Fizički fakultet Univerziteta u Beogradu

RAZVOJ ALATA ZA OBRADE HIJERARHIJSKI ORGANIZOVANIH KOLEKCIJA TABELARNIH PODATAKA

Mentor: prof. dr Zoran Nikolić

Student: Nikola Tešanović

Broj indeksa: 7002/2021

Beograd 2023.

Sadržaj:

1. Uvod

U današnjem digitalnom dobu, prikupljanje, obrada i skladištenje velikih količina podataka postalo je uobičajena praksa u mnogim industrijama kao i u nauci. Međutim, s rastom količine podataka, dolazi i do rasta potrebe za efikasnim načinima skladištenja i upravljanja tim podacima. Jedna od ključnih odluka koju treba doneti u procesu skladištenja podataka je izbor formata za skladištenje. Postoje mnogi formati dostupni danas, a svaki od njih ima svoje prednosti i nedostatke koji ih čine prikladnim ili manje prikladnim za određenu namenu.

U ovom radu težište istraživanja zasnovana je na dva konkretna zadatka:

1. Benčmarkovanje različitih tipova fajlova po veličini i brzini pisanja.
2. Razvoj alata sa kojim će se omogućiti lakši rad sa fajlovima tipa h5.

U prvom delu opisan proces benčmarkovanja različitih tipova fajlova korišćeni za čuvanje tabelarnih podataka. Posmatrana su četiri različita tipa fajlova: json, xml, parquet i h5. Razlog zašto su izabrani ovi tipovi jeste zato što su oni pogodni za rad sa tabelarnim tipovima podataka, međutim treba da se vidi njihova efikasnost pri skladištenju tih podataka i brzina kojom se oni pišu u fajl.

Cilj prvog zadatka jeste da se vidi efikasnost svakog od navedenih formata po pitanju skladištenja i brzine pisanja, i da se time odredi idealan kandidat za skladištenje velikih količina podataka u budućnosti prema tim kriterijumima.

Drugi deo rada se sastojao se iz kreiranja alatki za fajlove tipa h5 u okruženju R programskog jezika i u unapređenju već postojećih alatki.

Potreba za razvoj alata je došla iz jednog konkretnog problema korišćenja h5 formata na realne podatke. Taj problem jeste skladištenje podataka dobijeni satelitski snimcima noćnog neba od strane NASA-e koji su dostupni na sajtu LAADS DAAC (link je ispod odeljka za literaturu). Problematika se sastojala od toga da su atmosferski snimci noćnog neba sveta podeljeni u kvadrante, i ti kvadratni nisu određeni po državnih granicama, već po nekim drugim parametrima. Željeni podaci su konkretno bili podaci za državu Republiku Srbiju, koji su nažalost bili sačuvani u dva različita fajla čija veličina je velika. U tom cilju je napravljen alat da se od ta dva fajla, kreira jedan fajl koji će samo da sadrži podatke za Republiku Srbiju. Tim postupkom, nerelevantni podaci se filtriraju iz novog fajla, i cena prostora skladištenja se znatno smanjuje. Taj proces je sproveden na jednoj godini podataka (ukupno 24 fajla), pri čemu je broj fajlova smanjen na 12 (po jedan fajl za svaki mesec umesto prethodno dva). Zatim, kreirana je još jedna alatka (funkcija), čiji zadatak je bio da spoji tih 12 kreiranih fajlova u jedan kompaktan h5 fajl koji sadrži informacije za godinu dana podeljeni u grupama po mesecima.

2. Programski jezik R

U ovom radu je kao alatka za obradu podataka, korišćen programski jezik R. Jedan od glavnih razloga korišćenja R-a u odnosu npr. Python-a ili C++, je lakoća pri transformaciji podaka. Brzina nije bila glavni prioritet u ovom radu, međutim dobijeni kod je vektorizovan (na svim mestima gde je to imalo smisla), brzina izlaznih funkcija je zadovoljavajuća.

2.1 Uvod u R

R je jezik i okruženje za statističko izračunavanje i grafičko modeliranje. To je GNU (operativni sistem sastavljen od isključivo open source softvera) projekat koji je sličan S jeziku i okruženju koje su razvili u Bell Laboratories (ranije AT&T, sada Lucent Tehnologije) od strane Džona Čejmbersa i njegovih kolega. R se može smatrati različitom implementacijom S-a. Postoje neke važne razlike, ali mnogo koda napisanog za S radi nepromenjeno pod R-om.

R pruža širok spektar statističkih (linearno i nelinearno modeliranje, klasični statistički testovi, analiza vremenskih serija, klasifikacija, grupisanje,...) i grafičkih tehnika i veoma je proširiv. Jezik S je često sredstvo izbora za istraživanje u statističkoj metodologiji, a R pruža otvoreni izvorni put do učešća u toj aktivnosti.

Jedna od prednosti R-a je lakoća sa kojom se mogu proizvesti dobro osmišljeni grafikoni kvaliteta vrednim publikacijama, uključujući matematičke simbole i formule tamo gde je to potrebno. Velika pažnja je stavljena na dizajniranje i vizualizaciju podataka gde pri tom korisnik zadržava potpunu kontrolu o procesu (može da menja svaki parametar po želji).

R je dostupan kao slobodan softver pod uslovima GNU opšte javne licence Fondacije za slobodni softver u obliku izvornog koda. On kompajlira i radi na velikom broju UNIX platformi i sličnih sistema (uključujući FreeBSD i Linux), Windows i MacOS.

R je prvenstveno napisan u C, Fortranu i samom R-u (delimično samo-hostovanje). Unapred kompajlirani izvršni programi su obezbeđeni za različite operativne sisteme. R ima interfejs komandne linije. Takođe je dostupno više grafičkih korisničkih interfejsa nezavisnih proizvođača, kao što su RStudio, integrisano razvojno okruženje i Jupiter, interfejs za notebook računar.

The Comprehensive R Archive Network (CRAN) je mreža servera koji čuvaju i distribuiraju softver otvorenog koda za statistički programski jezik R. Napravljen je da obezbedi centralnu lokaciju za korisnike da pronađu, preuzmu i instaliraju R pakete, koji su kolekcije R funkcija, podataka i dokumentacije koja proširuje mogućnosti programskog jezika R.

CRAN mreža se sastoji od nekoliko ogledala sajtova širom sveta na kojima se nalaze identične kopije istih R paketa. Ova redundantnost osigurava da korisnici uvek mogu da pristupe softveru koji im je potreban, čak i ako se jedna lokacija za ogledalo pokvari. CRAN paketima doprinosi

raznolika zajednica programera i statističara i podležu strogim merama kontrole kvaliteta kako bi se osiguralo da su dobro dokumentovani, pouzdani i kompatibilni sa trenutnim verzijama R.

2.2 R okruženje

R je integrisani paket softverskih sredstava za manipulaciju sa podacima, proračun i njihov grafički prikaz. Sadrži:

- Efikasan objekat za rukovanje i skladištenje podataka
- Paket operatora za proračune nizova, posebno matrica
- Veliku, koherentnu, integrisanu kolekciju srednjih alata za analizu podataka
- Grafička sredstva za analizu i prikaz podataka na ekranu ili na štampanoj kopiji, i
- Dobro razvijen, jednostavan i efikasan programski jezik koji uključuje uslove, petlje, korisnički definisane rekurzivne funkcije i ulazne i izlazne mogućnosti.

Termin „okruženje“ ima za cilj da ga okarakteriše kao potpuno planiran i koherentan sistem, a ne kao postepeno povećanje veoma specifičnih i nefleksibilnih alata, kao što je čest slučaj sa drugim softverom za analizu podataka.

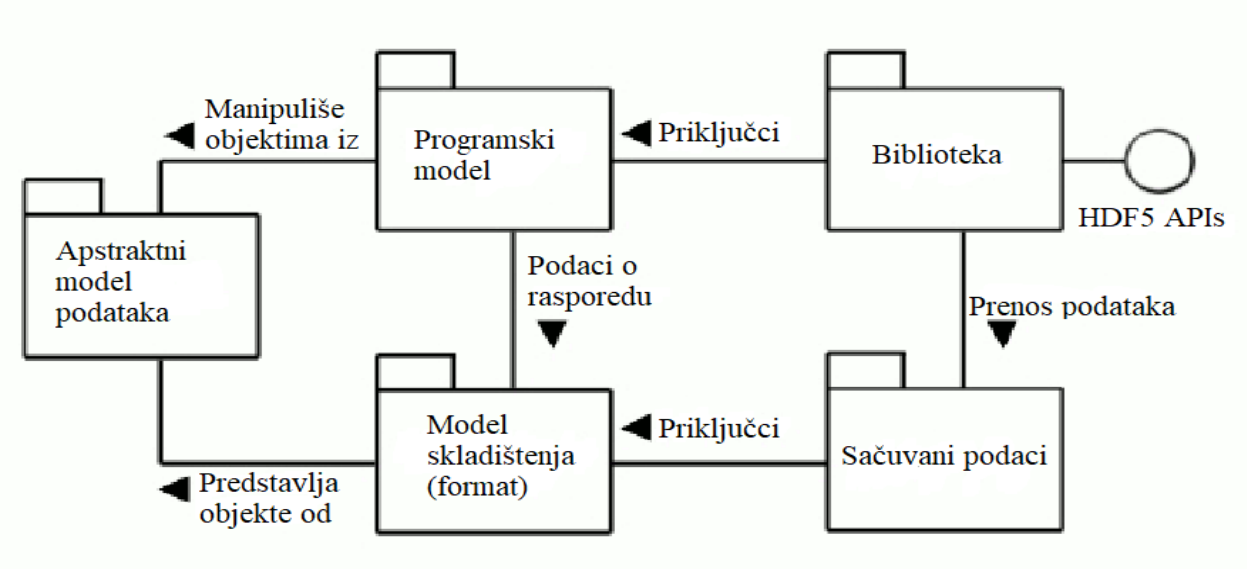
R, kao i S, je dizajniran oko pravog računarskog jezika i omogućava korisnicima da dodaju funkcionalnost definisanjem novih funkcija. Veći deo sistema je napisan na R dijalektu S-a, što korisnicima olakšava da prate algoritamske izbore. Za računarski intenzivne zadatke, C, C++ i Fortran kod se mogu povezati i pozvati tokom izvršavanja. Postoji i mogućnost da se kod direktno napiše u C kodu, i on da direktno manipuliše R objektima. Postoji i mogućnost korišćenja Python koda u R-u, time se mogu koristiti biblioteke napisane u Pythonu, učitaju se u Python kodu, pa zatim interpretiraju na R objektima.

O R-u se može razmišljati kao o okruženju u kojem se sprovode statističke tehnike. R se može (lako) proširiti preko paketa. Postoji oko osam paketa koji se isporučuju sa R distribucijom, a mnogi drugi su dostupni preko CRAN porodice internet sajtova koji pokrivaju veoma širok spektar modernih alatki. Taj spektar ide od obrade podataka pa do pravljenja aplikacija sa svim backend i frontend delovima u R-u.

3. HDF5 model podataka i struktura fajla

3.1 Uvod u HDF5

Hijerarhijski format podataka (HDF) implementira model za upravljanje i skladištenje podataka. Model uključuje apstraktni model podataka i apstraktni model skladištenja (format podataka), biblioteke za implementaciju apstraktnog modela i mapiranje modela skladištenja u različite mehanizme skladištenja. HDF5 biblioteka pruža programski interfejs za konkretnu implementaciju apstraktnih modela. Biblioteka takođe implementira model prenosa podataka, efikasnog kretanja podataka iz jedne uskladištene reprezentacije u drugu sačuvanu reprezentaciju. Na slici 3.1 ilustrovanu su odnosi između modela i implementacija.



Slika 3.1 HDF5 modeli i implementacije

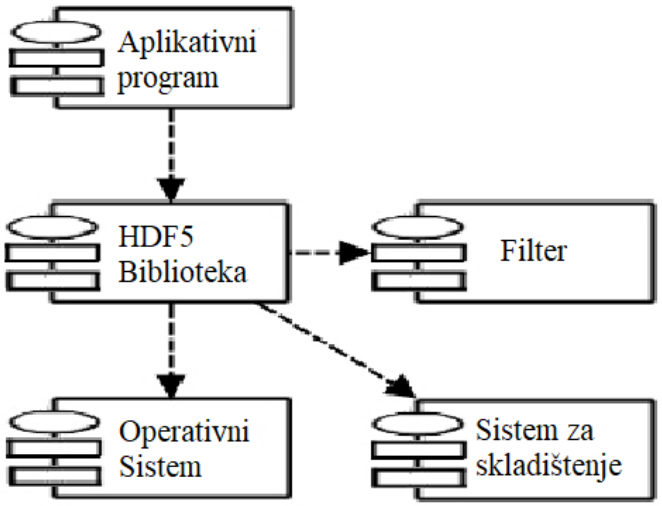
Apstraktni model podataka je konceptualni model podataka, tipova podataka i organizacije podataka. Apstraktni model podataka je nezavisan od medijuma za skladištenje ili programskog okruženja. Model skladištenja je standardni prikaz za objekte apstraktnog modela podataka. Specifikacija formata fajla HDF5 definiše model skladištenja.

Programski model je model računarskog okruženja i uključuje platforme od malih pojedinačnih sistema do velikih višeprocesorski sistema i klastera. Model programiranja manipuliše (instancira, popunjava i preuzima) objekte iz apstraktnog modela podataka.

Biblioteka je konkretna implementacija modela programiranja. Biblioteka izvozi HDF5 API-je kao svoj interfejs. Pored implementacije objekata apstraktnog modela podataka, biblioteka upravlja prenosom podataka iz jednog sačuvanog oblika u drugi. Primeri prenosa podataka uključuju čitanje sa diska na memoriju i pisanje sa memorije na disk.

Sačuvani podaci su konkretna implementacija modela skladištenja. Model skladištenja je mapiran na nekoliko mehanizama za skladištenje uključujući fajlove na jednom disku, više fajlova (familija fajlova) i memorijske reprezentacije.

HDF5 biblioteka je C modul koji implementira model programiranja i apstraktni model podataka. Biblioteka HDF5 poziva operativni sistem ili drugi softver za upravljanje skladištem (na primer, MPI/IO biblioteku) za skladištenje i preuzimanje trajnih podataka. HDF5 biblioteka se takođe može povezati sa drugim softverom kao što su filteri za kompresiju. HDF5 biblioteka je povezana sa aplikacijskim programom koji može biti napisan u C, C++, Fortranu ili Javi. Aplikacioni program implementira algoritme i strukture podataka specifične za probleme i poziva HDF5 biblioteku za skladištenje i preuzimanje podataka. Na slici 3.2 prikazane su zavisnosti ovih modula.



Slika 3.2 Biblioteka, aplikativni program i drugi moduli

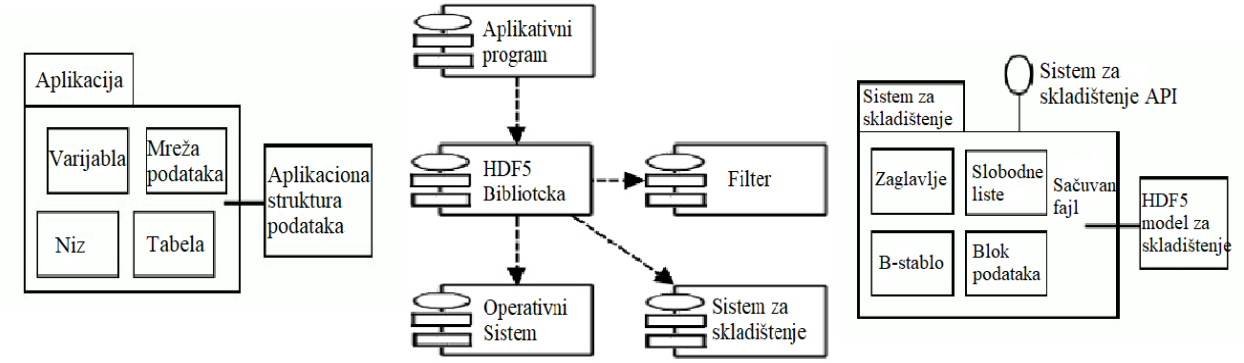
Svaka od softverskih komponenti upravlja podacima koristeći modele i strukture podataka koji su prikladni za komponentu. Kada se podaci prosleđuju između slojeva (tokom skladištenja ili preuzimanja), oni se transformišu iz jedne reprezentacije u drugu. Na slici 3.3 prikazane su neke od vrsta struktura podataka koje se koriste u različitim slojevima.

Aplikatiovni program koristi strukture podataka koje predstavljaju problem i algoritme uključujući promenljive, tabele, nizove i mreže između ostalih struktura podataka.

U zavisnosti od svog dizajna i funkcije, aplikacija može imati dosta različitih vrsta struktura podataka i različite brojeve i veličine objekata.

HDF5 biblioteka implementira objekte HDF5 apstraktnog modela podataka. Neki od ovih objekata uključuju grupe, skupove podataka i atribute. Aplikativni program mapira strukture podataka

aplikacije u hijerarhiju HDF5 objekata. Svaka aplikacija će kreirati mapu koja najbolje odgovara njenoj nameni.



Slika 3.3 Strukture podataka u različitim slojevima

Objekti HDF5 apstraktnog modela podataka se mapiraju u objekte HDF5 modela skladištenja i čuvaju u medijumu za skladištenje. Sačuvani objekti uključuju blokove zaglavlja, slobodne liste, blokove podataka, B-stabla i druge objekte. Svaka grupa ili skup podataka se čuva kao jedno ili više zaglavlja i blokova podataka.

