SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI SEMINARSKI RAD

PRIMJENA PARALELNE OBRADE U ANALIZI DRUŠTVENIH MREŽA

Ivan Validžić

Zagreb, travanj 2015.

Sadržaj

Uvod

Društvene mreže su jedan od najčešćih pojmova u svijetu Interneta. One virtualno okupljaju ljude i organizacije ovisno o tome za što su specijalizirane. Tako postoje poslovne, organizacijske i druge društvene mreže. Društvenu mrežu može se definirati kao društvenu strukturu koju sačinjavaju skupine društvenih aktera (osobe ili organizacije) i kompleksni skup između aktera. Cilj ovoga rada je obraditi temeljne principe obrade podataka te ih pokazati na jednom studijskom primjeru.

U prvom poglavlju navedena je definicija analize sadržaja i proces općenite analize sadržaja. Nakon teoretskog uvoda u općenitu analizu sadržaja, u drugom poglavlju se opisuje analiza sadržaja na društvenim mrežama, princip rada analize te prikladne metode. Treće poglavlje govori o dubinskoj analizi podataka, analizi koja se koristi za velike količine podataka. u četvrtom poglavlju se opisuju i postupci dubinske analize podataka. Takva analiza podataka je prikladna za društvene mreže jer se na njima nalaze ogromne količine podataka koje je najprikladnije obraditi primjenom paralelne obrade. Ta tema je obrađena u petom poglavlju. Paralelna obrada podataka se obavlja u raspodijeljenom datotečnom sustavu te je najčešći primjer korištenja paradigma *MapReduce.* Radni okvir koji će se koristiti i također opisati u ovom poglavlju je sustav *Hadoop*, pomoću softvera *Cloudera*. U šestom poglavlju je na studijskom primjeru opisan praktični dio gradiva koji je bio opisan u svim prethodnim poglavljima. Naposljetku, u sedmom poglavlju je opisana usporedba paralelne obrade sa klasičnom obradom podataka te su navedene prednosti paralelne obrade podataka.

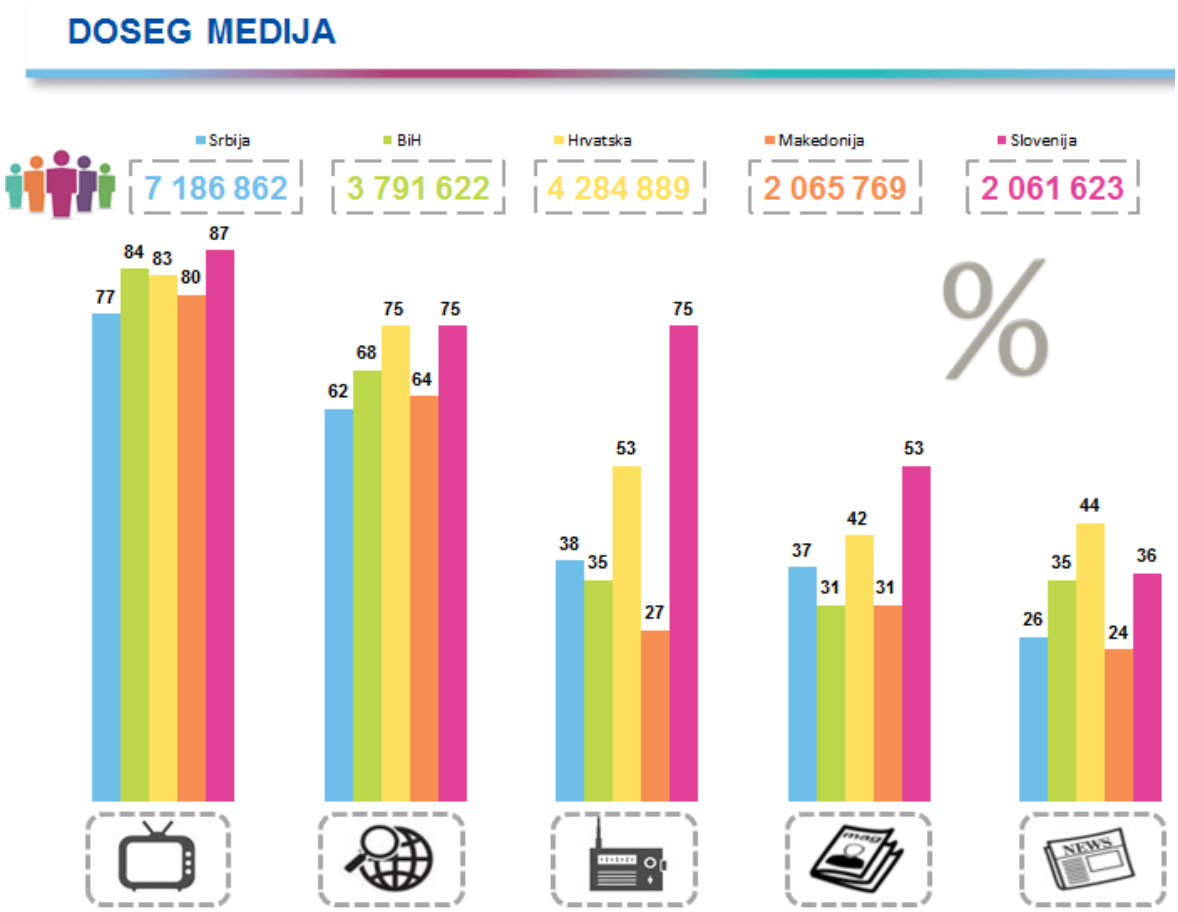
1. Analiza podataka

Analiza podataka u širem smislu je metoda istraživanja i analiziranja tekstualnih materijala. Temeljena je na društvenim znanostima te proučava učinke, utjecaje i svrhu komunikacije, međudruštvene integracije i degradacije, predviđa pojedina djelovanja i služi za sustavnu reprezentaciju prikupljenih podataka. Ozbiljniju analizu podataka je razvio sociolog Max Weber, koji se zalagao za trajnu i sistematičnu analizu podataka časopisa. No tek 70-ih godina prošlog stoljeća se analiza podataka počela koristiti kao važan alat u procjeni medijskih profila. Jedan od pionira u proučavanju analize podataka u medijima je američki sociolog Earl Babbie. Definirao ju je kao učenje o zabilježenim ljudskim komunikacijama, kao što su knjige, *web* stranice, slike i zakoni [1] [2].

Potpuniju definiciju analize podataka iznio je američki istraživač Ole Holsti, koji je definira kao bilo koji način za izradu zaključaka tako da se objektivno i sustavno identificiraju određena obilježja poruke. U samim počecima, analiza se smatrala metodologijom za pobijanje hipoteza. Time je postala poznata kao tehnika analize podataka. Ona omogućuje uključivanje velikih količina tekstualnih informacija te sistematično identificiranje njegovih karakteristika. Analiza podataka može se primijeniti na pisane, zvučne ili slikovne zapise. Kvantitativna analiza utvrđuje učestalost i ustrojstvo jedinica analize (sloga, riječi, sintagme, rečenice, slike i sl.) u nekome tekstu, dok je kvalitativna analiza podataka usmjerena na otkrivanje značenjskoga konteksta prebrojenih jedinica analize [2] [3].

U novije doba, analiza podataka je postala neizostavan element medijske procjene ili medijske analize. U analizama ovih kategorija, podaci iz analize podataka obično su kombinirani s medijskim podacima. Također je korištena za predviđanje i identificiranje trendova. S gledišta procjene, analiza podataka je kvaziprocjeniteljska, jer odluke analize podataka ne moraju se bazirati na vrijednosti izjava, ako subjekt istraživanja u prvi plan stavlja subjektivne doživljaje. Dakle, mogu biti temeljene na znanju svakodnevno proživljenih iskustava. Takvu analizu podataka ne smatramo procjeniteljskom. Odluke analize podataka temeljene na vrijednostima, kao što su studije, su procjeniteljske. Primjer procjeniteljskih odluka je sadržaj koji se nalazi na društvenim mrežama [4].

U počecima, koristeći prve novine krajem 19. stoljeća, analiza podataka radila se ručno, mjerenjem broja linija i dodijeljenoj količini medijskog prostora pojedinom predmetu. Razvojem računalnih tehnologija (najviše osobnih računala), popularnost računalnih metoda temeljenih na analizi je u eksponencijalnom porastu. Time svi sadržaji komunikacije u strojno čitljivom obliku mogu postati objekti sustavne analize tekstualnih podataka. Samim time sav sadržaj generiran od strane korisnika u strojno čitljivom obliku može biti sustavno analiziran sa bilo kojom metodom analize podataka. Primjer jedne analize je na Slici 1, gdje je prikazan doseg medija u državama jugoistočne Europe. Ovisno o vrsti medija, tvrtke koje se bave različitim medijima na temelju ovih podataka mogu planirati kvalitetniju promociju u budućnosti. Kao što se vidi na slici, najuspješniji medij u svim državama je televizija, a u stopu ga prati i korištenje Interneta. U preostalim vrstama medija se vide veće razlike. Najbolji primjer za to je radio, kod kojeg je najveća razlika u korištenju između država. Radio u Sloveniji svakodnevno koristi čak 75% ljudi, dok je u Makedoniji ta brojka samo 27%.



Slika 1. Doseg medija u državama JI Europe [5]

Općenito, primjena analize podataka se obavlja po idućim kategorijama:

- Rade se zaključci o prethodnicima komunikacije;
- Opisuju se i rade zaključci o karakteristikama komunikacije;
- Donose se zaključci o učincima komunikacije.

Kod procesa analize podataka, bitno je razlikovati stvarno i prikriveno značenje riječi, rečenica ili samih dijelova tekstova. Sadržaj generiran od strane autora ili govornika predstavlja njegov manifest. Manifest opisuje što je definitivno napisano, dok prikriveno značenje predstavlja što je autor želio reći, odnosno napisati. Prikriveno značenje predstavlja rizik kod analize podataka jer ne predstavlja činjenično, već fiktivno stanje koje nama nije dostupno. Zbog toga analize podataka mogu biti primijenjene samo na manifestni sadržaj: riječi, rečenice i dijelove teksta.

Proces analize podataka posvećuje pozornost na šest osnovnih pitanja analize pomoću kojih se detaljno analizira sadržaj određene teme:

- Koji podaci su analizirani?
- Kako su definirani?
- Koji je uzorak populacije iz koje su oni izvučeni?
- U kojem kontekstu su analizirani podaci?
- Koje su granice analize?
- Koji je cilj zaključaka?

