju.

4.1. Arhitektura modela

Digitalizacija svih aspekata ljudskog društva vodi ka izdvajanju značajnih resursa za upravljanje podacima. Poslednjih godina se pojavljuje sve veći broj kompanija koje pružaju servise za specijalizovano upravljanje određenim resursima. Na primer, servisi za skladištenje podataka u cloud-u, omogućuje čuvanje i deljenje fajlova, oslobađajući korisnike obaveze nabavke sve više skladišnog prostora ili slanja fajlova preko mreže.

Upravljanje podacima oblaka tačaka, takođe, iziskuje značajne ljudske i hardverske, a samim tim i finansijske resurse. Zbog toga, je sve više korisnika spremno da delegira upravljanje tom vrstom podataka specijalizovanim servisima.

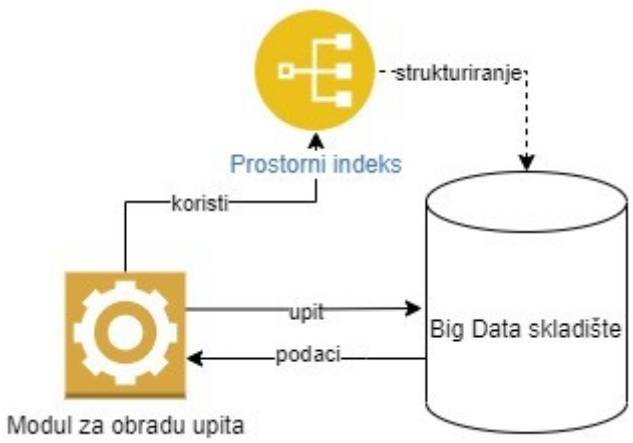
Arhitektura modela za upravljanje oblakom tačaka, prikazana na slici 4.1, je dizajnirana imajući na umu osnovne funkcionalnosti koje bi takav sistem trebalo da pruži korisniku, a koje su navedene u poglavlju 3.2.



Slika 4.1 Aritekura modela za upravljanje velikim oblacima tačaka

4.1.1 Jezgro arhitekture

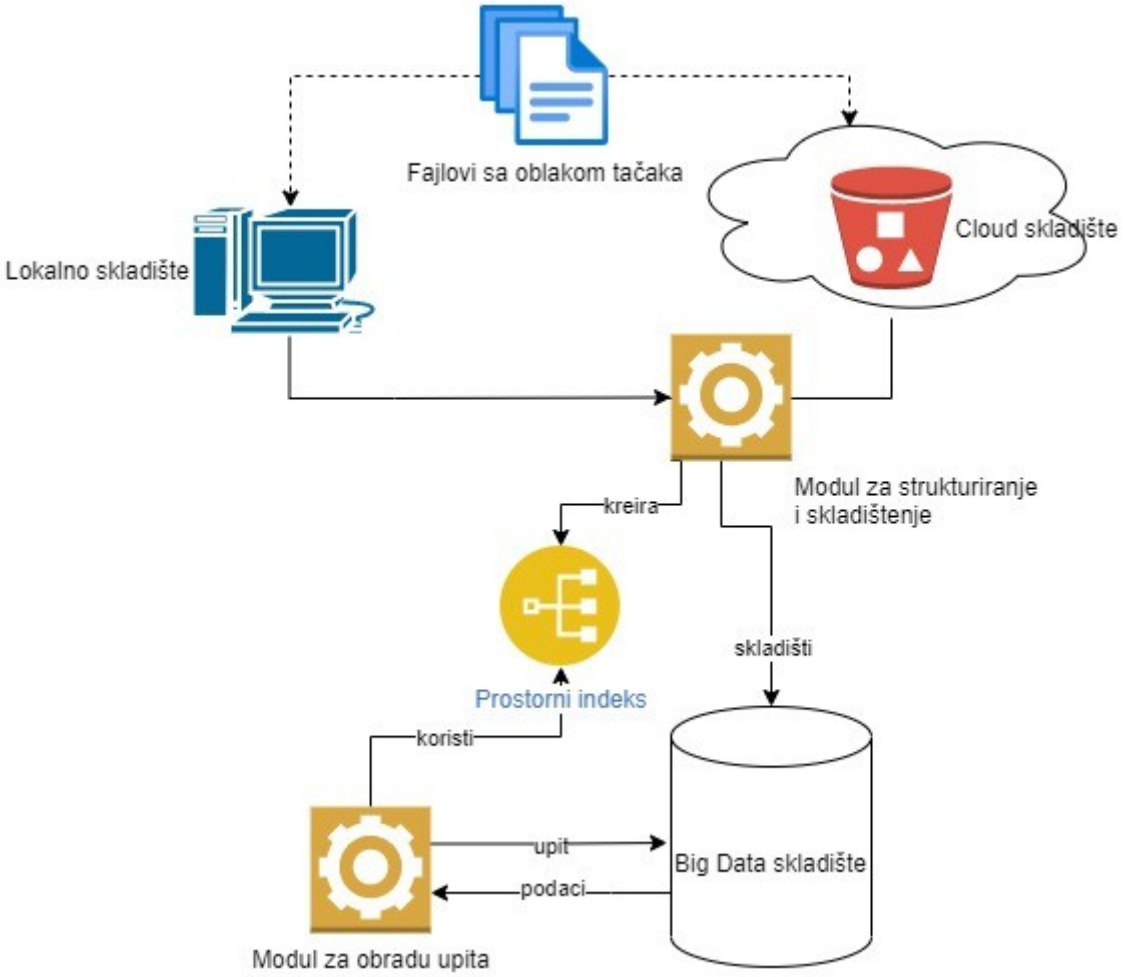
U središtu modela se nalazi modul obradu upita implementiran u Apache Spark-u. Zadatak ovog modula je da prihvata upite od drugih modula i distribuirano ih izvršava nad Big Data skladištem. Najčešći tipovi upita se odnose na selekciju prema zadatom opsegu. Radi postizanja optimalne performanse modul za obradu upita koristi prostorni indeks zasnovan na krivama za popunjavanje prostora. U nastavku ovog poglavlja će biti detaljno opisani logički i fizički modeli podataka na kojima se zasniva jezgro arhitekture, kao i realizacija modula za obradu upita sa implementacijom prostornih upita. Na slici 4.2 je prikazan dijagram jezgra arhitekture.



Slika 4.2 Jezgro aritekure modela za upravljanje velikim oblacima tačaka

4.1.2. Strukturiranje i skladištenje podataka

Modul za strukturiranje i skladištenje (Slika 4.3) je odgovoran za takozvani *Extract-Transform-Load* (ETL) proces, odnosno preuzimanje podataka, transformaciju i skladištenje prema strukturi pogodnoj za distribuiranu obradu. Ovde ćemo navesti ukratko funkcionalnosti ovog modula, a detalji će biti prikazani u nastavku poglavlja. Modul za strukturiranje i skladištenje preuzima podatke iz distribuiranog fajl sistema, kao što je HDFS, ili nekog cloud skladišta, transformiše i u format pogodan za distribuiranu obradu i skladišti u Big Data skladištu. Modul *za strukturiranje i skladištenje* je implementiran tako da koristi pridružene metapodatke iz ulaznih fajlova, ako su oni prisutni. Ukoliko to nije slučaj, modul izvršava distribuiranu ekstrakciju tih podataka, poput prostornog opsega oblaka tačaka. Proces transformacije obuhvata normalizaciju koordinata i izračunavanje Morton kodova. Na kraju, podaci se skladište prema fizičkom formatu koji je definisan u nastavku poglavlja.



Slika 4.3 Modul za strukturiranje i skladištenje podataka

4.1.3. Geoprostorne analize

Poput ostalih tipova geoprostornih podataka i oblak tačaka nosi veliku količinu informacija neophodnih za donošenje odluka. Radi ekstrakcije pomenutih informacija je razvijen veoma veliki broj algoritama za analizu oblaka tačaka. Na osnovu klasifikacije operacija nad oblakom tačaka prikazanih u poglavlju 3.2, kao i na osnovu operacija implementiranih u [3], mogu se izdvojiti nekoliko najznačajnijih kategorija algoritama:

- prostorno, vremensko i alfanumeričko filtriranje,
- izračunavanje statistike,
- spajanje oblaka tačaka iz različitih izvora,
- triangulacija,
- rasterizacija,
- filtriranje grešaka,
- ekstrakcija objekata,

- klasifikacija,
- izračunavanje površina i zapremina, i
- uklapanje geometrijskih primitiva.

S obzirom, količinu podataka i kompleksnost tipa podataka koji predstavlja oblaka tačaka njihova obrada postavlja značajne izazove za implementaciju efikasnih algoritama. Ukoliko je kompleksnost algoritma, na primer $O(n^2)$ povećanjem broja tačaka se vrlo brzo isrpljuju resursi računarskog sistema na kojem se vrši obrada. Zbog toga, mogućnost paralelizacije predstavlja jedan od osnovnih zahteva prilikom dizajniranja algoritma.

Dva osnovna načina za paralelnu obradu podataka predsavljaju distribuirani računarski sistem i grafička procesorska jedinica. U prvom slučaju je reč o sistemu sačinjenom od odvojenih računarskih jedinica sa sopstvenom memorijom i centralnim procesorskom jedinicom (obično sa više jezgara), dok u drugom slučaju imamo uređaj sa velikim brojem jezgara sposoban za masovno paralelno izvršavanje, ali sa ograničenim memorijskim resursima. Sa verzijom 3, Spark je pored dosadašnjeg modela zasnovanog na distribuiranom procesiranju dodao i podršku za izračunavanje na grafičkim procesorskim jedinicama i tako iskoristio prednosti oba modela izračunavanja [3].

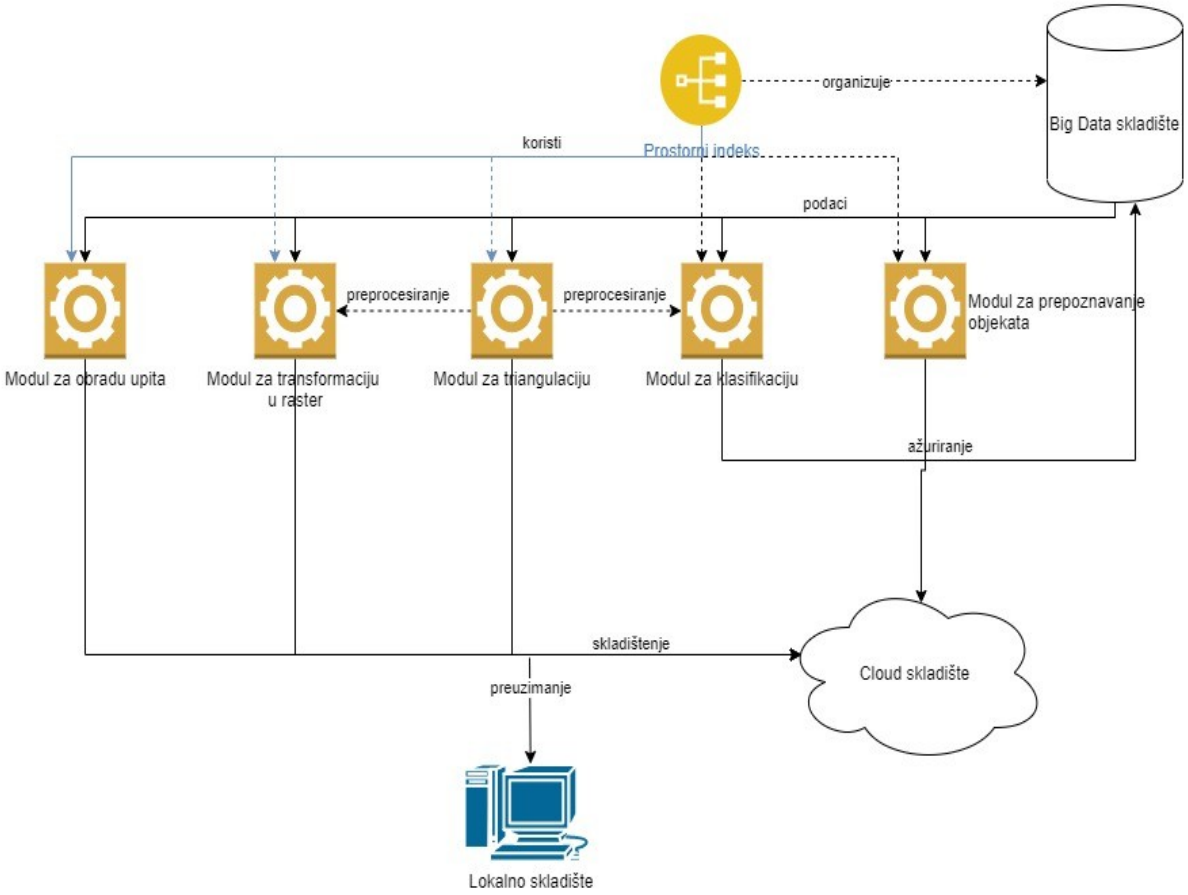
Kod većine navedenih algoritama je prostorna bliskost tačaka neophodna za efikasnu obradu. Prema tome organizacija podataka u okviru definisanog modela za upravljanje oblakom tačaka predstavlja dobru osnovu za efikasnu distribuiranu obradu podataka. U [33] je prikazana upotreba Morton kodova za efikasno (vremenska kompleksnost O(n)) generisanje Delaunejeve triangulacije. Indeksiranje tačaka primenom Morton kodova je omogućilo efikasnu paralelizaciju algoritma koji je zatim izvršavan na grafičkoj procesorsko jedinici. U okviru [48] je prikazano distribuirano poređenje oblaka tačaka iz različitih perioda radi detekcije promena. Autori su najpre generisali trodimenzionalnu mrežu voksela i odredili pripadnost voksela jednom ili više oblaka tačaka. Vokseli su su indeksirani Morton kodovima i distribuirani u okviru Apache Spark aplikacije radi detekcije promena. Upotreba Morton kodova za distribuiranu obradu oblaka tačaka je prikazana u [5]. Pored opšteg algoritma za distribuciju oblaka tačaka, prikazana je i konkretna implementacija za određivanje kNN.

U okviru modela za upravljanje oblakom tačaka je definisano nekoliko modula za geoprostorne analize oblaka tačaka (slika 4.4).

- Modul za obradu upita kao što je navedeno ranije u tekstu, omogućuje prostorno i alfanumeričko filtriranje podataka.
- Modul za triangulaciju predstavlja modul za distribuirano generisanje triangulacije. Rezultati obrade ovog modula se mogu koristiti u modulu za rasterizaciju, na primer za

interpolaciju digitalnih visinskih modela ili za generisanje osenčenog reljefa. Takođe, se rezultati iz ovog modula mogu koristiti i u modulu za klasifikaciju, na primer, za određivanje tačaka na terenu primenom progresivnog progušćavanja mreže trouglova [10].

- Modul za transformaciju u raster omogućuje transformaciju oblaka tačaka u rasterski oblik generisanjem sadržaja piksela na osnovu odabranih atributa oblaka tačaka, na primer, određivanje boje prema visini ili klasifikaciji.
- Modul za klasifikaciju predstavlja modul koji grupiše tačke prema određenim svojstvima.
- Modul za prepoznavanje objekata vrši analizu oblaka tačaka radi utvrđivanja detekcije postojanja zadatih oblika.



Slika 4.4 Modul za geoprostorne analize

4.1.4. Vizualizacija geoprostornih podataka

Vizualizacija geoprostornih podataka je jedna od najznačajnijih funkcionalnosti potrebnih korisnicima. Njena osnovna svrha je da prikaže podatke na korisniku razumljiv način. Kada je reč o oblaku tačaka vizualizacija predstavlja kompleksan zadatak s obzirom njegovu na prirodu. Ni jedan računarski sistem, bez obzira na njegove kapacitete, nije u stanju da u realnom vremenu renderuje

milijarde tačaka. Zbog toga je dugi niz godina preovlađujući način za vizualizaciju oblaka tačaka bila njegova transformacija u raster i prikaz generisanog rastera. Drugi problem kod vizualizacije oblaka tačaka je gustina renderovanih tačaka, gde često nije moguće razaznati prikazane objekta. Radi rešavanja tog problema su osmišljene tehnike kao što su generisanje nivoa detalja ili *splatting* [39]. Bez obzira na to i dalje se u praksi koriste obe navedene opcije.

Poslednjih godina je primetan trend sve veće migracije softverskih rešenja na *cloud*. Taj trend nije zaobišao ni sisteme za vizualizaciju oblaka tačaka. Dugi niz godina je direktna vizualzacija oblaka tačaka bila rezervisana samo za desktop računarske sisteme. Tek pre desetak godina, specifikacijom WebGL tehnologije [74], web browser je postao platforma za renderovanje kompleksnih više dimenzionalnih scena. Drugi preduslov za nastanak cloud-based sistema za renderovanje oblaka tačaka je bilo povećanje mrežnog protoka. Današnji mrežni sistemi omogućuju više nego dovoljne brzine protoka da podrže transport velikih količina podataka neophodnih za vizualizaciju oblaka tačaka. Dalji razvoj tehnologije, kao što je na primer 5G [2], će voditi daljem unapređenju pomenutih rešenja.