HDF5 biblioteka takođe može da koristi druge biblioteke i module kao što je kompresija.

Ne postoji nužno jednostavna korespondencija između objekata aplikacijskog programa, apstraktnog modela podataka i onih iz specifikacije formata. Organizacija podataka aplikacijskog programa i način na koji se oni mapiraju u HDF5 apstraktni model podataka zavisi od programera aplikacije.

Većina aplikacija ne mora da razmatra nikakve detalje o specifikaciji formata fajla HDF5 ili detalje o tome kako se objekti apstraktnog modela podataka prevode u i iz skladišta.

3.2 Apstraktni model podataka

Apstraktni model podataka (ADM) definiše koncepte za definisanje i opisivanje složenih podataka uskladištenih u fajlovima. ADM je opšti model koji je dizajniran da konceptualno pokrije mnoge specifične modele. Mnogo različitih vrsta podataka može se mapirati na objekte ADM-a i stoga uskladištiti i preuzeti pomoću HDF5. ADM, međutim, nije model nekog određenog problema ili domena aplikacije. Korisnicima je prepusteno da mapiraju svoje podatke sa konceptima ADM-a.

Ključni koncepti uključuju:

- Fajl (File) - neprekidni niz bajtova u računarskoj prodavnici (memorija, disk, itd.), a bajtovi predstavljaju nula ili više objekata modela
- Grupa (Group) - kolekcija objekata (uključujući grupe)
- Skup podataka (Dataset) – višedimenzionalni niz elemenata podataka sa atributima i drugim metapodacima
- Prostor podataka (Dataspace) – opis dimenzija višedimenzionalnog niza
- Tip podataka (Datatype) – opis specifične klase elementa podataka uključujući njegov raspored skladištenja kao obrazac bitova
- Atribut (Attribute) – imenovana vrednost podataka povezana sa grupom, skupom podataka ili imenovanim tipom podataka
- Lista svojstava (Property List) - kolekcija parametara (neki trajni i neki prolazni) koji kontrolišu opcije u biblioteci
- Veza (Link) - način na koji su objekti povezani

- Fajl (File)

Apstraktno, HDF5 fajl je kontejner za organizovanu kolekciju objekata. Objekti su grupe, skupovi podataka i drugi objekti. Objekti su organizovani kao ukorenjeni, usmereni graf. Svaki HDF5 fajl ima najmanje jedan objekat, root (ukorenju) grupu.

HDF5 objekti imaju jedinstven identitet unutar jednog HDF5 fajla i mogu im se pristupiti samo po njihovim imenima unutar hijerarhije fajla. HDF5 objekti u različitim fajlovima nemaju nužno jedinstvene identitete i nije moguće pristupiti trajnom HDF5 objektu osim preko fajla.

Kada se fajl kreira, svojstva kreiranja fajla određuju podešavanja za fajl. Svojstva kreiranja fajla uključuju informacije o verziji i parametre globalnih struktura podataka. Kada se fajl otvori, svojstva pristupa fajlu određuju podešavanja za trenutni pristup. Svojstva pristupa fajlu uključuju parametre drajvera za skladištenje, parametre za keširanje i sakupljanje smeća. Svojstva kreiranja fajla su trajno podešena tokom trajanja fajla, a svojstva pristupa njemu se mogu promeniti zatvaranjem i ponovnim otvaranjem fajla.

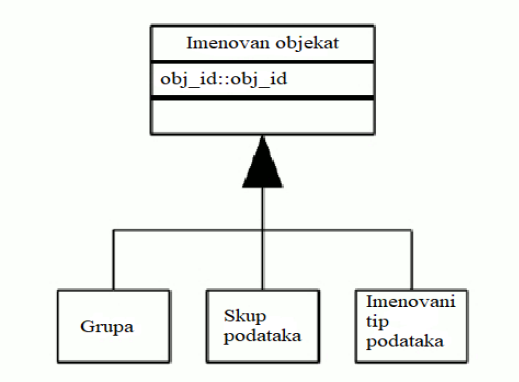
Fajl HDF5 se može „montirati“ kao deo drugog HDF5 fajla. Ovo je analogno montiranju Unix sistema fajlova. Koren montiranog fajla je pridružen grupi u fajlu za montiranje, a svom sadržaju se može pristupiti kao da je montirani fajl deo fajla za montiranje.

- Grupa (Group)

HDF5 grupa je analogna direktorijumu sistema fajlova. Apstraktno, grupa sadrži nula ili više objekata, a svaki objekat mora biti član najmanje jedne grupe. Root grupa je poseban slučaj; ne može biti član nijedne grupe.

Članstvo u grupi se zapravo implementira preko objekata veze. Objekat veze je u vlasništvu grupe i ukazuje na imenovani objekat. Svaka veza ima ime i svaka veza upućuje na tačno jedan objekat. Svaki imenovani objekat ima najmanje jednu, a moguće i mnogo veza do njega.

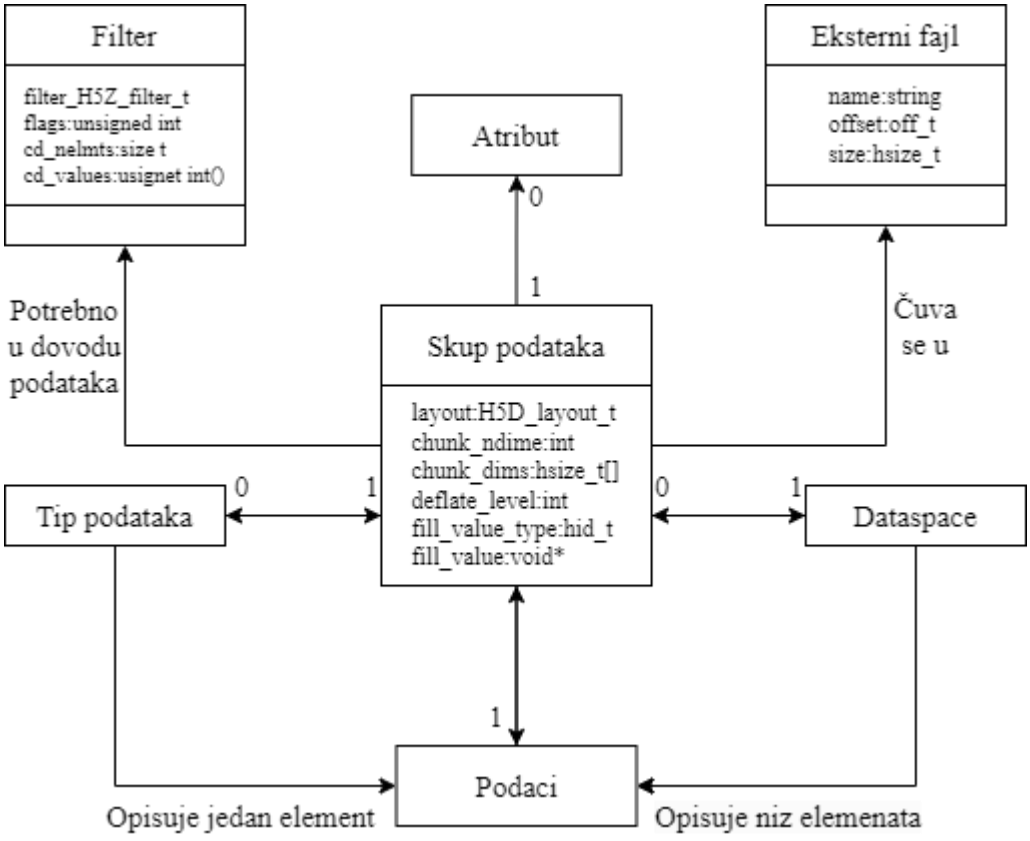
Postoje tri klase imenovanih objekata: grupa, skup podataka i urezani (imenovani) tip podataka. Na slici 3.4 može se videti ovaka struktura. Svaki od ovih objekata je član najmanje jedne grupe, a to znači da postoji bar jedna veza do njega.



Slika 3.4 Klase imenovanih objekata

- Skup podataka (Dataset)

HDF5 skup podataka je višedimenzionalni (pravougaoni) niz elemenata podataka. Prikazan je na slici 3.5. Oblik niza (broj dimenzija, veličina svake dimenzije) opisuje objekat prostora podataka.



Slika 3.5 Skup podataka

Element podataka je jedna jedinica podataka koja može biti broj, znak, niz brojeva ili znakova ili zapis heterogenih elemenata podataka. Element podataka je skup bitova.

Prostor podataka i tip podataka se postavljaju kada je skup podataka kreiran i ne mogu se menjati tokom trajanja skupa podataka. Svojstva kreiranja skupa podataka se postavljaju kada se skup podataka kreira. Svojstva kreiranja skupa podataka obuhvataju vrednost punjenja i svojstva skladištenja kao što su chunking i kompresija. Ova svojstva se ne mogu promeniti nakon kreiranja skupa podataka.

Objekat skupa podataka upravlja skladištenjem i pristupom podacima. Dok su podaci konceptualno neprekidni pravougaoni niz, oni se fizički čuvaju i prenose na različite načine u zavisnosti od svojstava skladištenja i korišćenog mehanizma skladištenja. Stvarno skladište može biti skup komprimovanih delova, a pristup može biti kroz različite mehanizme skladištenja i keš memorije. Skup podataka mapira između konceptualnog niza elemenata i stvarno sačuvanih podataka.

- Prostor podataka (Dataspace)

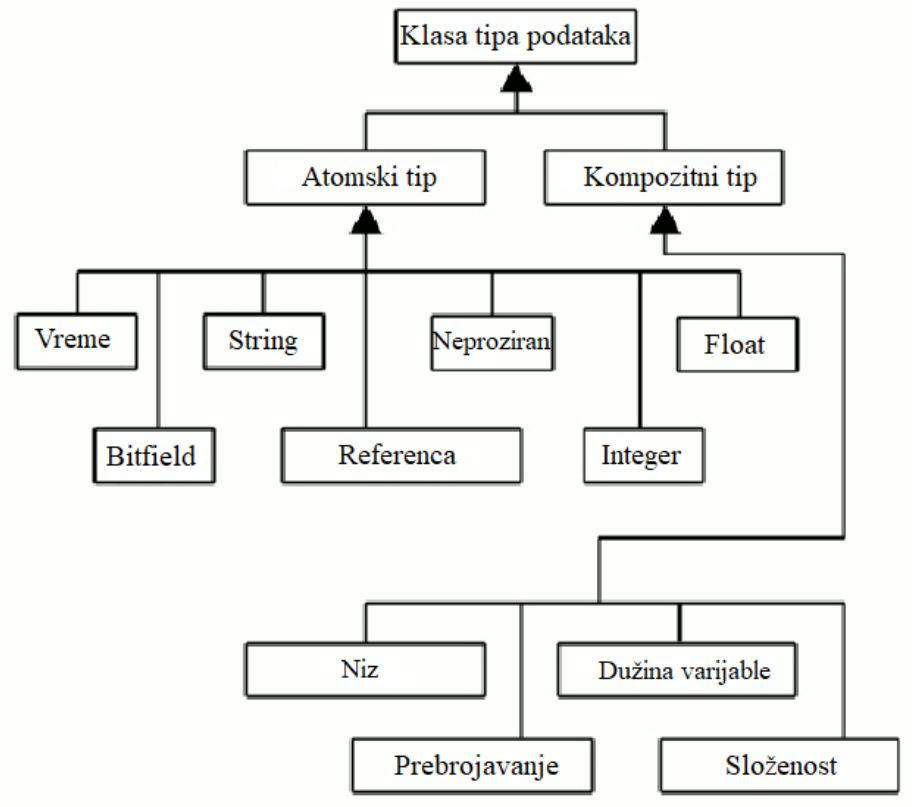
Prostor podataka HDF5 opisuje raspored elemenata višedimenzionalnog niza. Konceptualno, niz je hiperpravougaonik sa jednom do 32 dimenzije. Prostori podataka HDF5 mogu se proširiti. Svaka dimenzija ima trenutnu veličinu i maksimalnu veličinu, a maksimum može biti neograničen. Prostor podataka opisuje hiperpravougaonik: to je lista dimenzija sa trenutnom i maksimalnom (ili neograničenom) veličinom.

Objekti prostora podataka se takođe koriste za opisivanje selekcija hiperploče iz skupa podataka. Bilo koji podskup elemenata skupa podataka može se izabrati za čitanje ili pisanje navođenjem skupa hiperploča. Nepravougaona oblast se može izabrati unijom nekoliko (pravougaonih) prostora podataka.

- Tip podataka (Datatype)

HDF5 objekat tipa podataka opisuje izgled jednog elementa podataka. Element podataka je jedan element niza, to može biti jedan broj, znak, niz brojeva ili nosilaca ili drugi podaci. Objekat tipa podataka opisuje raspored skladištenja ovih podataka.

Tipovi podataka su kategorisani u 11 klasa tipa podataka. Svaka klasa se tumači u skladu sa skupom pravila i ima specifičan skup svojstava koja opisuju njeno skladištenje. Na primer, brojevi sa pokretnim zarezom imaju eksponentni položaj i veličine koje se tumače u skladu sa odgovarajućim standardima za predstavljanje brojeva. Dakle, klasa tipa podataka govori šta element znači, a tip podataka opisuje kako se čuva.



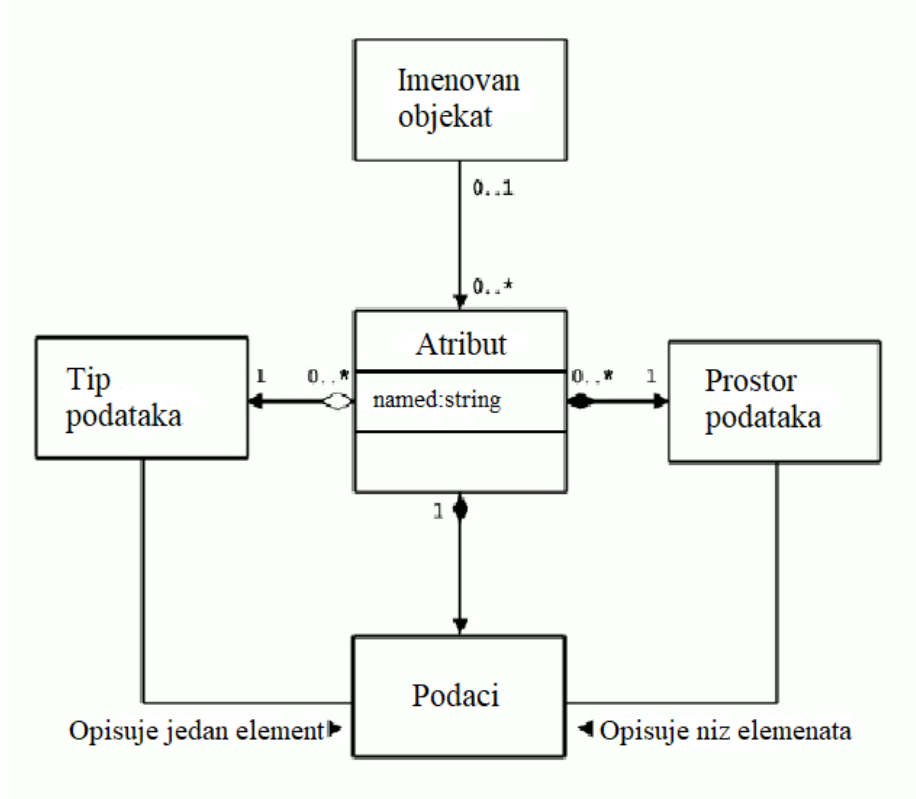
Slika 3.6: Klasifikacije tipova podataka

Na slici 3.6 prikazana je klasifikacija tipova podataka. Atomski tipovi podataka su nedeljivi. Svaki može biti jedan objekat kao što je broj ili niz. Kompozitni tipovi podataka se sastoje od više elemenata atomskih tipova podataka. Pored standardnih tipova, korisnici mogu definisati dodatne tipove podataka kao što su 24-bitni ceo broj ili 16-bitni float. Skup podataka ili atribut ima jedan objekat tipa podataka povezan sa njim, to se može videti na slici 3.7. Objekat tipa podataka može da se koristi u definiciji nekoliko objekata, ali će po podrazumevanoj vrednosti kopija objekta tipa podataka biti privatna za skup podataka.

Opciono, objekat tipa podataka može biti sačuvan u HDF5 fajlu. Tip podataka je povezan u grupu i stoga mu je dodeljeno ime. Urezani tip podataka (ranije nazvan imenovani tip podataka) može se otvoriti i koristiti na bilo koji način na koji se može koristiti objekat tipa podataka.

- Atribut (Attribute)

Bilo koji HDF5 imenovani objekat podataka (grupa, skup podataka ili imenovani tip podataka) može imati nula ili više korisnički definisanih atributa. Atributi se koriste za dokumentovanje objekta. Atributi objekta se čuvaju sa objektom.



Slika 3.7: Elementi podataka atributa

Atribut HDF5 ima ime i podatke. Deo podataka je sličan po strukturi skupu podataka: prostor podataka definiše izgled niza elemenata podataka, a tip podataka definiše raspored skladištenja i interpretaciju elemenata, može se videti na slici 3.7.

U stvari, atribut je veoma sličan skupu podataka sa sledećim ograničenjima:

- Atributu se može pristupiti samo preko objekta
- Imena atributa su značajna samo unutar objekta
- Atribut treba da bude mali objekat
- Podaci atributa moraju biti pročitani ili upisani u jednom pristupu (delimično čitanje ili pisanje nije dozvoljeno)
- Atributi nemaju atribute

Vrednost atributa može biti referenca objekta. Deljeni atribut ili atribut koji je veliki niz može se primeniti kao referenca na skup podataka.