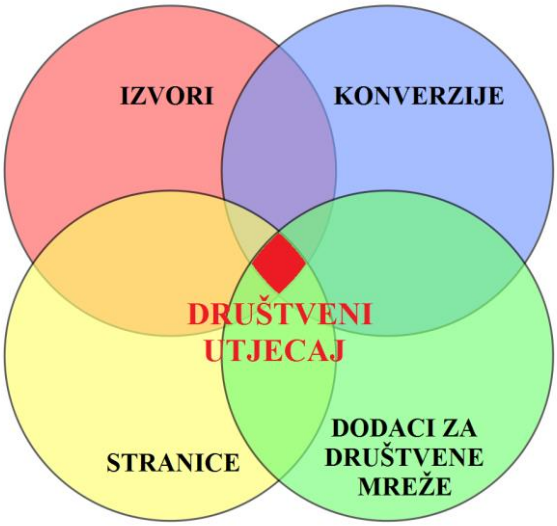
Pouzdanost u bilo kojem obliku istraživanja je stupanj točnosti analize rezultata koji se dobivaju nizom mjerenja. Svojstva pouzdanosti u analizi podataka su dosljednost i ponovljivost. Neke vrste pouzdanosti su :

- Unutarnje vrednovanje- različite skupine ispitanika rješavaju isti test;
- Ponavljajući testovi- ista skupina ispitanika je ispitivana u različitim trenucima;
- Paralelni oblici- različite skupine ispitanika u istim vremenskim trenucima pristupaju istom testu;
- Unutarnja dosljednost- postavljaju se različita pitanja sa istim značenjem [4].

2. Analiza podataka na društvenim mrežama

Društvene mreže predstavljaju pristupačni internetski servis koji istovremeno uključuje i tehnologiju i društvenu interaktivnost. Najčešće su zasnovane na internetskim i pokretnim tehnologijama. Osim međusobne povezanosti između više korisnika, imaju svoju ulogu u marketingu i promociji. Marketing i promocija su noviji termini na društvenim mrežama koji iz dana u dan dobivaju sve veći značaj. Mjerenje uspješnosti učinaka marketinga i promocije vrši se upravo analizom podataka na društvenim mrežama. Bitan čimbenik koji utječe na uspješnost učinaka marketinga i promocije je društveni utjecaj [6].

Društveni utjecaj definiran je sa četiri elemenata koji su prikazani na Slici 2.



Slika 2. Četiri elementa društvenog utjecaja na društvenim mrežama

Izvori predstavljaju cjelokupni sadržaj generiran na društvenim mrežama. Sadržaj se međusobno razlikuje po tipu i po značenju. Tipovi podataka mogu biti: video, slika, audio i običan tekst. Sadržaj po svojoj semantici postaje jedinstveni između ostalih podataka, a određuje samu informaciju i njezinu primjenu.

Konverzije predstavljaju interakcije na društvenim mrežama. Interakcije mogu biti između korisnika preko međusobne komunikacije ili preko dijeljenja podataka jedne društvene mreže u drugoj društvenoj mreži. Mjerenjem tog prometa može se steći bolji uvid u društveni utjecaj na društvenim mrežama.

Stranice predstavljaju mjesta na društvenim mrežama na kojima korisnici u sve većoj mjeri stupaju u interakciju sa sadržajem, dijele ga i raspravljaju o njemu.

Dodaci za društvene mreže predstavljaju dodatne mogućnosti koje nude društvene mreže. Ponuđene dodatne mogućnosti imaju za cilj olakšati dijeljenje podataka sa drugih internetskih lokacija na društvenu mrežu ili obratno.

2.1. Metode analize podataka na društvenim mrežama

Analiza podataka na društvenim mrežama usko je vezana uz znanstvenu disciplinu ekonometrije. Ekonometrija je relativno mlada grana ekonomske znanosti koja se intenzivnije počinje razvijati tridesetih godina prošlog stoljeća. Cilj metodologije ekonometrijskih istraživanja je, u biti, povezivanje ekonomske teorije i stvarnih podataka, koristeći teoriju i tehnike inferencijalne statistike kao spone između njih. Ekonometrija se može definirati kao društvena znanost u kojoj se instrumentarij ekonomske teorije, matematike i inferencijalne statistike primjenjuje u analizi ekonomskih fenomena. Oslanja se na matematičku ekonomiju pri specifikaciji relacija, a osnovni cilj joj je mjerenje, ocjenjivanje i testiranje relacija na temelju stvarnih opažanja.

Ekonometrija zapravo vrednuje u kojoj mjeri je ekonomska teorija konzistentna sa stvarnim podacima. Omogućuje dublje promicanje u bit stvarnih pojava i procesa, a vrednovane ekonometrijske relacije i modeli mogu poslužiti za definiranje ekonomskih parametara potrebnih za predviđanje tendencija budućih kretanja vrijednosti međusobno ovisnih ekonomskih varijabli. Ako u svojoj analizi uključuje sredstva i rezultate istraživanja teorijske ekonometrije tada prerasta u primijenjenu ekonometriju. Primijenjena ekonometrija koristi ekonometrijske metode u istraživanjima primijenjenim u okvirima određenih ekonomskih područja. Neke od ekonometrijskih metoda su:

- Regresija;
- Višestruka regresija;
- Faktorska analiza;
- Složena analiza varijacije [7].

2.2. Princip rada

Pitanje koje se prirodno postavlja je zašto je uopće važna analiza društvenih mreža. Općenito, mreže se analiziraju prema teoriji grafova. To znači da se sama mreža analizira kao graf. Najčešće analizirane mreže su financijske, mreže električne energije, političke mreže, mreže organizacija, prometne mreže, itd. Prebacivanje mreže u graf ne mora nužno značiti i njeno pojednostavljenje, ali iz njega se dobivaju važne mogućnosti za obradu. Ovakav pogled omogućava uočavanje veza koje prije nisu mogle biti uočene, pruža vizualizaciju, daje uvid u strukturu, omogućava mjerenje, ispitivanje povezanosti i optimizaciju. Može se također promatrati kako struktura utječe ne procese u njoj te što je bitno za društvene mreže, mogu se uspoređivati različiti parametri ovisno o povezanosti čvorova. Društvene mreže, čak ni one napravljene primarno za zabavu, obuhvaćaju puno veći spektar aktivnosti kao što je marketing, reklamiranje, širenje poslova i slično. Za takve aktivnosti izuzetno je važna analiza zbog izraženog ekonomskog faktora. Teorija grafova omogućava predikciju, organizaciju, drugačiji pogled na sigurnost mreže ili njenih djelova, prati širenje korisnih informacija, ali i štetnih komponenata kao što su virusi. Sve ovo omogućava, ovisno o interesima, odluku o pristupu i primjeni strategije na društvenu mrežu, ali i mreže općenito. Činjenice na kojima se temelji analiza:

- aktori i njihove akcije su međusobno ovisne;
- linkovi između aktora su kanali između koji se prenose resursi (materijalni i nematerijalni);
- modeli mreža se fokusiraju na davanje pogleda individualcu na strukturalni pogled na mrežu kao okolinu koja mu daje mogućnosti za djelovanje ali i ukazuje na moguća ograničenja;

3. Dubinska analiza podataka

Dubinska analiza podataka (engl. *data mining*) je proces traženja i analiziranja podataka u svrhu pronalaženja implicitne, ali potencijalno korisne informacije. Naziv *data mining* sugerira da se radi o postupcima rudarenja za grumenima znanja u gomilama podataka (engl. *mining for knowledge nuggets in mountains of data*). Ono uključuje označavanje, istraživanje i grupiranje velike količine podataka kako bi se izvukle relevantne informacije. Dubinska analiza podataka koristi mnoge metode kao što su statistička analiza, stabla odluke, neuronske mreže i grafička vizualizacija.

Osnovu za dubinsku analizu podataka čini konačan skup podataka dobivenih iz nekog procesa koji se odvija u stvarnom svijetu. Na temelju podataka o procesu koji su dostupni moguće je izraditi model ponašanja nekog određenog dijela procesa koji je od interesa. Dubinska analiza podataka se zbog svoje složenosti treba odvijati na računalu. U teoriji, svaki postupak dubinske analize podataka ostvariv je i bez računala, no praktično vrijeme potrebno za njegovo provođenje je u tom slučaju predugo.

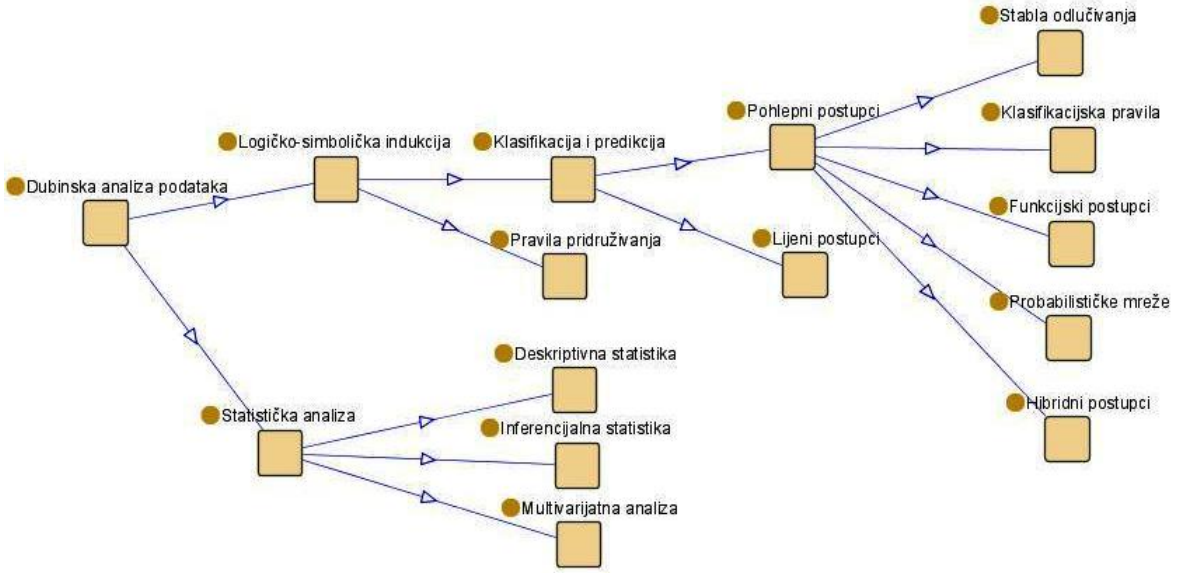
Pri analizi nekog procesa pomoću postupaka dubinske analize podataka pretpostavljamo sljedeće nužne preduvjete:

- Proces koji analiziramo je malen, ograničen i točno određen dio svijeta;
- Cilj analize je jasan;
- Podaci koji su dostupni dovoljno su kvalitetni za opis procesa;
- Podaci su prikazani u obliku tablice varijabli i objekata.