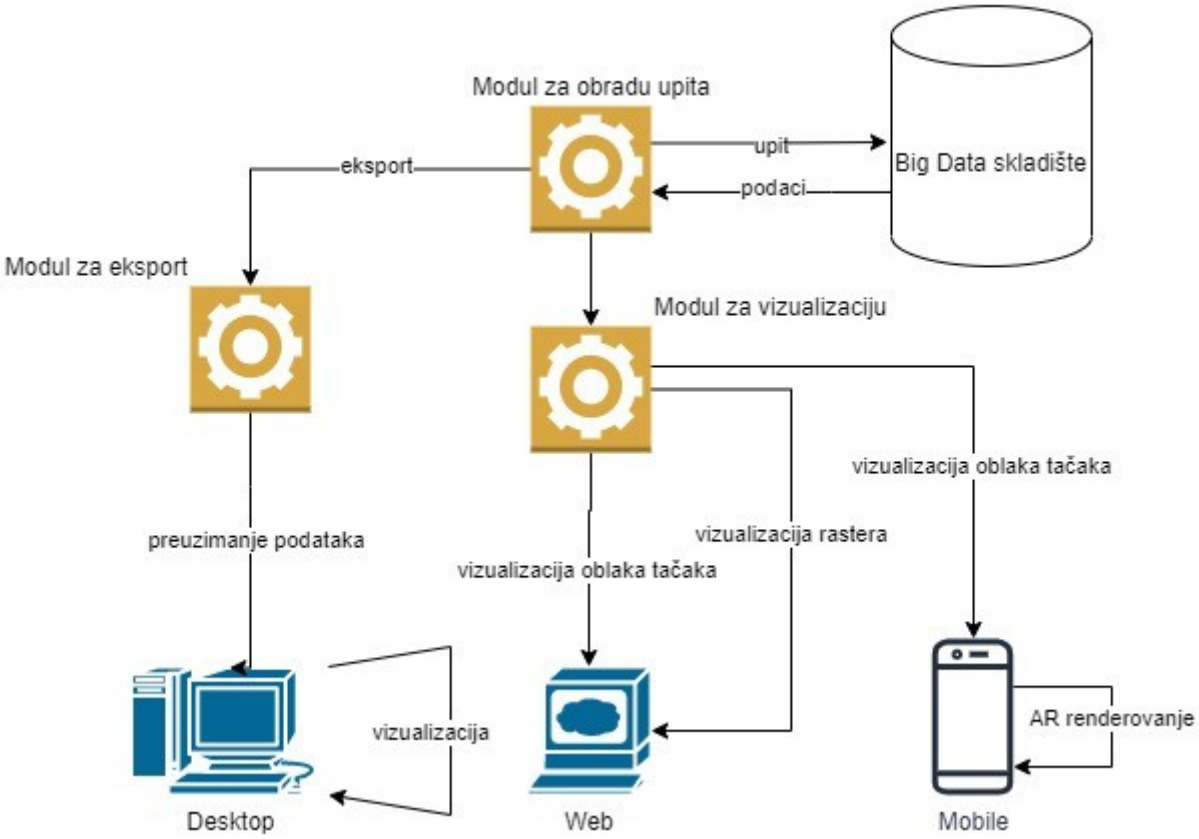
Iako trenutno ne postoje rešenja (bar ne previše upotrebljiva) za vizualizaciju oblaka tačaka na mobilnim okruženjima, verovatno je da će se u budućnosti i takva rešenja sve više koristiti. Takođe, na razvoj rešenja za vizualizaciju oblaka tačaka na mobilnim uređajima će uticati i razvoj sistema za proširivu realnost na kojima će paralelnim pregledom oblaka tačaka i podataka sa kamere moći da se donose odluke na terenu, poput Trimble SiteVision [67].

Arhitektura modula za vizualizaciju oblaka tačaka (slika 4.5) je definisana na sledeći način.

- Modul za obradu upita omogućuje obradu upita koje drugi moduli mogu imati ka Big Data skladištu.
- Modul za vizualizaciju omogućuje distribuciju oblaka tačaka u obliku pogodnom za renderovanje na klijentskim aplikacijama poput potree [45] ili Cesium [18]. Postojeći sistemi, kao i njima pridruženi alati, omogućuju generisanje hijerarhijske strukture pripremljene za distribuciju do klijentskih aplikacija putem interneta, koje su odgovorne za samo renderovanje. Razlaganjem oblaka tačaka u takvu strukturu omogućuje da se sa renderovanjem započne vrlo brzo, tj. inicijalni skup podataka sadrži relativno mali broj ključnih tačaka. Prikaz se zatim progušćuje pristizanjem podataka koji sadrže veći broj tačaka. Različite aplikacije koriste različite šeme za generisanje hijerarhijskih podataka, primeri su Entwine Point Tiles [29] i Cesium 3D Tiles [54]. Pored distribucije samog oblaka tačaka korisnicima je često značajno da mogu da pregledaju oblak tačaka u rasterkoj formi,

na primer, u obliku digitalnog modela visina, osenčenog reljefa, ili mape dubina. Za pomenute potrebe, modul za vizualizaciju, može koristiti servise koje pružaju moduli za triangulaciju i rasterizaciju.

- S obzirom na to da su se desktop alati za vizualizaciju oblaka tačaka, kao i uopšte desktop alati za renderovanje, mnogo ranije pojavili nego Web bazirani alati, mnogi korisnici preferiraju rad u njima. Druga prednost takvih alata je što na njihove performanse ne utiče brzina prenosa podataka putem mreže i samim tim omogućuju bolje korisničko iskustvo. Modul za eksport podataka omogućuje korisnicima da preuzmu podatke u celini ili na odabranim lokacijama i da vizualizuju podatke u lokalnom okruženju. U poglavlju o distribuciji podataka će detaljnije biti opisan sam proces eksportovanja.

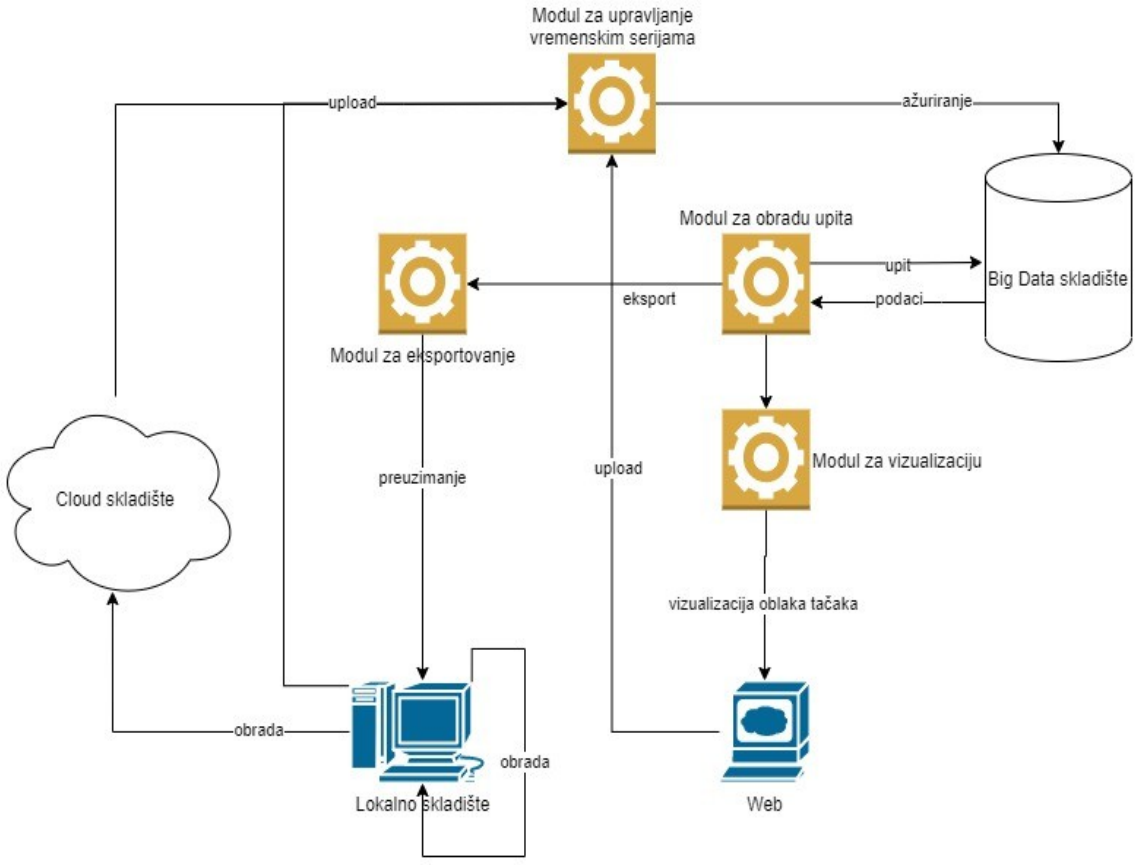


Slika 4.5 Modul za vizualizaciju podataka

4.1.5. Upravljanje vremenskim serijama

Upravljanje vremenskim serijama geoprostornih podataka je kompleksno, naročito u situacijama kad se podaci nalaze skladišteni po fajlovima. Najčešće svaka faza obrade generiše novi skup podataka na osnovu podataka iz prethodne faze. Praćenje takvog toka podataka je veoma

kompleksno i obično zahteva implementaciju kompletnih sofverskih alata. Model za upravljanje oblakom tačaka definiše način za uniformisanje pomenutog postupka tako što se svakoj tački pridružuje jedinstveni identifikator zasnovan na nekoliko parametara kao što su Morton kod i identifikator oblaka tačaka. Definisana su dva toka podataka, koji su prikazani na slici 4.6. Prvi od njih je cloud orijentisan, gde kroz web interfejs korisnik direkno manipuliše podacima i sve izmene se automatski čuvaju u okviru Big Data skladišta. Trenutno najveći nedostatak takvog rešenja je što još uvek ne postoje Web bazirani alati za obradu oblaka tačaka. Drugi pristup je da se korisnicima omogući da preuzmu deo podataka za oblast koju obrađuju, izvrše obardu u moćnim desktop alatima, poput aplikacija iz TerraSolid [66] familije, i zatim, uploaduju podatke u cloud skladište. Sistem za upravljanje podacima na osnovu jedinstvenih idenifikatora zatim svakoj tački pridružuje odgovarajuću izmenu. S obzirom da u okviru Big Data sistema skladišni prostor nije ograničavajući faktor, prethodne verzije tačaka nije potrebno brisati, nego samo dodati novo stanje tačke. Na takav način je moguće u svakom trenutku imati potpunu istoriju podataka. Samim tim korisnici podatka mogu u svakom trenutku imati uvid u sve procese obrade, na primer može se vizualizovati nad kojim delom oblaka tačaka je izvršena klasifikacija.



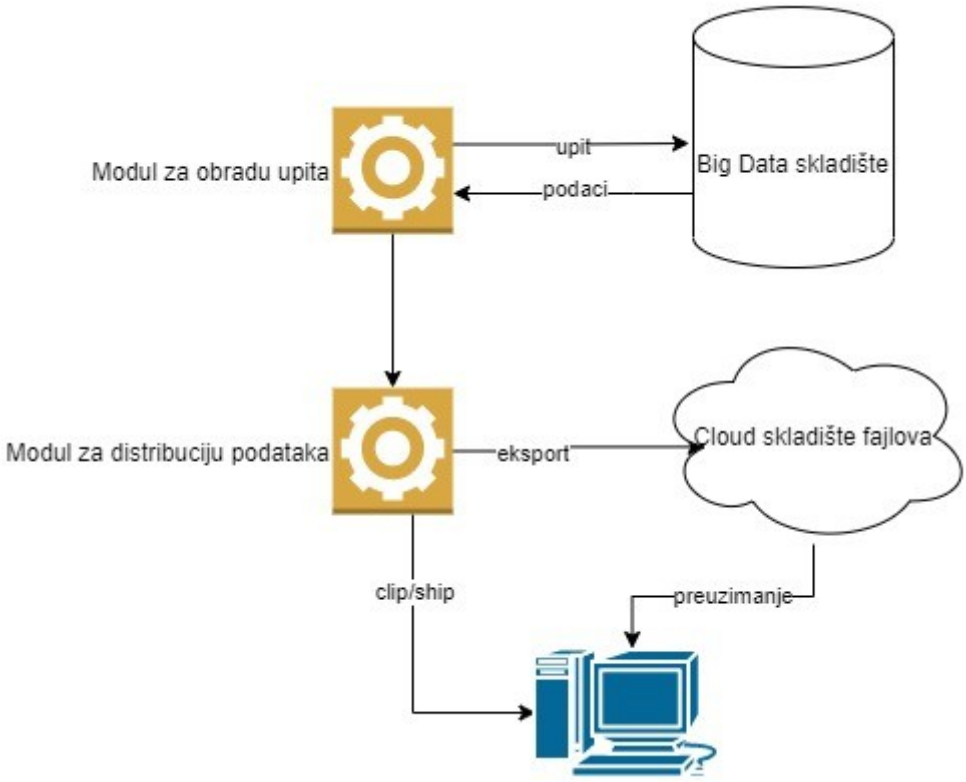
Slika 4.6 Modul za upravljanje vremenskim serijama

4.1.6. Distribucija podataka

Trend migracije softverskih platformi na cloud, ne zaobilazi ni aplikacije za skladištenje i distribuciju geoprostornih podataka, kao što su oblaci tačaka dobijeni Lidar snimanjima ili iz drugih izvora [1]. Takav pristup ubrzava tok podataka od prikupljanja do preuzimanja podataka od strane korisnika, s obzirom da se u potpunosti eliminiše transfer podataka putem fizičkih medijuma poput hard diskova. Takođe, korisnicima je omogućeno da odmah nakon akvizicije vrše kontrolu kvaliteta podataka i da odmah ukažu na moguće probleme. Na taj način se izbegava dugotrajni postupak slanja hard diskova, učitavanja podataka u desktop softverska rešenja, prijavljivanja grešaka, i zatim čekanja sledeće isporuke. Više reči o tome biće dato u sledećem poglavlju koje se bavi distribuiranom kontrolom kvaliteta oblaka tačaka.

Sam modul za distribuciju podataka, omogućuje korisicima dve osnovne funkcionalnosti.

- Preuzimanje podataka u celosti, odnosno takozvani *bulk* transfer. U tom slučaju korisnik eksportuje celokupne podatke oblaka tačaka. S obzirom da je u tom slučaju količina podataka veoma velika, preuzimanje podataka na lokalni računar nije moguće, te se podaci obično skladište u obliku fajlova u okviru nekog cloud skladišta, kao što je na primer Amazon Simple Storage Service (S3). Odatle se podaci mogu po potrebi preuzeti primenom Web konzole ili odgovarajućih desktop alata. Sam postupak pripreme podataka može biti izvršen paralelno, gde svaki Spark izvršilac generiše određeni skup fajlova ili određeni deo nekog većeg fajla.
- Drugi način za distribuciju podataka predstavlja selekciju podskupa podataka od strane korisnika, na primer putem zadavanja prostornog opsega, i preuzimanje podataka na lokalni računar. Drugim rečima u pitanju je *clip-zip-ship* proces. Modul za eksport podataka, prosleđuje zadati prostorni opseg modulu za obradu upita, koji zatim izdvaja podatke koji pripadaju tom opsegu i dostavlja ih nazad. Kako bi se redukovala količina prenetih podataka, oni se kompresuju, i prosleđuju korisniku. Jedan primer na koji se podaci mogu kompresovati predstavlja LasZIP [44].