Ime, prostor podataka i tip podataka atributa se navode kada se kreira i ne mogu se menjati tokom trajanja atributa. Atribut se može otvoriti po imenu, indeksu ili iteracijom kroz sve atribute objekta.

- Lista svojstava (Property List)

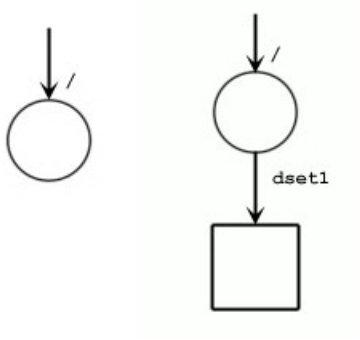
HDF5 ima generički objekat liste svojstava. Svaka lista je kolekcija parova ime-vrednost. Svaka klasa liste svojstava ima određeni skup svojstava. Svako svojstvo ima implicitno ime, tip podataka i vrednost. Objekt liste svojstava se kreira i koristi na način sličan drugim objektima HDF5 biblioteke.

Liste svojstava su priložene objektu u biblioteci i može ih koristiti bilo koji deo biblioteke. Neka svojstva su trajna (na primer, strategija razdvajanja podataka za skup podataka), druga su prolazna (na primer, veličine bafera za prenos podataka). Uobičajena upotreba liste svojstava je da se proslede parametri iz pozivajućeg programa VFL drajveru ili modulu cevovoda.

Liste svojstava su konceptualno slične atributima. Liste svojstava su informacije relevantne za ponašanje biblioteke, dok su atributi relevantni za podatke i aplikaciju korisnika.

3.3 Struktura HDF5 fajla

HDF5 fajla je organizovana kao ukorenjeni, usmereni graf. Imenovani objekti podataka su čvorovi grafa, a veze su usmereni lukovi. Svaki luk grafa ima ime, a grupa korena ima ime "/". Objekti se kreiraju i zatim ubacuju u grafik operacijom veze koja kreira imenovanu vezu od grupe do objekta. Na primer, slika ispod ilustruje strukturu HDF5 fajla kada se kreira jedan skup podataka. Objekat može biti cilj više od jedne veze. Imena na vezama moraju biti jedinstvena unutar svake grupe, ali može postojati mnogo veza sa istim imenom u različitim grupama. Imena veza su nedvosmislena: neki predak će imati drugačije ime, ili su isti objekat. Grafikon se kreće pomoću imena putanja sličnih Unix sistemima fajlova. Objekat se može otvoriti sa punom putanjom koja počinje od koren grupe ili sa relativnom putanjom i početnim čvorom (grupom). Imajte na umu da se sve putanje odnose na jedan HDF5 fajl. U tom smislu, HDF5 fajl je analogan jednom Unix sistemu fajlova.



Slika 3.8: HDF5 fajl sa jednim skupom podataka

Treba napomenuti da, baš kao i Unix sistem fajlova, HDF5 objekti nemaju imena. Imena su povezana sa putanjama. Objekt ima jedinstveni (unutar fajla) identifikator objekta, ali jedan objekat može imati mnogo imena jer može postojati mnogo putanja do istog objekta. Objekat se može preimenovati (premestiti u drugu grupu) dodavanjem i brisanjem veza. U ovom slučaju, sam objekat se nikada ne pomera. Što se toga tiče, članstvo u grupi nema implikacije na fizičku lokaciju uskladištenog objekta. Brisanje veze ka objektu ne znači nužno brisanje objekta. Objekat ostaje dostupan sve dok postoji bar jedna veza do njega. Nakon što se sve veze ka objektu izbrišu, on se više ne može otvoriti iako se skladište može, ali i ne mora vratiti.

- HDF5 imena putanja i navigacija

Struktura fajla čini prostor imena za objekte u fajlu. Ime putanje je niz komponenti razdvojenih sa '/'. Svaka komponenta je naziv veze ili specijalni znak "." za trenutnu grupu. Imena veza (komponente) mogu biti bilo koji niz ASCII znakova koji ne sadrži '/' (osim stringa "." koji je rezervisan). Međutim, korisnicima se savetuje da izbegavaju upotrebu znakova interpunkcije i znakova koji se ne štampaju jer mogu stvoriti probleme za drugi softver. Slika 3.13 daje BNF gramatiku za nazive putanja HDF5:

```
PathName ::= AbsolutePathName | RelativePathName
Separator ::= "/" ["/"]*
AbsolutePathName ::= Separator [ RelativePathName ]
RelativePathName ::= Component [ Separator RelativePathName ]*
Component ::= "." | Name
Name ::= Character+ - {"."}
Character ::= {c: c in {{ legal ASCII characters } - {'/'}}
```

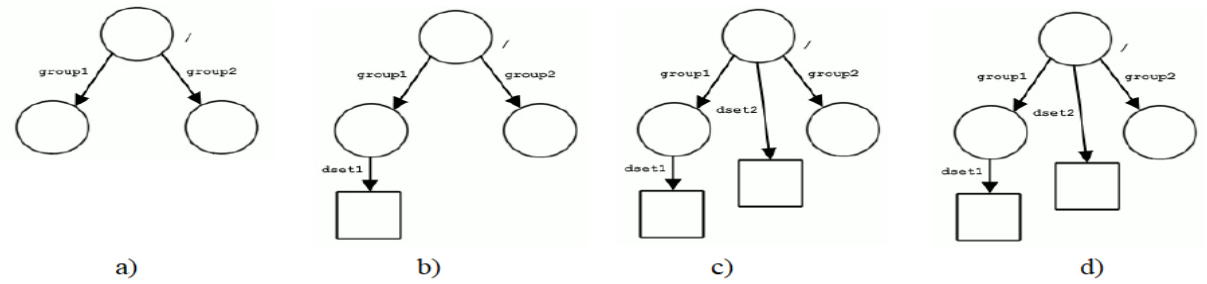
3.13: BNF gramatika za nazive putanja

Objekat se uvek može adresirati punom ili apsolutnom putanjom koja bi počela od koren grupe. Dati objekat može imati više od jednog punog imena putanje. Objekt se takođe može adresirati relativnom putanjom koja bi počinjala od grupe i uključivala bi putanju do objekta.

Struktura HDF5 fajla je „samoopisujuća“. To znači da je moguće kretati fajlom da bi se otkrili svi objekti u njemu. U osnovi, struktura se prelazi kao graf počevši od jednog čvora i rekurzivno obilazeći čvorove grafa.

- Primeri strukture HDF5 fajla

Na slici 3.14 prikazane su neke moguće strukture HDF5 fajla sa grupama i skupovima podataka. Pod a) prikazana je struktura fajla sa tri grupe. Pod b) prikazuje se skup podataka kreiran u „/group1“. Pod c) prikazuje strukturu nakon što je skup podataka pod nazivom dset2 dodat u koren grupu. I konačno pod d) prikazuje strukturu nakon dodavanja druge grupe i skupa podataka.



Slika 3.14: Primeri različitih struktura HDF5 fajla

3.4 Prednosti i slabosti HDF5 formata

Prednosti:

- Skladištenje visokih performansi: HDF5 je dizajniran za skladištenje visokih performansi, omogućavajući brzo čitanje i pisanje velikih skupova podataka.
- Kompresija podataka: HDF5 podržava nekoliko algoritama kompresije, omogućavajući efikasnije skladištenje podataka. Ovo pomaže da se smanji prostor za skladištenje potreban za velike skupove podataka.
- Fleksibilni model podataka: HDF5 format podržava širok spektar tipova podataka i struktura, uključujući nizove, tabele i grupe, što ga čini pogodnim za skladištenje raznolikog spektra naučnih skupova podataka.
- Nezavisno od platforme: HDF5 fajlovi se mogu čitati i pisati na mnogo različitih platformi, što olakšava deljenje podataka između različitih sistema.
- Performanse: HDF5 je dizajniran za visoke performanse i može se optimizovati za specifična hardverska i softverska okruženja, što ga čini brzim i efikasnim za rad sa velikim skupovima podataka.

Slabosti:

- Opterećenje performansi: Dok je HDF5 dizajniran za skladištenje visokih performansi, postoje određeni troškovi povezani sa pristupom podacima uskladištenim u formatu. Ovo može rezultirati sporijim performansama za male skupove podataka.
- Nedostatak interoperabilnosti: Iako se HDF5 široko koristi, to nije jedini format podataka za skladištenje naučnih podataka, a može postojati i problem kompatibilnosti između HDF5 i drugih formata.
- Ograničenja pristupa podacima: HDF5 format je optimizovan za nasumičan pristup velikim skupovima podataka i može biti sporiji kada se pristupa malim količinama podataka.

4. JSON format

Format JSON (JavaScript Object Notation) prvi je predložio Daglas Krokford 2002. godine kao lagan format za razmenu podataka koji je jednostavan za korišćenje. U to vreme, XML je postajao sve složeniji i teži za raščlanjivanje, a postojala je potreba za pojednostavljenim i jednostavnijim formatom.

JSON je dizajniran da bude jednostavan format zasnovan na tekstu koji se lako može koristiti za razmenu podataka između servera i veb aplikacija. Inspirisan je sintaksom JavaScript objekata i dizajniran je da bude podskup JavaScript programskog jezika.

U godinama nakon njegovog predloga, JSON je brzo stekao popularnost među veb programerima, posebno za upotrebu u API-jima. Jednostavnost i lakoća korišćenja JSON-a u poređenju sa XML-om učinili su ga atraktivnim izborom za razmenu podataka preko veba.

Danas se JSON široko koristi kao format za razmenu podataka u mnogim veb aplikacijama i API-jima, a podržavaju ga mnogi programski jezici, uključujući Java, Pithon i Rubi. JSON format je postao standard za laganu i efikasnu razmenu podataka i nastavlja da se široko koristi i usvaja.

4.1 JSON sintaksa

JSON format fajla ima specifičnu sintaksu koju morate pratiti da bi se smatrao važećim JSON fajlom. Osnovna pravila za sintaksu JSON fajla:

- Podaci su predstavljeni kao parovi ključ-vrednost: U JSON-u podaci su predstavljeni kao parovi ključ-vrednost, gde je ključ string, a vrednost može biti string, broj, logički, niz ili drugi JSON objekat.
- Parovi ključ-vrednost su odvojeni zarezima: Parovi ključ-vrednost u JSON objektu su odvojeni zarezima.
- Parovi ključ-vrednost su zatvoreni u vitičaste zagrade: Parovi ključ-vrednost u JSON objektu su zatvoreni u vitičaste zagrade {}.
- Stringovi se stavljaju u dvostruke navodnike: U JSON-u, stringovi moraju biti stavljeni u dvostruke navodnike.
- Nizovi su predstavljeni uglastim zagradama: Nizovi u JSON-u su predstavljeni uglastim zagradama [], a vrednosti u nizu su odvojene zarezima.
- Bez zareze na kraju: JSON ne dozvoljava zareze na kraju u objektima ili nizovima.

4.2 Primeri JSON fajlova

Jednostavan JSON objekat ima oblik:

```
{
  "ime": "Pero Perić",
  "starost": 30,
  "adresa": "123 Nebojšina, Beograd, Srbija"
}
```

U ovom primeru imamo jednostavan JSON objekat sa tri para ključ/vrednost. Prvi ključ je "ime", drugi je "starost", a treći je "adresa". Vrednosti svakog ključa su string, broj i drugi string, respektivno.

JSON objekat sa ugnježdenim objektima:

```
{
  "ime": "Pero Perić",
  "starost": 30,
  "adresa": {
    "ulica": "123 Nebojšina",
    "grad": "Beograd",
    "država": "Srbija"
  }
```

U ovom primeru JSON objekat ima ista tri ključa kao u prethodnom primeru. Međutim, ključ „adresa“ sada ima drugi JSON objekat kao vrednost. Ovaj ugnježdeni JSON objekat ima tri para ključ/vrednost: „ulica“, „grad“ i „država“.

JSON objekat sa nizovima:

```
{
  "ime": "Pero Perić",
  "starost": 30,
  "interesovanja": "čitanje","pešačenje","kuvanje"
}
```

JSON objekat ima tri ključa. Ključ "interesi" ima niz stringova kao svoju vrednost. Ovaj niz predstavlja interese osobe.

Poslednji primer jeste jedan složeni JSON objekat sa ugnježdenim objektima i nizovima:

```
{
 "ime": "Pero Perić",
  "starost": 30,
  "adresa": {
    "ulica": "123 Nebojšina",
    "grad": "Beograd",
    "država": "Srbija"
  },
  "interesovanja": [
    {
      "tip": "na_otvorenom",
      "aktivnosti": ["planinarenje", "kampovanje"]
    },
    {
      "tip": "u_zatvorenom",
      "aktivnosti": ["čitanje", "igranje"]
    }
  ]
}
```

U ovom primeru predstavljen je složeniji JSON objekat koji uključuje ugnježdene objekte i nizove. Ključ „interesovanja“ ima niz JSON objekata kao svoju vrednost, gde svaki objekat predstavlja tip interesovanja i povezane aktivnosti. Ovaj primer pokazuje kako se JSON može koristiti za predstavljanje složenijih struktura podataka.

Ovo su samo nekoliko primera kako JSON fajl može da izgleda. Format je veoma fleksibilan i može se koristiti za predstavljanje širokog spektra struktura podataka, od jednostavnih parova ključ-vrednost do složenih ugnježdenih objekata i nizova.

4.3 Prednosti i slabosti JSON formata

Prednosti JSON formata:

- Čovjeku čitljiv: JSON je format zasnovan na tekstu, što ga čini čitljivim ljudima. Ovo olakšava ljudima da razumeju i tumače podatke bez potrebe za posebnim alatima.
- Lagan: JSON je lagan format, što ga čini brzim i efikasnim za prenos i raščlanjivanje. Ovo je posebno važno za veb aplikacije gde je brz prenos podataka ključan.
- Kompatibilnost među platformama: JSON je format koji zavisi od platforme, što znači da se podaci mogu lako deliti između različitih sistema i programskih jezika.
- Lako za raščlanjivanje: JSON ima jednostavnu i dobro definisanu sintaksu, koja programerima olakšava raščlanjivanje i pristup podacima u svojim programima.
- Široko prihvaćen: JSON je postao široko prihvaćen format za razmenu podataka, a mnogi API-ji i web usluge koriste JSON kao svoj preferirani format za razmenu podataka.

Slabosti JSON formata:

- Ograničeni tipovi podataka: JSON podržava samo ograničen skup tipova podataka, uključujući nizove, brojeve, logičke vrednosti, nizove i objekte. Ovo možda neće biti dovoljno za neke slučajeve upotrebe koji zahtevaju složenije strukture podataka.
- Ograničenja veličine: JSON podaci mogu postati prilično veliki, posebno za složene strukture podataka. Ovo može dovesti do povećane upotrebe memorije i sporijeg vremena prenosa podataka.
- Sklon greškama: JSON je format zasnovan na tekstu, što znači da može biti sklon greškama kao što su greške u kucanju ili nedostajući navodnici. Ove greške može biti teško otkriti i popraviti, posebno za velike i složene strukture podataka.
- Nema podrške za verzionisanje: JSON nema ugrađenu podršku za verzionisanje, što može otežati upravljanje promenama strukture podataka tokom vremena.

5. XML format

XML (Ektensible Markup Language) je jezik za označavanje podataka koji je kreiran za skladištenje i transport podataka na način koji je i čitljiv i mašinski čitljiv. Razvio ga World Wide Web Consortium (W3C) kasnih 1990-ih kao odgovor na sve veću potrebu za fleksibilnim i efikasnim načinom skladištenja i razmene podataka preko Interneta.

Razvoj XML-a bio je motivisan potrebom za zajedničkim formatom za razmenu strukturiranih podataka između različitih računarskih sistema. U to vreme, HTML (Hypertext Markup Language) je bio najčešće korišćeni format za razmenu podataka, ali je bio ograničen u svojoj sposobnosti da opiše složene strukture podataka i nije bio pogodan za razmenu podataka između različitih sistema.

XML je dizajniran da prevaziđe ova ograničenja obezbeđivanjem fleksibilnog i proširivog jezika za označavanje koji se može koristiti za opisivanje širokog spektra struktura podataka. Dizajniran je tako da bude jednostavan, lako čitljiv i ljudima i mašinama i da može da se obrađuje širokim spektrom softverskih alata.

Od svog nastanka, XML je postao jedan od najčešće korišćenih formata za razmenu podataka i koristio se u širokom spektru aplikacija, uključujući veb usluge, upravljanje dokumentima i razmenu podataka između različitih računarskih sistema. Takođe je korišćen kao osnova za druge jezike za označavanje, kao što su XHML (Extensible Hypertext Markup Language) i XSLT (Extensible Stylesheet Language Transformations).