Nužno je da analizirani proces bude ograničen na jedan malen dio, budući da su korišteni računalni postupci prostorno i vremenski zahtjevni. Prije nego što se krene u postupak dubinske analize podataka, nužno je znati točan cilj istraživanja. Za svaki proces koji se promatra, podaci koji su dostupni moraju ga dovoljno dobro opisivati. Ako su podaci o procesu nedostatni ili neprimjereni, ili ako je previše podataka nepoznato, analiza se ne može zadovoljavajuće provesti ili dovodi do netočnih modela. Gotovo svi postupci dubinske analize podataka podrazumijevaju da su podaci predočeni u obliku tablice (ako nisu u tom obliku, najprije se pretvaraju u taj oblik, a tek onda se analiziraju). Takva tablica sadrži objekte i atribute po kojima se dotični objekti mjere ili opisuju [8].

4. Podjela postupaka dubinske analize podataka

Podjela postupaka kod otkrivanja znanje u skupovima podataka je prikazana na Slici 3. Potrebno je napomenuti da je ova podjela prilično gruba i nije jedina moguća, no njome se ipak obuhvaća daleko najveći dio dostupnih postupaka danas.



Slika 3. Podjela postupaka

Na najvišoj razini, dubinsku analizu podataka možemo podijeliti na statističku analizu podataka i na indukciju znanja logičko-simboličkim postupcima. Iako je granica među tim područjima često vrlo diskutabilna, može se reći da je statistička analiza namijenjenija opisivanju osnovne prirode procesa koji se promatra i većinom je deduktivne naravi, dok logičko-simbolička indukcija problemu prilazi na praktičniji način i gradi općenitije modele s ciljem iskorištavanja rezultata analize.

Statističku analizu može podijeliti na deskriptivnu, inferencijalnu i multivarijatnu analizu. Deskriptivna statistika ima za zadatak na što bolji način opisati postojeće podatke. Obuhvaća opis putem mjera središnje težnje, mjera rasipanja kao i teorijskih ili empirijskih razdioba za vrijednosti neke varijable. Inferencijalna statistika istražuje načine dobivanja znanja o populaciji iz uzorka koji nam je dostupan. Uključuje procjenu intervala pouzdanosti, testiranje hipoteze, pogreške pri procjeni, kao i niz postupaka za analizu zavisnosti i međuzavisnosti varijabli. Multivarijatna statistička analiza obuhvaća postupke koji se provode nad više od jednom varijablom.

Logičko-simboličku indukciju može se podijeliti prema tome koji je cilj istraživanja na klasifikacijsko-predikcijske postupke i pravila pridruživanja (engl. *association rules*). Pravila pridruživanja za svoj cilj imaju pronalaženje skrivenih odnosa među raznim atributima, dok je za klasifikacijsko-predikcijske postupke najčešća pretpostavka jednog ili nekoliko ciljnih (kriterijskih, zavisnih) atributa.

Klasifikacijsko-predikcijski postupci mogu se podijeliti s obzirom na način predočavanja objekata na pohlepne (engl. *eager*) i na lijene (engl. *lazy*) postupke. Kod lijenih postupaka, klasifikacija određenog objekta u ciljni razred se odgađa dok god ne dođe upit upravo za tog pojedinca. Za razliku od toga, kod pohlepnih postupaka klasifikacija svih objekata provodi se najčešće odjednom. Lijeni postupci klasificiraju nove objekte tako da analiziraju samo njima slične objekte, dok zanemaruju objekte koji nisu slični novom objektu. Zbog svega navedenog, lijeni postupci brzo uče, ali sporo klasificiraju nove objekte, dok pohlepni postupci uče sporije budući da u fazi učenja stvaraju jednu globalnu aproksimaciju podataka, ali zato klasificiraju brže nove objekte.

Pohlepni postupci za klasifikaciju i predviđanje mogu se podijeliti na razne načine. Dijele se na stabla odlučivanja, indukcijska ili klasifikacijska pravila, vjerojatnosne ili Bayesove mreže, funkcijske postupke i hibridne postupke. Stabla odlučivanja su uz funkcijske postupke možda najšire područje budući da uključuju nekoliko desetaka načina izgradnje lako interpretabilnih stabala koji služe bilo klasifikaciji vrijednosti kategoričkih bilo regresijskom predviđanju numeričkih atributa. Indukcijska ili klasifikacijska pravila su takva pravila koja nastoje inducirati novo znanje na temelju dovoljno dobrog prekrivanja postojećeg činjeničnog prostora, najčešće izvedenog putem podijeli-pa-vladaj strategije. Bayesove mreže pretpostavljaju da je znanje koje imamo o nekom događaju u svijetu opisano s vjerojatnosti pojave tog događaja. Svaki događaj u modelu je moguć i ima makar malenu i uvijek postojeću vjerojatnost pojave. Ta vjerojatnost se zadaje *a priori* ili se izvodi iz podataka. Kod funkcijskih postupaka, skup postupaka koji koriste određeni oblik funkcije da bi opisali odnos ulaz-izlaz u sustavu za klasifikaciju ili predviđanje nazivaju se funkcijski klasifikatori. U općenitom obliku, funkcija preslikavanja ulaznih podataka u izlazne rezultate pri klasifikaciji može biti bilo koja nelinearna funkcija. Ipak, zbog prirode svijeta često su odnosi između ciljne i prediktivne varijable linearni. Hibridni postupci su dobili ime po tome što kombiniraju dva ili više osnovna klasifikacijsko-predikcijska postupaka s ciljem dobivanja što učinkovitijeg modela [8].

5. Paralelna obrada podataka

Količina podataka koju danas prikupljaju najpopularnije i najkorištenije *web*-usluge zahtijeva potpuno drukčije programske sustave i modele programiranja od onih na koje su programeri naviknuli. Nije neuobičajeno da volumen dnevnika velikih mrežnih sjedišta raste brzinom od 1 TB na dan. Prirodno je za očekivati da ce ovaj trend jos samo rasti s padom cijena sklopovlja i pojavom boljih i bržih načina stvaranja podataka. Moderne primjene dubinske analize podataka, koje se često nazivaju "big-data" analize, zahtijevaju brzo upravljanje golemom količinom podataka. U mnogim od tih primjena, podaci su izuzetno pravilni te postoji dovoljno prilika za iskorištavanje paralelne obrade podataka. Paralelna obrada je postupak kod kojega se više instrukcija obrađuje istovremeno. Obrada se zasniva na principu da se veliki problemi gotovo uvijek mogu podijeliti na manje te onda obraditi istovremeno (paralelno). Ovaj tip obrade postoji u nekoliko oblika:

- Paralelizam na razini bita;
- Paralelizam na razini instrukcije;
- Podatkovni paralelizam;
- Zadaćni paralelizam.

Neki važniji primjeri paralelne obrade podataka koji se koriste u svakodnevnom životu su:

- Rangiranje *web* stranica po važnosti, što uključuje ponovljeno množenje matrica-vektora gdje su dimenzije u milijardama;
- Pretraživanje "prijatelja" na društvenim mrežama, koja uključuju grafove sa stotinama milijuna čvorova i milijardama rubova.

Rješavanje takvih primjena je dovelo do evolucije softverskog stoga (engl. *software stack*). Takvi programerski sustavi su dizajnirani kako bi dobili svoj paralelizam iz računalnih grozdova (engl. *computing clusters*), a ne iz "super-računala". Softverski stog koristi novi oblik datotečnog sustava, raspodijeljeni datotečni sustav (engl. *distributed file system*, DFS) [9].

5.1. Raspodijeljeni datotečni sustav

DFS (engl. *Distributed Filesystem*) je raspodijeljeni datotečni sustav za spremanje i slijedno čitanje vrlo velikih datoteka u spletu računala. Oblikovan je za brzo slijedno (engl. *streaming*) čitanje, jer je za programski model *MapReduce* važnija optimizacija čitanja cijelog skupa podataka od optimizacije vremena pristupa prvom zapisu u skupu podataka. Dodatno, DFS se izvršava na spletu običnih računala (engl. *commodity hardware*) gdje je vjerojatnost kvara vrlo velika. Osim toga, DFS osigurava i ispravan rad sustava bez obzira na prisutnost kvarova ili grešaka u komunikaciji.

DFS sprema datoteke u raspodijeljeno spremište koristeći apstrakciju blokova. Veličina bloka je velika (u odnosu na veličinu bloka običnih datotečnih sustava) zbog optimizacije vremena čitanja. Nakon što je blok pozicioniran, cijeli blok se može učitati deklariranom brzinom čitanja. Prednosti korištenja blokova fiksne veličine u raspodijeljenom datotečnom sustavu su jednostavnost ostvarenja te mogućnost spremanja datoteka čija veličina nadmašuje kapacitet bilo kojeg diska u spletu računala.

Dostupnost i otpornost sustava na pogreške osigurava se replikacijom blokova u spletu računala. Ako promatrani blok postane nedostupan, DFS održava replike promatranog bloka i njihovo čitanje i održavanje je transparentno korisnicima datotečnog sustava [10].