Slika 4.7 Modul za distribuciju podataka

4.1.7. Kontrola kvaliteta

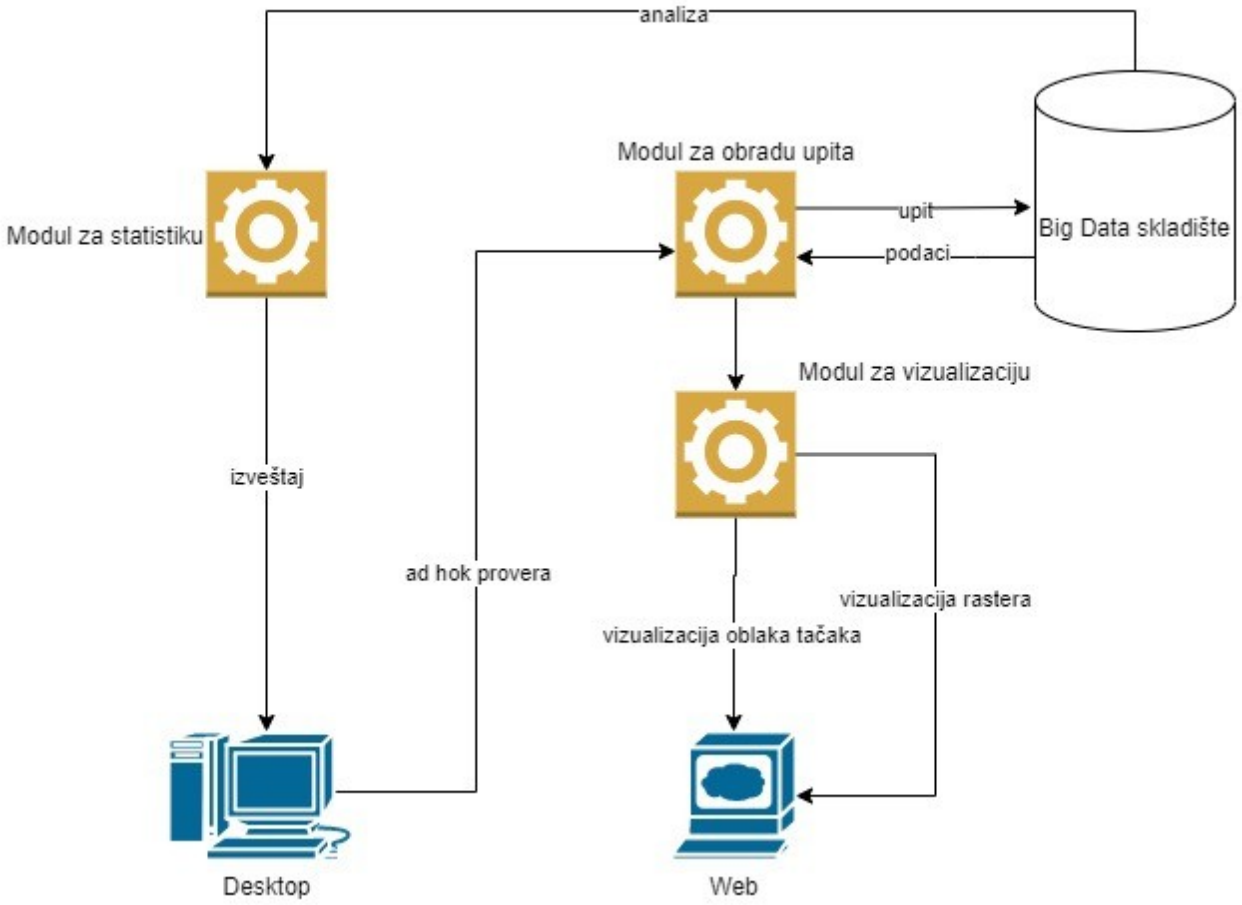
Kontrola kvaliteta geoprostornih podataka je jedan od najvažnijih aspekata njihovog celokupnog životnog veka. Kao i u drugim aspektima opisanim u ovom dokumentu, količina podataka predstavlja značajan izazov i za postupak kontrole kvaliteta. Kada je reč o oblaku tačaka, proces kontrole kvaliteta je opisan u [40]. Prema tom dokumentu osnovni podaci neophodni za kontrolu kvaliteta su:

- Gustina tačaka i rastojanje između tačaka.
- Detekcija dupliranih tačaka.
- Apsolutna tačnost.
  - Horizontalna.
  - Vertikalna.
- Relativna tačnost.
  - Horizontalna.
  - Vertikalna.

- Tačnost klasifikacije.

Model za upravljanje oblakom tačaka može da pruži podršku u kontroli tačnosti na višestruke načine (slika 4.8). Pre svega, moguće je eliminisati duplirane tačke na vrlo jednostavan način, poređenjem Morton kodova, prilikom samog importa. Distribuiranom obradom tačaka je moguće generisati detaljnu statistiku, koja uključuje podatke, kao što su prosečno rastojanje između tačaka i gustina tačaka, i generisati zone koje ukazuju na probleme u podacima. Provera relativne i apsolutne tačnosti je moguća postavljanjem ad hoc upita kroz modul za obradu upita, kao i automatski distribucijom većeg broja kontrolnih tačaka i poređenjem sa njima.

Pored navedenog, vizualna inspekcija je veoma značajna za kontrolu kvaliteta. Kao što je prikazano u prethodnim poglavljima, u okviru modela za upravljanje oblakom tačaka, su definisani moduli za vizualizaciju. Za kontrolu kvaliteta su značajne obe varijante za vizualizaciju, direktim prikazom oblaka tačaka i vizualizacijom rasterskih derivata. S obzirom da je osnovna stavka u kontroli kvaliteta ispitivanje kompletnosti podataka, potrebno je omogućiti generisanje rastera koji prikazuje gustinu podataka, čijim pregledom je vrlo lako uočiti problematične zone. Problemi u klasifikaciji se mogu lako uočiti prikazom osenčenog reljefa (primenom različitih pozicija i uglova osvetljenja). Prikaz poprečnih preseka nad oblakom tačaka je takođe od velikog značaja za vizualnu inspekciju. Klijentske aplikacije obično poseduju takve alate i mogu koristiti podatke iz modula za vizualizaciju radi vizualizacije poprečnih profila.

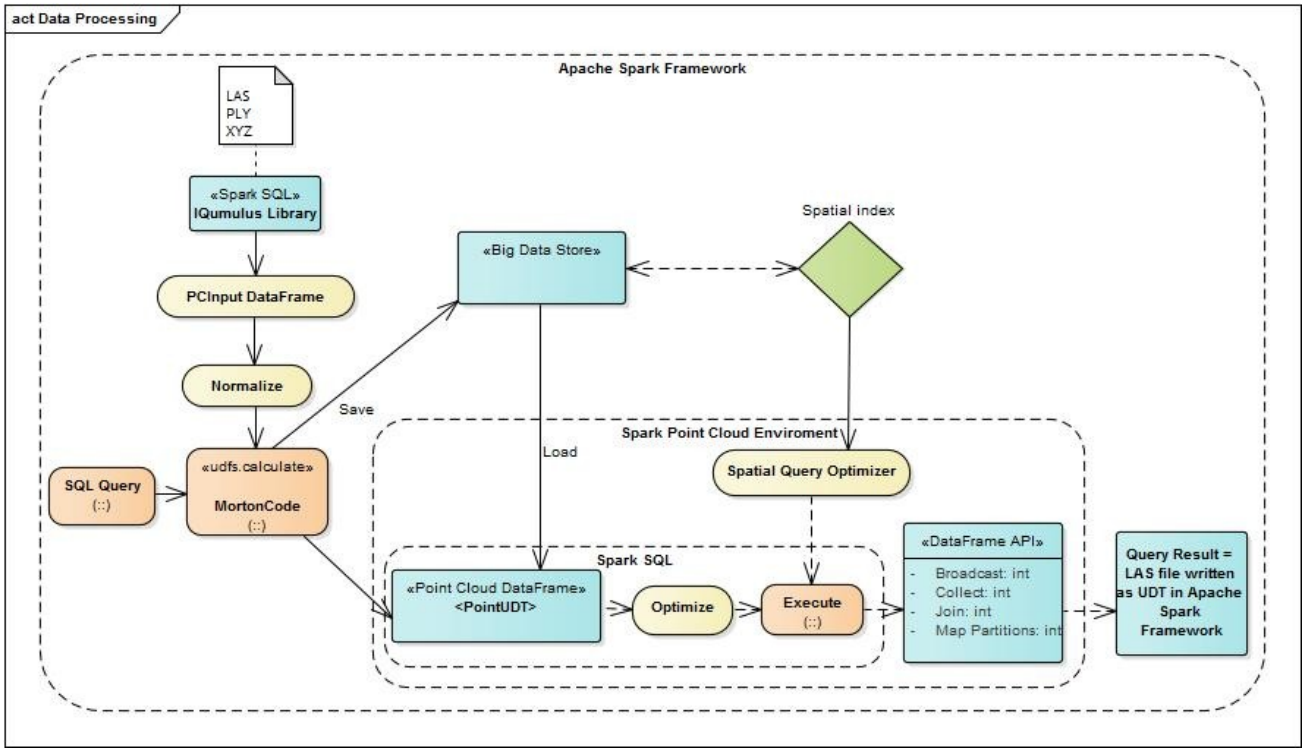


Slika 4.8 Modul za kontrolu kvaliteta podataka

4.2. Konceptualni model podataka

Kao što je već prikazano, uobičajen način skladištenja oblaka tačaka je u okviru fajlova. Zbog ograničenja fajl sistema podaci oblaka tačaka moraju biti podeljeni u veliki broj fajlova. Pri tome, fajlovi mogu biti skladišteni, na jednom računaru, više umreženih računara, ili od (relativno) nedavno u okviru cloud skladišta. Najčešći format za skladištenje oblaka tačaka predstavlja LAS binarni format, koji pored samih tačaka skladišti i pridružene metapodatke. Sledeći često korišćeni format predstavlja tekstualni format, gde svaki red teksta sadrži podatke o jednoj tački odvojene nekim specijalnim karakterom, na primer zarezom ili tabulatorom. U ovom slučaju, tačkama nisu pridruženi metapodaci. Pored dva navedena, koriste se i *Polygon File Format* (PLY), E57 i ***P**oint **C**loud **D**ata* (PCD), međutim opis njihove strukture izlazi iz okvira teze.

U prethodnom tekstu je definisan modul za strukturiranje i skladištenje podataka, čija će funkcionalnost ovde biti detaljnije objašnjena. Takođe će biti opisana detaljnija implementacija modula za obradu upita nad učitanim podacima podacima.

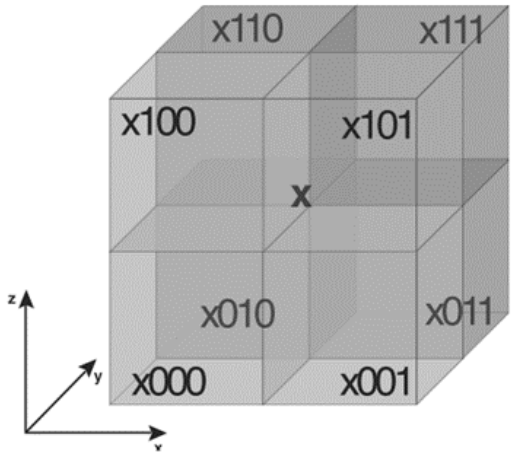


Slika 4.9 Postupak importovanja, izvršavanja upita i (opciono) eksportovanja podataka zasnovan na Spark platformi

Funkcionalnost importovanja i eksportovanja podataka je zasnovana na Spark SQL IQmulus softverskoj biblioteci koja omogućuje ulazno-izlalzne operacije za LAS, PLY i tekstualne fajlove (slika 4.9). U toku učitavanja podataka vrši se normalizacija koordinata kako bi se svele na celobrojne vrednosti, na osnovu kojih se kasnije izračunavaju Mortonovi kodovi. Za implementaciju prostornog indeksa oblaka tačaka je izabrana Z-kriva, iz sledećih razloga:

- jednostavno izračunavanje,
- jednostavno mapiranje na oktalno stablo,
- solidno očuvanje prostorne bliskosti podataka.

Morton kod predstavlja poziciju tačke ili skupa tačaka u okviru oktalnog stabla. Morton kod određenog segmenta oblaka tačaka se može izvesti rekurzivno počevši od korena koji obuhvata kompletan prostorni opseg oblaka tačaka. Sledeći nivo stabla se dobija tako što se korenski čvor podeli na osam segmenata kodiranih binarno od 000 do 111 (slika 4.10). Dublji nivoi se generišu na identičan način, odnosno daljom podelom postojećih čvorova.



Slika 4.10 Rekurzivno generisanje oktalnog stabla

Prema slici 4.9 tačke se učitavaju u DataFrame strukturu na osnovu putanje u okviru distribuiranog fajl sistema. Prilikom učitavanja podataka se određuje i njihov prostorni opseg na jedan od sledećih načina:

1 Čitanjem pridruženih metapodataka, na primer onih koji su zapisani u zaglavlju LAS fajlova,

2 Izračunavanjem statistike na osnovu učitanih tačaka.

Da bi se izvršila normalizacija tačaka neophodno je poznavati ili definisati preciznost koordinata tačaka. Za to mogu poslužiti metapodaci iz zaglavlja LAS fajla, a ukoliko oni ne postoje, neophodno ih je uneti kao ulazni parametar. Izračunati prostorni opseg i preciznost se zatim koriste za određivanje Morton koda za svaku tačku i na taj način prostorno indeksiranje tačaka. Tako organizovane tačke se skladište u okviru distribuiranog, ili takozvanog Big Data, skladišta podataka. Za implentaciju je odabrano Hbase skladište podataka. Format zapisa, odnosno fizički model podataka, je opisan u nastavku poglavlja.

4.2.1. Logički model podataka

Većina postojećih sistema za upravljanje oblakom tačaka, bilo da su bazirani na fajl sistemu, bazama podataka ili Big Data platformama, se zasnivaju na sličnom logičkom modelu koji je baziran na prostornoj podeli na blokove. Obično se generišu blokovi pravougaonog oblika koji sadrže tačke čije koordinate pripadaju prostornom opsegu bloka [60][58][70]. Radi efikasnijeg pristupa, sami blokovi mogu biti indeksirani prostornim indeksom, kao i same tačke unutar blokova. Međutim, prema istraživanju opisanom u [70], pokazano je da pristup zasnovan na blokovima donosi začajnu količinu dodatne obrade, jer se zahteva pristup na nivou pojedinačnih tačaka. Može se zaključiti da je pristup zasnovan na blokovima najviše doprinosi prevazilaženju

ograničenja koje nosi izvršavanje na pojedinačnim radnim stanicama, to omogućuje maksimalnu efikasnost u takvim uslovima. S obzirom da Big Data arhitektura pruža skoro „neograničene“ resurse, moguće je skalirati i rešenja koja se zasnivaju na model pojedinačnih tačaka, odnosno takozvanom *flat* modelu. Prema tome, u okviru teze je odabran flat model za skladištenje oblaka tačaka. Takođe, dosta istraživanja uzima u obzir samo takozvane, dvoipodimenzionalne oblake tačaka. Takva istraživanja pre svega uzimaju u obzir topografske podatke koji su rezultat Lidar ili fotogrametrijkih snimanja na velikim oblastima. U takvim slučajevima je opravdano posmatrati podatke kao 2,5D jer je opseg horizontalnih koordinata za nekoliko redova veličine veći od opsega vertikalne koordinate. U okviru teze je odabrano rešenje koje će biti primenjivo ne samo na 2,5D oblake tačaka, već i na one koji se mogu tumačiti kao puni 3D podaci, proizašli iz terestričkog laserskog skeniranja ili bliskopredmetne fotogrametrije. Takođe, rešenje je primenjivo na sve opsege oblaka tačaka, od milimetarskog do kilometarskog nivoa.

S obzirom da u distribuiranom okruženju nije jednostavno upravljati kompleksim strukturama podataka, najjedostavniji pristup je transformacija višedimenzionalnih podataka na jednu dimenziju. Krive za popunjavanje prostora predstavljaju najbolje rešenje za redukciju dimenzionalnosti. Postojeći sistemi za rad sa prostorno-vremenskim Big Data, poput GeoMesa i GeoTrellis, koriste ili eksperimentišu sa nekoliko tipova SFC, prvenstveno Z-krivom ili Hilbertovom krivom. U okviru teze je odabrana Z-kriva iz prethodno opisanih razloga.

Z-kriva je bazirana na Morton kodovima koji se mogu izračunati na osnovu koordinata tačaka, na jednostavan način. Pre izračunavanja Morton koda neophodno je izvršiti normalizaciju koordinata kako bi one bile prevedene u celobrojne vrednosti. Nakon toga se Morton kod dobija jednostavnim preplitanjem binarnih vrednosti normalizovanih koordinata. Da bi se izvršila normalizacija koordinata, neophodno je odrediti prostorni opseg koordinata, kao i njihovu preciznost (na primer preciznost 0.001 predstavlja nivo jednog milimetra). Bez gubitka opštosti se može pretpostaviti da je preciznost ista po sve tri koordinate.

Normalizovane koordinate se dobijaju prema sledećoj jednačini:

$$x_{norm}=\frac{x-x_{min}}{p}, y_{norm}=\frac{y-y_{min}}{p}, z_{norm}=\frac{z-z_{min}}{p}, m=z_{21}\,y_{21}\,x_{21}\ldots z_2\,y_2\,x_2\,z_1\,y_1\,x_1$$

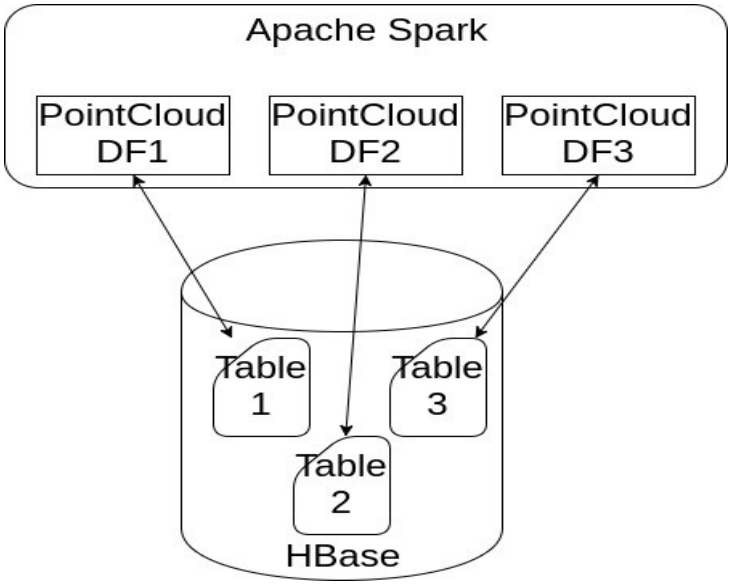
gde je *p* preciznost, a *m* Morton kod dobijen preplitanjem bitova koordinata.

U okviru teze je odabrano da se koriste 64 bitni Morton kodovi zbog implementacije na sistemima sa 64-bitnom arhitekturom, pa zbog toga postoji ograničenje da svaka koordinata može biti predstavljena sa 21-im bitom. Proširenje ovog ograničenja je relativno jednostavno.