5.1 Sintaksa XML fajlova

Sintaksa XML fajlova se sastoji o par ključnih elemenata:

1. Elementi: Elementi su građevni blokovi KSML dokumenta i koriste se za opisivanje strukture i sadržaja podataka. Elementi su definisani početnom i završnom oznakom, a sadržaj elementa je smešten između dve oznake. Na primer:

```
<person>Pero Perić</person>
```

2. Atributi: Atributi se koriste za pružanje dodatnih informacija o elementu. Oni su specificirani unutar početne oznake elementa i imaju oblik parova ime-vrednost. Na primer:

```
<person ime=">Pero Perić" starost="35"/>
```

3. Instrukcije za obradu: Instrukcije za obradu pružaju informacije XML procesorima o tome kako da rukuju podacima. Počinju sa <? i završava se sa ?>. Na primer:

```
<?xml version="1.0"?>
```

4. Komentari: Komentari se koriste za uključivanje beleški ili opisa u XML dokument. Počinju sa <!-- i završavaju sa -->. Na primer:

```
<!-- Ovo je komentar -->
```

5. CDATA odeljak: CDATA odeljak se koristi za uključivanje velikih blokova teksta u XML dokument bez potrebe za izbegavanjem specijalnih znakova. Počinje sa <![CDATA[i završava se sa]]>. Na primer:

```
<description><![CDATA[Ovo je opis]]></description>
```

6. Reference entiteta: Reference entiteta se koriste za uključivanje specijalnih znakova u XML dokument. Na primer:

```
<description>Simbol za Euro valutu je &#8364;</description>
```

7. Imenski prostori: Imenski prostori pružaju način da se razlikuju elementi sa istim imenom koji se koriste u različitim kontekstima. Prostori imena se deklarišu pomoću atributa xmlns. Na primer:

```
<person xmlns="http://example.com/persons">Pero Perić</person>
```

Ovo su osnovni elementi XML sintakse. Razumevanje ovih elemenata je od suštinskog značaja za kreiranje dobro oblikovanih XML dokumenata koji se lako čitaju i obrađuju.

5.2 Primer XML fajla

Primer jednostavnog XML dokumenta:

```xml
<?xml version="1.0"?>
<osoba>
  <ime>Pero Perić</ime>
  <email adresa="pero.perić@example.com"/>
  <telefon vrsta="posao">555-555-1212</telefon>
</osoba>
```

Prvi red govori da je ovo XML dokument i navodi verziju XML-a koja se koristi. Osnovni element je „osoba“ i sadrži tri podređena elementa: „ime“, „email“ i „telefon“. Element „email“ ima atribut „adresa“, a element „telefon“ ima atribut „vrsta“.

XML dokument sa imenskim prostorom:

```xml
<?xml version="1.0"?>
<osoba xmlns="http://example.com/persons">
  <ime>Pero Perić</ime>
 <email adresa="pero.perić@example.com"/>
  <telefon vrsta="posao">555-555-1212</telefon>
</osoba>
```

U ovom primeru, imenski prostor je deklarisan pomoću atributa xmlns na osnovnom elementu. Imenski prostor pruža način za razlikovanje elemenata sa istim imenom koji se koriste u različitim kontekstima. U ovom slučaju, prostor imena „http://example.com/persons“ je povezan sa elementom „osoba“.

5.3 Prednosti i slabosti XML formata

Prednosti XML-a:

- Fleksibilnost: XML je veoma fleksibilan i može se koristiti za predstavljanje širokog spektra struktura podataka. Nije ograničen na određenu vrstu podataka i može se koristiti za bilo šta, od jednostavnih tekstualnih dokumenata do složenih naučnih podataka.
- Čoveku čitljiv: XML je čitljiv čoveku, što programerima olakšava razumevanje i pisanje XML dokumenata. To ga čini popularnim izborom za razmenu podataka između različitih sistema.
- Široko je podržan i korišćen.
- Nezavisno od platforme: XML podaci su nezavisni od platforme, što znači da se mogu koristiti na bilo kom operativnom sistemu i bilo kom programskom jeziku koji podržava XML.
- Skalabilan: XML je dizajniran da rukuje velikim količinama podataka, što ga čini skalabilnim i pogodnim za upotrebu u velikim sistemima za skladištenje i pronalaženje podataka.

Slabosti XML formata:

- Opširno: XML fajl mogu biti veće od binarnih formata jer sadrže mnogo suvišnih podataka i meta informacija.
- Sporija obrada: raščlanjivanje i obrada XML podataka može biti sporija od obrade binarnih podataka, posebno za velike fajlove.
- Složena: Struktura XML dokumenta može postati složena, što otežava rad i održavanje.
- Stroga sintaksa: XML zahteva striktno poštovanje svoje sintakse, a čak i manje greške mogu dovesti do toga da se XML dokument smatra nevažećim.
- Nedostatak kucanja podataka: XML nema ugrađen način za navođenje tipova podataka, što može otežati obradu i validaciju podataka.

6. Parquet format

Apache parquet je format podataka optimizovan za skladištenje i obradu velikih količina strukturiranih podataka, često korišćen u okviru Hadoop i drugih Big Data sistema.

Parquet je napavljen prednosti komprimovanog, efikasnog predstavljanja podataka u kolonama učinili dostupnim bilo kom projektu u Hadoop ekosistemu.

Napravljen je od temelja sa složenim ugnježdenim strukturama podataka na umu i koristi algoritam za sečenje i sklapanje zapisa opisan u Dremel dokumentu.

Parquet je napravljen da podržava veoma efikasne šeme kompresije i kodiranja. Više projekata je pokazalo učinak primene odgovarajuće šeme kompresije i kodiranja na podatke. Parquet omogućava da se šeme kompresije specificiraju na nivou po koloni, i ima mogućnost za budućnost da omogući dodavanje više kodiranja kako se izmišljaju i implementiraju.

6.1 Terminologija

Blok (*hdfs blok*): Ovo znači blok u hdfs-u i značenje je nepromenjeno za opisivanje ovog formata fajlova. Format fajla je dizajniran da dobro radi na hdf-ovima.

Fajl (*File*): hdfs fajl koja mora da sadrži metapodatke za fajl. Ne mora zapravo da sadrži podatke.

Grupa redova (*Row group*): Logičko horizontalno razdvajanje podataka u redove. Ne postoji fizička struktura koja je zagarantovana za grupu redova. Grupa redova se sastoji od dela kolone za svaku kolonu u skupu podataka.

Komad kolone (*Column chunk*): Deo podataka za određenu kolonu. Oni žive u određenoj grupi redova i garantovano su uzastopni u fajlu.

Stranica (*Page*): Delovi kolona su podeljeni na stranice. Stranica je konceptualno nedeljiva jedinica (u smislu kompresije i kodiranja). Može postojati više tipova stranica koje se prepliću u komadu kolone.

Hijerarhijski, fajl se sastoji od jedne ili više grupa redova. Grupa redova sadrži tačno jedan deo kolone (komad kolone) po koloni. Delovi kolona sadrže jednu ili više stranica.

Jedinica paralelizacije

- MapReduce - Grupa fajlova/redova
- IO – komad kolone
- Kodiranje/Kompresija - Stranica

6.2 Format parquet fajla

Primer definisane tabele u parquet formatu:

```
4-byte magic number "PAR1"
<Column 1 Chunk 1 + Column Metadata>
<Column 2 Chunk 1 + Column Metadata>
...
<Column N Chunk 1 + Column Metadata>
<Column 1 Chunk 2 + Column Metadata>
<Column 2 Chunk 2 + Column Metadata>
...
<Column N Chunk 2 + Column Metadata>
...
<Column 1 Chunk M + Column Metadata>
<Column 2 Chunk M + Column Metadata>
...
<Column N Chunk M + Column Metadata>
File Metadata
4-byte length in bytes of file metadata
4-byte magic number "PAR1"
```

U ovoj tabeli ima N kolona, podeljenih u M grupa redova. Metapodaci fajla sadrže lokacije svih početnih lokacija metapodataka kolone. Svaki red podataka je podeljen na jedan ili više „Column chunk“, gde svaki deo sadrži podatke za jednu kolonu. Unutar svakog dela, podaci se dalje dele na „Pages“, gde svaka stranica sadrži deo podataka kolone. Ova struktura omogućava brze operacije po kolonama, pošto čitanje podataka sa diska zahteva samo učitavanje relevantnih stranica za željene kolone, a ne ceo red.

Metapodaci su u obliku podnožja fajla koje opisuje sadržaj fajla, uključujući šemu, informacije o tipu podataka i statistiku o podacima, kao što su vrednosti minimuma i maksimuma, koje mogu da koriste mašine za upite za optimizaciju upita. Metapodaci se pišu nakon podataka da bi se omogućilo jednokratno pisanje.

Parquet takođe koristi napredne tehnike kompresije i kodiranja kako bi minimizirao količinu prostora na disku potrebnom za skladištenje podataka. Na primer, koristi kodiranje dužine pokretanja, kodiranje rečnika i pakovanje bitova da bi se podaci kompresovali na efikasan način.

Od strane korisnika se očekuje da prvo pročitaju metapodatke fajla kako bi pronašli sve delove kolona za koje su zainteresovani. Delove kolona onda treba čitati uzastopno.

Format je eksplicitno dizajniran da odvoji metapodatke od podataka. Ovo omogućava razdvajanje kolona u više fajlova, kao i da jedan fajl metapodataka upućuje na više fajlova parqueta.

6.3 Konfiguracije fajlova

Veličina grupe redova (Row Group Size):

Veće grupe redova omogućavaju veće komade kolona što omogućava veći sekvencijalni IO. Veće grupe takođe zahtevaju više baferovanja u putanji pisanja (ili upisivanje u dva prolaza). Preporučuje se da velike grupe redova (512MB - 1GB). Pošto će možda morati da se pročita cela grupa redova, u cilju je da ona u potpunosti stane na jedan HDFS blok. Stoga, veličine HDFS blokova takođe treba da budu podešene na veće. Optimizovano podešavanje čitanja bi bilo: 1GB grupe redova, 1GB HDFS veličina bloka, 1 HDFS blok po HDFS fajlu.

Veličina stranice sa podacima (Data Page Size):

Stranice sa podacima treba smatrati nedeljivim tako da manje stranice sa podacima omogućavaju detaljnije čitanje (npr. traženje u jednom redu). Veće veličine stranica izazivaju manje prostora (manje zaglavlja stranice) i potencijalno manje raščlanjivanja (zaglavlja obrade). Preporučujemo 8 KB za veličine stranica.

6.4 Prednosti i slabosti parquet formata:

Prednosti parquet formata:

- Columnar Storage: Apache parquet koristi kolonasti format skladištenja, koji omogućava efikasno kompresovanje i kodiranje podataka. Ovo rezultira bržom obradom podataka i smanjenim I/O diskom.
- Evolucija šeme: Apache parquet podržava evoluciju šeme, što znači da se podaci mogu čuvati u različitim verzijama šeme. Ovo je korisno za organizacije koje imaju evoluirajuću strukturu podataka i moraju da skladište podatke u fleksibilnom formatu.
- Kompresija: Apache parquet podržava različite algoritme kompresije, uključujući Snappy, Gzip i LZO, da smanji veličinu podataka na disku. Ovo rezultira bržim vremenom prenosa podataka i smanjenim troškovima skladištenja.
- Spuštanje predikata: Parquet podržava spuštanje predikata, što omogućava filtriranje podataka na sloju za skladištenje, a ne na sloju za obradu upita. Ovo može značajno smanjiti količinu podataka koje treba obraditi, što dovodi do poboljšanih performansi.
- Interoperabilnost: Parquet je otvoreni standard, što znači da se može koristiti sa raznim alatima za obradu velikih podataka i sistemima za skladištenje podataka, uključujući Apache Hadoop, Apache Spark, Apache Hive i Apache Impala.

Slabosti parquet formata:

- Ograničena podrška za ažuriranje: Parquet nema izvornu podršku za ažuriranja i brisanja. Ako je potrebno da se ažuriraju podaci, mora se kreirati novi parquet fajl sa ažuriranim podacima i zameniti sa starim fajlom.
- Ograničeno indeksiranje: Apache parquet nema ugrađenu podršku za indeksiranje, što znači je potrebno koristiti druge alate za indeksiranje podataka ako je potreban brz pristup određenim redovima.
- Opterećenje performansi: Parquet nije najbolji izbor za male skupove podataka, jer prostor koji je povezan sa samim formatom fajla mogu da nadmaše prednosti njegovog načina skladištenja i kompresije.
- Samo struktuirani podaci: Nije pogodan za skladištenje nestrukturiranih podataka, kao što su tekst ili slike. Prvenstveno je dizajniran za skladištenje i obradu strukturiranih podataka, koji uključuju podatke sa dobro definisanom šemom i fiksnim tipovima podataka.

7. Metodologija benčmarkovanja

Ovaj deo se posmatra u dva dela, prvi deo je benčmarkovanje veličina fajlova kreirani u različitim formatima, a drugi deo predstavlja benčmarkovanje brzine kreiranja tih fajlova.

7.1 Metodologija procesa benčmarkovanja različitih tipova fajlova

Benčmarkovanje veličina fajla se može podeliti u par koraka:

1. Kreiranje funkcija koje će omogućitit pravljenje test fajlova u različitim formatima, ti formati su json, xml, parquet i h5.
2. Definisati podatke koji će fajlovi da sadrže.
3. Kreirati fajlove u različitim uslovima i beležiti njihovu veličinu.
4. Veličine fajlova vizualizovati.

Na slici 7.1 može se videti suženi dijagram toka celokupnog procesa rada za benčmarkovanje veličine fajla. Detaljniji opisi svakog procesa biće dat u narednim delovima teksta.

Ukupno je generisano po 50 tačaka u zavisnost od slučaja. Ima tri razmatrana slučaja. Razmatrani slučaji su: Kada je broj redova fiksiran a broj kolona se menja, kada je broj kolona fiksiran a broj redova se menja i slučaj kada se menja i broj kolona i broj redova.

Opseg kolona je od 1 do 50 sa korakom 1. Opseg redova je od 5000 do 250000 sa korakom od 5000. Ovaj proces je rađen radi potrebe da se vidi da li postoji bias između načina upisivanja podataka u fajl, u slučaju da postoji, bilo bi neophodno dodatno optimizovanje podataka pri upisu.

Fajl se briše nakon što se njegova veličina zapamti, radi očuvanja prostora na korišćenom disku.

Slika 7.1: Dijagram toka prvog procesa

7.2 Metodologija procesa benčmarkovanja brzine pisanja fajlova

Benčmarkovanje brzine pisanja fajlova može biti opisano narednim koracima:

1. Pozivanjem funkcija za kreiranje fajlova.
2. Definisati podatke koji će kreirani fajlovi da sadrže.
3. Kreirati fajlove u različitim uslovima i beležiti njihovo vreme pisanja.
4. Brzinu pisanja fajlova vizualizovati.

Brzina je posmatrana u umanjenom broju tačaka nego u prvom procesu, ukupno 10 po slučaju. Na slici 7.2 može se videti suženi dijagagram toka celokupnog procesa rada za benčmarkovanje brzine.

Raspon kolona je od 5 do 50 sa korakom od 5. Broj redova je išao od 25000 do 250000 u korako od 25000.

Posmatranjem brzine se mogu videti dosta neočekivani rezultati. Brzina može i da utiče u biranju formata u kriznim slučajevima, npr. kad se hitno mora uraditi rezervna kopija fajlova, u takvom slučaju znati koji je tip najefikasniji ipak može mnogo značiti.

Treba da se naglasi da su ovo brzine funkcija koje su napravljene u R programskom jeziku, tako da ovaj test jeste validan samo u R okruženju. Ovi rezultati ne moraju da imaju istu validnost kada se primene drugi, programskim jezici.

Za merenje brzine samo je razmatrana brzina kreiranja i pisanja podataka u fajla.

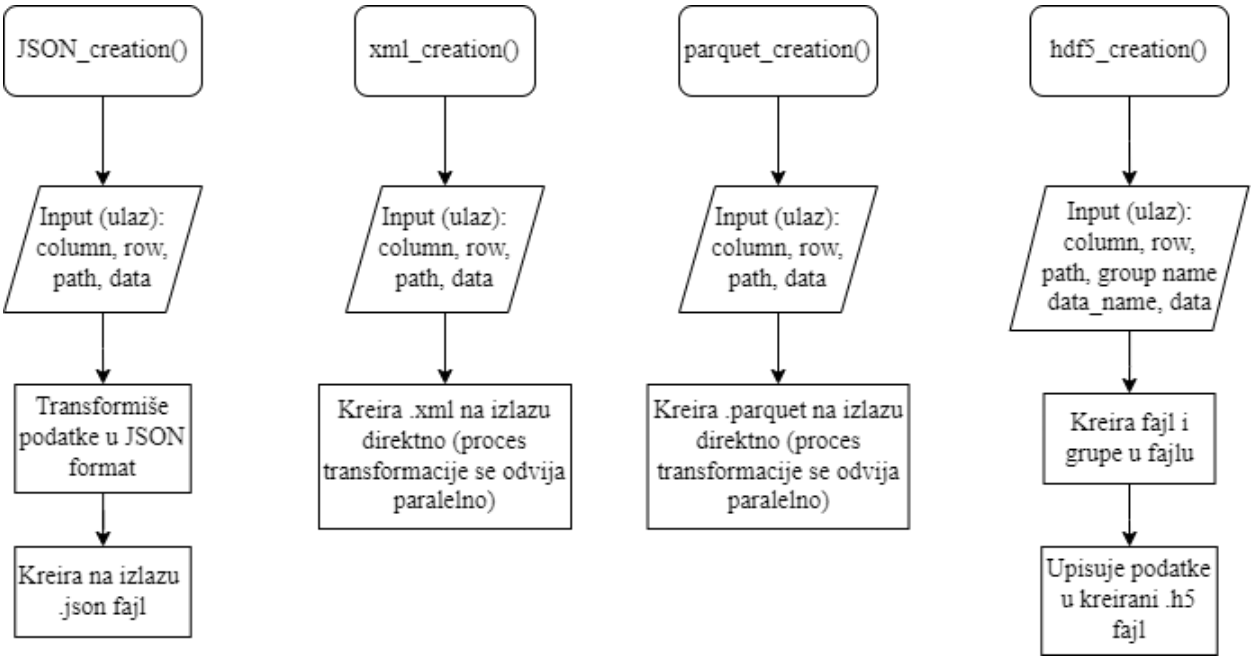
Slika 7.2: Dijagram toka drugog procesa

7.3 Kreiranje funkcija za generisanje fajlova

Za prvi korak je bilo neophodno da se konstruišu funkcije kojima će se generisati fajlovi tipa json, xml, parquet i h5. Kreirane su jednostavne funkcije za kreaciju pod imenima:

- JSON_creation – koristi biblioteku jsonlite.
- xml_creation – koristi paket biblioteka MESS.
- parquet_creation – koristi biblioteku arrow.
- hdf5_creation – koristi biblioteku rhdf5.