5.2. Paradigma *MapReduce*

MapReduce je programski model za raspodijeljenu obradu podataka sa svojstvom linearnog razmjernog rasta. *MapReduce* se može promatrati kao sustav za izvršavanje upita u pozadini (engl. *batch query processor*). Programski model *MapReduce* može se direktno programirati u Javi, ali je u mnogim programskim jezicima (Python, Perl...) implementiran kao biblioteka. *MapReduce* model obrade podataka funkcionira po idućim koracima:

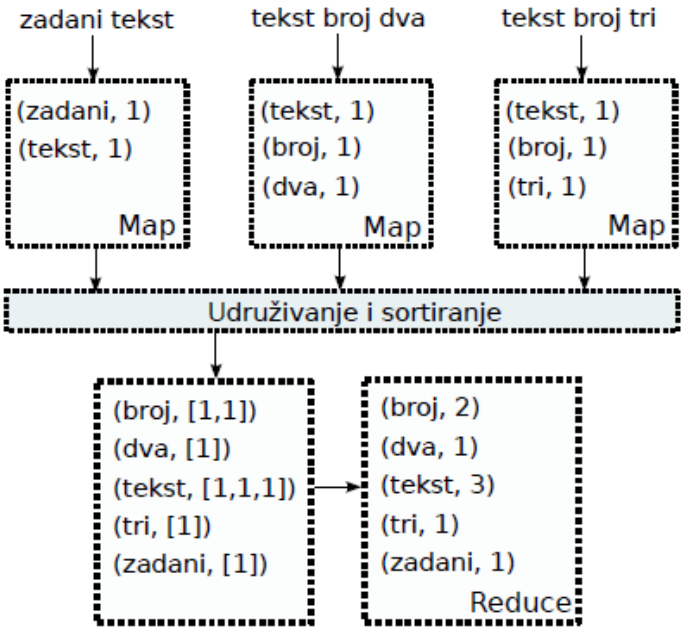
- Podjela ulaznih podataka na manje dijelove;
- Obrada dijelova tako da se dobije međurezultat;
- Kombinacija međurezultata kako bi se dobili krajnji rezultati obrade.

Kako programski model *MapReduce* odlikuje svojstvo linearnog razmjernog rasta, podaci koji se obrađuju paralelno mogu biti jako veliki te je ovaj način pogodan za analizu velikih

količina podataka. Bez obzira na količinu podataka, sustav obrađuje cijeli skup podataka za svaki upit. Obrada se definira dvjema funkcijama:

- *Map*- transparentno čitanje „sirovih“ podataka iz raspodijeljenog datotečnog sustava, filtriranje te generiranje parova ključ-vrijednost;
- *Reduce*- obrada udruženih i sortiranih parova generiranih *Map* funkcijama te generiranje izlaza u obliku parova ključ-vrijednost.

Funkcija *Map* se poziva nad svakim dijelom ulaznih podataka i proizvodi međurezultat u obliku ključ/vrijednost. Zatim se grupiraju sve vrijednosti povezane s ključem, dok naposljetku funkcija *Reduce* obrađuje sve vrijednosti povezane s istim ključem kako bi se dobio konačan rezultat obrade. Kanonski primjer *MapReduce* programa prikazan na Slici 4, a radi se o programu za brojanje ponavljanja riječi u zadanom tekstu.



Slika 4. Kanonski primjer

Zadani tekst podijeljen je u tri particije. Funkcija *Map* paralelno čita odgovarajuće particije i za svaku riječ unutar teksta generira par (riječ, 1) koji označava da je pročitana promatrana riječ. Nakon sto su pročitane sve particije, sustav *MapReduce* udružuje i sortira generirane parove u obliku para (riječ, [1, ... , 1]). Ulaz u funkciju *Reduce* su udruženi i sortirani parovi iz prethodnog koraka. Broj ponavljanja pojedine riječi predstavlja duljina liste u drugom elementu para.

Za razliku od sustava za upravljanje relacijskim bazama podataka (engl. *Relational database management system*, RDBMS), *MapReduce* model pokazuje svojstvo razmjernog rasta jer funkcije *Map* i *Reduce* ne ovise o veličini ulaznog skupa podataka niti o veličini spleta računala na kojem se sustav izvršava. *MapReduce* model obrađuje cijeli skup podataka za vrijeme izvršavanja upita, dok RDBMS sustavi obično održavaju dodatne strukture podataka (npr. B-stabla) koje ubrzavaju izvršavanje upita ili ažuriranje manje količine zapisa, ali značajno usporavaju ažuriranje većine zapisa u bazi. Dodatno, *MapReduce* programski model je zamišljen za obradu nestrukturiranih (ili polustrukturiranih) podataka kao sto su tekst ili binarni podaci [9] [10].

5.3. Sustav *Hadoop*

Hadoop je nastao 2004. godine kao otvoreno ostvarenje *MapReduce* radnog okvira razvijenog u tvrtki Google. Radni okvir *Hadoop* razvijen je za *ad-hoc* paralelnu obradu nestrukturiranih podataka koristeci programski model *MapReduce* i raspodijeljeni datotečni sustav HDFS (*Hadoop Distributed Filesystem*). Danas je *Hadoop* prihvaćen u industriji kao standard pozadinske obrade podataka sa svojstvom razmjernog rasta. Između ostalih, koriste ga Yahoo!, Facebook, Twitter, Last.fm i mnogi drugi.

Radni okvir *Hadoop* je programski sustav za analiziranje velikih količina podataka koji se nalaze u pouzdanom i raspodijeljenom spremistu. Velike količine podataka odnose se na baze podataka programskih sustava koji učinkovito upravljaju volumenom podataka na Internet razini (engl. *Internet scale*), npr. sustavi za *web* indeksiranje ili dubinsku analizu podataka. Radni okvir *Hadoop* ostvaren je u programskom jeziku Java.

Jezgru radnog okvira čine programski model *MapReduce* i raspodijeljeni datotečni sustav HDFS. Koristeći osnovne *Hadoop* primitive, razvijeno je nekoliko programskih sustava koji olakšavaju korištenje Hadoopa ili ostvaruju potpuno nove funkcionalnosti. Sustav *Pig* olakšava pisanje *MapReduce* programa uvodeći poseban jezik *Pig Latin* te okolinu za izvršavanje programa. *Pig* prevodi programe iz jezika više razine u *MapReduce* programe koje izvodi na spletu računala. *HBase* je raspodijeljena baza podataka razvijena po uzoru na *BigTable* bazu podataka koja se koristi u tvrtki Google. *HBase* je namijenjen za čitanje i pisanje zapisa u stvarnom vremenu za vrlo velike skupove podataka. *ZooKeeper* je pouzdani sustav za koordinaciju raspodijeljenih primjenskih sustava. [10]

5.3.1. Slanje poslova na izvođenje

MapReduce posao šalje se na izvođenje koristeći *Hadoop* klijentsku aplikaciju *JobClient*. Klijentska aplikacija traži od glavnog čvora(*JobTracker*) novi jedinstveni identifikator posla i izračunava particije ulaznih podataka. Nakon što su ulazni podaci podijeljeni u particije i nakon što su provjereni parametri posla (npr. postojanje izlaznog direktorija), *JobClient* kopira komponente posla u raspodijeljeni datotečni sustav u direktorij naziva jednakog identifikatoru posla generiranom u prvom koraku. Komponente posla uključuju JAR[1] arhivu sa samim programom, konfiguracijske datoteke te particije ulaznih podataka.

Nakon što su komponente posla postali dostupni u raspodijeljenom datotečnom sustavu, posao se sprema u interni red poslova (engl. *job queue*). Raspoređivač poslova (engl. *job scheduler*) zatim dohvaća posao te inicijalizira zadatke potrebne za njegovo izvođenje. Inicijalizirani zadatci uključuju *Map* funkcije (po jednu *Map* funkciju za svaku particiju ulaznih podataka) i *Reduce* funkcije (broj *Reduce* funkcija definiran je u konfiguracijskoj datoteci).

5.3.2. Izvođenje zadataka

Izvođenje zadataka je u potpunosti orkestrirano glavnim čvorom. Prije samog izvođenja pojedinih zadataka, *JobTracker* mora izabrati kojem poslu pripadaju zadatci koje će izvoditi. Pretpostavljeni raspoređivač bira posao koji je prvi stigao u red poslova. Nakon što odabere posao, *JobTracker* dodjeljuje zadatke koji čine odabrani posao slobodnim radnicima. Radnik (*TaskTracker*) periodički javlja svoje stanje glavnom čvoru. Stanje uključuje informaciju o slobodnim „utičnicama“ (engl. *slots*) za *Map* i *Reduce* zadatke. Važna optimizacija događa se kod dodijeljivanja *Map* zadataka. *Map* zadaci pokušavaju se dodijeliti čvorovima radnicima na kojima se nalaze podaci koje obrađuje upravo dodijeljeni zadatak. Na taj način se izbjegava skupa mrežna komunikacija jer su podaci lokalni *Map* zadatku. S ciljem optimizacije preklapanja čitanja i obrade podataka, radni okvir *Hadoop* pokreće više *Map* i *Reduce* zadataka konkurentno na čvorovima radnicima. Pretpostavljeni broj utičnica za zadatke je 4 (2 za *Map* i 2 za *Reduce* zadatke).

---

[1] JAR (engl. Java Archive)- vrsta podatkovnog paketa pisanog u programskom jeziku Javi

Nakon što je čvoru radniku dodijeljen zadatak, radnik dohvaća JAR arhivu s programom, pokreće posebnu instancu Java prividnog stroja (JVM) za izvođenje dodijeljenog zadatka. *MapReduce* programi mogu trajati satima, stoga čvorovi radnici periodički dojavljuju napredak izvršavanja zadatka. Posao je završio kada čvor radnik koji izvršava posljednji zadatak u poslu javi glavnom čvoru da je završio sa izvršavanjem dodijeljenog zadatka.

5.3.3. Oporavak od pogreške

Izvođenje zadataka je podložno programskim ili sklopovskim pogreškama koje se mogu manifestirati na razne načine. Jedna od prednosti radnog okvira *Hadoop* je praćenje stanja poslova i zadataka, rukovanje pogreškama i zastojima u radu te omogućavanje izvršavanja posla u potpunosti u nesigurnoj okolini računalnog spleta. Pogrešku u čvoru radniku može uzrokovati iznimka za vrijeme izvodenja *Map/Reduce* zadatka ili neka druga pogreška u JVM sustavu. Čest slučaj je i pojava sporih radnika ili radnika u zastoju. Kada glavni čvor primi periodičku poruku o stanju radnika s dojavom pogreške, on ponovo pokreće neuspješan zadatak (izbjegavajući pokretanje zadatka na istom čvoru radniku gdje se dogodila pogreška). Ako glavni čvor uopće na primi periodičku poruku o stanju radnika, glavni čvor briše radnika u kvaru iz skupa radnika za raspoređivanje poslova (engl. *tasktrackers pool*). Dodatno, dostupnost sredstava u sustavu je vrlo visoka koristeći raspodijeljeni datotečni sustav s replikacijom blokova (npr. stupanj replikacije sredstava poslova je 10). Na taj način osim očitih prednosti smanjenja mrežne komunikacije tijekom izvođenja zadataka, povećava se i pouzdanost sustava redundancijom podataka. *Hadoop* trenutno ne razmatra pogrešku u glavnom čvoru koji predstavlja jedinstvenu točku pada sustava (engl. *single point of failure*). Jedna od mogućnosti zaštite je uvođenje redundancije korištenjem sustava *ZooKeeper* koji bi upravljao spletom čvorova i određivao primarni (glavni) čvor.