4.2.2. Fizički model podataka

Za skladištenje podataka izabrano je Hbase distribuirano Big Data skladište. Detalji o Hbase su dati u poglavlju 2.3.5. Pored Hbase, razmatrana su i druga Big Data skladišta, Accumulo i Cassandra, ali je Hbase izabran zbog bolje podrške za integraciju sa Sparkom u trenutnku realizacije istraživanja.

Prema definisanom modelu, svaki oblak tačaka je uskladišten kao posebna tabela, prema flat modelu (slika 4.11).



Slika 4.11 Zapis oblaka tačaka u Hbase

Da bi se obezbedile optimalne performanse upita neophodno je definisati odgovarajuću šemu zapisa tačaka. Prema preporukama za definsanje Hbase strukture podataka izabran je zapis tačke prikazan u tabeli 4.1. Pri tome su definisani sledeći obavezni elementi:

- identifikator reda – za identifikator reda je izabran samo Morton kod što omogućuje efikasnu prostornu distribuciju podataka i prostorno filtriranje.

- idenitifikator familije kolona – uticaj organizacija kolona u familije kolona na skladištenje podataka i performansu izlazi iz okvira teze pa su zbog toga sve kolone smeštene u istu familiju. Prema preporukama za nomenklaturu u okviru Hbase, familijama kolona se daju kratka imena, te je ovde izabrano slovo P.

- identifikatori kolona – u okviru istraživanja je odlučeno da se svaki atribut oblaka tačaka mapira u pojedinačnu kolonu kako bi se omogućilo filtriranje prema uskladištenim vrednostima. Drugi način bi bio da se podaci uskladište u obliku binarnog zapisa bez

definisane strukture, ali bi to uključilo potrebu za serijalizacijom/deserijalizacijom pri svakom upitu.

Tabela 4.1 Struktura zapisa tačke u Hbase

	Column Family								
	P								
Row Key	**Column Qualifiers**								
morton code	x	y	z	class.	intens.	red	green	blue	...
9920124610	6.739	−14.289	−1.548	0	−361	215	234	188	

4.2.3. Korisnički definisani tipovi podataka

Značajna funkcionalnost koju pruža SparkSQL za podršku naprednoj analitičkoj obradi su korisnički definisani tipovi (*user defined types* – UDT). Oni omogućuju da korisnik definiše svoje klase koje su interoperabilne sa SparkSQL. Kreiranjem korisnički definisanog tipa za, na primer, klasu X, omogućuje se definisanje DataFrame-a koji u svoju šemu uključuje X.

```
@SQLUserDefinedType(udt = classOf[PointUDT])
case class Point (x: Double,
                  y: Double,
                  z: Double,
                  normx: Integer,
                  normy: Integer,
                  normz: Integer,
                  var mortonCode: Long,
                  intensity: Int,
                  classification: Short,
                  red: Short,
                  green: Short,
                  blue: Short) {
class PointUDT extends UserDefinedType[Point] {
  def dataType = StructType(Seq(
  StructField("x", DoubleType),
  StructField("y", DoubleType),
  StructField("z", DoubleType),
  StructField("normx", IntegerType),
  StructField("normy", IntegerType),
  StructField("normz", IntegerType),
  StructField("mortonCode", LongType),
  StructField("intensity", IntegerType),
  StructField("classification", ShortType),
  StructField("red", ShortType),
  StructField("green", ShortType),
  StructField("blue", ShortType)
 ))
```

Slika 4.12 Korisnički definisan tip Point

Da bi se definisao korisnički definisan tip neophodno je opisati njegovu strukturu na način razumljiv Sparku. Na slici 4.12 je prikazan tip Point koji opisuje tačku kao element oblaka tačaka. Najpre je definisana klasa PointUDT koja definiše tipove podataka za svako polje klase Point

prema logičkoj šemi opisanoj u prethodnom tekstu. Koordinate tačaka su definisane kao DoubleType, a ostalim poljima su pridruženi tipovi definisani prema LAS standardu. Na primer, klasifikacija je definisana kao ShortType, a Morton kod kao LongType.

Nakon registracije Point tipa, on će biti prepoznat od strane Sparka u okviru DataFrame operacija i SQL upita. Takođe, moguće je definisati korisnički definisane funkcije koje vrše operacije nad tim tipom podataka. Na slici 4.13 su prikazani primeri takvih funkcija.

```
val normalize: UserDefinedFunction = udf ((value: Double, offset: Double) => {
        ((value – offset) * resolution).toInt} )
val calculateMortonCode: UserDefinedFunction = udf((
        x: Int, y: Int, z: Int) => {
                val mortonCode = z3.apply(x, y ,z)
                mortonCode.z})
```

Slika 4.13 Korisnički definisane funkcije *normalize* i *calculateMortonCode*

4.3. Procesiranje upita

4.3.1. Prostorno filtriranje

Prostorno filtiranje predstavlja jednu od osnovnih operacija nad oblakom tačaka. Ono predstavlja izdvajanje tačaka koje pripadaju definisanom prostornom opsegu. U slučaju distribuiranog skladištenja i obrade oblaka tačaka prostorno filtriranje nije trivijalna operacija, s obzirom da su podaci organizovani u linerane strukture primenom SFC.

Naivan način prostornog filtriranje nad SFC se definiše na sledeći način:

1 na osnovu minimalnih koordinata se odredi minimalan Morton kod za prostorni opseg Z_{min},

2 na osnovu maksimalnih koordinata se odredi maksimalan Morton kod za prostorni opseg Z_{max},

3 izdvoje se sve tačke za koje važi $Z_{min} \leq Z_{point} \leq Z_{max}$

4 odbace se sve tačke selektovane u prethodnom koraku, a koje ne pripadaju definisanom prostornom opsegu.

Ovaj pristup je veoma neefikasan jer se u koraku može 3. obuhvatiti veliki broj takozvanih lažno pozitivnih rezultata (slika 4.14).

Prema [68] efikasniji pristup kojim se inicijalni set Mortonovih kodova značajno umanjuje je definisan na sledeći način:

1. odrede se nizovi opsega Morton kodova koji pripadaju definisanom prostornom opsegu (slika 4.15).

$(Z_{min}, Z_{max}) \geq ((Z^1_{min}, Z^1_{max}), (Z^2_{min}, Z^2_{max}), \ldots, (Z^m_{min}, Z^m_{max}))$

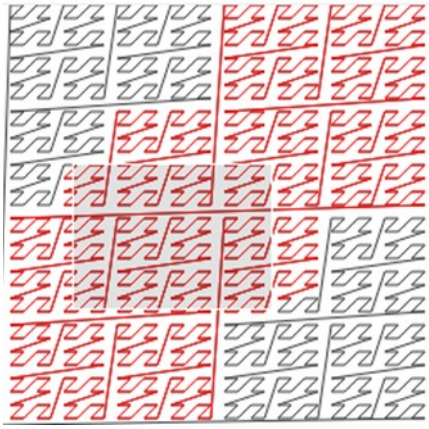
2. Ako je $Z^{i+1}_{min} - Z^i_{max} < delta$, nizovi se spajaju kako bi se redukovao broj opsega pretrage. Na taj način skup tačaka kandidata se ne uvećava previše, a ubrazava se postupak pretrage.

   $((Z^1_{min}, Z^1_{max}), (Z^2_{min}, Z^2_{max}), \ldots, (Z^m_{min}, Z^m_{max})) \geq ((Z^1_{min}, Z^1_{max}), (Z^2_{min}, Z^2_{max}), \ldots, (Z^k_{min}, Z^k_{max}))$

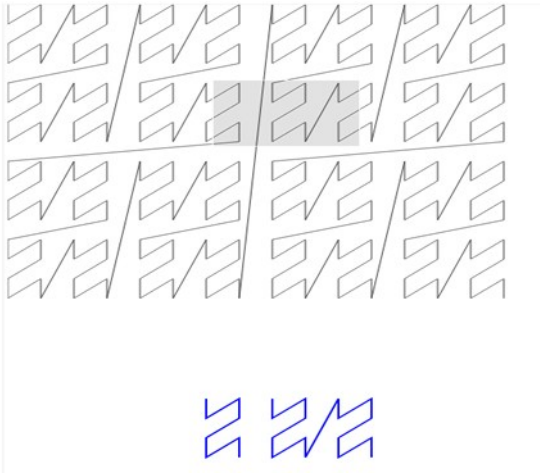
   Gde je m broj opsega u definisanom opsegu pretrage, a k je broj opsega nakon spajanja, i važi $k < m$.

3. Spark broadcast join operacijom se niz opsega distribuira svim izvršiocima koji izdvajaju tačke čiji Morton kodovi su obuhvaćeni nekim opsegom, odnosno važi $Z^i_{min} \leq Z_{point} \leq Z^i_{max}$, gde je i = 1 do k.

4. Koordinate tačaka se zatim porede sa definisanim protornim opsegom kako bi se dobio konačan rezultat (slika 4.15).



Slika 4.14 Crvenom bojom su prikazane svi Morton kodovi koji pripadaju opsegu Z_{min} do Z_{max}



Slika 4.15 Plavom bojom su prikazani izdvojeni opsezi

4.3.2. K najbližih suseda - kNN

Određivanje K najbližih suseda predstavlja osnovu za veliki broj operaija nad oblakom tačaka, kao što su klasifikacija, hijerarhijska podela, računarska vizija, redukcija dimenzionalnosti, detekcija grešaka, određivanje normala i druge. Zbog osobine da zadržavaju prostornu bliskost tačaka, SFC predstavljaju osnovu algoritama za određivanje kNN.

Prema [68], algoritam za određivanje najbližih suseda za svaku tačku i, na osnovu niza tačaka sortiranih prema Morton kodu, koji izdvaja tačke u opsegu $(I - a \times k, i + a \times k)$ se definiše na sledeći način:

1. Niz tačaka je sortiran prema rastućoj vrednosti Morton koda,
2. RDD particije se kreiraju korišćenjem Custom Range Partitioner kojiomogućuje preklapanje između particija za $a \times k$ redova. Konstanta a omogućuje podešavanja brzine izvršavanja i tačnosti. Veća vrednost omogućuje bolju tačnost, ali vodi ka sporjem izvršvanju.
3. Korišćenjem operacije MapPartitions nad definisanim particijama, kreira se novi RDD koji sadrži parove (tačka, skup kandidata za najbliže suseda) za svaku tačku iz inicijalnog RDD-a, u obliku $(P_i, (P_{i-a\times k}, P_{i-a\times k+1}, \ldots, P_{i-1}, P_{i+1}, \ldots, P_{i+a\times k-1}, P_{i+a\times k}))$.
4. Korišćenjem operacije MapPartitions još jednom, nad novim RDD-om, sortira se skup kandidata za najbliže susede prema udaljensti od posmatrane tačke P_i i izvaja se k najbližih suseda.

4.4. Poboljšanje rezultata primenom višestrukih krivih za popunu prostora

Za isti skup tačaka se mogu generisati različite Z-krive promenom redosleda koordinata ili primenom određene transformacije, na primer translacije ili rotacije. Na taj način tačke koje su udaljene u originalnoj Z-krivoj postaju bliske u transformisanoj Z-krivoj. Primenom višestrukih Z-krivih može postići bolja tačnost, na primer kod pretrage najbližih suseda. Sa druge strane, dodavanje još jedne (ili više) Z-krivih vodi ka povećanju memorijskih zahteva i vremena obrade [51]. Taj nedostatak gubi na značaju u slučaju distribuirane obrade, jer postoji značajno veći memorijski prostor, a performansa obrade se podiže primenom paralelizacije algoritama.

Osnovni cilj korišćenja višestrukih Z-krivih za prostorno filtriranje je pronalaženje krive koja proizvodi najmanji skup opsega za odabrani prostorni filter, što vodi ka manjem broju opsega koji se distribuiraju kroz klaster. Da bi se odredio najmanji skup opsega za zadati prostorni filter primenjen je sledeći pristup:

1. Generisana su 4 skupa primenom slučajno generisanih vektora (X^i_{shift}, Y^i_{shift}, Z^i_{shift}) rotacijom primenom slučajno generisanih uglova (W^i, F^i, K^i), gde je $1 \leq i \leq 4$.
2. Generisani su Morton kodovi za svaki od 4 skupa koordinata.
3. Za skup tačaka kandidata je izdvojen iz najkraćeg niza opsega.

5. Implementacioni model i verifikacija

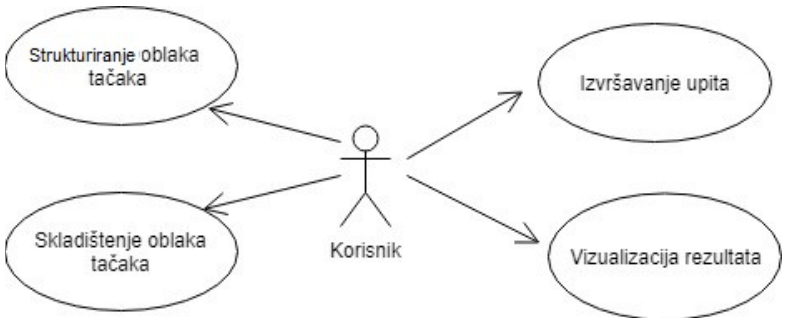
Na osnovu konceptualnog modela opisanog u poglavlju 4., u cilju verifikacije i validacije prodloženog modela, formiran je implentacioni model u obliku Spark aplikacije. Verifikacije je obuhvatila dva eksperimenta, poređenje osobina preloženog modela sa PostgreSQL relacionom bazom podataka i ispitivanje skalabilnosti modela u odnosu na količinu podataka i broj računara u klasteru.

5.1. Implementacioni model

Arhitektura implementacionog modela je predstavljena u formi UML dijagrama. Use case dijagramom je predstavljena interakcija korisnika sa sistemom. Dijagramima sekvenci su opisani tipovi i redosled interakcija objekata u sistemu. Dijagramom klasa su predstavljeni entiteti u sistemu i odnosi među njima.

5.1.1. Use Case dijagram

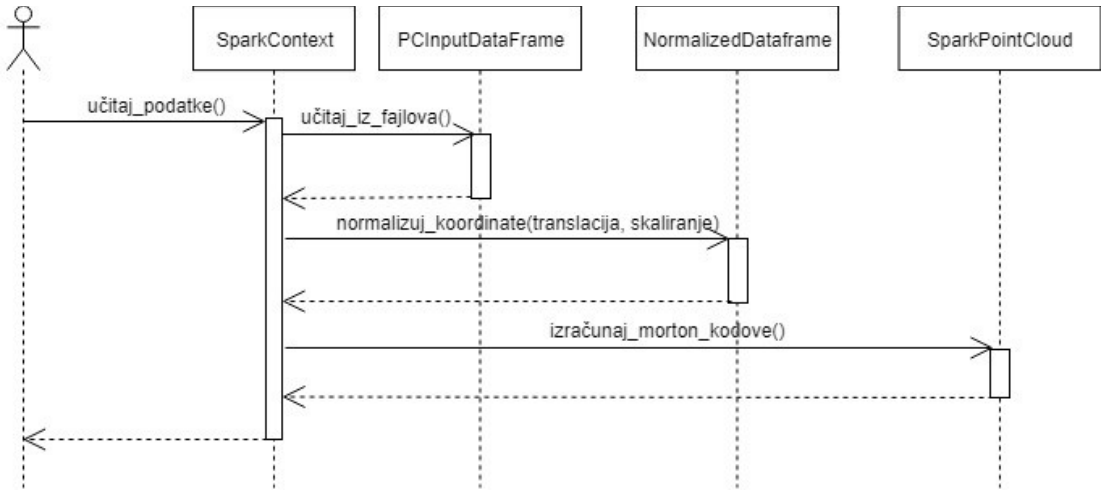
Slučajevi korišćenja implemeniranog sistema se mogu predstaviti relativno jednostavnim *Use Case* dijagramom. Prvi korak je učitavanje oblaka tačaka uskladištenih u fajl sistemu i transformacija u odgovarajuću distribuiranu strukturu podataka. Tako transformisani podaci se zapisuju, odnosno skladište, u HDFS ili neko od *key-value* skladišta prema fizičkoj strukturi zapisa definisanoj u prethodnom poglavlju. Nad učitanim podacima se mogu izvršavati definisani prostorni upiti, odnosno prostorno filtriranje i KNN. Vizualizacija rezultata je važan korak koji bi omogućio korisniku da proveri rezultate upita. Implementirani model bi mogao da bude integrisan sa Potree [55] alatom koji omogućuje vizualizaciju oblaka tačaka kroz web okruženje. Međutim, takva implementacija bi zahtevala velike vremenske resurse te je zbog toga izostavljena iz ovog rada.