Procesi koji se dešavaju u funkcijama su prikazani u dijagramu toka 7.3:



Slika 7.3 Dijagram toka funkcija za kreiranje fajlova

Razlog pravljenja funkcija jeste da se smanji količina koda koja se ponavlja na više mesta, tj. da se smanji finalna kompleksnost koda. Kreiranje funkcija takođe se omogućava efikasniji metod za benčmarkovanja brzine u R prgoramskom jeziku.

U funkciji JSON_creation() korišćena je funkcija iz jsonlite biblioteke toJSON(). To je funkcija koja služi da pretvori kreirani data frame (objekat sa podacima u R jeziku) generisanih podataka u json format. Zatim se koristi generička funkcija (funkcija koja dolazi sa R jezikom) write() da se sačuva fajl u definisanoj fascikli. U R programskom jeziku ima dosta različitih biblioteka za rad sa json formatom, odluka na jsonlite je pala radi brzine i elegantnosti pri korišćenju. Takođe je funkcija dobro optimizovana u smislu da se pri korišćenju memorija pametno iskoristi (fajl se postepeno konvertuje).

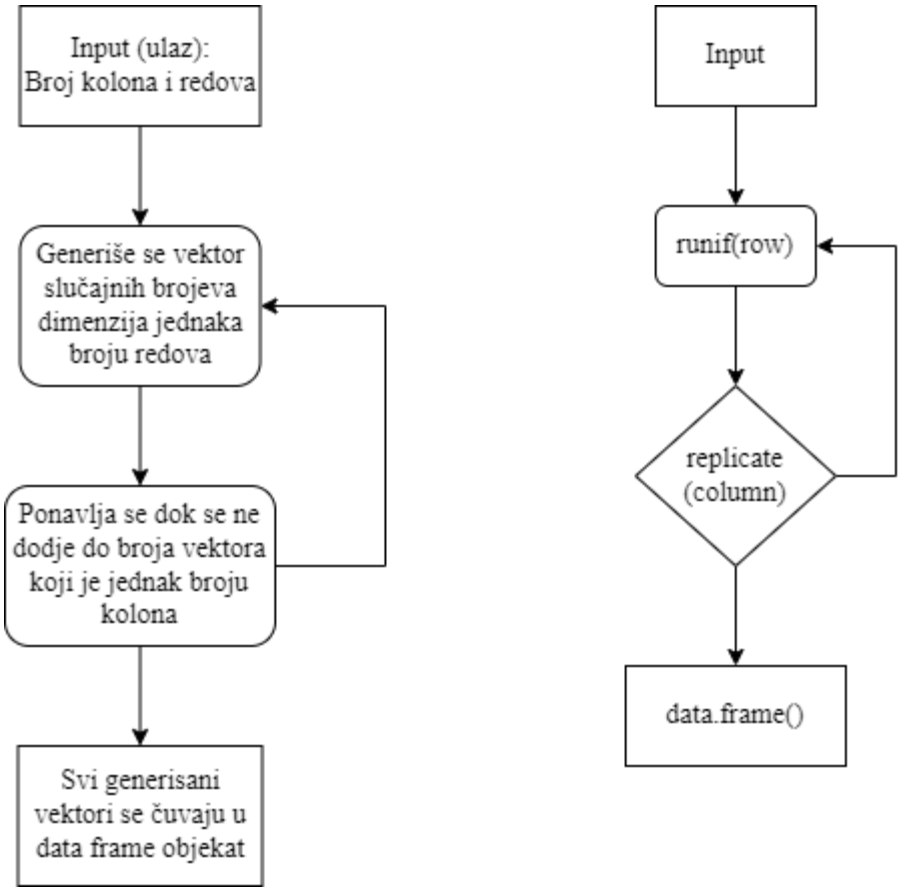
Za funkciju xml_creation() koristio se paket biblioteka MESS. Razlog za to je izbegavanje konvertovanja data frame - a u xml format pre upisavanja u fajl. Pomoću funkcijes write_xml() omogućeno je paralelno pisanje fajla i konvertovanje podataka u xml format. Što se tiče xml formata, on je generalno jako spor što se tiče brzine pisanja, i to je poprilično tačno i u R programskom jeziku. Pokušane su i druge biblioteke u ovom procesu kao što su XML i xml2, međutim njihove perfomanse su bile malo slabije u poređenju sa funkcijom iz MESS biblioteke.

Funkcija parquet_creation() je nastala korišćenjem biblioteke arrow. Nažalost to je i jedina biblioteka trenutno u R programskom jeziku koja omogućava rad sa parquet fajl formatom. Arrow biblioteka i sve funkcije u njoj su napisane u C++ programskom jeziku. Te funkcije koriste C++ interpreter, pa se zatim interpretira u R objekat. Sve funkcije koje su kreirane u arrow biblioteci oslanjaju se na funkcije koje je napravio Apache (oni su i napravili parquet format). Za kreiranje parquet fajla korišćena je funkcija write_parquet() iz arrow biblioteke. U ovom radu nije korišćena dodatna kompresija, koja predstavlja jedan od odlika parquet formata. To nije rađeno iz razloga da bi prestavljanje bilo identično kao sa ostalim formatima, međutim dodatna kompresija, npr. gzip je moguća pri kreiranju parquet fajla (pri kreiranju fajla se gzip-uju podaci).

Poslednja funkcija za kreiranje jeste hdf5_creation(). Ona se oslanja na funkcije iz biblioteku rhdf5. Kao i kod parquet formata, rhdf5 je jedina biblioteka u R programskom jeziku koja omogućava rad sa h5 fajlovima. Biblioteka rhdf5 nije dostupna na CRAN, već se koristi preko Bioconductor instalacionog menađera koji jeste podržan od strane CRAN-a. Funkcije koje su korišćene su: h5createFile, napravi prazan h5 fajl u definisanom repozitorijumu, zatim h5createGroup čime se kreira grupa u prethodno kreiranom h5 fajlu, pa se pomoću funkcije h5write upisuju podaci u kreirani h5 fajl.

7.4 Generisanje podataka za upis u fajlove

Da bi se fajlovi kreirali bilo je potrebno da se generišu podaci kojima će ti fajlovi biti ispunjeni. Za generisanje podataka korišćen je generator slučajno generisanih brojeva. Generisani brojevi su bili tipa double, sa 15 decimalnih mesta. Interval u kom su generisani je 0 do 1. Funkcija koja je korišćena za njihovo generisanje je runif(), koja generiše brojeve po normalnoj raspodeli. Količina generisanih brojeva je zavisila od broja redova, a koliko puta će se funkcija pokrenuti zavisilo je od broja kolona. U dijagramu toka 7.4 može se videti kako su generisani generisani podaci:



Slika 7.4 Dijagram toka generisanja podataka

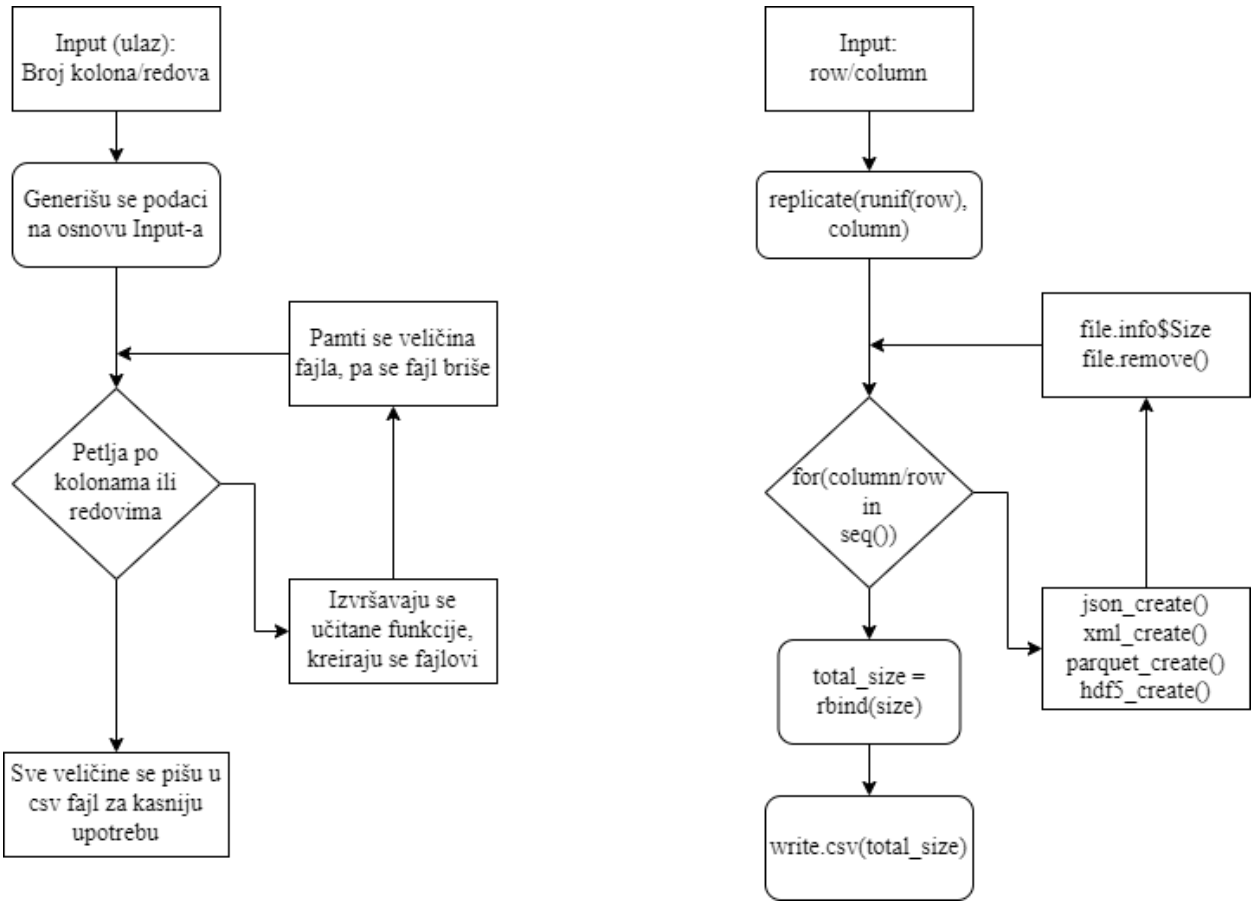
Funkcija runif() generiše vektor dužine ekvivalentnom definisanom broju redova. Taj vektor predstavlja jednu kolonu u finalnom data frame – u. Posao funkcije replicate() jeste da ponovi proces funkcije runif() toliko puta koliki je broj kolona na ulazu, time se generiše data frame koji ima broj kolona jednak broju kolona dat na ulazu sa dužinom od broja redova koji je definisan na ulazu. U svakoj iteraciji petlje, ovaj proces se vrši samo jednom, te se ti podaci posle generisanja čuvaju u različitim formatima.

7.5 Postupak merenja veličine fajlova

Razmatrana su tri slučaja, slučaj kada je fiksiran broj redova a menja se broj kolona, drugi je kada se menja broj redova a fiksiran je broj kolona i poslednji slučaj je kada se menja i broj redova i broj kolona. Opseg kolona je od 1 do 50 u koraku od 1, opseg redova je od 5000 do 250000 sa korakom od 5000. Dimenzije su određene radi hardverskog ograničenja korišćenog računara.

Razlog žašto su razmatrana tri slučaja jeste da se proveri da li postoji bias između redova i kolona u korišćenom formatu. Ako postoji bias, to bi značilo da bi pri upisu podataka, trebalo bi se voditi računa o broju kolona/redova za dati fajl, ako je bias veliki to bi značilo da je efikasnije podeliti podatke u dva fajla naspram sve staviti u jedan, što povećava kompleksnost skladištenja.

Proces nakon učitavanja biblioteka i funkcija se može videti na dijagramu toka 7.5:



Slika 7.5: Dijagram toka izvršavanja petlje za benčmarkovanje brzine

Na ulazu se definišu parametri neophodni za funkcije, to su broj kolona, redova i putanja ka folderu za izlaz (lokacija gde će se kreirati fajlovi). Zatim se ulazi u jednu for petlju koja broji po broju kolona ili redova, sa korakom 1 ako je slučaj kolona ili po koraku od 5000 ako je u pitanju slučaj kada se redovi koriste. U petlji se prvo generišu podaci, ti generisani podaci će biti samo sačuvani u petlji, kada se izađe iz petlje oni se ne čuvaju. Primenjuje se svaka od create() funkcija na tim podacima iterativno, tj. jedna po jedna. Prvo se generiše npr. parquet fajl, pamti se njegova vrednost veličine u jedan odvojeni objekat, te se taj fajl briše, tim korakom se završava deo za parquet format u toj iteraciji petlje. Onda se kreira json fajl od generisanih podataka, pamti se njegova veličina, pa se fajl zatim briše. Postupak je identičan za xml i h5 format. Petlja se vrti dok ne prebroji ceo interval kolona ili redova. Mora se naglasiti da u svaku novu iteraciju petlje se podaci definišu iznova, tj. njihove dimenzije se menjaju u zavisnost od broja kolona ili broja redova.

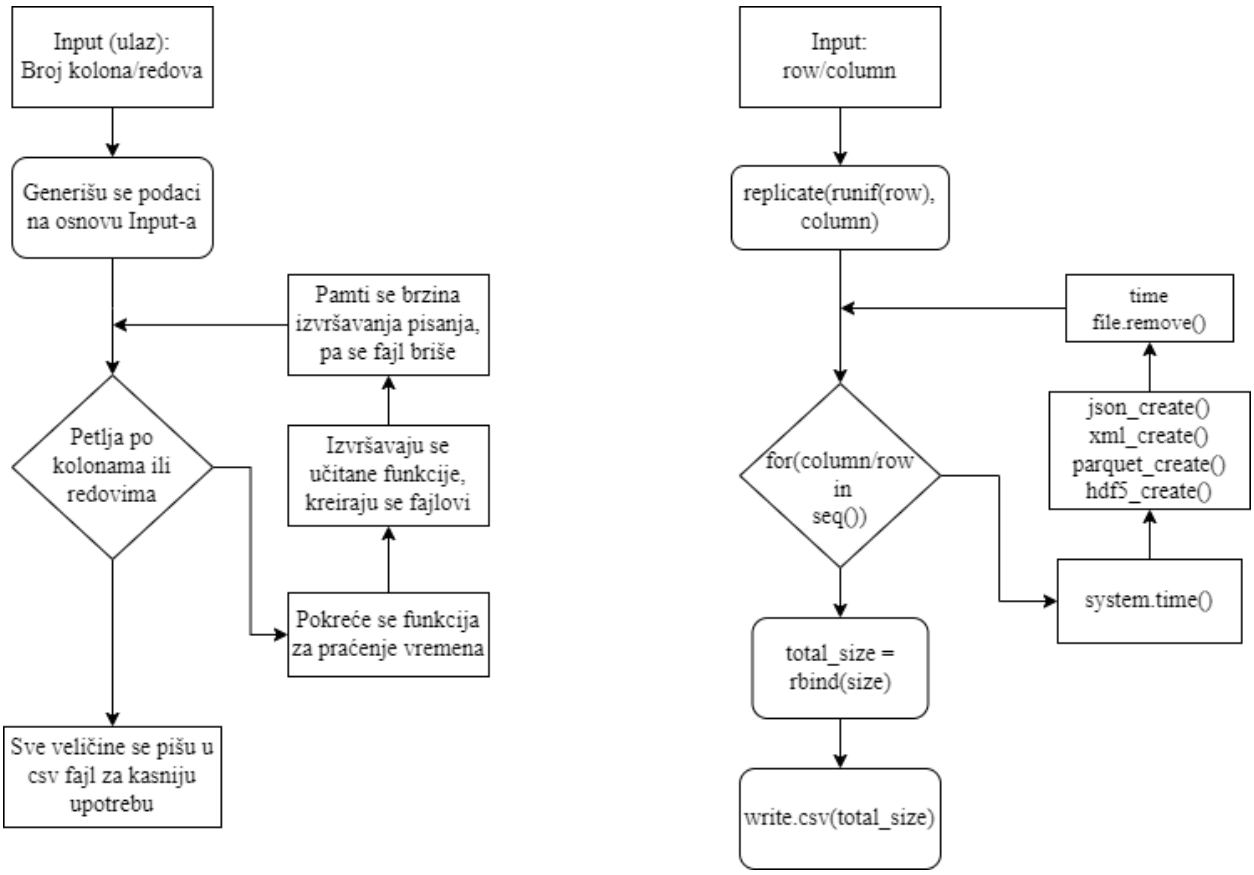
Ceo proces merenja vezano za benčmarkovanje veličine trajalo je oko 4 sata. To je inače i razlog žašto se finalni rezultat merenja čuvao u odvojeni csv fajl. Mogućnost da se proces samo ostavi, i da rezultati budu sačuvani na odvojenom mestu pri završetku merenja ili u slučaju prekida procesa ili neočekivanih okolnosti.