6. Studijski primjer: brojanje riječi

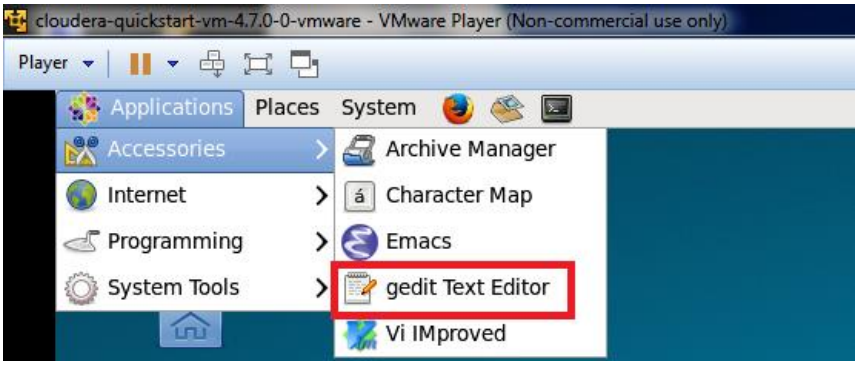
Nakon uspješne instalacije, čiji postupak se nalazi na kraju ovog rada (u Privitku A), na jednom primjeru će se objasniti kako *MapReduce* točno funkcionira te kako se treba podesiti konfiguracija kako bi sve ispravno radilo. Nakon otvaranja virtualnog stroja *VMware Player* i pokretanja softvera *Cloudera*, koji je zapravo pojednostavljena simulacija radnog okvira *Hadoop*, obradit će se primjer koji broji riječi u datotekama. U nastavku je prikazan kôd pomoću kojega će se u programskom jeziku Java implementirati spomenuti program brojanja riječi.

```
package org.myorg;

import java.io.IOException;
import java.util.regex.Pattern;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.log4j.Logger;

public class WordCount extends Configured implements Tool {

	private static final Logger LOG = Logger.getLogger(WordCount.class);

	public static void main(String[] args) throws Exception {
		int res = ToolRunner.run(new WordCount(), args);
		System.exit(res);
	}

	public int run(String[] args) throws Exception {
		Job job = Job.getInstance(getConf(), "wordcount");
		job.setJarByClass(this.getClass());
		FileInputFormat.addInputPath(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(args[1]));
		job.setMapperClass(Map.class);
		job.setReducerClass(Reduce.class);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(IntWritable.class);
		return job.waitForCompletion(true) ? 0 : 1;

	}
```

```
	public static class Map extends
		Mapper<LongWritable, Text, Text, IntWritable> {
	private final static IntWritable one = new IntWritable(1);
	private Text word = new Text();
	private long numRecords = 0;
	private static final Pattern WORD_BOUNDARY = Pattern
		.compile("\\s*\\b\\s*");

	public void map(LongWritable offset, Text lineText, Context
context)
		throws IOException, InterruptedException {
		String line = lineText.toString();
		Text currentWord = new Text();
		for (String word : WORD_BOUNDARY.split(line)) {
			if (word.isEmpty()) {
				continue;
			}
			currentWord = new Text(word);
			context.write(currentWord, one);
		}
	}
	}
	public static class Reduce extends
		Reducer<Text, IntWritable, Text, IntWritable> {
	@Override
	public void reduce(Text word, Iterable<IntWritable> counts,
			Context context) throws IOException,
InterruptedException {
		int sum = 0;
		for (IntWritable count : counts) {
			sum += count.get();
		}
		context.write(word, new IntWritable(sum));
	}
	}
}
```

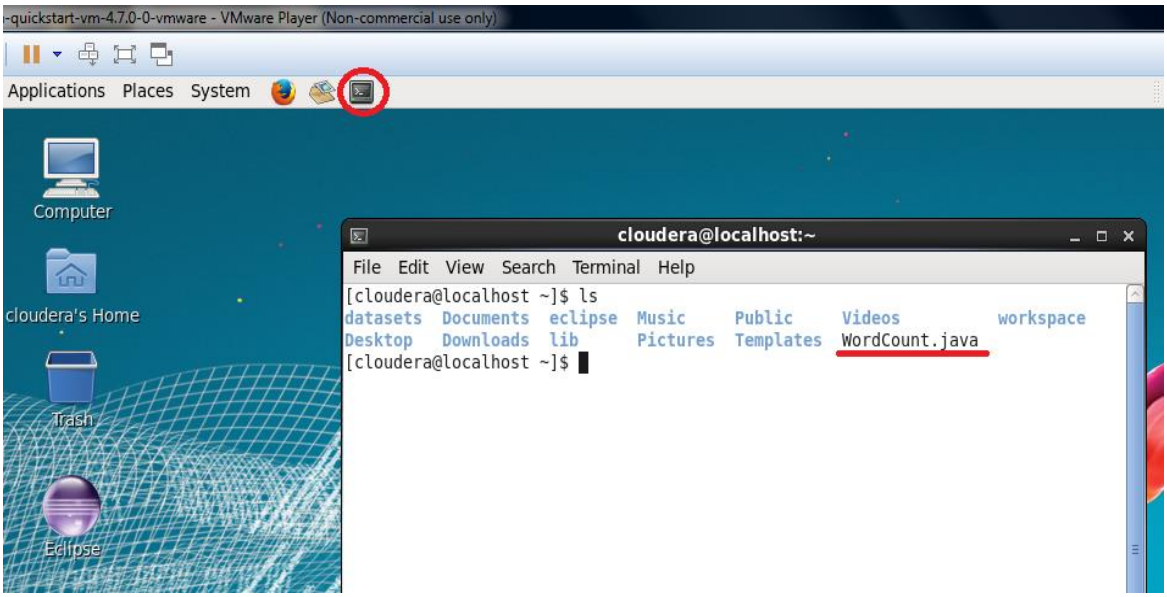
Kôd 1. Klasa za brojanje riječi

Kako bi se kôd mogao koristiti u virtualnom stroju, potrebno je prvo otvoriti uređivač teksta. Taj postupak je prikazan na Slici 5. Prikazana datoteka se sprema pod nazivom *WordCount.java*. Potrebno je pripaziti na točan naziv datoteke jer su nazivi datoteka i naredbe osjetljivi na velika i mala slova.



Slika 5. Otvaranje uređivača teksta

Kako bi se provjerilo je li datoteka uistinu spremljena, može se otvoriti terminal (na Slici 6 je označen crvenim krugom) te se upisivanjem naredbe „*ls*“ ispisuju nazivi svih datoteka koje se nalaze u tom direktoriju. Kao što se vidi na slici, datoteka je zaista spremljena te je na slici podcrtana crvenom linijom.



Slika 6. Ispis datoteka u terminalu

Za uspješno izvršavanje ovog programa, prvo je potrebno postaviti *CLASSPATH* varijablu kako bi se moglo pristupiti *Hadoop* knjižnici, što je označeno brojem 1 na Slici 7. Zatim treba stvoriti direktorij (u ovom slučaju, naziv je *wordcount_classes*) u koji će se spremati *Hadoop* ulazni i izlazni podaci, kao i kompajlirani kôd. To se radi tako što se u konzolu upisuje naredba *„mkdir wordcount_classes“* (naredba 2 na slici). Kako bi se prvobitno napisani kôd *WordCount.java* uspješno kompajlirao i spremio u novostvoreni direktrorij, u konzolu se upisuje naredba *„javac –d wordcount_classes/ WordCount.java“*(broj 3). Naposljetku, potrebno je stvoriti Java JAR datoteku kako bi se pomoću konzole mogla izvršavati obrada ulaznih podataka, upisivanjem naredbe *„jar –cvf wordcount.jar –C wordcount_classes/ .“* (četvrti korak na slici).

```
cloudera@localhost:~
File  Edit  View  Search  Terminal  Help
[cloudera@localhost ~]$ export CLASSPATH=/usr/lib/hadoop/client-0.20/\*:/usr/lib
/hadoop/\* 1
[cloudera@localhost ~]$ mkdir wordcount_classes 2
[cloudera@localhost ~]$ javac -d wordcount_classes/ WordCount.java 3
[cloudera@localhost ~]$ jar -cvf wordcount.jar -C wordcount_classes/ . 4
added manifest
adding: org/(in = 0) (out= 0)(stored 0%)
adding: org/myorg/(in = 0) (out= 0)(stored 0%)
adding: org/myorg/WordCount$Reduce.class(in = 1643) (out= 686)(deflated 58%)
adding: org/myorg/WordCount.class(in = 1985) (out= 988)(deflated 50%)
adding: org/myorg/WordCount$Map.class(in = 2209) (out= 983)(deflated 55%)
[cloudera@localhost ~]$
```

Slika 7. Obrada napisanog kôda u naredbenoj konzoli

Kako bi se program mogao testirati, potrebno je prvo stvoriti ulazne podatke koje će program obrađivati. Prva dva retka na Slici 7 prikazuju taj postupak (označeno je brojem 1 na slici). Naredbom *echo* se tekst pod navodnicima sprema u tekstualne datoteke naziva *file0* i *file1*. Datoteke je potrebno spremiti u *Hadoop* raspodijeljeni datotečni sustav za spremanje i slijedno čitanje. Kako bi se mogle koristiti svojstva datotečnog sustava, sve naredbe koje su specifične za *Hadoop* datotečni sustav trebaju počinjati naredbom *„hadoop fs“*. Prvo treba stvoriti odgovarajući HDFS direktorij, u ovom slučaju naziva *wordcount* (oznaka 2 na slici). Naredbom 3 se unutar novostvorenog direktorija stvara još jedan, u koji će se spremati ulazni podaci. Posljednje dvije naredbe, označene brojem 4, spremaju datoteke *file0* i *file1* u stvoreni direktorij HDFS.

```
cloudera@localhost:~
File  Edit  View  Search  Terminal  Help
[cloudera@localhost ~]$ echo "Hello World This Is Hadoop" >file0
[cloudera@localhost ~]$ echo "Hello Hadoop Bye Hadoop" >file1 1
[cloudera@localhost ~]$ hadoop fs -mkdir /user/cloudera/wordcount 2
[cloudera@localhost ~]$ hadoop fs -mkdir /user/cloudera/wordcount/input 3
[cloudera@localhost ~]$ hadoop fs -put file0 /user/cloudera/wordcount/input
[cloudera@localhost ~]$ hadoop fs -put file1 /user/cloudera/wordcount/input 4
[cloudera@localhost ~]$
```