Slika 5.1. *Use Case* dijagram

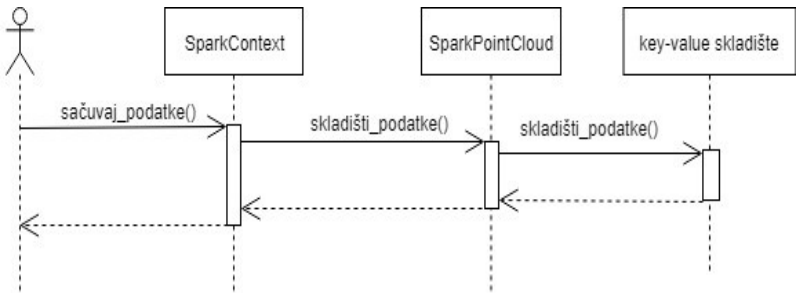
5.1.2. Dijagrami sekvenci

Svaka vrsta interakcije korisnika sa sistemom je opisana dijagramom sekvenci koji detaljno opisuje redosled izvršavanja operacija. Korisnik koji inicira učitavanje podataka predstavlja učesnika u modelu. SparkContext objekat prima poruku i inicira IO akciju učitavanja podataka. Učitani podaci se skladište u PCInputDataFrame. Nakon toga se inicira akcija za normalizaciju koordinata nakon čega se kreira NormalizedDataframe. Da bi se izvršila normalizacija koordinata neophodno je odrediti minimalne koordinate oblaka tačaka, kao i preciznost koordinata. Ti podaci se mogu unapred zadati ili odrediti iz ulaznih fajlova na način opisan u poglavlju 4. Na osnovu NormalizedDataframe se kreira SparkPointCloud dataframe koji uključuje tačke sa pridruženim Morton kodovima, sortirane u rastućem redosledu Morton kodova (slika 5.2.).



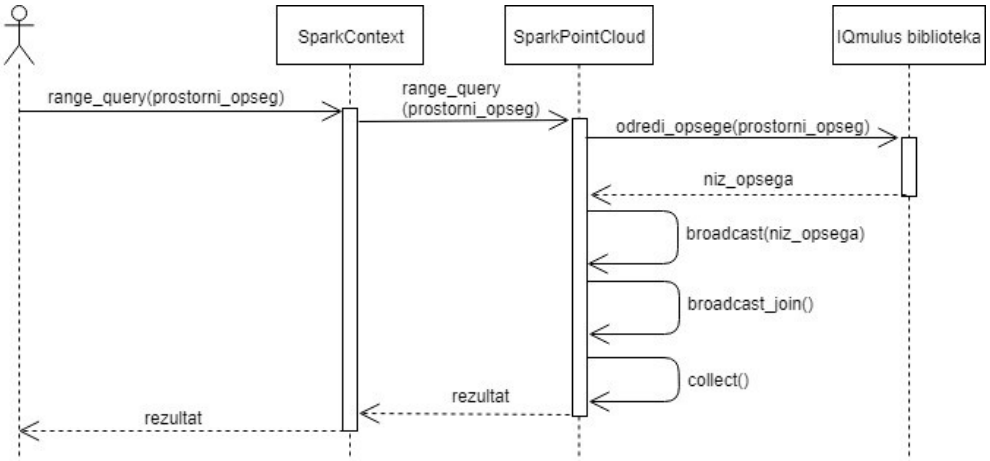
Slika 5.2. Dijagram sekvenci učitavanja podataka

Dijagram sekvenci prikazan na slici 5.3. prikazuje akcije vezane za skladištenje podataka iz SparkPointCloud dataframe-a na permanentno key-value skladište. Prema fizičkom modelu opisanom u poglavlju 5.



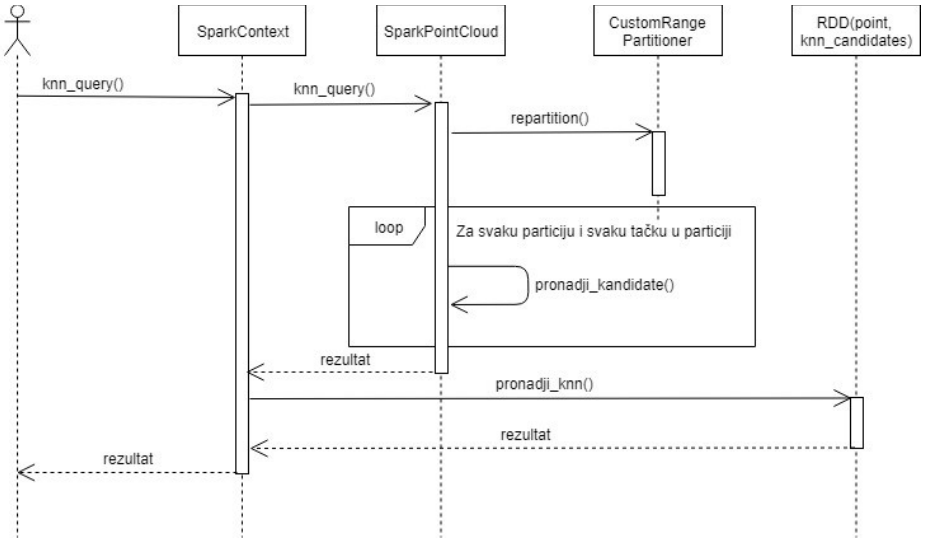
Slika 5.3. Dijagram sekvenci skladištenja podataka

Slika 5.4. predstavlja tok izvršavanja prostorno filtriranje upita nad SparkPointCloud dataframe na način koji je opisan u poglavlju 4. Objekti predstavljeni na dijagramu osim učesnika i SparkPointCloud uključuju i Iqmulus biblioteku koja omogućuje određivanje prostornih opsega za prostorno filtriranje, na osnovu kojih se kreira Ranges dataframe koji se broadcast-uje i zatim se vrši spatial join sa SparkPointCloud dataframe.



Slika 5.4. Dijagram sekvenci za *prostorno filtriranje*

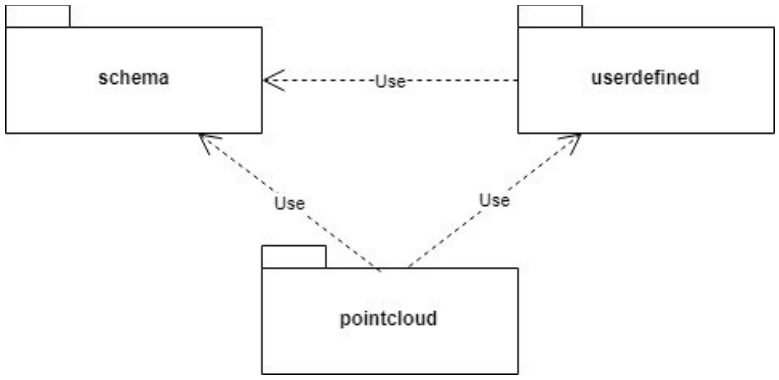
KNN query je opisan dijagramom sa slike 5.5. S obzirom da KNN posmatra susedne određeni broj susednih tačaka potrebno je bilo reparticionisati SparkPointCloud dataframe kreiranjem posebnog range patritioner-a, a zatim vršiti operacije nad svakom particijom posebno. To je predstavljeno na dijagramu povratnom vezom (loop).



Slika 5.5 Dijagram sekvenci za *KNN query*

5.1.3. Dijagram klasa

Dijagram klasa uključuje tri paketa (slika 5.6). U okviru paketa *schema* definisane su klase koje opisuju strukturu podataka prema logičkom modelu prikazanom u poglavlju 4. Paket *userdefined* definiše korisničke tipove podataka na osnovu klasa iz schema paketa. Paket *pointcloud* sadrži klase u okviru kojih je definisana logika sistema, odnosno operacije nad oblakom tačaka. Dijagram klasa je predstavljen na slici 5.7.

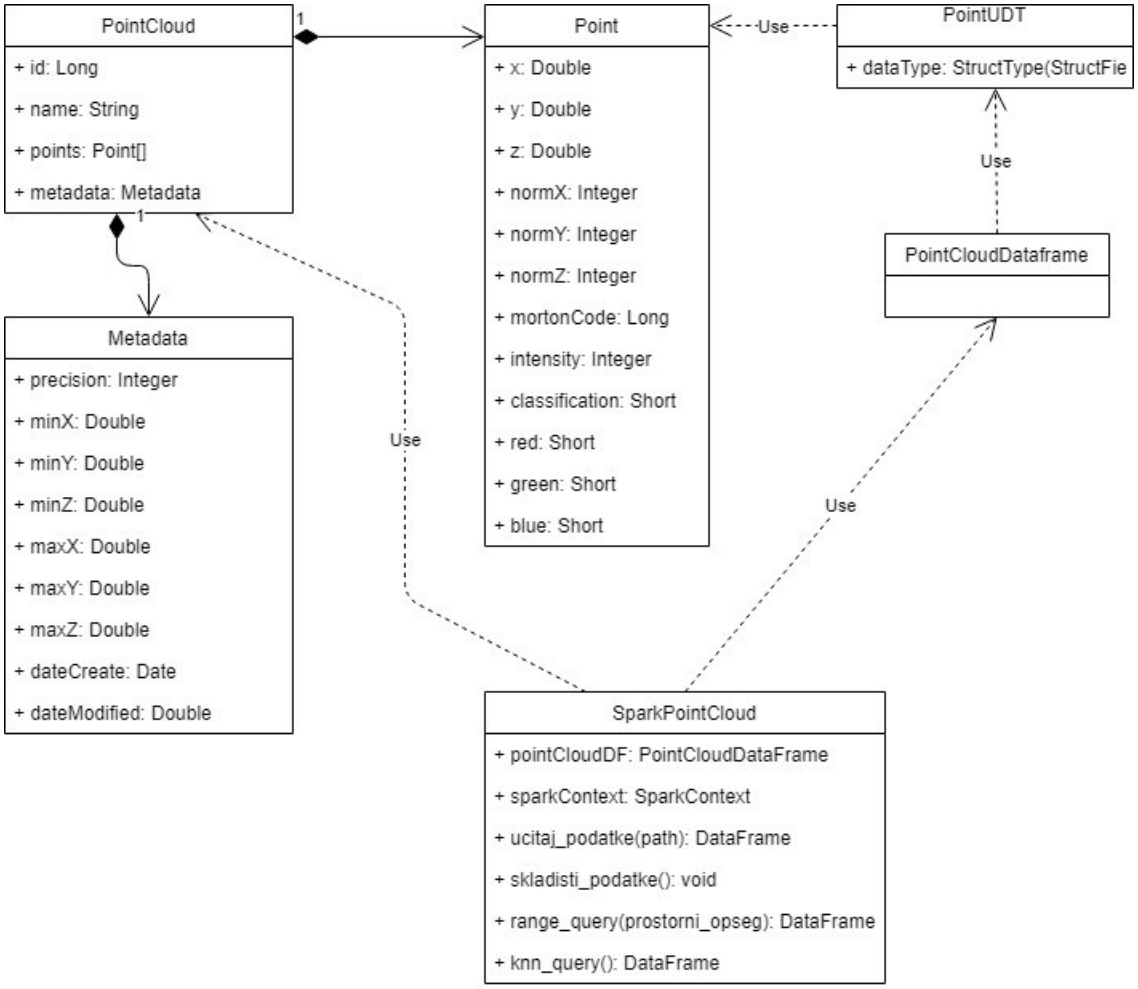


Slika 5.6. Dijagram paketa

Osnovne klase u paketu *schema* su *PointCloud*, *Metadata*, i *Point*. Klasa *PointCloud* opisuje entitet oblaka tačaka koji je definisam jedinstvnim identifikatorom i nazivom. Pored toga sadrži 0 ili više entiteta tipa *Point,* i stoji u odnosu 1 prema 1 sa entitetom *Metadata*. Entitet *Metadata* predstavlja metapodatke za entitet *PointCloud*. Metapodaci sadrže prostorni opseg koji obuhvata sve tačke iz oblaka, kao i preciznost koordinata tačaka. Te vrednosti se koriste za normalizaciju koordinata tačaka pre izračunavanja Morton kodova. Klasa *Point* opisuje entitet tačke iz oblaka sa odgovarajućim atributima koji su opisani u prethodnim poglavljima.

Paket *userdefined* sadrži korisnički definisane tipove i funkcije. Klasa *PointUDT* definiše šemu putem koje se podaci iz entiteta *Point* učitava u odgovarajući dataframe. Takođe, ovaj paket sadrži korisnički definisane fukcije koje su prikazane u poglavlju 4. i koje se koriste u okviru SQL iskaza kojima se definišu transformacije nad *Dataframe*-ovima.

Paket *pointcloud* sadrži klase koje predstavljaju Dataframe-ove, kao i klasu *SparkPointCloud* u okviru koja pruža interfejs prema operacijama nad Dataframe entitetima. Klasa *SparkPointCloud* uključuje *SparkContext* preko kojeg se pokreće izvršavanje operacija nad Dataframe koji sadrže podatke oblaka tačaka i koji je odgovoran za ulazno/izlazne operacije.



Slika 5.7. Dijagram klasa

5.2. Verifikacija modela

5.2.1. Skupovi podataka

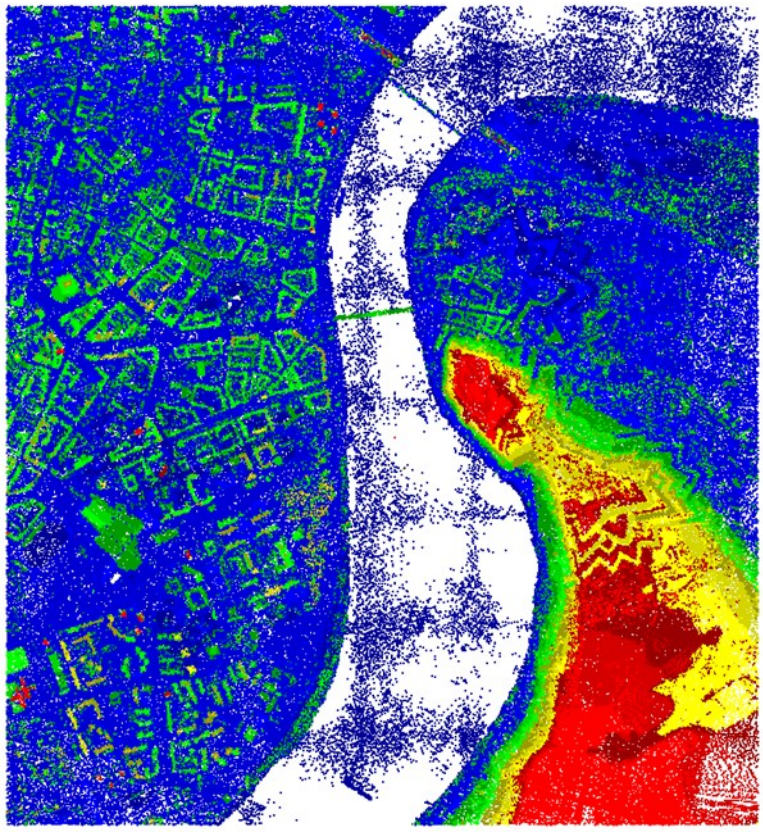
Radi verifikacije modela za upravljanje velikim oblacima tačaka korišćeni su podaci Laboratorije za Geoinformatiku, Fakulteta tehničkih nauka. Počevši od 2008. u okviru Laboratorije za Geoinformatiku, prikupljena je velika količina podataka u obliku oblaka tačaka, primenom avionskog i terestričkog laserskog skeniranja. S obizrom da se laboratorija nalazi u Novom Sadu, najveći deo podataka pokriva područje grada. Za potrebe izrade ove teze, izabrani su skupovi podataka prikazani u tabeli 5.1.

Tabela 5.1. Skupovi podatka za verifikaciju

Skup podataka	Opis podataka	Metoda prikupljanja	Broj tačaka
Skup 1	Oblak tačka za područje Univerzita u NS i Petrovaradina	avionsko	190 miliona
Skup 2	Oblak tačaka za područje podgrađa petrovaradinske tvrđave	terestričko	95 miliona
Skup 3	Oblak tačaka za područje FTN	terestičko	45 miliona

Skupovi podataka obuhvataju različite izvore i različite gustine tačaka. Primena podataka dobijenih terestričkim laserskim skeniranjem su omogućila je bolju distribuciju tačaka po sve 3 ose te tako dala bolji uvid u primenu odabranog modela nad oblacima tačaka koji imaju ravnomernu raspodelu po sve tri ose. Prostorni opsezi navedenih skupova podataka sa prikazani na slici 5.8, a izgled oblaka tačaka iz skupa 1 je prikazan na slici 5.9.

Slika 5.8. Prostorni opsezi korišćenih skupova podataka; Skup 1 (narandžasto); Skup 2 (plavo); Skup 3 (zeleno)



Slika 5.9. Oblak tačaka za područje Novog Sada prikupljen avionskim laserskim skeniranjem

5.2.1. Eksperimentalna platforma

Eksperiment je izveden u okviru Laboratorije za Geoinformatiku na Fakultetu tehničkih nauka, Univerziteta u Novom Sadu. Laboratorija je opremljena sa 16 računara povezanih 10-gigabitnom mrežom. Ukupan broj od šest računara je korišćen u okviru eksperimenta. Za poređenje funkcionalnosti su izabrane dve konfiguracije podataka:

- PostrgeSQL (verzija 9.5.10) sa flet tabelom gde svaki zapis predstavlja jednu tačku.
- Apache Spark (version 1.6.2).

Na jednom računaru je instaliran PostgreSQL. Spark je instaliran na pet računara, sa jednim masterom i četiri izvršioca. Hardveska konfiguracija računara je data u tabeli 5.2.