Pored svega ovoga potrebno je naglasiti da ovaj proces ipak koristi for petlju, tj. nije vektorizovan kao što je to poželjno u R programskom jeziku. Primaran razlog za tim jeste konkretna kompleksnost izlaznog koda, sa vektorizanom varijantom količina koda se poveća za tri puta, što ga čini još više kompleksnijim za razumevanje sa stanovišta drugog korisnika. Međutim potrebno je naglasiti da je optimizacija for petlje u R programskom jeziku sasvim zadovoljavajuća, što se tiče perfomansi u ovom konkretnom slučaju, nema velike razlike između for petlje i vektorizovane varijante koda. Konkretno slučaj kada se moraju izbegavati for petlje u R programskom jeziku jeste kada se pojavi ugnježdena for petlja, onda jeste krucijalno da se kod vektorizuje bez uzimanja u obzir finalnu kompleksnost konkretnog koda.

Razlog žašto se rezultat čuva u csv fajl jeste zato što najlakši format za rad u R programskom jeziku.

7.6 Postupak merenja brzine pisanja fajlova

Postpupak za brzinu je sličan kao za benčmarkovanje veličine. Postpupak merenja može se videti na dijagramu toka 7.6:



Slika 7.6: Dijagram toka izvršavanja petlje za benčmarkovanje brzine

Merenje brzine predstavlja samo merenje brzine pisanja fajla, okolni procesi nisu uzeti u obzir.

Posmatrana su tri slučaja, jedan je kada je fiksiran broj redova a menja se broj kolona, drugi je kada se menja broj redova a fiksiran je broj kolona i poslednji slučaj je kada se menja i broj redova i broj kolona.

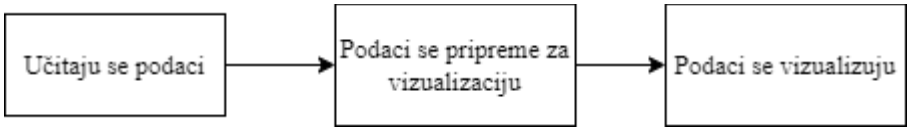
Za merenje brzine koristio se manji broj tačaka, tj. broj se smanjio sa 50 tačaka na 10 tačaka. Broj kolona ide od 5 do 50 sa korakom od 5, broj redova ide od 25000 do 250000 sa korakom od 25000. Kad je u pitanju merenje brzine ne dobija se mnogo od većeg broja tačaka, zavinost je jasna i od malog broja tačaka.

Na ulazu se definišu potrebni parametri, to su broj kolona ili broj redova u zavisnost od posmatranog slučaja i putanja ka folderu za izlaz (lokacija gde će se kreirati fajlovi). Zatim se ulazi u petlju u zavisnost od slučaja, tj. da li broji po kolonama ili redovima. Podaci se definišu za datu iteraciju. Zatim se poziva funkcija system.time() koja je generička funkcija R programskog jezika (ne dolazi sa bibliotekom, već direktno sa R programskim jezikom). Funkcija system.time() služi za merenje brzine funkcija. Zatim se poziva npr. parquet_create() funkcija koja je smeštena u funkciju system.time(). Kreira se parquet fajl sa generisanim podacima za tu iteraciju, nakon završetka funkcije parquet_create(), definiše se parametar time od strane system.time() funkcije, koji pamti vrednosti vremena pisanja fajla. Ti podaci se čuvaju u odvojeni data frame (objekat) isto kao u procesu opisanim u 7.5. Zatim se isti proces vrši i za json, xml i h5 formate. Nakon što petlja pređe ceo interval zadat na ulazu, piše se csv fajl sa dobijenim merenjima.

Stvar koja se mora naglasiti jeste da benčmarkovanje brzine je u potpunosti zavisno od efikasnosti implementacije funkcija u datom programskom jeziku, tj. u ovom slučaju od R programskog jezika. Ovi rezultati ne važe za implementaciju istih funkcija u drugim programskim jezicima, već su striktno vezani za R programsko okruženje.

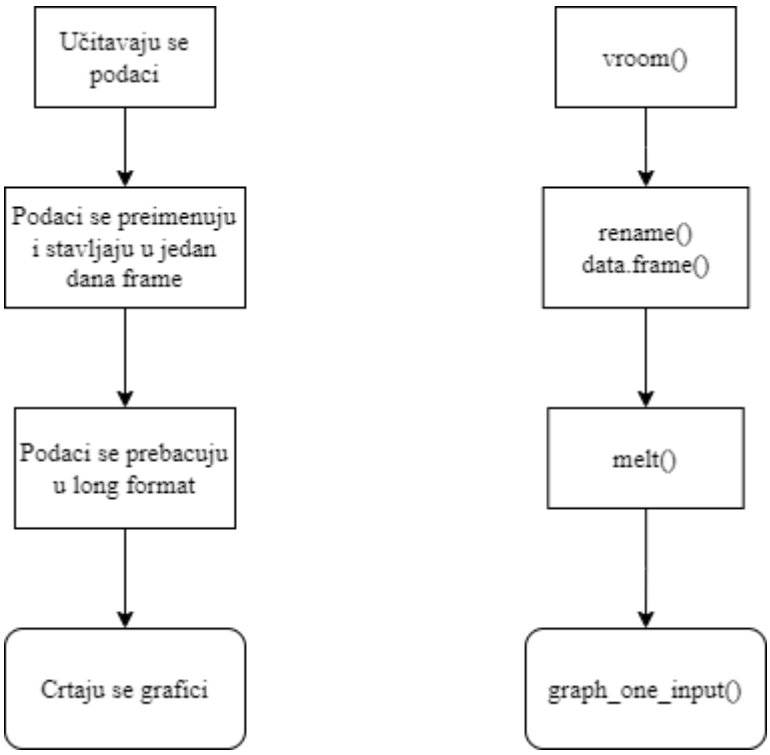
7.7 Proces vizualizacije

Proces vizuelizacije sproveden je korišćenjem R programskog jezika. Za crtanje grafika korišćena je biblioteka ggplot2, a za manipulisanje podataka korišćene su biblioteke dplyr i reshape2. Za učitavanje podataka korišćena je biblioteka vroom. U dijagramu 7.7 se može videti ukratko kako je izgledao proces vizualizacije podataka



Slika 7.7: Koraci pri vizualizaciji

Prvi korak u ovom postupku je bio da se podaci učitaju, učitavanje se vršilo pomoću funkcije vroom() iz biblioteke vroom. Nakon što su podaci učitani, vršilo se sređivanje podataka u smislu dodavanja adekvatnih imena kolonama, pretvaranje u čitljive jedinice (sekundi [s] i megabajti [MB]). Zatim se svi podaci za specifičan slučaj (u zavisnost šta se menja, kolone, redovi ili oba) spoje u jedan objekat koji zadrži podatke za sve formate u jednom objektu. To je rađeno jer smanjuje količinu koda, i samim tim njegovu kompleksnost pri crtanju grafika. Zatim na taj objekat se primenjuje funkcija melt() iz reshape2 biblioteke, koji stavlja fajla u long format (pretvori 4 kolone u dve). Zatim se taj ti podaci vizualizuju sa funkcijom napravljena za crtanje, naziv funkcije je graph_one_input(). Detaljan tok se može videti na dijagramu toka 7.8:



Slika 7.8 Detaljni tok procesa vizualizacije

Ovaj proces je primenjen i na benčmarkovanje veličine i na benčmarkovanje brzine. Podaci za dati slučaj, kad su fiksirani redovi a menjaju se kolone su crtane na jedan grafik, kad su fiksirane kolone a menjaju se redove na jedan i isto tako za slučaj kada se oba menjaju.

8. Metodologija izrade alata za HDF5 fajlove

Potreba za razvoj alata za rad sa HDF5 fajlovima se pojavila iz jednog konkretnog problema, taj problem jeste skladištenje snimaka noćnog neba sveta koji su dobijeni od strane NASA. Ti snimci su dostupni na sajtu LAADS DAC (Level-1 and Atmosphere Archive & Distribution System), i njihova problematika jeste da se podaci čuvaju u kvadrantima koji ne uzimaju u obzir državne granice. I kao što je u ovom slučaju, snimci od interesa su snimci za Republiku Srbiju, nalaze se u dva različita h5 fajla. Cilj ovog dela jeste razvoj alata koji će omogućiti prepakivanje podataka za Republiku Srbiju iz dva fajla u jedan fajla, a kasnije i više fajlova u jedan konkretan.

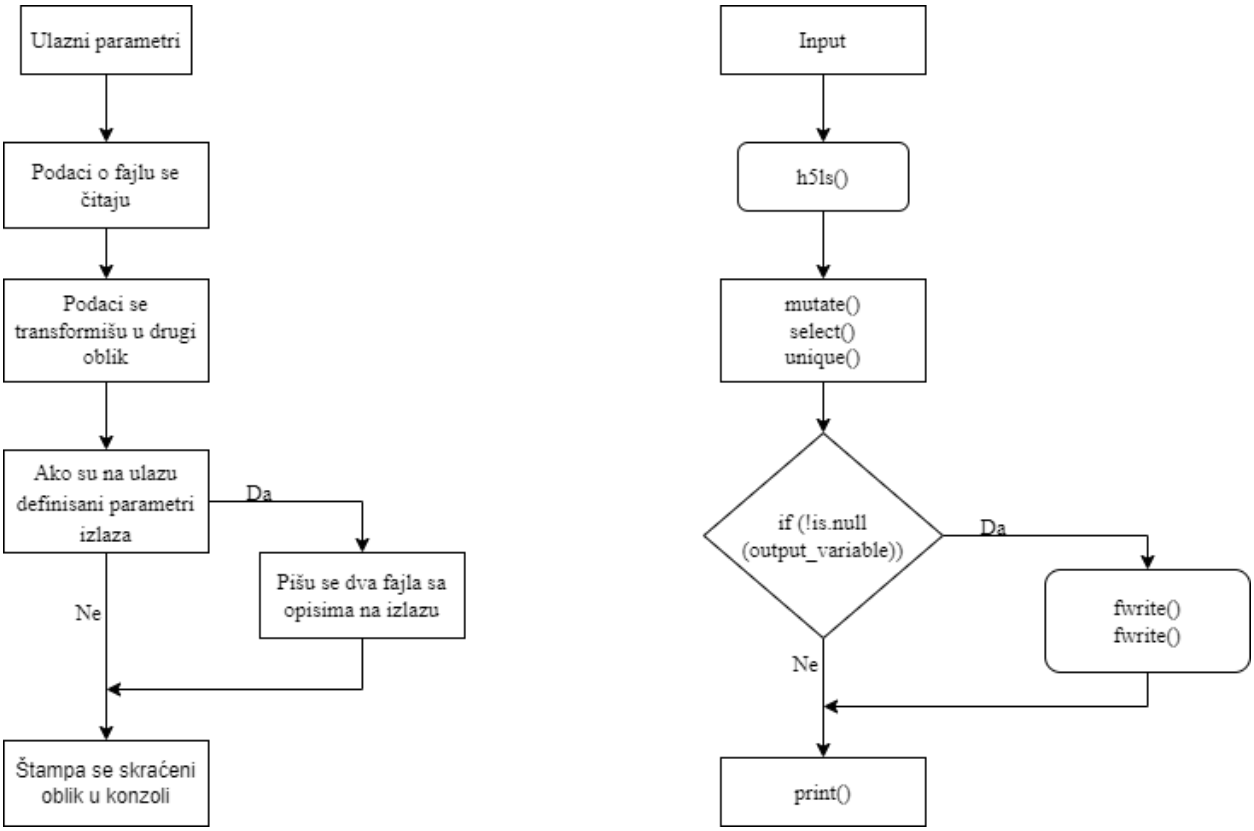
Dostupni h5 fajlovi dolaze na mesečnom nivou. Napravljena je i alatka koja od svih tih podataka filtriranih stavlja u jedan fajl, tj. od 12 h5 fajlova koji predstavljaju snimke za godinu dana za Republiku Srbiju, dobiće se jedan h5 fajl koji ima sve godišnje snimke spakovane u 12 različitih grupa koji predstavljaju mesec u datoj godini.

Pored toga je napravljena još jedna alatka koja omogućava uvid u strukturu h5 fajl. Najveći problem sa radom h5 fajlova jeste što je teško odrediti šta se konkretno nalazi u fajlu bez njegovog direktnog otvaranja. To znači da ogromne fajlove kao što su korišćeni snimci (po 400 MB podataka), moraju se učitati u memoriju računara da bi se saznalo šta je u njima. Ako se radi na računarima sa malom količinom dostupne RAM memorije, ovaj proces može biti veoma rizičan jer može dovesti do preopterećenja memorije i samim tim dovesti do isključenja računara. Da bi se sprečio takav scenario napravljena je funkcija koja će na izlazu dati putanju ka podacima h5 fajla. Ovime može se samo deo sa podacima učitati, ili samo jedna grupa sa podacima na primer. Detaljni podaci se mogu dobiti korišćenjem funkcije iz rdhf5 bibliotek h5ls(), međutim iako rezultat govori o tome šta se nalazi u datom fajlu, nezgodno je za otkrivanje konkretne putanje ka podacima. Ta činjenica je motivacija razvoja ove alatke.

8.1 Metodologija razvoja funkcije za uvid fajla

Kreirana funkcija ima ime h5_lookup(). Za ovu funkciju korišćene biblioteke su rdhf5, data.table, tidyr i magrittr. Glavni oslonac ove funkcije jesu povratne informacije koje se dobijaju iz funkcije h5ls(). Glavni parametar na ulazu je file_path, koji predstavlja putanju ka fajlu čija informacija se traži. Pored toga dodata su još dva opciona parametra output_name i file_name koja predstavljaju mogućnost da se na izlazu piše csv fajl sa dobijenim informacijama. Ako se opcioni parametri definišu, onda će se na izlazu kreirati dva csv fajla, jedan fajl se završava sa file_nameShort.csv a drugi sa file_nameDetailed.csv. Jedan predstavlja putanje podataka (short), a drugi sadrži sve informacije o konkretnom fajlu (detailed).

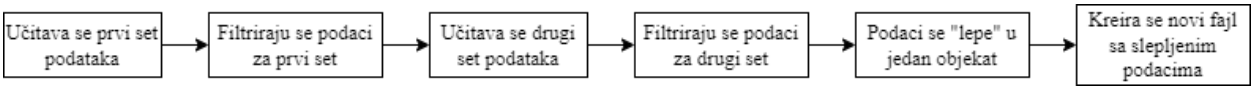
Konkretan tok rada funkcije može se videti na dijagramu toka 8.1:



Slika 8.1 Tok funkcije h5_lookup()

8.2 Metodologija razvoja funkcije filtriranja i lepljenja dva HDF5 fajla

Kreirana funkcija je specifično napravljena da radi sa podacima koji su obrađeni u ovom radu, funkcija je samo validna za primenu na njima sličnim podacima. Obično se funkcije ovog tipa prilagođavaju specifičnom problemu i teško je stvoriti funkciju koja bi opšte rešila problem sa različitim vrstama ulaznih podataka. Sa tim na umu, ova funkcija se sastoji iz par koraka koji su prikazani na slici 8.2:



8.2 Koraci pri izvršavanju funkcije hdf5_2file_transformer()

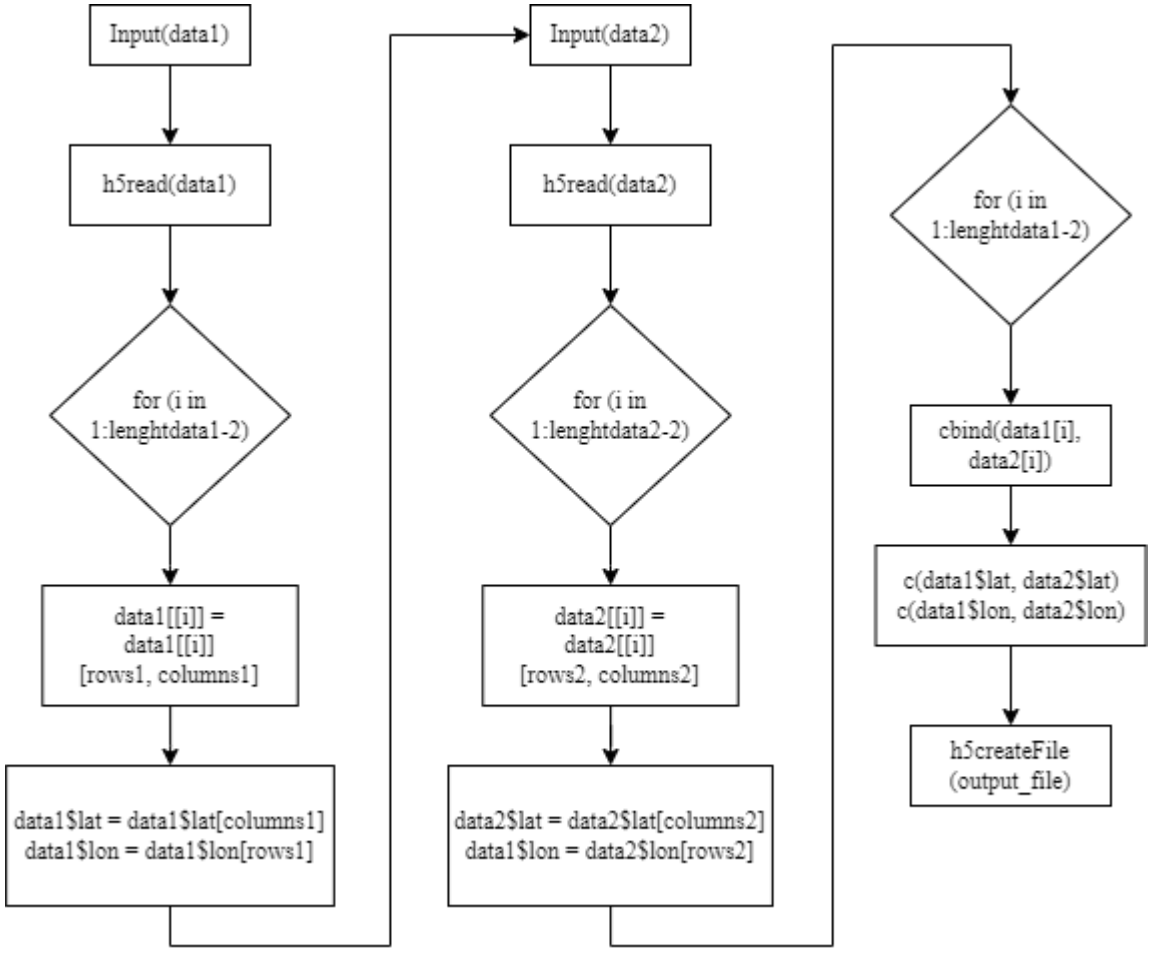
Kreirana funkcija se zove hdf5_2file_transformer(). Kao input se definišu putanje fajlova (folder u kome se nalazi prvi i drugi fajl), putanja podataka u fajlu (putanja se može dobiti korišćenjem funkcije napravljene u delu 8.1), broj kolona i redova koje je potrebno izdvojiti iz fajla (ove varijable prestavljaju nizove brojeva) i konačno putanja i ime izlaznog fajla koji će se kreirati na izlazu.