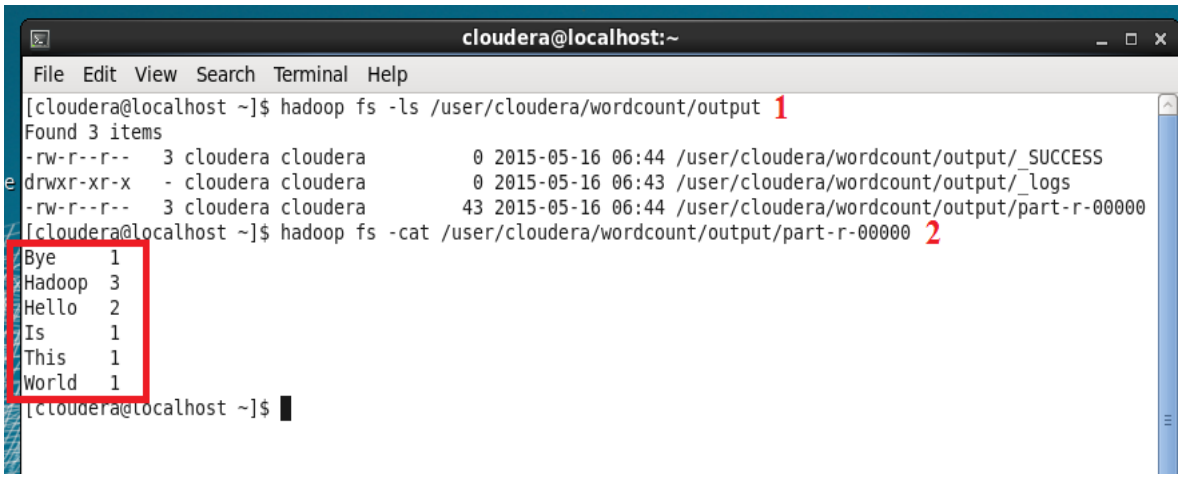
Slika 8. Stvaranje ulaznih podataka i njihovo spremanje u HDFS

Za pokretanje obrade podataka pomoću *Hadoopa*, upisuje se naredba prikazana na Slici 9. Potrebno je označiti da se program pokreće pomoću JAR datoteke te se upisuju njezin naziv, naziv glavne klase, putanja kojom se dolazi do ulaznih podataka iz HDFS-a te mjesto gdje će se podaci obrade (izlazni podaci) spremati.

```
cloudera@localhost:~
File  Edit  View  Search  Terminal  Help
[cloudera@localhost ~]$ hadoop jar wordcount.jar org.myorg.WordCount /user/cloudera/wordcoun
cloudera/wordcount/output
15/05/16 06:43:57 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
should implement Tool for the same.
15/05/16 06:43:58 INFO input.FileInputFormat: Total input paths to process : 2
15/05/16 06:43:59 INFO mapred.JobClient: Running job: job_201505160627_0001
15/05/16 06:44:00 INFO mapred.JobClient:  map 0% reduce 0%
15/05/16 06:44:22 INFO mapred.JobClient:  map 100% reduce 0%
15/05/16 06:44:32 INFO mapred.JobClient:  map 100% reduce 100%
15/05/16 06:44:36 INFO mapred.JobClient: Job complete: job_201505160627_0001
15/05/16 06:44:37 INFO mapred.JobClient: Counters: 32
15/05/16 06:44:37 INFO mapred.JobClient:   File System Counters
15/05/16 06:44:37 INFO mapred.JobClient:     FILE: Number of bytes read=86
15/05/16 06:44:37 INFO mapred.JobClient:     FILE: Number of bytes written=492943
15/05/16 06:44:37 INFO mapred.JobClient:     FILE: Number of read operations=0
15/05/16 06:44:37 INFO mapred.JobClient:     FILE: Number of large read operations=0
15/05/16 06:44:37 INFO mapred.JobClient:     FILE: Number of write operations=0
15/05/16 06:44:37 INFO mapred.JobClient:     HDFS: Number of bytes read=319
15/05/16 06:44:37 INFO mapred.JobClient:     HDFS: Number of bytes written=43
15/05/16 06:44:37 INFO mapred.JobClient:     HDFS: Number of read operations=4
15/05/16 06:44:37 INFO mapred.JobClient:     HDFS: Number of large read operations=0
15/05/16 06:44:37 INFO mapred.JobClient:     HDFS: Number of write operations=1
15/05/16 06:44:37 INFO mapred.JobClient:   Job Counters
15/05/16 06:44:37 INFO mapred.JobClient:     Launched map tasks=2
15/05/16 06:44:37 INFO mapred.JobClient:     Launched reduce tasks=1
15/05/16 06:44:37 INFO mapred.JobClient:     Data-local map tasks=2
15/05/16 06:44:37 INFO mapred.JobClient:     Total time spent by all maps in occupied slots
15/05/16 06:44:37 INFO mapred.JobClient:     Total time spent by all reduces in occupied slo
15/05/16 06:44:37 INFO mapred.JobClient:     Total time spent by all maps waiting after rese
s)=0
15/05/16 06:44:37 INFO mapred.JobClient:     Total time spent by all reduces waiting after r
 (ms)=0
15/05/16 06:44:37 INFO mapred.JobClient:   Map-Reduce Framework
15/05/16 06:44:37 INFO mapred.JobClient:     Map input records=2
15/05/16 06:44:37 INFO mapred.JobClient:     Map output records=9
15/05/16 06:44:37 INFO mapred.JobClient:     Map output bytes=87
15/05/16 06:44:37 INFO mapred.JobClient:     Input split bytes=268
15/05/16 06:44:37 INFO mapred.JobClient:     Combine input records=0
15/05/16 06:44:37 INFO mapred.JobClient:     Combine output records=0
15/05/16 06:44:37 INFO mapred.JobClient:     Reduce input groups=6
15/05/16 06:44:37 INFO mapred.JobClient:     Reduce shuffle bytes=120
15/05/16 06:44:37 INFO mapred.JobClient:     Reduce input records=9
15/05/16 06:44:37 INFO mapred.JobClient:     Reduce output records=6
15/05/16 06:44:37 INFO mapred.JobClient:     Spilled Records=18
15/05/16 06:44:37 INFO mapred.JobClient:     CPU time spent (ms)=4580
15/05/16 06:44:37 INFO mapred.JobClient:     Physical memory (bytes) snapshot=496148480
15/05/16 06:44:37 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=2074976256
15/05/16 06:44:37 INFO mapred.JobClient:     Total committed heap usage (bytes)=302907392
[cloudera@localhost ~]$
```

Slika 9. Pokretanje obrade podataka

Time je proces obrade podataka završen. Naposljetku, na Slici 10 su prikazani rezultati te obrade. Kako bi se vidjelo koje datoteke su stvorene u direktoriju sa izlaznim podacima, upisuje se naredba označena brojem 1 na slici. Direktorij se sastoji od 3 dijela: statusa obrade, dnevnika zapisa te rezultata obrade. Broj 2 označava naradbu koja se opisuje kako bi se rezultati obrade prikazali u konzoli te nam pokazali konačne rezultate brojanja riječi.

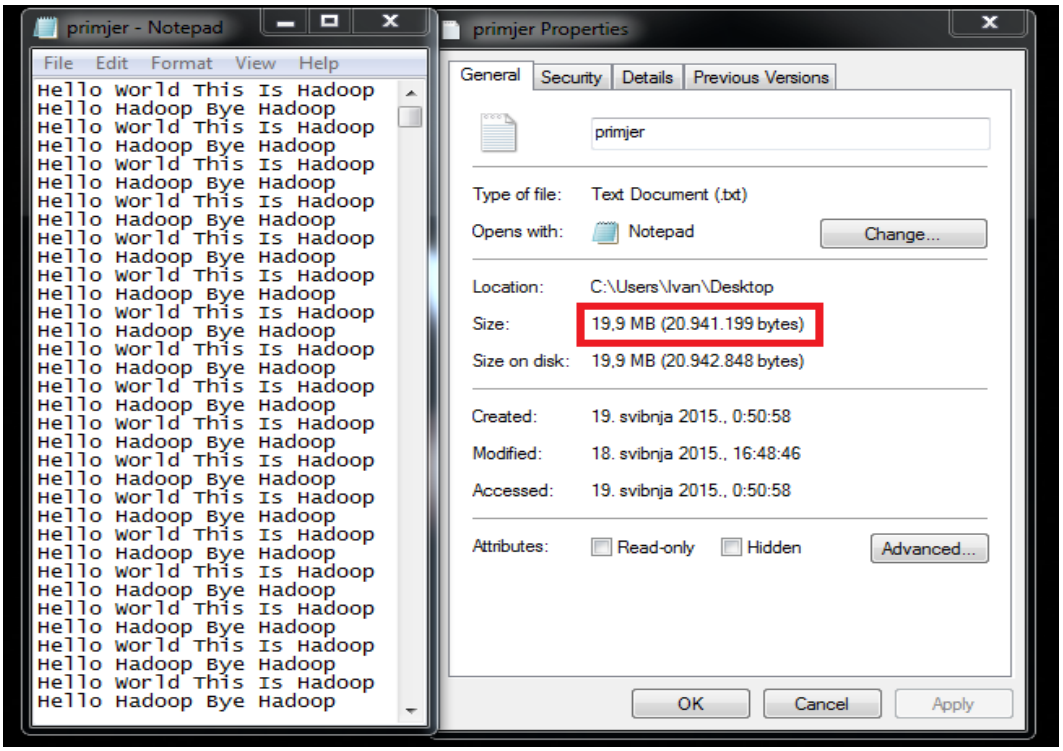


Slika 10. Rezultati obrade podataka

Kao što je vidljivo na slici, sve riječi koje se nalazi unutar dvije ulazne datoteke su zaista i obrađene te se nalaze na popisu unutar crvenog kvadrata označenog na Slici 10. Popis je poredan abecedno, a broj uz riječ označava koliko puta se koja riječ ponovila u obrađenim datotekama.