Tabela 5.2. Hardverska konfiguracija računara u klasteru

CPU	RAM	HDD
INTEL Core i7-6700 3.4 GHz	16 GB, DDR 4, 3200 MHz	1 TB, 7200 rpm

5.2.1.1. PostgreSQL

Radi učitavanja podataka u PostgreSQL tabelu podaci iz LAS fajlova su transformisani u tekstualni format i učitani korišćenjem PostgreSQL COPY komande (Slika 5.10).

```
CREATE TABLE point_cloud
(
  id bigint NOT NULL DEFAULT nextval('point_cloud_id_seq'::regclass),
  x numeric(15,2) NOT NULL,
  y numeric(15,2) NOT NULL,
  z numeric(15,2) NOT NULL,
  morton_code bigint,
  CONSTRAINT point_cloud_pkey PRIMARY KEY (id)
)
COPY point_cloud(x, y, z)
FROM 'pc_data.txt' DELIMITER ', ' CSV HEADER;
```

Slika 5.10. SQL upit za importovanje podataka u PostgreSQL

Nakon toga, za tačke su izračunati Morton kodovi korišćenjem pgmorton proširenja koje je modifikovano da omogući rad sa trodimenzionalnim Morton kodovima (https://github.com/Oslandia/pgmorton). Na kraju je generisan indeks tipa B-stabla nad kolonom koja sadrži Morton kodove i tabela koja predstavlja oblak tačaka je klasterovana radi reorganizacije redova prema rastućem Morton kodu (Slika 5.11).

```
CREATE INDEX point_cloud_morton_code_idx
  ON point_cloud
  USING btree
  (morton_code);

CLUSTER point_cloud USING point_cloud_morton_code_idx;
```

Slika 5.11. SQL upit indeksiranje oblaka tačaka prema Morton kodu

5.2.1.2 Apache Spark

U slučaju Spark instanci, odabrana je verzija 1.6.2 kako bi se iskoristila spark-iqmulus biblioteka za čitaje LAS fajlova. LAS fajlovi su distribuirani na izvršioce kako bi se optimizovala procedura

učitavanja. Podaci su importovani u odgovarajući DataFrame, kao i u čitav sistem, korišćenjem procedure opisane u prethodnom poglavlju.

5.2.1.3. Upit za prostorno filtriranje

Radi poređenja performansi Spark-a sa tradicionalnim SUBP, nad oba sistema je izvršen isti prostorni upit. Kao što je već objašnjeno, prostorni upit obuhvata ekstrakciju tačaka unutar zadatog trodimenzionalnog prostora opisanog minimalnim i maksimalnim x, y i z koordinatama. Sam upit je izvršen u dva koraka. Prvo, tačke kandidati su određene poređenjem Morton kodova, a zatim je konačan rezultat dobijen poređenjem samih koordinata.

5.2.1.4. PostgreSQL upit

Kao što je već rečeno, prvi upit sadrži poređenje Morton kodova sa minimalnom i maksimalnom vrednošću unutar zadatog prostora pretrage (Slika 5.12).

```
CREATE TABLE candidate_points AS
SELECT x, y, z FROM point_cloud WHERE morton_code >= Morton_Encode_3d(409800, 5010800, 70) AND morton_code <=
Morton_Encode_3d(409810, 5010810, 80)
```

Slika 5.12. SQL upit za izdvajanje tačaka kandidata

Konačni rezultat je dobijen upiton sa slike 5.13.

```
SELECT x, y, z FROM candidate_points WHERE x <= 409810 AND x >= 409800 AND y <= 5010810 AND y >= 5010800 AND z <=
80 AND z >= 70
```

Slika 5.13. SQL upit za dobijanje konačnog rezultata

5.2.1.5. Apache Spark upit

Spark implementacija prostornog filtriranja je izvršena korišćenjem Scala DataFrame API-ja, ali zbog čitljivosti ovde je prikazana ista implementacija primenom SQL koda. Granične koordinate prostornog filtera su transformisane u minimalni i maksimalni Morton kod. Radi redukcije prostora pretrage, celokupni opseg Moroton kodova je podeljen u niz kraćih opsega koji pripadaju prostoru pretrage. Takav niz je zatim transformisan u odgovarajuću DataFrame i distrubuiran kroz sistem, kako bi bio korišćen za JOIN operaciju. Skup tačaka kandidata je određen primenom upita sa slike 5.14.

```
CREATE TABLE candidate_points AS
SELECT x, y, z FROM point_cloud AS pc JOIN ranges AS r ON pc.morton_code >= r.min AND pc.morton_code <= r.max
```

Slika 5.14. SQL upit za dobijanje skupa tačaka kandidata

Konačni rezultat je dobijen upiton sa slike 5.15.

```
SELECT x, y, z FROM candidate_points WHERE x <= 409810 AND x >= 409800 AND y <= 5010810 AND y >= 5010800 AND z <=
80 AND z >= 70
```

Slika 5.15. SQL upit za dobijanje konačnog rezultata

5.2.2. Performansa upita

Kako bi se utvrdila skalabilnost obe metode uportebljena su tri skupa podataka sa različitim brojem tačaka. Prvi skup je generisan Lidar snimanjem iz vazduha, drugi terestričkim laserskim skeniranjem, a treći je dobijen spajanjem tri skupa podataka opisanih prethodno u poglavlju. Skupovi podataka su sadržali 190 miliona tačaka, 95 miliona tačaka, a treći 330 miliona tačaka.

Utvrđeno je, prema očekivanju, da Spark rešenje skalira mnogo bolje od PostgreSQL rešenja sa povećanjem broja tačaka. Vreme neophodno da Spark okruženje podesi svaki upit je najviše uticalo na performansu upita i bilo je skoro identično u sva tri slučaja (Tabela 5.3).

Tabela 5.3. Analiza performanse prostornog upita za obe metode

	PostgreSQL	**Apache Spark**
Skup 1	7038 ms	2307 ms
Skup 2	4519 ms	1807 ms
Skup 3	10,612 ms	2614 ms

Drugi eksperiment je izvršen kako bi se utvrdio uticaj veličine klastera na brzinu izvršavanja. Pri tom su korišćene tri različite postavke. Prvi klaster se sastojao od jednog master čvora i dva izvršioca, drugi od master čvora i četiri ivršioca, a treći od master čvora i osam izvršioca. Eksperiment je izveden korišćenjem prvog skupa podataka iz prethodnog eksperimenta. Ponovo, vreme podešavanja za svaki upit je nosilo najveći deo samog izvršavanja. Osim toga Spark je pokazao dobru skalabilnost, odnosno dodavanjem novih čvorova u klaster se povećavala brzina obrade (Tabela 5.4).

Tabela 5.4. Analiza performanse prostornog upita za Spark u zavisnosti od veličine klastera

Veličina klastera	**Skup 1**
3 nodes	3620 ms
5 nodes	2307 ms
9 nodes	1477 ms

6. Diskusija

Izvedeni eksperimenti su pokazali opravdanost istraživanja i dobre performanse predloženog modela za upravljanje oblakom tačaka zasnovanog na Spark-u. Ovakav pristup je pokazao bolje performanse od tradicionalnog SUBP za rad sa velikim oblacima tačaka i bolju skalabilnost u slučaju povećanja broja tačaka. Kritičan uticaj na performanse Spark sistema je bilo vreme neophodno za distribuciju koda i podešavanje izvršenja, takozvani hladan start. U produkcionom okruženju se ovaj problem se najčešće rešava podizanjem posvećenog Spark klastera na nekoj cloud infrastrukturi. Ključni benefit je da povećanje broja tačaka nije imalo značajniji uticaj na vreme izvršavanja na Spark sistemu dok je PostgreSQL rešenje usporavalo gotovo linearno.

Takođe, prema očekivanjima, dodavanje čvorova u klaster je uticalo na povećanje performanse izvršavanja. Klaster sa tri čvora se pokazao dvostruko bržim u odnosu na jednu instancu PostgreSQL-a zbog dodatnog vremena za podešavanje klastera. Sa povećanjm klastera rastao je i dobitak u performansi pa tako instanca sa devet čvorova se pokazala pet puta brža u odnosu na PostgreSQL.

Iako je metod zasnovan na Sparku pokazao dobre performanse za rad sa velikim oblacima tačaka, to ima svoju cenu. Unutar memorijska obrada može postati usko grlo kada je neophodno obratiti pažnju na minimizaciju troškova, jer čuvanje podataka u memoriji je skupo, zahteva veliku količinu memorije, i ne može se upravljati na jednostavan način. Zbog toga što Spark zahteva značajne memorijske resurse, troškovi Spark klastera mogu biti veliki, te je u okviru ove teze korišćeno okruženje postojeće računarske mreže koje je manje optimizovano u odnosu na postojeća cloud rešenja. Prema tome, podešavanje Sparka je izvršeno ručno, izmenom određenih konfiguracionih parametara, što zahteva značajan napor. Dodatni problem predstavlja to što se moraju pažljivo definisati algoritmi jer neoptimalna rešenja vode ka gubitku performansi zbog dodatnog prenosa podataka kroz mrežu, odnosno takozvanog shuffling-a. Zbog toga uvek treba imati na umu troškove i kompleksnot rešenja kada se Spark koristi za obradu podataka.

U okviru teze su izvršeni određeni izbori, u pogledu logičke i fizičke strukture podataka, kao i definicije algoritama. Ti izbori su načinjeni svesno sa ciljem provere osnovne hipoteze. Zbog navedenog postojeće rešenje može biti manje optimalno od drugačijih pristupa za neke oblasti primene. U tabeli 6.2 su prikazani primeri uspešne primene modela za upravljanje velikim serijama geoprostornih podataka.

Odluka koja može najviše da utiče na kompleksnost i performansu sistema je izbor krive za popunjavanje prostora. Kao što je navedeno, najčešći izbor predstavljaju Z-kriva i Hilbertova kriva. Prvenstveni razlozi za odabir Z-krive su bili solidno očuvanje prosorne bliskosti, logička bliskost oktalnom stablu i jednostavnost prostorne pretrage. Implementacija rešenja primenom Hilbertove je značajno kompleksnija. Iako je dokazano da Hilbertova kriva najbolje očuvava prostornu bliskost dobit koju to nosi je umanjena samom kompleksnošću izračunavanja. Zbog toga upotreba Hilbertove krive može najbolje pokazati svoje prednosti u primenama gde je performansa kritičan faktor.

Dalje unapređenje sistema bi se moglo izvesti skladištenjem više oblaka tačaka u okviru iste tabele u Big Data skladištu. To bi zahtevalo proširenje identifikatora reda tako da se ispred Morton koda doda identifikator oblaka tačaka. Takođe, u tom slučaju bi bilo potrebno sve tačke dovesti u zajednički referentni sistem, gde bi se mogao iskoristiti neki od načina za globalno geokodiranje poput Geohash, kojim bi se zatim proširio identifikator reda.

Količina podataka zapisanih u Big Data skladištu, u smislu memorijskog prostora, nije detaljno razmatrana u okviru teze, ali ona može imati uticaja u produkcionim okruženjima. Jednostavan način za redukciju podataka bi bio da se umesto definisanja odgovarajuće kolone za svaki atribut tačke, oni budu zapisani u okviru jedne kolone u kompaktnom binarnom zapisu. S druge strane takav pristup bi zahtevao serijalizaciju/deserijalizaciju prilikom svake ulazno/izlazne operacije. Drugi problem takvog pristupa bi bio filtriranje prema nekom od atributa, jer bi dizajn upita bio značajno kompleksniji, što bi moralo biti rešeno uvođenjem korisnički definisanih funkcija na nivou obrade. Svakako, rešenja koja bi služila samo za prostorno filtriranje bi mogla da profitiraju primenom takvog pristupa.

Otvoreno pitanje prilikom definisanja strukture oblaka tačaka je izbor između flat ili blokovske strukture. Prema istraživanjim blokovska struktura ima nedosatak jer proizvodi značajno više false pozitive vrednosti prilikom upita. Sa druge strane, skladištenje tačaka po blokovima omogućuje manji broj zapisa, veću kompresiju podataka, što u određenim slučajevima može povećati performanse sistema. Zbog toga je neophodno imati na umu zahteve koji se stavljaju pred sistem prilikom definisanja strukture zapisa. Tabela 6.1 prikazuje uporedni pregled prednosti flat i blokovske strukture.

Tabela 6.1 Poređenje prednosti flat i blok logičkog modela podataka

FUNKCIONALNOST		FLAT MODEL	BLOK MODEL
Strukturiranje i skladištenje	**Indeksiranje**	Jednostavno generisanje i skladištenje.	Manji skladišni zahetvi zbog mogućnosti kompresije podataka.
	Nivoi detalja	Nema značajnijih razlika u odnosu na blok model.	Nema značajnijih razlika u odnosu na flat model.
Geoprostorne analize		Jednostavniji i brži pristup potrebnim podacima. Manja potreba za mrežnom komunikacijom između izvršioca.	Umanjenje performanse zbog serijalizacije/deserijalizacije podataka i mrežne komunikacije.
Vizualizacija geoprostornih podataka	**Rasterizacija**	Jednostavnije izdvajanje neophodnih tačaka (npr. prema klasifikaciji)	Jednostavna rasterizacija kompletnog bloka.
	Renderovanje oblaka tačaka	Jednostavno pronalaženje ključnih tačaka.	Može biti veoma efikasno ukoliko se blokovi pripreme prema hijerarhijskom modelu pogodnom za renderovanje.
Upravljanje vremenskim serijama		Moguće na nivou pojedinačnih tačaka (npr. izmena klasifikacije)	Moguće na nivou bloka. Kompleksnija izmena podataka.
Distribucija podataka		Ekstrakcija podataka u jednom koraku obrade *clip-zip-ship*.	Ekstrakcija podataka u više koraka (ukoliko se ne preuzima kompletan blok).
Kontrola kvaliteta	**Kontrola tačnosti**	Nema značajnijih razlika.	Nema značajnijih razlika.
	Ekstrakcija metapodataka	Nema značajnijih razlika.	Nema značajnijih razlika.

Kao što je prethodno navedeno, u okviru nekoliko istraživanja su celokupan model ili pojedini koncepti, iskorišćeni kao osnova za implementaciju rešenja za upravljanje geoprostornim Big Data u nekoliko oblasti primene: vizualizaciji, geoprostornim analizama, praćenju promena i pametnim gradovima. Pregled tih istraživanja, koja se mogu tumačiti kao praktična verifikacija predstavljenog modela, je dat u tabeli 6.2.

Tabela 6.2 Oblasti primene modela za upravljane velikim serijama geoprostornih podataka

OBLAST PRIMENE	PRIMER
Vizualzacija geoprostornih podataka	U [50] je prikazana vizualizacija oblaka tačaka zasnovana na voxel-ima primenom CesiumJS softverske biblioteke. Sama izrada voxel-a se vrši na jednom računaru, a podaci se skladište u okviru MongoDB (granice voxela) i PostgreSQL (oblak tačaka). Opisani proces se može jednostavno unaprediti primenom koncepata opisanih u okviru ove teze kako bi se ceo sistem skalirao za primenu nad velikim oblacima tačaka.
Geoprostorne analize - klasifikacija	U [7] je predstavljena distribuirana klasifikacija i segmentacija podataka dobijenih tehnologijama daljinske detekcije. Kavetrnalno stablo (u principu Z-kriva) se koristi za particionisaje i distribuirano indeksiranje. Kao platforma za distribuiranu obradu se koristi Apache Pig koji se izvršava u okviru Hadoop klastera. Sama implementacija je izvršena adaptacijom Apache Zepellin *notebook* sistema.
Geoprostorne analize – ekstrakcija semantike	Proširenje oblaka tačaka pridruženom semantikom i korišćenje iste za efikasno izvršavanje upita je predstavljeno u okviru [27]. Model je implementiran za izvršavanje na jednom računaru. S obzirom na to, primena na većim skupovima podataka je rezultovala slabijom performansom. Model za upravljanje velikim serijama geoprostornih podataka se može primeniti za ekstrakciju semantike, pridruživanje semantike geoprostornim podacima (oblacima tačaka) i upite zasnovane na semantici.
Podrška odlučivanju	Autori u [46] su analizirali primenu koncepata zasnovanih na modelu opisanom u disertaciji za analizu prostorno-vremenskih šema mobilnosti na osnovu GPS podataka kako bi redukovali vreme potrebno za otkrivanje korisnih informacija radi podrške procesima odlučivanja.
Upravljanje pametnim gradovima	U okviru [6] je prikazan GAMINESS, model za upravljanje pametnim gradovima primenom Big Data koncepata radi prevazilaženja ograničenja tradicionalnih modela zasnovanih na relacionim bazama podataka. GAMINESS koristi koncepte predstavljene u kviru ove disertacije za konverziju i strukturiranje podataka iz oblasti pamentih gradova, kao što su geoprostorni podaci i podaci sa senzora u strukturu definisanu kroz Spark DataFrame-ove.
Geoprostorne analize – ekstrakcija objekata	Istraživanje višenivojske ekstrakcije objekata i semantičke klasifikacija velikih oblaka tačaka primenom *random forest* klasifikatora je prikazano u [62]. Rešenje je zasnovano na sličnim konceptima onima predstavljenim u okviru disertacije, odnosno na Spark platformi sa tom razlikom da se kao skladište koristi Cadsandra umesto Hbase.
Praćenje promena	U okviru [76] je analizirana upotreba distribuirane arhitekture za praćenje promena na oblacima tačaka pri čestoj akviziciji podataka primenom mobilnog laserskog skeniranja. Detekcija promena je zasnovana na indekxiranju u dva nivoa, R-stablo indeksu za indeksiranje blokova tačaka i octree indeksu za indeksiranje unutar bloka. Poređenje pojedinačnih tačaka je vršeno metodom *self-join*. Autori nisu razmatrali skladištenje različitih verzija tačaka na način predstavljen u okviru ove disertacije koje bi omogućilo praćenje istorije promena. Performase predstavljenog rešenja takođe nisu razmatrane, ali je za očekivati da bi migracija rešenja na Spark značajno unapredila performanse.