Za ovu funkciju korišćena je samo biblioteka rdhf5 za čitanje fajla, a za obradu i filtriranje podatka su korišćenje alatke koje dolaze sa R programskim jezikom (nisu korišćene dodatne biblioteke za obradu podataka).

Uslov da bi ova funkcija mogla da radi jeste da broj redova u oba fajla bude isti, u slučaju da broj redova nije isti, data frame od slepljenih podataka neće moći da se konstruiše i time izaziva grešku.

Kod se izvršava postepeno, na ulazu se definišu sve potrebne varijable, zatim se učitavaju podaci iz prvog fajla. Putanje ka podacima su određene korišćenjem funkcije iz dela 8.1. Zatim podaci iz prvog fajla ulaze u for petlju koja broj po dužini (broj redova u ovom slučaju) podataka umanjeno za dva. Te poslednje dve tačke su rezervisane za podatke lat i lon, koji nisu prestavljeni kao matrica podataka, već su prestavljeni kao vektori, što znači da se moraju drugačije tretirati. Fajl filtira sve kombinacije kolona i redova koji su definisani na ulazu, pa se zatim filtriraju lat in lon u odnosu na redove i kolone respektivno. Nakon završetka filtriranja prvih podataka, učitavaju se podaci iz drugog fajla, proces se ponavlja kao za podatke jedan, samo se uzimaju parametri sa sufiksom 2. Kada se i drugi fajl filtrira, ulazi se u petlju za slepljenje tih fajlova. Broji se po dužini fajla (broj redova) umanjeno za dva (lat, lon). Matrice se spajaju pomoću funkcije cbind() a podaci za lat i lon koriste operator concatenate c(). Zatim se novi h5 kreira pod imenom definisanom na ulazu. Mora se naglasiti da dužina fajla predstavlja broj redova u ovom slučaju a ne broj kolona, to je zato što objekat koji vraća funkcija h5read() je lista a ne data frame. Kastovanjem liste u data frame se gubi validnost podataka, pa je bilo neophodno raditi sa listama u ovom slučaju.

Tok izvršavanja koda može da se vidi na dijagramu 8.3:



Slika 8.3 Tok izvršavanja koda funkcije hdf5_2file_transformer()

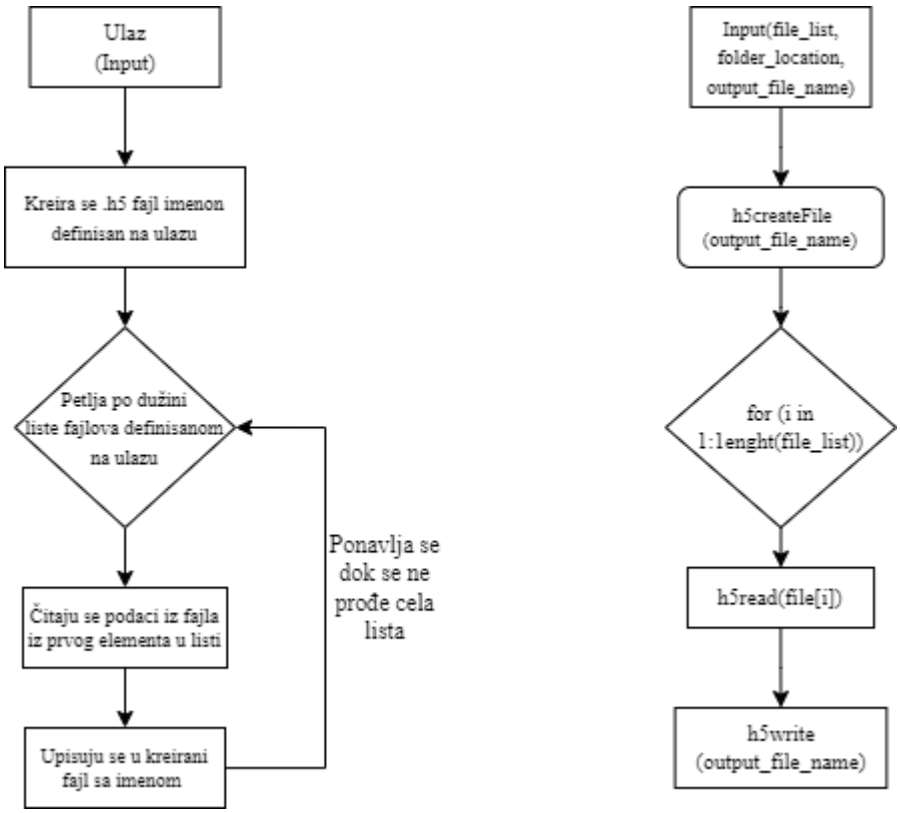
8.3 Metodologija izrade funkcija spajanja više fajlova u jedan

Funkcija koja je napravljena u ovom odeljku naziva se hdf5_file_combiner(). Funkcija spaja definisan broj fajlova u jedan podeljen po grupama. Broj koliko fajlova zavisi od ulaza koji definiše korisnik, znači proizvoljan je broj.

Uslov da bi ova funkcija radila jeste da ime fajla odgovara imenu grupe definisanu u h5 fajlu, to je i jedini uslov za ovu funkciju da bi radila. Ovaj uslov je nametnut da olakša proces kreiranja pedantnog izlaznog fajla, međutim po potrebi moguće je ovo promeniti u samom kodu.

Izlazni fajl se sastoji od tačno toliko grupa koliko je definisano na ulazu. U tim grupama su upisani podaci respektivnih fajlova.

Opis rada funkcije dat je u dijagramu toka 8.4:



Slika 8.4 Proces rada hdf5_file_combiner() funkcije

Kreira se fajl pod definisanim imenom sa funkcijom h5createFile() (prazan h5 fajl). Uzima se lista iz inputa fajlova, ulazi se u for petlju po dužini liste fajlova. Svaki fajl je jedan korak u petlji. Čita se lista fajlova iterativno pomoću funkcije h5read(), pa se ti podaci pišu u kreirani h5 fajl u grupu pod nazivom imena fajla definisanom na ulazu. Podaci u fajl se upisuju pomoću funkcije h5write().

9. Dobijeni rezultati

9.1 Rezultati benčmarkovanja veličine fajlova

Prvi korak je gledanje da li postoji bias između kolona i redova u ispitivanim formatima, u slučaju da postoji treba se uzeti u obzir dodatna optimizacija fajla pri upisivanju u taj format. Kao merilo da li ima bias jeste konkretno oblik krive koji se dobija, međutim uzete su i vrednosti aritmetičke sredine i medijana kao koristan indikator za postojanje biasa.

Aritmetička sredina je definisana:

$$\text{mean}(x) = \frac{1}{n}\left(\sum_{i}^{n} x_i\right) = \frac{x_1 + x_2 + \cdots + x_n}{n} \tag{9.1}$$

Broj n predstavlja ukupan broj merenja.

Mediana je definisana za dva slučaja, kad je n parno i kad je neparno.

Kad je n parno:

$$\text{median}(x) = \frac{x_{\frac{n}{2}} + x_{\left(\frac{n}{2}\right)+1}}{2} \tag{9.2}$$

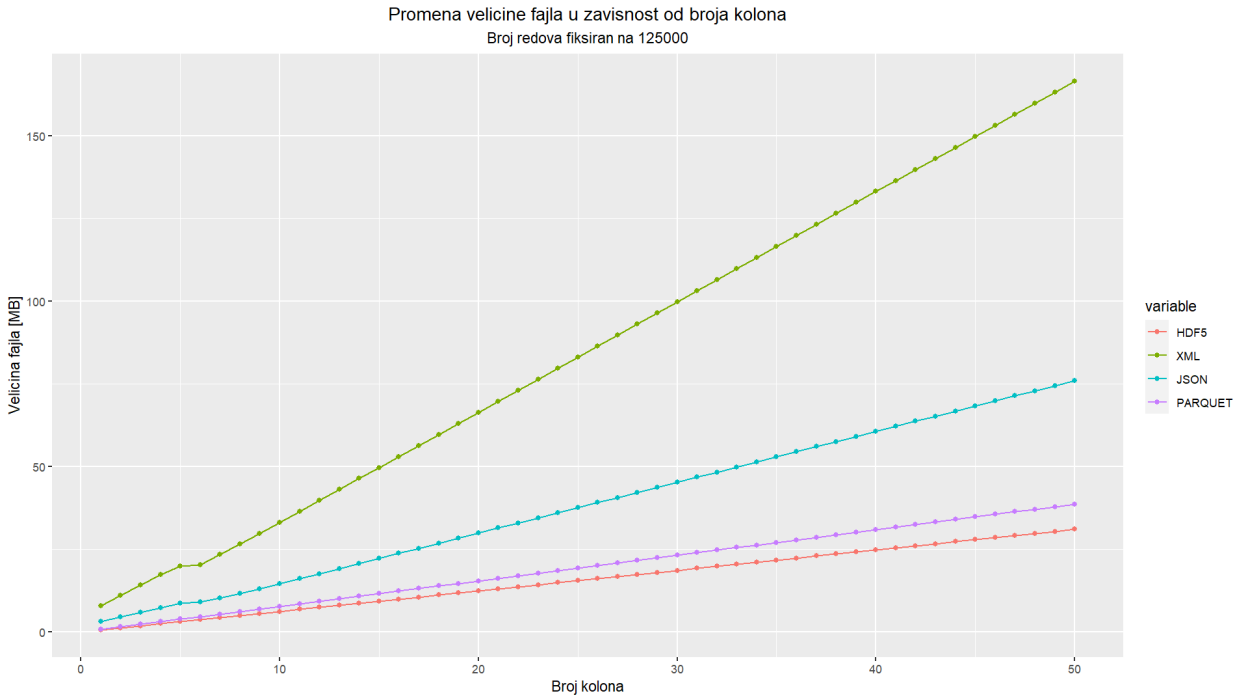
Kad je n neparno:

$$\text{median}(x) = x_{\frac{n+1}{2}}$$

Ako vrednosti između medijane i srednje vrednosti odstupaju dosta jedna od druge, to bi značilo da ipak postoji određeni bias u merenim podacima iako kriva to ne prikazuje.

Za vrednosti efikasnosti skladištenja će se porediti maksimalne vrednosti dobijeni u slučaju kada se menjaju i broj kolona i broj redova.

Prvo je razmatran slučaj kada je broj redova fiksiran na 125000, a broj kolona se menja od 1 do 50 sa korakom od 1. Dobijeni grafik je:



Slika 9.1 Grafik zavisnosti veličine fajla od broja kolona za fiksirane vrednosti redova

Kao što se vidi na grafiku 9.1, sve vrednosti su manje više linearne, što je i poželjno za svaki tip podataka. Dobijene vrednosti za sve formate su:

$$\text{mean}(x)_{\text{hdf5}} = 15.82\ \text{MB} \quad \text{median}(x)_{\text{hdf5}} = 15.82\ \text{MB}$$

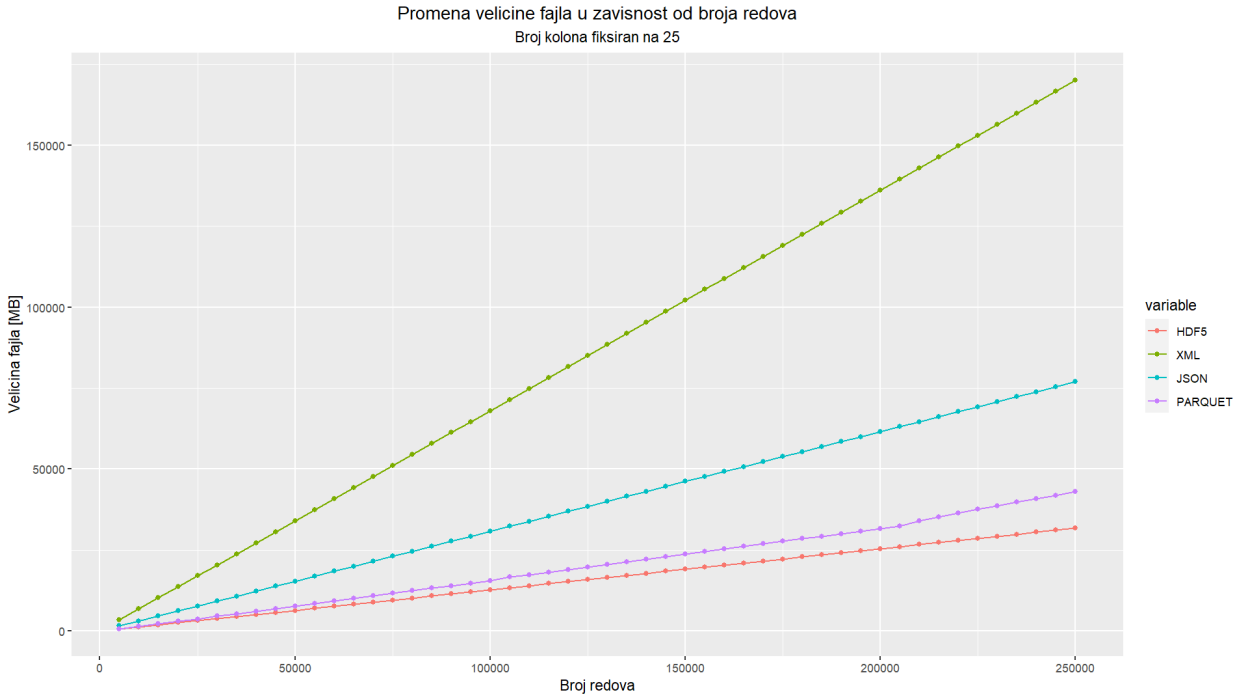
$$\text{mean}(x)_{\text{xml}} = 85.23\ \text{MB} \quad \text{median}(x)_{\text{xml}} = 84.76\ \text{MB}$$

$$\text{mean}(x)_{\text{json}} = 38.58\ \text{MB} \quad \text{median}(x)_{\text{json}} = 38.35\ \text{MB}$$

$$\text{mean}(x)_{\text{parquet}} = 19.71\ \text{MB} \quad \text{median}(x)_{\text{parquet}} = 19.71\ \text{MB}$$

Srednje vrednosti i vrednosti medijane ne odstupaju jedna od druge za korišćene formate značajno. Te sa tim činjenicama i sa samim izgledom krivih na nacrtanom grafiku može se zaključiti da ovi formati na ovim vrednostima merenja nemaju bias prema broju kolona.