7. Usporedba studijskog primjera u različitim okolinama

Kako bi se mogle vidjeti prednosti paralelne obrade pomoću paradigme *MapReduce* i radnog okvira *Hadoop*, potrebno je rezultate obrade usporediti sa klasičnom obradom podataka. Pošto je glavno mjerilo usporedbe upravo vrijeme izvođenja, potrebno je stvoriti veću datoteku kako bi razlike u vremenima izvođenja bile vidljive. U tu svrhu je stvorena datoteka *primjer.txt* od ~20 MB, čiji se sadržaj i svojstva nalaze na Slici 11. Datoteka se sastoji od dvije različite rečenice koje su ponovljene 395117 puta (u datoteci se nalaze ukupno 790234 rečenice) kako bi se mogao pokrenuti primjer u kojem se nalaze riječi koje se ponavljaju, a da je datoteka pritom dovoljne veličine.



Slika 11. Datoteka *primjer.txt*

Novostvorenu datoteku se prvo pokrenulo na radnom okviru *Hadoop*, pomoću softvera *Cloudera*. Postupak pokretanja je isti kao i u prethodnom poglavlju, samo su ulazni podaci puno opširniji. Postupak i pojedinosti pokretanja su prikazane Slikom 12.

```
cloudera@localhost:~
File  Edit  View  Search  Terminal  Help
[cloudera@localhost ~]$ hadoop jar wordcount.jar org.myorg.WordCount /user/cloudera/wordcount
cloudera/wordcount/output1
15/05/18 14:03:33 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
hould implement Tool for the same.
15/05/18 14:03:33 INFO input.FileInputFormat: Total input paths to process : 1
15/05/18 14:03:36 INFO mapred.JobClient: Running job: job_201505181356_0001
15/05/18 14:03:37 INFO mapred.JobClient:  map 0% reduce 0%
15/05/18 14:03:55 INFO mapred.JobClient:  map 50% reduce 0%
15/05/18 14:03:58 INFO mapred.JobClient:  map 73% reduce 0%
15/05/18 14:04:01 INFO mapred.JobClient:  map 94% reduce 0%
15/05/18 14:04:04 INFO mapred.JobClient:  map 100% reduce 0%
15/05/18 14:04:21 INFO mapred.JobClient:  map 100% reduce 100%
15/05/18 14:04:24 INFO mapred.JobClient: Job complete: job_201505181356_0001
15/05/18 14:04:24 INFO mapred.JobClient: Counters: 32
15/05/18 14:04:24 INFO mapred.JobClient:   File System Counters
15/05/18 14:04:24 INFO mapred.JobClient:     FILE: Number of bytes read=3930622
15/05/18 14:04:24 INFO mapred.JobClient:     FILE: Number of bytes written=6223047
15/05/18 14:04:24 INFO mapred.JobClient:     FILE: Number of read operations=0
15/05/18 14:04:24 INFO mapred.JobClient:     FILE: Number of large read operations=0
15/05/18 14:04:24 INFO mapred.JobClient:     FILE: Number of write operations=0
15/05/18 14:04:24 INFO mapred.JobClient:     HDFS: Number of bytes read=20941337
15/05/18 14:04:24 INFO mapred.JobClient:     HDFS: Number of bytes written=74
15/05/18 14:04:24 INFO mapred.JobClient:     HDFS: Number of read operations=2
15/05/18 14:04:24 INFO mapred.JobClient:     HDFS: Number of large read operations=0
15/05/18 14:04:24 INFO mapred.JobClient:     HDFS: Number of write operations=1
15/05/18 14:04:24 INFO mapred.JobClient:   Job Counters
15/05/18 14:04:24 INFO mapred.JobClient:     Launched map tasks=1
15/05/18 14:04:24 INFO mapred.JobClient:     Launched reduce tasks=1
15/05/18 14:04:24 INFO mapred.JobClient:     Data-local map tasks=1
15/05/18 14:04:24 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (
15/05/18 14:04:24 INFO mapred.JobClient:     Total time spent by all reduces in occupied slot
15/05/18 14:04:24 INFO mapred.JobClient:     Total time spent by all maps waiting after reser
)=0
15/05/18 14:04:24 INFO mapred.JobClient:     Total time spent by all reduces waiting after re
(ms)=0
15/05/18 14:04:24 INFO mapred.JobClient:   Map-Reduce Framework
15/05/18 14:04:24 INFO mapred.JobClient:     Map input records=790234
15/05/18 14:04:24 INFO mapred.JobClient:     Map output records=3556053
15/05/18 14:04:24 INFO mapred.JobClient:     Map output bytes=34375179
15/05/18 14:04:24 INFO mapred.JobClient:     Input split bytes=137
15/05/18 14:04:24 INFO mapred.JobClient:     Combine input records=0
15/05/18 14:04:24 INFO mapred.JobClient:     Combine output records=0
15/05/18 14:04:24 INFO mapred.JobClient:     Reduce input groups=6
15/05/18 14:04:24 INFO mapred.JobClient:     Reduce shuffle bytes=1964038
15/05/18 14:04:24 INFO mapred.JobClient:     Reduce input records=3556053
15/05/18 14:04:24 INFO mapred.JobClient:     Reduce output records=6
15/05/18 14:04:24 INFO mapred.JobClient:     Spilled Records=10668159
15/05/18 14:04:24 INFO mapred.JobClient:     CPU time spent (ms)=27790
15/05/18 14:04:24 INFO mapred.JobClient:     Physical memory (bytes) snapshot=357933056
15/05/18 14:04:24 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=1396850688
15/05/18 14:04:24 INFO mapred.JobClient:     Total committed heap usage (bytes)=220266496
[cloudera@localhost ~]$
```

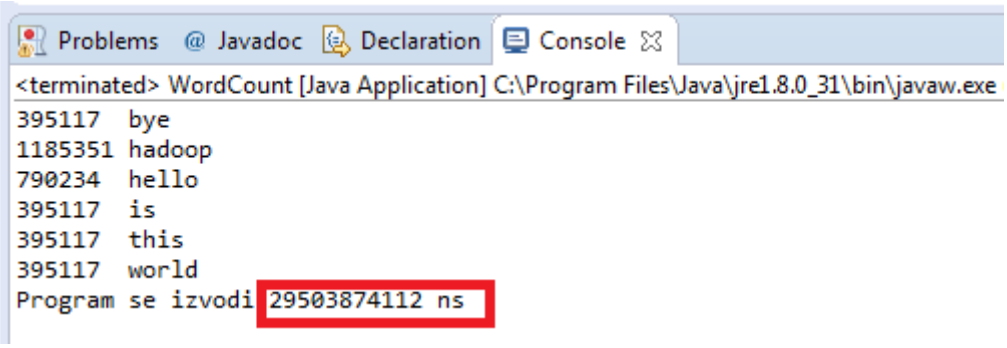
Slika 12. Pojedinosti obrade datoteke *primjer.txt*

Najvažniji podatak sa slike je vrijeme izvođenja, koje je označeno crvenim pravokutnikom. Vrijeme izvođenja je prikazano u milisekundama te iznosi 27790 milisekundi, odnosno 27.79 sekundi. Kako bi se provjerilo da je obrađena upravo datoteka *primjer.txt*, moguće je pomoću konzole provjeriti učestalost pojavljivanja riječi (Slika 13).

```
cloudera@localhost:~
File  Edit  View  Search  Terminal  Help
[cloudera@localhost ~]$ hadoop fs -ls /user/cloudera/wordcount/output1
Found 3 items
-rw-r--r--   3 cloudera cloudera          0 2015-05-18 14:04 /user/cloudera/wordcount/output1/_SUCCESS
drwxr-xr-x   - cloudera cloudera          0 2015-05-18 14:03 /user/cloudera/wordcount/output1/_logs
-rw-r--r--   3 cloudera cloudera         74 2015-05-18 14:04 /user/cloudera/wordcount/output1/part-r-00000
[cloudera@localhost ~]$ hadoop fs -cat /user/cloudera/wordcount/output1/part-r-00000
Bye     395117
Hadoop  1185351
Hello   790234
Is      395117
This    395117
World   395117
[cloudera@localhost ~]$
```

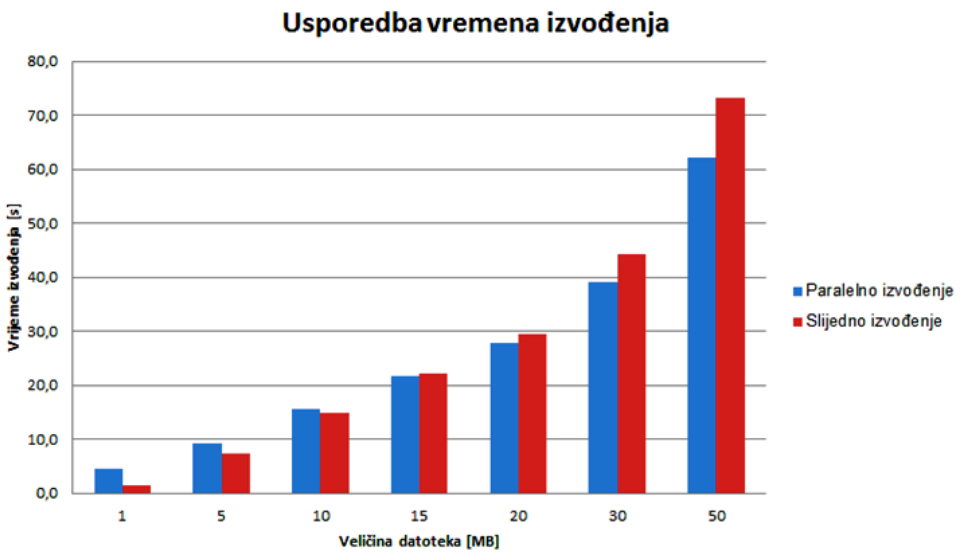
Slika 13. Rezultati obrade datoteke *primjer.txt* u radnom okviru *Hadoop*

Kako bi usporedba sa radnim okvirom *Hadoop* bila potpuna, u programskoj razvojnoj okolini *Eclipse* je također napisan program za brojanje riječi iz određene datoteke. Kôd je napisan u programskom jeziku Java. Datoteka koja se obrađivala je također datoteka *primjer.txt*, da se rezultati obrade mogu precizno usporediti. Slika 14 prikazuje rezultate izvođenja, koji su identični onima nastalim korištenjem softvera *Cloudera*. Osim toga, na slici je crvenom bojom označeno i vrijeme izvođenja, koje je u ovom slučaju izraženo u nanosekundama te iznosi 29.5 sekundi.