7. Zaključak

U današnje vreme, više nego ikad ranije prikupljaju se ogromne količine geoprostornih podataka. Takođe brzina pristizanja takvih serija podataka zahteva obradu u realnom vremenu. Zbog toga je neophodno pronaći rešenja za skladištenje i obradu takvih količina podataka. Rezultat toga je rast broja istraživanja koja se bave tom temom iz godine u godinu. Najveći deo tih istraživanja ima za temu rad sa rasterskim i vektorskim podacima, a samo je jedan manji deo orijentisan ka oblaku tačaka. Zbog toga je fokus diseratcije na oblaku tačaka koji, zbog strukture, raznovrsnosti, i količine prikupljenih podataka zahteva definisanje metoda zasnovanih na Big Data paradigmi.

Pre definisanja samog modela predstavljeni su koncepti na kojima se zasniva rad sa velikim serijama i količinama geoprostornih podataka. Osnovu predstavlja Map-Reduce programski model, distribuirani fajl sistemi, i Big Data skladišta podataka. Najčešće korišćeni sistem za distribuiranu obradu je Hadoop koji predstavlja prvu open-source implementaciju Map-Reduce programskog modela. Poslednjih godina rešenja migriraju ka platformama koje pružaju bolje performanse, kao što je Spark. Osnovni problem koji moraju da reše Big Data platforme za rad sa geoprostornim ili prostorno-vremenskim podacima je indeksiranje u okviru distribuiranog sistema gde nije moguće jednostavno predstaviti višedimenzionalne strukture. Zbog toga vrši redukcija dimenzionalnosti na jednu dimenziju i delovi nizova podataka distribuiraju kroz sistem. Osnovni način za svođenje višedimenzionalnih podataka na jednodimenzionalne su krive za popunjavanje prostora, jer omogućuju zadržavanje prostorne bliskosti podataka nakon linearizacije.

U okviru disertacije je definisan model za upravljanje velikim serijama geoprostornih podataka, sa fokusom na oblak tačaka. Najpre je definisana kompletna arhitektura sistema na osnovu mogućih načina korišćenja takvog sistema.

Kao metod za indeksiranje oblaka tačaka je izabrana Z-kriva, zbog toga što zadržava solidnu prostornu bliskost, omogućuje jednostavno mapiranje na oktalno stablo, i jednostavnu implementaciju prostornih upita. Oblak tačaka indeksiran Z-krivom se distribuira u okviru klastra radi paralelne obrade. Kao platforma za obradu je izabran Spark jer pruža odličnu performansu i skalabilnost. U okviru Sparka je oblak tačaka predstavljen kao DataFrame sačinjen od korisnički definisanog tipa Point. Oblak tačaka se indeksira u toku postupka importovanja podataka. Sistem je definisan tako da može da importuje tačke iz LAS, tekstualnih ili JSON fajlove. Na osnovu

importovanih tačaka se formira pomenuti DataFrame, a same tačke se skladište u Hbase prema definisanoj fizičkoj strukturi podataka.

Pored toga, definisane su operacije nad takvim DataFrame zasnovane na SQL upitima. Prednost SparkSQL je što se moguće definisanjem korisnički definisanih tipova i funkcija opisati operacije jezičkim konstruktima iz domena primene. Prikazane su dve veoma često korišćene operacije nad oblakom tačaka, prostorno filtriranje i pretraga najbližih suseda. Takođe, su objašnjene mogućnosti za unapređenje tačnosti i efikasnosti obrade. Konačno, definisani model se može posmatrati kao prvi korak ka sveobuhvatnoj distribuiranoj bazi podataka oblaka tačaka.

U poređenju sa tradicionalnim načinima upravljanja oblakom tačaka, zasnovanim na fajl sistemu i jednoj procesorskoj jedinici, distribuirani pristup zasnovan na Sparku, pruža bolju skalabilnost, performansu, robustnost i otpornost na greške. Spark platforma se horizontalno sklaira jednostavnim dodavanjem čvorova u klaster. Predstavljeno rešenje je moguće jednostavno proširiti implementacijom dodatnih operacija nad oblakom tačaka primenom korisnički definisanih tipova i funkcija, kao što su na primer, segmentacija ili ekstrakcija objekata.

Rezultati eksperimenta su prikazali da Spark ima bolju performansu u poređenju sa PostgreSQL kada se posmatra izvšavanje prostornog filtriranja. Prema očekivanjima, Spark je pokazao odličnu skalabilnost. On je pokazao robusnu performansu sa povećanjem količine podataka, kao i unapređenje performansi dodavanjem čvorova u klaster. Prema tome, može se reći da su postavljene hipoteze potvrđene u potpunosti.

U okviru disertacije je fokus bio na rad sa pojedinačnim oblacima tačaka, sa osvrtom na mogućnost proširenja sistema za podršku višestrukim oblacima tačaka. Jedan od budućih pravaca istraživanja može biti u pravcu integracije oblaka tačaka prikupljenih različitim metodama, u različito vreme i/ili u različitom koordinatnom sistemu. Jedan od načina bi mogao da bude proširenje Morton koda preko 64 bita čime bi bilo moguće indeksirati veća područja ili dodati još jednu koordinatu, na primer vreme. Ako se Morton kod proširi na 128 bita, moguće je indeksirati sve koordinate u prostoru od 2.2 miliona kilometara sa milimetarskom preciznošću. Na taj način je moguće u okviru jedne Hbase tabele uskladištiti sve oblake tačaka, na primer, na nivou države. Na taj način bi državne geodetske organizacije mogle da uskladište oblake tačaka u sistem koji obezbeđuje praktično neograničen skladišni prostor, uz sprečavanje gubitka podataka i brzi pristup kroz definisane interfejse. To bi predstavilo unapređenje u odnosu na dosadašnji pristup baziran na

fajlovima, gde postoje problemi sa praćenjem verzija, bekapovanjem, i konkurentnim pristupom podacima.

Pored navedenog, još jedan pravac budućeg istraživanja bi bio korelacija sa drugim tipovima geoprostornih podataka, odnosno integracija sa rešenjima za rad sa rasterskim i vektorskim podacima, kao što su GeoMessa i GeoTrellis.

Literatura

[1] 3D Data Solved. Actionable 3D information powering digital asset management solutions. https://www.pointerra.com/ (poslednji pristup 15.04.2021)

[2] 5G, https://en.wikipedia.org/wiki/5G (poslednji pristup 18.04.2021)

[3] Accelerating Apache Spark 3.0 with GPUs and RAPIDS, https://developer.nvidia.com/blog/accelerating-apache-spark-3-0-with-gpus-and-rapids/ (poslednji pristup 22.04.2021)

[4] ACID, https://en.wikipedia.org/wiki/ACID, (poslednji pristup 20.05.2020)

[5] Alis, C.; Boehm, J.; Liu, K. Parallel processing of big point clouds using Z-Order-based partitioning. ISPRS Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. 2016, 41, 71–77.

[6] Amović, Mladen & Govedarica, Miro & Radulović, Aleksandra & Janković, Ivana. (2021). Big Data in Smart City: Management Challenges. Applied Sciences. 11. 10.3390/app11104557.

[7] Antunes, Rodrigo & Blaschke, Thomas & Tiede, Dirk & Bias, Edilson & Costa, Gilson & Nigri Happ, Patrick. (2018). Proof of concept of a novel cloud computing approach for object-based remote sensing data analysis and classification. GIScience & Remote Sensing. 56. 1-18. 10.1080/15481603.2018.1538621.

[8] Apache HBase Reference Guide, https://hbase.apache.org/book.html (poslednji pristup 23.03.2019)

[9] APACHE HIVE TM, https://hive.apache.org/, (poslednji pristup 12.11.2020)

[10] Axelsson, P.: DEM generation from laser scanner data using adaptive TIN models. Int. Arch. Photogram. Remote Sens. Spatial Inf. Sci. 33, 111–118 (2000)

[11] Bader M., Bungartz HJ., Mehl M. (2011) Space-Filling Curves. In: Padua D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09766-4_145

[12] Big Data y normas ISO relacionadas, https://www.pmg-ssi.com/2019/12/big-data-y-normas-iso-relacionadas/ (poslednji pristup 15.03.2021)

[13] Big data, https://en.wikipedia.org/wiki/Big_data (poslednji pristup 01.02.2021)

[14] Binarno stablo pretrage, https://sr.wikipedia.org/wiki/Binarno_stablo_pretrage (poslednji pristup 17.04.2018)

[15] Boehm J., File-centric organization of large LiDAR Point Clouds in a Big Data context, IQmulus Workshop - Processing Large Geospatial Data, Cardiff, Jul 08, 2014

[16] Boehm, J.; Liu, K.; Alis, C. Sideloading—Ingestion of large point clouds into the Apache Spark big data engine. ISPRS Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. 2016, XLI-B2, 343–348.

[17] Castro R., Lewiner T., Lopes H., Tavares G., Bordignon A., "Statistical optimization of octree searches", Computer Graphics Forum, v. 27, p. 1557–1566, 2008.

[18] Cesium, https://cesium.com/ (poslednji pristup 17.02.2021)

[19] Dai, Jing. (2011). Efficient Range Query Using Multiple Hilbert Curves. 10.5772/22413.

[20] Daniar Achakeev, Z-Order curve range query, http://bl.ocks.org/daniarleagk/278709dedf09451b794ff72c4c05cda1 (poslednji pristup 17.02.2020)

[21] Dasgupta A., Big data: the future is in analytics, http://www.geospatialworld.net/Magazine/MArticleView.aspx?aid=30512 (poslednji pristup 10.04.2014.)

[22] Dean, J. and Ghemawat, S. 2004. MapReduce: Simplified data processing on large clusters. In Proceedings of Operating Systems Design and Implementation (OSDI). San Francisco, CA. 137-150.

[23] Deep Dive into Spark SQL’s Catalyst Optimizer, https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html (poslednji pristup 18.03.2017)

[24] Deibe, D.; Amor, M.; Doallo, R. Big Data Geospatial Processing for Massive Aerial LiDAR Datasets. Remote Sens. 2020, 12, 719. https://doi.org/10.3390/rs12040719

[25] Dobos, L., Csabai, I., Szalai-Gindl, J. M., Budavari, T., and Szalay, A. S. (´ 2014). Point cloud databases. In Proceedings of the 26th International Conference on Scientific and Statistical Database Management, page 33. ACM.

[26] Du, Q. & Li, X. & Zhang, X.. (2014). A novel kNN algorithm with the Hilbert curve by random shifts. 8. 2907-2913.

[27] El-Mahgary, Sami & Virtanen, Juho-Pekka & Hyyppä, Hannu. (2020). A Simple Semantic-Based Data Storage Layout for Querying Point Clouds. ISPRS International Journal of Geo-Information. 9. 72. 10.3390/ijgi9020072.

[28] Elseberg, J., Borrmann, D., & Nüchter, A. (2013). One billion points in the cloud – an octree for efficient processing of 3D laser scans. Isprs Journal of Photogrammetry and Remote Sensing, 76, 76-88.

[29] Entwine Point Tile https://entwine.io/entwine-point-tile.html (poslednji pristup 17.02.2021)

[30] Faloutsos, C. and Roseman, S. (1989). Fractals for secondary key retrieval. In Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pages 247–252. ACM.

[31] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, Bigtable: a distributed storage system for structured data. 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI), {USENIX} (2006), pp. 205-218

[32] Fox A., Eichelberger C., Hughes J., Lyon S., “Spatiotemporal indexing in non-relational distributed databases,” in Proceeding of IEEE Conference on Big Data, pp. 291-299, 2013.

[33] Fuetterling, Valentin & Lojewski, C. & Pfreundt, Franz-Josef. (2014). High-performance delaunay triangulation for many-core computers. High-Performance Graphics 2014, HPG 2014 - Proceedings. 97-104. 10.2312/hpg.20141098.

[34] Functional programming, https://en.wikipedia.org/wiki/Functional_programming (poslednji pristup 10.12.2020)

[35] Geohash. http://en.wikipedia.org/wiki/Geohash. (poslednji pristup 01.02.2015)

[36] GeoTrellis is a geographic data processing engine for high performance applications, https://geotrellis.io/ (poslednji pristup 18.03.2021)

[37] GeoWave Overview, https://locationtech.github.io/geowave/overview.html (poslednji pristup 10.03.2021)

[38] Ghemawat S., Gobioff H., Leung S., "The Google file system", In 19th Symposium on Operating Systems Principles, pages 29–43, Lake George, New York, 2003.

[39] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: surface elements as rendering primitives. In SIGGRAPH '00, pages 335–342, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[40] Heidemann Hans Karl, Lidar Base Specification, https://pubs.usgs.gov/tm/11b4/pdf/tm11-B4.pdf (poslednji pristup 15.12.2020)

[41] Hilbert, D. (1891). Ueber die stetige abbildung einer line auf ein flachenst ¨ uck. ¨ Mathematische Annalen, 38(3):459–460.

[42] Integrated Geospatial Information Framework (IGIF), https://ggim.un.org/IGIF/ (poslednji pristup 12.10.2020)

[43] IQmulus, 2012. URL http://iqmulus.eu/ (poslednji pristup 11.10.2018)

[44] Isenburg Martin, LASzip: lossless compression of LiDAR data, https://laszip.org/ (poslednji pristup 10.12.2020)

[45] Jia Yu, Jinxuan Wu, Mohamed Sarwat. "A Demonstration of GeoSpark: A Cluster Computing Framework for Processing Big Spatial Data". (demo paper) In Proceeding of IEEE International Conference on Data Engineering ICDE 2016, Helsinki, FI, May 2016

[46] Kamal, Muhammad & Tahir, Ali & Kamal, Muhammad & Naeem, Muhammad Asif. (2020). Future Location Prediction for Emergency Vehicles Using Big Data: A Case Study

of Healthcare Engineering. Journal of Healthcare Engineering. 2020. 1-11. 10.1155/2020/6641571.

[47] Karau, H.; Konwinski, A.;Wendell, P.; Zaharia, M. Learning SPARK; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2015; ISBN 9781449358624.

[48] Liu, K. & Boehm, Jan. (2015). Classification of Big Point Cloud Data Using Cloud Computing. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XL-3/W3. 553-557. 10.5194/isprsarchives-XL-3-W3-553-2015.

[49] Magellan: Geospatial Processing made easy, https://magellan.ghost.io/magellan-geospatial-processing-made-easy/ (poslednji pristup 21.06.2018.)

[50] Marx, Andrew & Chou, Yu-Hsi & Mercy, Kevin & Windisch, Rich. (2019). A Lightweight, Robust Exploitation System for Temporal Stacks of UAS Data: Use Case for Forward-Deployed Military or Emergency Responders. Drones. 3. 29. 10.3390/drones3010029.

[51] Megiddo, N.; Shaft, U. Efficient nearest neighbor indexing based on a collection of space filling. RJ 10093 (91909). Math. Comp. Sci. 1997.

[52] Pavlovic, Mirjana & Bastian, Kai-Niklas & Gildhoff, Hinnerk & Ailamaki, Anastasia. (2017). Dictionary Compression in Point Cloud Data Management. 10.1145/3139958.3139969.

[53] Peano, G. (1890). Sur une courbe, qui remplit toute une aire plane. Mathematische Annalen, 36(1):157–160.

[54] Point Cloud, https://github.com/CesiumGS/3d-tiles/blob/master/specification/TileFormats/PointCloud/README.md (poslednji pristup 18.02.2021)

[55] Potree, https://github.com/potree/potree (poslednji pristup 17.02.2021)

[56] Psomadaki, S., Using a Space Filling Curve for the Management of Dynamic Point Cloud Data in a Relational DBMS, master thesis, Faculty of Architecture and The Built Environment, TU Delft

[57] Radoev, M. (2017). “A Comparison between Characteristics of NoSQL Databases and Traditional Databases”. Computer Science And Information Technology, 5(5), 149-153. doi: 10.13189/csit.2017.050501

[58] Ramsey P., "LIDAR in PostgreSQL with PointCloud", FOSS4G, September 2013, Nottingham

[59] Random Forest, https://en.wikipedia.org/wiki/Random_forest, (poslednji pristup 12.11.2020)

[60] Ravada S., Horhammer M., Kazar B. M., "Point Cloud: Storage, Loading, and Visualization", National Science Foundation TeraGrid Workshop on Cyber-GIS, February 2010, Washington

[61] Schon B., Mosa A.S.M., Laefer D.F., Bertolotto M., “Octree-based indexing for 3D pointclouds within an Oracle Spatial DBMS”, Computers & Geosciences 51 (2013) 430-438

[62] Singh, Satendra & Sreevalsan-Nair, Jaya. (2020). A Distributed System for Multiscale Feature Extraction and Semantic Classification of Large-Scale Lidar Point Clouds. 74-77. 10.1109/InGARSS48198.2020.9358938.