Drugi slučaj koji je razmatran jeste kada je broj kolona fiksiran na 25, a broj redova se menja od 25000 do 250000 sa korakom od 5000. Dobijeni grafik je:



I u ovom slučaju vrednosti sa grafika 9.2, su manje više linearne, što je i poželjno za svaki tip podataka. Dobijene vrednosti za sve formate su:

$$\text{mean}(x)_{\text{hdf5}} = 15.82\ \text{MB} \quad \text{median}(x)_{\text{hdf5}} = 15.82\ \text{MB}$$

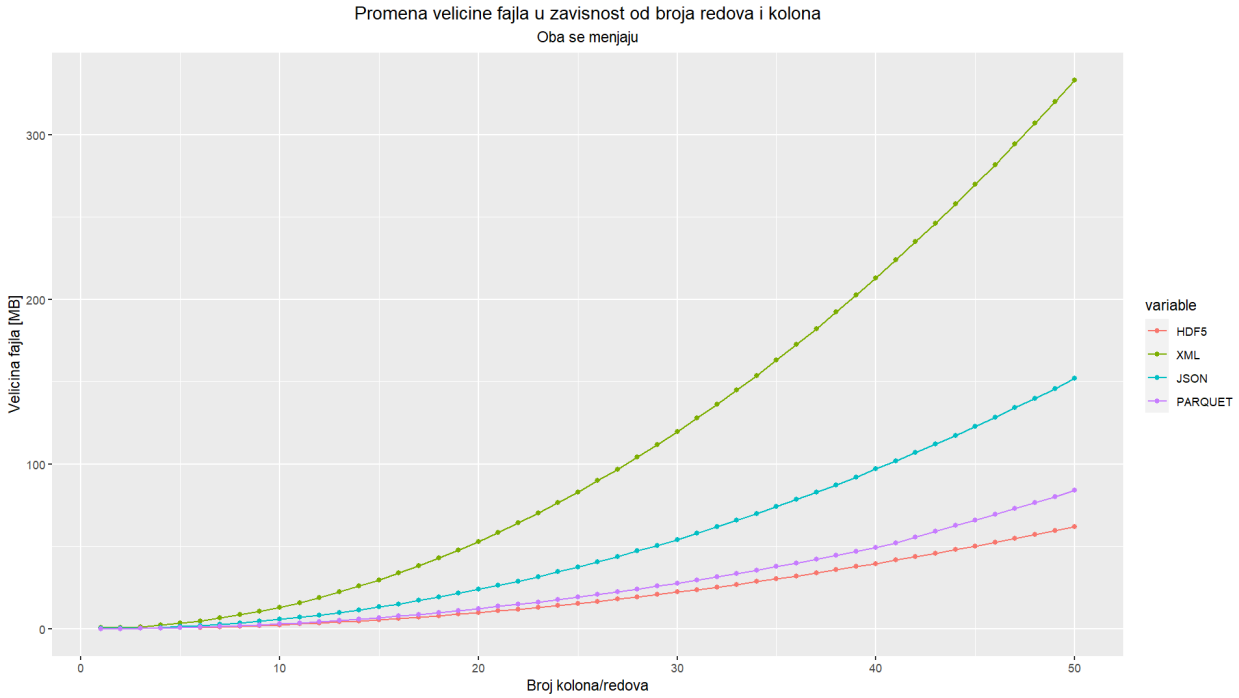
$$\text{mean}(x)_{\text{xml}} = 84.75\ \text{MB} \quad \text{median}(x)_{\text{xml}} = 84.75\ \text{MB}$$

$$\text{mean}(x)_{\text{json}} = 38.33\ \text{MB} \quad \text{median}(x)_{\text{json}} = 38.33\ \text{MB}$$

$$\text{mean}(x)_{\text{parquet}} = 20.02\ \text{MB} \quad \text{median}(x)_{\text{parquet}} = 19.71\ \text{MB}$$

Srednje vrednosti i vrednosti medijane ne odstupaju jedna od druge za korišćene formate značajno. Te sa tim činjenicama i sa samim izgledom krivih na nacrtanom grafiku može se zaključiti da ovi formati na ovim vrednostima merenja nemaju bias prema broju redova.

Na poslednjem grafiku se menjao broj kolona i broj redova. Broj kolona je išao od 1 do 50 sa korakom od 1, a broj redova od 25000 do 250000 sa korakom od 5000. Dobijeni grafik:



Slika 9.3 Grafik zavisnosti veličine fajla od broja redova i kolona

Dobijene maksimalne vrednosti za svaki format su:

$$\max(x)_{\text{hdf5}} = 62.04 \text{ MB}$$

$$\max(x)_{\text{xml}} = 333.07 \text{ MB}$$

$$\max(x)_{\text{json}} = 151.98 \text{ MB}$$

$$\max(x)_{\text{parquet}} = 83.96 \text{ MB}$$

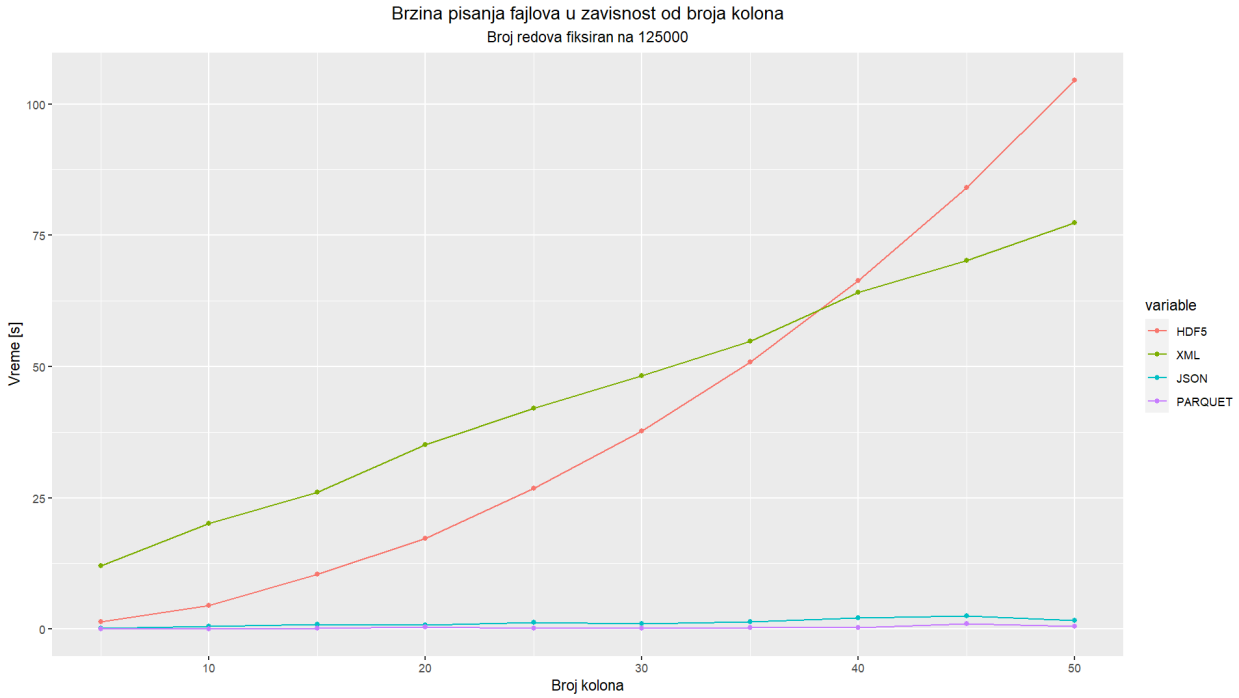
Kao što se može videti sa dobijenih rezultata, najloše perfomanse je imao xml format dok najbolje perfomanse je imao hdf5 format. Ovim rezultatima se može reći da u opsegu podataka od 1 do 50×250000 format hdf5 je najefikasniji za skladištenje tabelarnih podataka.

9.2 Rezultati benčmarkovanja brzine pisanja fajlova

U ovom delu biće razmatrane brzine pisanja fajlova za sva tri slučaja, kada su fiksirane vrednosti redova a menjaju se kolone, kada su fiksirane kolone a menjaju se redovi i kada se oba menjaju.

Kao merene vrednosti će se prikazati ukupno vreme potrebno da se napiše 10 tačaka svakog fajl formata za dati slučaj.

Za prvi slučaj kada su redovi fiksirani a kolone se menjaju, dobijeno je:



Slika 9.4 Grafik zavisnosti vremena od broja kolona za fiksirane vrednosti redova

Dobijene vrendosti su:

$$t_{\text{hdf5}} = 6 \text{ min } 43 \text{ sec}$$

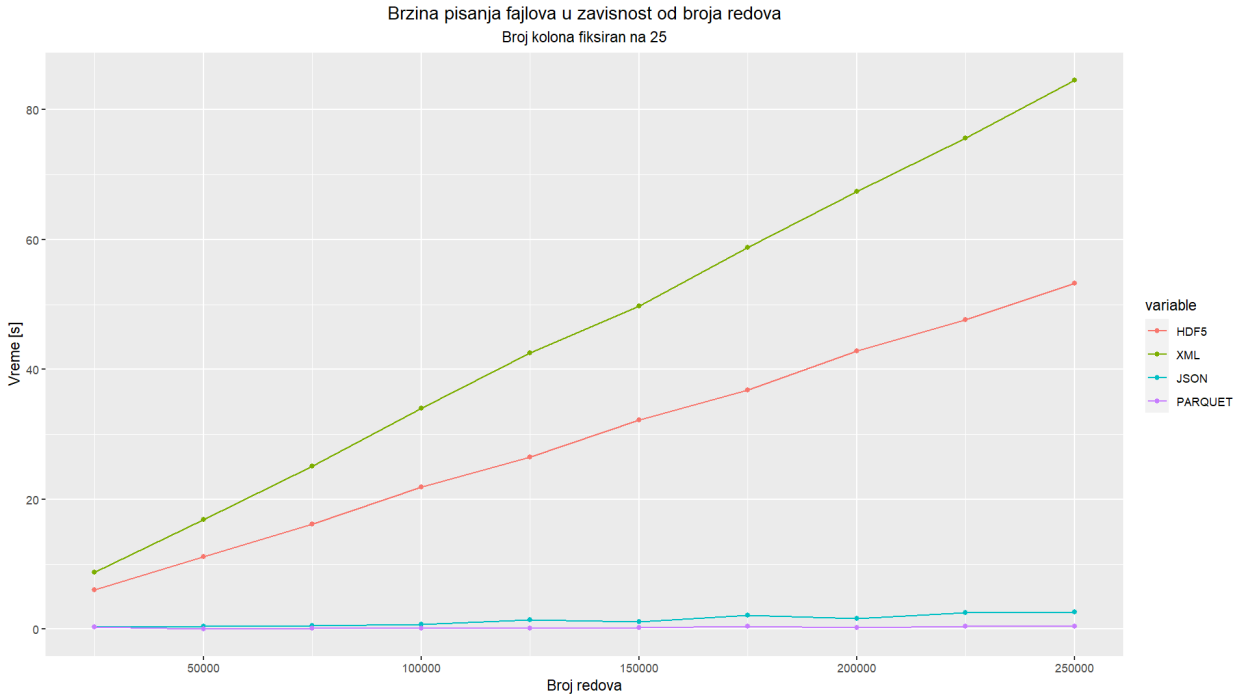
$$t_{\text{xml}} = 7\ min\ 30 \text{ sec}$$

$$t_{\text{json}} = 12 \text{ sec}$$

$$t_{\text{parquet}} = 3 \text{ sec}$$

Ovde se mogu videti ogromne razlike između formata, za ovaj slučaj ubedljivo najefikasniji je parquet format.

Za drugi slučaj kada su kolone fiksirani a redovi se menjaju, dobijeno je:



Slika 9.5 Grafik zavisnosti vremena od broja redova za fiksirane vrednosti kolona

Dobijene vrendosti su:

$$t_{\text{hdf5}} = 4 \text{ min } 54 \text{ sec}$$

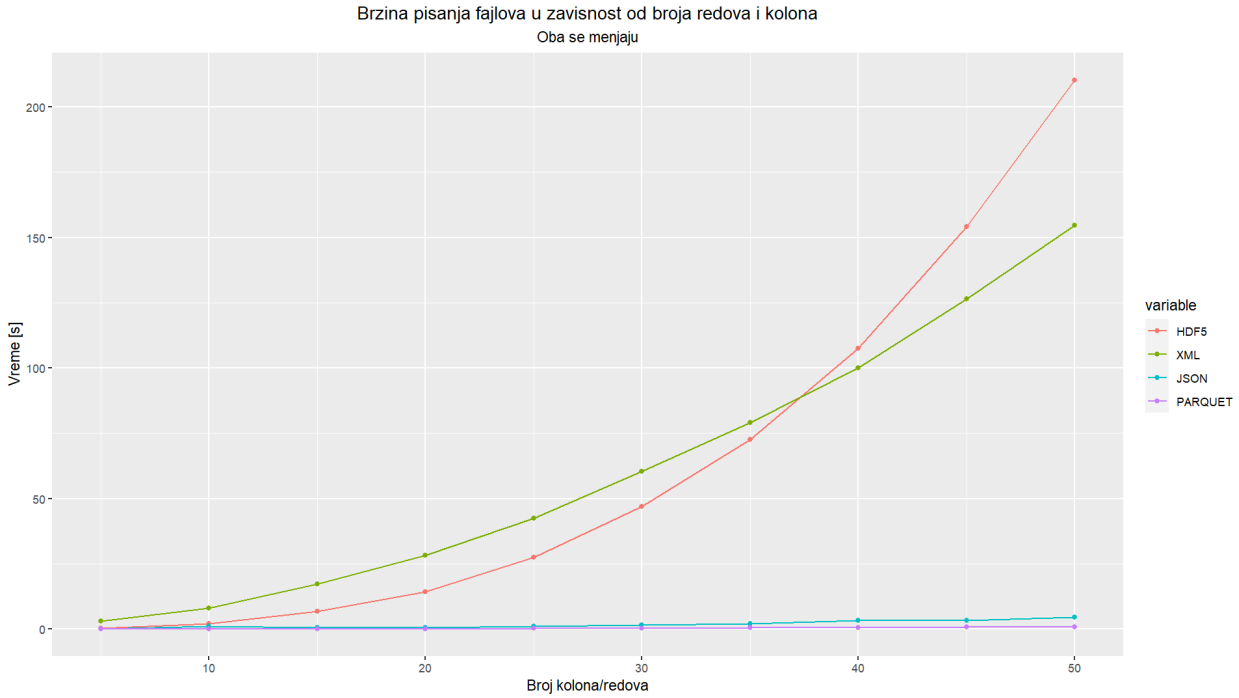
$$t_{\text{xml}} = 7 \text{ min } 43 \text{ sec}$$

$$t_{\text{json}} = 13 \text{ sec}$$

$$t_{\text{parquet}} = 3 \text{ sec}$$

I u ovom slučaju dominantni format jeste parquet format.

Poslednji slučaj gde se menjaju i redovi i kolone:



Slika 9.6 Grafik zavisnosti vremena od broja redova i kolona

Dobijene vrendosti su:

$$t_{\text{hdf5}} = 10 \text{ min } 41 \text{ sec}$$

$$t_{\text{xml}} = 10 \text{ min } 19 \text{ sec}$$

$$t_{\text{json}} = 18 \text{ sec}$$

$$t_{\text{parquet}} = 3 \text{ sec}$$

I u poslednjem slučaju pokazuje se da je, bar što se tiče brzine pisanja (pravljenja) fajlova, parquet format najbrži.

Međutim iako dobijeni rezultati su takvi, treba i dalje da se naglasi da je ovo brzina funkcija pisana koji su korišćeni u R programskom okruženju, i oni samo važe za R programsko okruženje.

9.3 Rezultati primenjenih alatki

Kao rezultat u ovom delu biće predstavljena ukupna veličina svih fajlova za godinu dana pre filtriranja i posle filtriranja i spajanja u jedan fajl.

Veličina svih fajlova 24 fajla pre primene alata na njih:

$$x_{pre} = 522 \text{ MB}$$

Veličina fajla koji sadrži sve filtrirane podatke iz svih 24 ulazna fajla jeste:

$$x_{posle} = 70.2 \text{ MB}$$

Ukupno vreme potrebno da se ceo proces izvrši je 24 sekunde.

10. Zaključak

Kada je u pitanju izbor pravog formata fajla za skladištenje podataka, mora se uzeti u obzir nekoliko faktora. Na primer, JSON (JavaScript Object Notation) je lagan format pogodan za razmenu strukturiranih podataka između aplikacija. Popularan je zbog svog lako čitljivog formata i kompatibilnosti sa različitim programskim jezicima. XML (eXtensible Markup Language) je takođe pogodan za strukturirane podatke i može se koristiti za razmenu podataka između različitih sistema. Parquet je stubni format za skladištenje dizajniran za efikasnu obradu podataka, posebno za velike skupove podataka. Podržava kompresiju i omogućava brz pristup za čitanje podataka. HDF5 (hijerarhijski format podataka) je format koji se obično koristi za skladištenje i upravljanje velikim i složenim skupovima podataka. Podržava kompresiju, chunking i druge funkcije za efikasno skladištenje i preuzimanje podataka. Odabir iz ovih opcija je primarno zavisio od toga šta je najbitniji uslov koji treba da se ispuni u konkretnom problemu.

Sa rezultatima iz ovog rada, što se tiče skladištenja podataka, HDF5 format se pokazao kao najefikasniji što se tiče efikasnosti skladištenja.

Slučaj kada je brzina bila primaran faktor, parquet format je pokazao neverovatno dobre perfomanse u R programskom okruženju.

Uzimanjem oba faktora u obzir, parquet fajl format se pokazao kao najbolja opcija.

U drugom delu cilj je bio da se dokaže da je moguće izdvojiti (isfiltrirati) podatke iz dva h5 fajla, spojiti povezane podatke, i zatim od ukupno 24 fajla konstruisati jedan fajl koji će da sadrži godinu dana vrednosti podataka noćnog neba za Republiku Srbiju. To je uspešno i urađeno.

11. Literatura

Garrett Grolemund, Hadley Wickham, Hands-On Programming with R: Write Your Own Functions and Simulations

Hadley Wickham, Garrett Grolemund, R for Data Science

Tom Marrs, JSON at Work: Practical Data Integration for the Web 1st Edition

Tom White, Hadoop: The Definitive Guide, 4th Edition

https://www.r-project.org

https://docs.hdfgroup.org/hdf5/develop/_h5_d_m__u_g.html#subsec_data_model_intro

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

https://docs.fileformat.com/web/json/

https://www.educba.com/xml-file-format/

https://docs.fileformat.com/web/xml/

https://parquet.apache.org/docs/

https://ladsweb.modaps.eosdis.nasa.gov

Link za korišćene podatke:

https://ladsweb.modaps.eosdis.nasa.gov/archive/allData/5000/VNP46A3/2022/

Github link za repozitorijom koji sadrži kreirani kod:

https://github.com/Engatsuo/Master.git