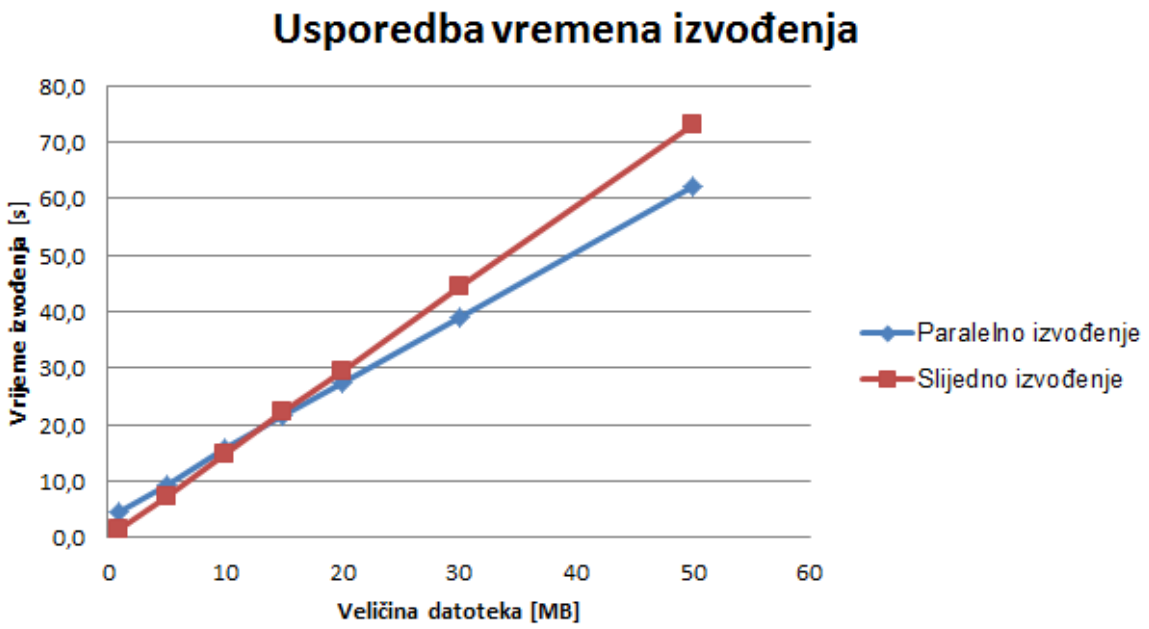
Slika 14. Rezultati obrade datoteke *primjer.txt* u razvojnoj okolini *Eclipse*

Za potpuniju usporedbu ove dvije vrste obrade, potrebno je obaviti bar nekoliko različitih testiranja. Po uzoru na prethodno opisani primjer, napravljeno je nekoliko identičnih tekstualnih datoteka, a jedina razlika je bila količina teksta u datotekama te sukladno tome su imale i različite veličine. Kao što je vidljivo na Slici 15, napravljene datoteke su redom imale 1, 5, 10, 15, 20, 30 te 50 MB. Sve datoteke su testirane i slijednim i paralelnim izvođenjem. Njihov prikaz na stupčanom dijagramu je prikazan na slici.



Slika 15. Usporedba slijednog i paralelnog izvođenja- stupčani dijagram

Kako bi se rezultati mogli uspješno analizirati te predvidjeti razlika za još veće tekstualne datoteke, prikazan je i linijski dijagram (Slika 16). Po njemu se može zaključiti kako slijedno izvođenje ima skoro pa ravnu liniju te će mu takav i biti rast za veće datoteke, dok paralelno izvođenje s veličinom datoteke lagano pada, ima oblik logaritamske krivulje. Za više datoteka većih veličina bi taj oblik bio još i izraženiji te vidljiviji.



Slika 16. Usporedba slijednog i paralelnog izvođenja- linijski dijagram

Zaključak

U sklopu diplomskog seminarskog rada opisani su vrste i procesi analize podataka. Osim toga, uspješno je realiziran studijski primjer na kojem je detaljno opisan proces izvođenja paralelne obrade pomoću paradigme *MapReduce*, radnog okvira *Hadoop* te softvera *Cloudera*. Tijekom razvoja su usvojena vrijedna znanja dubinske i paralelne obrade podataka te korištenja različitih programskih alata i tehnologija. Korištenje bogate baze stručne literature i podrške na Internetu uvelike su pomogli u razumijevanju teoretskog dijela ovog rada te njegovog uspješnog izvršenja. Zahvaljujući otvorenosti programskog kôda razvojnim programerima omogućen je uvid u već postojeće primjere kôda. Također, izvršena je i usporedna klasične te paralelne obrade podataka. Njihovo glavno mjerilo je vrijeme izvođenja, koje je manje kada se koristi paradigma *MapReduce* i radni okvir *Hadoop*. Razlika u vremenima izvođenja je samo par sekundi, ali se treba uzeti u obzir da su ulazni podaci imali samo 20 MB. Kada bi ulazni podaci bili još i veći, razlike vremena izvođenja bi bile osjetno vidljive. Time se uočavaju neke od velikih prednosti paradigme *MapReduce* te se dolazi do zaključka kako je paralelna obrada prilično korisna kada su u pitanju ulazni podaci jako velikog sadržaja (engl. *big data*). Ova zapažanja će biti od iznimne važnosti za velike baze podataka, kao npr. one koje se nalaze na društvenim mrežama.

Literatura

[1] Babbie, E.- The Practice of Social Research
Wadsworth Publishing, 13th edition, 2007

[2] Analiza sadržaja- Institut za hrvatski jezik i jezikoslovlje
http://struna.ihjj.hr/naziv/analiza-sadrzaja/25628/

[3] Holsti, O. R.- Content Analysis for the Social Sciences and Humanities
Longman Higher Education, 1st edition, 1969.

[4] An overview of content analysis,
http://pareonline.net/getvn.asp?v=7&n=17

[5] Medijska slika zemalja zemalja u regiji,
http://www.prglas.com/medijska-slika-zemalja-u-regiji/

[6] Wasserman, S.; Faust, K.- Social Network Analysis
Cambridge University Press, 1st edition, 1994.

[7] Belullo, A.- Uvod u ekonometriju
Sveučilišni udžbenik, Sveučilište Jurja Dobrile u Puli, 2011.

[8] Jović, A.- Postupci dubinske analize podataka
Fakultet elektrotehnike i računarstva, Zagreb

[9] Leskovac, J.; Rajaraman, A.; Ullman, J.- Mining of Massive Datasets
Cambridge University Press, 2nd edition, 2014.

[10] Upute za 2. laboratorijsku vježbu: Hadoop
http://www.fer.unizg.hr/_download/repository/lab2-zadatak.pdf

Sažetak

Primjena paralelne obrade u analizi društvenih mreža

U seminarskom radu obrađeni su temeljne principe analize i obrade podataka te su pokazani na jednom studijskom primjeru. Analiza sadržaja definira se kao bilo koji način za izradu zaključaka tako da se objektivno i sustavno identificiraju određena obilježja poruke. U novije doba, bitan čimbenik koji utječe na uspješnost učinaka marketinga i promocije je društveni utjecaj te je upravo zbog toga potrebno analizirati sadržaj sa društvenih mreža. Postoji nekoliko učinkovitih načina obrade podataka, od kojih je dubinska analiza podataka najčešće korištena. Dubinska analiza podataka je proces traženja i analiziranja podataka u svrhu pronalaženja implicitne, ali potencijalno korisne informacije. Takva analiza podataka je prikladna za društvene mreže jer se na njima nalaze ogromne količine podataka koje je najprikladnije obraditi primjenom paralelne obrade. Paralelna obrada je postupak kod kojega se više instrukcija obrađuje istovremeno. Obrada se zasniva na principu da se veliki problemi gotovo uvijek mogu podijeliti na manje te onda obraditi istovremeno. Paralelna obrada podataka se obavlja u raspodijeljenom datotečnom sustavu te je najčešći primjer korištenja paradigma *MapReduce.* Radni okvir koji se koristiti za studijski primjer je radni okvir *Hadoop*.

Ključne riječi: analiza sadržaja, društvene mreže, dubinska analiza podataka, paralelna obrada, *MapReduce*, DFS, *Hadoop*

Privitak A

Instalacija kompletne programske podrške

- S adrese https://my.vmware.com/web/vmware/downloads dohvatiti novu verziju virtualnog stroja *VMware Player* (u ovom slučaju, dohvaćena je verzija 7.1., za operacijski sustav *Windows* 7)
- Instalirati virtualni stroj prateći korake instalacije
- S adrese http://www.cloudera.com/content/cloudera/en/downloads.html skinuti *QuickStart VM* softver, verziju koja je napravljena za virtualnu mašinu *VMware Player* (zbog stabilnosti se za ovaj seminarski rad koristila verzija *Cloudera QuickStart 4.7*)
- Raspakirati sadržaj u odabrani direktorij
- Pokrenuti virtualnu mašinu, odabrati *Player→File→Open*, pozicionirati se u direktorij gdje se raspakirao sadržaj *Cloudere* i odabrati datoteku koju će virtualni stroj prepoznati kao jedinu koju može otvoriti
- Kada se *Cloudera* učita, otići u postavke (*Edit Virtual machine settings→ Hardware→ Processors→ Number of processor corres*) i broj jezgri postaviti na neki broj veći od 1 (u ovom slučaju, postavio se na 2), kako bi se mogao pokretati *Cloudera Manager* u virtualnom stroju
- Pokrenuti *Clouderu* pritiskom na *Play virtual machine*
- Pričekati nekoliko minuta dok se postavke ne podese