[63] Spark SQL: Catalyst Optimizer, https://blog.bi-geek.com/en/spark-sql-optimizador-catalyst/ (poslednji pristup 03.05.2021)

[64] T. Kim, K. Kim, J. Lee, A. Matono and K. Li, "Efficient Encoding and Decoding Extended Geocodes for Massive Point Cloud Data," in 2019 IEEE International Conference on Big Data and Smart Computing (BigComp), Kyoto, Japan, 2019 pp. 1-8. doi: 10.1109/BIGCOMP.2019.8679177

[65] The American Society for Photogrammetry & Remote Sensing, LAS Specification 1.4 - R15, http://www.asprs.org/wp-content/uploads/2019/07/LAS_1_4_r15.pdf (poslednji pristup 04.02.2020)

[66] The industry standard softwarefor point cloud and image processing, https://terrasolid.com/, (poslednji pristup 23.04.2021)

[67] Trimble SiteVision, https://fieldtech.trimble.com/en/products/mixed-reality-visualization/trimble-sitevision?utm_medium=cpc&utm_source=google&utm_campaign=FTG-2012-OA-Brand-Search-RDI&utm_content=homepage&utm_term=&utm_tstvar=&utm_spec=&gclid=Cj0KCQjwp86EBhD7ARIsAFkgakgOjDByV_uREbEufhVxBiDwlYV10DIaTg1Y4XH3_QYesmvyVjFl8qAaAjqgEALw_wcB (poslednji pristup 10.01.2021)

[68] V. Pajic, M. Govedarica, and M. Amovic, Model of pointcloud data management system in big data paradigm, ISPRSInternational Journal of Geo-Information, vol. 7, p. 265, 2018

[69] V. Pajic, M. Govedarica, Z. Galic, I. Alargic, OBRADA OBLAKA TAČAKA NA APACHE SPARK PLATFORMI POINT CLOUD PROCESSING ON APACHE SPARK, YU INFO 2015 ZBORNIK RADOVA, Beograd, 2015. ISBN: 978-86-85525-15-5

[70] van Oosterom, P.; Martinez-Rubi, O.; Ivanova, M.; Horhammer, M.; Geringer, D.; Ravada, S.; Tijssen, T.; Kodde, M.; Gonalves, R. Massive point cloud data management: Design, implementation and execution of a point cloud benchmark. Comput. Graph. 2015, 49, 92–125.

[71] Vo, A. V., Laefer, D. F., Trifkovic, M., Hewage, C. N. L., Bertolotto, M., Le-Khac, N. A., Ofterdinger, U., Highly Scalable Data Management System for Point Cloud and Full Waveform LIDAR Data, ISPRS International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLIII-B4-2020, 2020, pp.507-512

[72] Vo, Anh-Vu & Konda, Nikita & Chauhan, Neel & Aljumaily, Harith & Laefer, Debra. (2018). Lessons Learned with Laser Scanning Point Cloud Management in Hadoop HBase. 10.1007/978-3-319-91635-4_13.

[73] Vo, Anh-Vu & Truong-Hong, Linh & Laefer, Debra & Bertolotto, Michela. (2015). Octree-based region growing for point cloud segmentation. ISPRS Journal of Photogrammetry and Remote Sensing. 104. 10.1016/j.isprsjprs.2015.01.011.

[74] WebGL, https://en.wikipedia.org/wiki/WebGL (poslednji pristup 10.01.2021)

[75] White, T., Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale (2. ed.). O'Reilly 2011, ISBN 978-1-449-38973-4, pp. I-XXII, 1-600

[76] Yoon, Sanghyun & Ju, Sungha & Park, Sangyoon & Heo, Joon. (2019). A Framework Development for Mapping and Detecting Changes in Repeatedly Collected Massive Point Clouds. 10.22260/ISARC2019/0080.

[77] Z-order curve, https://en.wikipedia.org/wiki/Z-order_curve, (poslednji pristup 27.10.2019)

Prilog 1 – Implementacija u Scala programskom jeziku

Paket `point_cloud`

```
package point_cloud

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.types.StringType
import org.apache.spark.sql.Row
import org.apache.spark.sql.functions._
import org.apache.spark.rdd.PairRDDFunctions
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types.{StructType,StructField,LongType,ArrayType};
import java.io._

import point_cloud.schema._
import point_cloud.sfc._
import point_cloud.udf.CollectPointsInArray

object PointCloud {

  def load(path: String): DataFrame = {

    val points = SparkUtils.spark.textFile(path).map(_.split(" ")).map {fields => {
        val coordinates = Coordinates(fields(0).trim().toDouble,
                                fields(1).trim().toDouble,
                                fields(2).trim().toDouble)

        val normalizedCoordinates = NormalizedCoordinates(0, 0, 0)

        Point(coordinates,
              normalizedCoordinates,
              0,
              0,
              fields(3).trim().toInt,
              0,
              fields(4).trim().toShort,
```

```
            fields(5).trim().toShort,
            fields(6).trim().toShort)
      }
  }

  import SparkUtils.sql.implicits._
  points.toDF()

}

def findExtent(points: DataFrame): BoundingBox = {
  val stats: Row = points.agg(min(points.col("coordinates.x")),
                              min(points.col("coordinates.y")),
                              min(points.col("coordinates.z")),
                              max(points.col("coordinates.x")),
                              max(points.col("coordinates.y")),
                              max(points.col("coordinates.z")))
                         .collect().apply(0)

  val minCoords = Coordinates(stats.getDouble(0), stats.getDouble(1), stats.getDouble(2))
  val maxCoords = Coordinates(stats.getDouble(3), stats.getDouble(4), stats.getDouble(5))

  BoundingBox(minCoords,
              maxCoords,
              NormalizedCoordinates(0, 0, 0),
              NormalizedCoordinates(0, 0, 0))

}

def normalizeBoundingBox(bbox: BoundingBox, precision: Double): BoundingBox = {
  val f = (1 / precision).toInt
  BoundingBox(bbox.min,
              bbox.max,
              bbox.normalizedMin, //it is already 0,0,0
              NormalizedCoordinates((bbox.max.x - bbox.min.x).toInt * f,
                                    (bbox.max.y - bbox.min.y).toInt * f,
                                    (bbox.max.z - bbox.min.z).toInt * f)
              )
}
def normalizeCoordinates(points: DataFrame,
```

```
                            bbox: BoundingBox,
                            precision: Double,
                            resolution: Int): DataFrame = {

    import SparkUtils.sql.implicits._

    var pointsBeforeNormalization = points.select("coordinates.x", "coordinates.y", "coordinates.z",
        "mortonCode", "blockMortonCode",
        "intensity", "classification",
        "red", "green", "blue")


    pointsBeforeNormalization = pointsBeforeNormalization.withColumn("precision", lit((1 /
precision).toInt))
                                                         .withColumn("minX", lit(bbox.min.x))
                                                         .withColumn("minY", lit(bbox.min.y))
                                                         .withColumn("minZ", lit(bbox.min.z))

    println("Precision: %d".format((1 / precision).toInt))
    println("Min coords: %f, %f, %f".format(bbox.min.x, bbox.min.y, bbox.min.z))

    var pointsAfterNormalization = pointsBeforeNormalization.selectExpr("x",
                                                                        "y",
                                                                        "z",
                                                                        "round((x - minX) *
precision) AS normX",
                                                                        "round((y - minY) *
precision) AS normY",
                                                                        "round((z - minZ) *
precision) AS normZ",
                                                                        "mortonCode",
                                                                        "blockMortonCode",
                                                                        "intensity",
                                                                        "classification",
                                                                        "red",
                                                                        "green",
                                                                        "blue")


    pointsAfterNormalization = pointsAfterNormalization.withColumn("resolution", lit(resolution *
3))
```

```
    SparkUtils.sql.udf.register("calculateMortonCode", (x: Double, y: Double, z: Double) =>
Z3.apply(x.toInt, y.toInt, z.toInt).z)


    var pointsWithMortonCodes = pointsAfterNormalization.selectExpr("x",
                                                                    "y",
                                                                    "z",
                                                                    "normX",
                                                                    "normY",
                                                                    "normZ",
                                                                    "calculateMortonCode(normX,
normY, normZ) AS mortonCode",
                                                                    "blockMortonCode",
                                                                    "intensity",
                                                                    "classification",
                                                                    "red",
                                                                    "green",
                                                                    "blue",
                                                                    "resolution")
                                                           .orderBy("mortonCode")


        pointsWithBlockMortonCodes


  }
}
```

Paket schema

```
package schema


case class Coordinates(x: Double, y: Double, z: Double) {}


case class NormalizedCoordinates(var x: Int, var y: Int, var z: Int) {}


case class BoundingBox(min: Coordinates,
                       max: Coordinates,
                       var normalizedMin: NormalizedCoordinates,
                       var normalizedMax: NormalizedCoordinates) extends Serializable {


}


private[schema] case class Offsets(var x: Double, var y: Double, var z: Double);
```

```
private[schema] case class ScaleFactors(var x: Double, var y: Double, var z: Double);

case class Metadata(name: String,
                    /*
                     * Number of bits to shift point morton code in order to produce block morton
code.
                     * eg. if we have mm precision of point coordinates (0.001m) then with
resolution of 3
                     * one block will be cube with side dimension of 0.008m. (4 => 0.016m, 5 =>
0.032m).
                     * With resolution 0 size of block will be equal to precision of coordinates.
                     */
                    resolution: Int,
                    extent: BoundingBox,
                    offsets: Offsets,
                    scaleFactors: ScaleFactors) {

}

case class Point (coordinates: Coordinates,
                  var normalizedCoordinates: NormalizedCoordinates,
                  var mortonCode: Long,
                  intensity: Int,
                  classification: Byte,
                  red: Short,
                  green: Short,
                  blue: Short)
{}

case class PointCloud(var id: Int,
                      var name: String,
                      extent: BoundingBox,
                                      point: Array[Point])
{}

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SQLContext

object SparkUtils {
```

```
        val conf = new SparkConf().setAppName("SparkPointCloud").setMaster("local")


        val spark = new SparkContext(conf)


        val sql = new SQLContext(spark)

}
```

Paket udf

```
package udf

import org.apache.spark.sql.expressions.UserDefinedFunction

import org.apache.spark.sql.functions.udf

import org.locationtech.sfcurve.zorder.Z3


  val normalize: UserDefinedFunction = udf((value: Double, offset: Double, resolution: Double) => {

    ((value - offset) * 0.01).toInt

  })


  val calculateMortonCode:UserDefinedFunction = udf((x: Int, y: Int, z: Int) => {

    val mortonCode = Z3.apply(x, y, z)

    mortonCode.z

  })
```

Овај Образац чини саставни део докторске дисертације, односно докторског уметничког пројекта који се брани на Универзитету у Новом Саду. Попуњен Образац укоричити иза текста докторске дисертације, односно докторског уметничког пројекта.

План третмана података

Назив пројекта/истраживања
Модел управљања великим серијама геопросторних података
Назив институције/институција у оквиру којих се спроводи истраживање
а) Универзитет у Новом Саду **б) Факултет техничких наука** **в) Департман за рачунарство и аутоматику**
Назив програма у оквиру ког се реализује истраживање
1. Опис података
1.1 Врста студије *Укратко описати тип студије у оквиру које се подаци прикупљају* **Докторска дисертација** _____ _____ 1.2 Врсте података **а) квантитативни** **б) квалитативни** 1.3. Начин прикупљања података а) анкете, упитници, тестови б) клиничке процене, медицински записи, електронски здравствени записи

в) генотипови: навести врсту _____

г) административни подаци: навести врсту _____

д) узорци ткива: навести врсту_____

ђ) снимци, фотографије: ___**авионско и терестричко ласерско скенирање**_____

е) текст, навести врсту _____

ж) мапа, навести врсту _____

з) остало: описати _____**рачунарски експерименти**_____

1.3 Формат података, употребљене скале, количина података

1.3.1 Употребљени софтвер и формат датотеке:

а) Excel фајл, датотека _____

b) SPSS фајл, датотека _____

c) PDF фајл, датотека _____

d) Текст фајл, датотека _____

e) JPG фајл, датотека _____

f) Остало, датотека __**.csv, .xyz, .txt, .las, .laz**_____

1.3.2. Број записа (код квантитативних података)

а) број варијабли _____**Велики број**_____

б) број мерења (испитаника, процена, снимака и сл.) _____**Велики број**_____

1.3.3. Поновљена мерења

а) да

б) не

Уколико је одговор да, одговорити на следећа питања:

а) временски размак између поновљених мера је _____**1 сат**_____

б) варијабле које се више пута мере односе се на _____**време обраде података**_____

в) нове верзије фајлова који садрже поновљена мерења су именоване као _____

Напомене: _____

Да ли формати и софтвер омогућавају дељење и дугорочну валидност података?

а) Да

б) Не

Ако је одговор не, образложити _____

_____

2. Прикупљање података

2.1 Методологија за прикупљање/генерисање података

2.1.1. У оквиру ког истраживачког нацрта су подаци прикупљени?

а) експеримент, навести тип **Рачунарски експеримент обраде података типа облак тачака**

б) корелационо истраживање, навести тип _____

ц) анализа текста, навести тип ___**Анализа доступне литературе**_____

д) остало, навести шта _____

2.1.2 Навести врсте мерних инструмената или стандарде података специфичних за одређену научну дисциплину (ако постоје).

_____

_____

2.2 Квалитет података и стандарди

2.2.1. Третман недостајућих података

а) Да ли матрица садржи недостајуће податке? Да **Не**

Ако је одговор да, одговорити на следећа питања:

а) Колики је број недостајућих података? _____

б) Да ли се кориснику матрице препоручује замена недостајућих података? Да Не

в) Ако је одговор да, навести сугестије за третман замене недостајућих података

_____

2.2.2. На који начин је контролисан квалитет података? Описати

___**Квалитет података типа облак тачака је контролисан поређењем са контролним мерењима и визуалном инспекцијом**_____

_____

2.2.3. На који начин је извршена контрола уноса података у матрицу?

_**Контрола уноса података изведена је на бази експертског знања** __

_____

3. Третман података и пратећа документација

3.1. Третман и чување података

3.1.1. Подаци ће бити депоновани у **Репозиторијуму докторских дисертација на Универзитету у Новом Саду.**

3.1.2. URL адреса __*https://www.cris.uns.ac.rs/searchDissertations.jsf*_____

3.1.3. DOI _____

3.1.4. Да ли ће подаци бити у отвореном приступу?

а) ***Да***

б) *Да, али после ембарга који ће трајати до* _____

в) *Не*

Ако је одговор не, навести разлог _____

3.1.5. Подаци неће бити депоновани у репозиторијум, али ће бити чувани.

Образложење

_____

_____

3.2 Метаподаци и документација података

3.2.1. Који стандард за метаподатке ће бити примењен? **Стандард који примењује Репозиторијум докторских дисертација Универзитета у Новом Саду**

3.2.1. Навести метаподатке на основу којих су подаци депоновани у репозиторијум.

_____

_____

Ако је потребно, навести методе које се користе за преузимање података, аналитичке и процедуралне информације, њихово кодирање, детаљне описе варијабли, записа итд.

_____

_____

_____

_____

3.3 Стратегија и стандарди за чување података

3.3.1. До ког периода ће подаци бити чувани у репозиторијуму? _____

3.3.2. Да ли ће подаци бити депоновани под шифром? **Да** Не

3.3.3. Да ли ће шифра бити доступна одређеном кругу истраживача? **Да** Не

3.3.4. Да ли се подаци морају уклонити из отвореног приступа после извесног времена?

Да **Не**

Образложити

_____

_____

4. Безбедност података и заштита поверљивих информација

Овај одељак МОРА бити попуњен ако ваши подаци укључују личне податке који се односе на учеснике у истраживању. За друга истраживања треба такође размотрити заштиту и сигурност података.

4.1 Формални стандарди за сигурност информација/података

Истраживачи који спроводе испитивања с људима морају да се придржавају Закона о заштити података о личности *(https://www.paragraf.rs/propisi/zakon_o_zastiti_podataka_o_licnosti.html)* и

одговарајућег институционалног кодекса о академском интегритету.

4.1.2. Да ли је истраживање одобрено од стране етичке комисије? Да **Не**

Ако је одговор Да, навести датум и назив етичке комисије која је одобрила истраживање

_____

4.1.2. Да ли подаци укључују личне податке учесника у истраживању? Да **Не**

Ако је одговор да, наведите на који начин сте осигурали поверљивост и сигурност информација везаних за испитанике:

а) Подаци нису у отвореном приступу

б) Подаци су анонимизирани

ц) Остало, навести шта

_____

_____

5. Доступност података

5.1. Подаци ће бити

а) јавно доступни

б) доступни само уском кругу истраживача у одређеној научној области

ц) затворени

Ако су подаци доступни само уском кругу истраживача, навести под којим условима могу да их користе:

_____

_____

Ако су подаци доступни само уском кругу истраживача, навести на који начин могу приступити подацима:

_____

_____

5.4. Навести лиценцу под којом ће прикупљени подаци бити архивирани.

Ауторство - некомерцијално _____

6. Улоге и одговорност

6.1. Навести име и презиме и мејл адресу власника (аутора) података

___**Владимир Пајић, pajicv@uns.ac.rs** _____

6.2. Навести име и презиме и мејл адресу особе која одржава матрицу с подацима

___**Владимир Пајић, pajicv@uns.ac.rs**_____

6.3. Навести име и презиме и мејл адресу особе која омогућује приступ подацима другим истраживачима

____**Владимир Пајић, pajicv@uns.ac.rs**_____