Analiza velikih količina podataka korištenjem Apache Spark platforme

Nevajdić, Vanja

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:597888

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Vanja Nevajdić

ANALIZA VELIKIH KOLIČINA PODATAKA KORIŠTENJEM APACHE SPARK PLATFORME

Diplomski rad

Voditelj rada:
dr. sc. Ognjen Orel

Zagreb, rujan 2021.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Mojoj mami

Sadržaj

Sadržaj **iv**

Uvod **1**

1 Distribuirani sustavi **2**

2 Apache Spark **5**
2.1 Glavne karakteristike 5
2.2 Distribuirano procesiranje 6
2.3 Particije podataka 8
2.4 Sparkove strukture podataka 9
2.5 Sparkov plan izvedbe 13
2.6 Sparkove operacije i lijena evaluacija 14

3 Analiza podataka u Apache Sparku **16**
3.1 Yelpov skup podataka 16
3.2 Analiza kvalitete podataka 17
3.3 Performanse 34

Zaključak **37**

Bibliografija **38**

Uvod

Prikupljanje podataka, njihova obrada te donošenje zaključaka na temelju analiza tih podataka prisutno je u ljudi još od davnina. Već prije 7000 godina ljudi su dokumentirali procese poljodjelstva i stočarstva, u svrhu prenošenja znanja i unaprjeđenja postojećih sustava [14].

Razvojem ljudskog znanja, u svim djelatnostima javila se potreba za prikupljanjem i analizom podataka. Tehnološkim napretcima 20. stoljeća postalo je moguće provoditi složene matematičke operacije, a time i analize, pomoću računala. To je ljudima uvelike olakšavalo provođenje analiza te je postalo nezaobilazan dio svakog uspješnog poduzeća.

U devedesetim godina prošlog stoljeća, povezivanje sve većeg broja uređaja na internet potaklo je stvaranje dotad neviđene količine podataka, koja ne prestaje rasti. Podatke više ne generiraju samo ljudi, nego velikom većinom strojevi, poput mobilnih telefona, pomoću kojih se spremaju kretnje korisnika i odabiri u aplikacijama koje koristi.

Zbog sve veće i veće količine korisnih podataka o njihovim proizvodima i korisnicima, u velikim poduzećima poput Googlea i Yahoo!-a, razvile su se prve tehnologije za pohranu i obradu velike količine podatka. 2003. godine Google je razvio Google File System distribuirani datotečni sustav, Big Table bazu podataka i MapReduce programski model, što se smatra začetkom *big data* tehnologija. Nedugo nakon, 2006. godine, Yahoo! je razvio Hadoop[1] ekosustav nad kojim su do danas razvijene brojne *big data* tehnologije i alati, jedna od kojih je Apache Spark[2], predmet ovog rada.

U ovom radu objasnit će se potreba za naprednim tehnologijama za obradu podataka, prikazati arhitektura i način rada Apache Spark platforme te pomoću iste u konačnici provesti analiza velike količine podataka.

[1] https://hadoop.apache.org/
[2] https://spark.apache.org/

Poglavlje 1

Distribuirani sustavi

Temelj ranije spomenutih *big data* tehnologija i alata su distribuirani računalni sustavi, odnosno sustavi koji se sastoje od više računala, fizičkih ili virtualnih, koja funkcioniraju kao cjelina. Distribuirani sustavi imaju mnogo prednosti nad centraliziranim sustavima, koje sačinjava jedno računalo, no u kontekstu Apache Sparka najbitnije je da se na njima može provoditi distribuirano računanje. Jednostavno govoreći, to što se sustav sastoji od više računala omogućuje da se računanje na njima provodi paralelno, čime se vrijeme izvršavanja može drastično smanjiti.

Klaster

Jedan od oblika distribuiranih sustava je klaster, koji se sastoji od servera. Na serverima se pohranjuju podaci i izvršavaju programi. Među svim serverima izdvajaju se *master* serveri, koji obavljaju administrativne uloge u klasteru, odnosno koji organiziraju rad cijelog klastera. Na ostale servere može se gledati kao na prave radnike u klasteru jer se na njima izvršavaju programi i pohranjuju podaci. Master serveri imaju uvid u stanje svih drugih servera, što znači da imaju informaciju koliko svaki server ima resursa (memorijskih i procesorskih) te koliko je tih resursa dodijeljeno kojoj aplikaciji. Pri pokretanju neke aplikacije na klasteru, temeljem zatraženih resursa te aplikacije, master određuje na kojim serverima će se ta aplikacija izvršavati te joj dodjeljuje resurse.

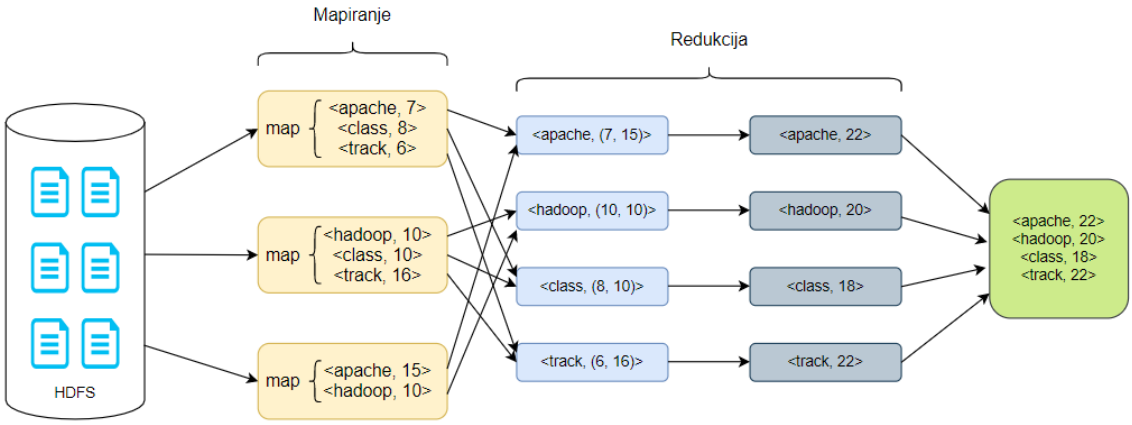
Ranije spomenuti Yahoo!-ov Hadoop, danas je najpoznatiji ekosustav (skup) *big data* tehnologija. Na klasteru s instaliranim Hadoopom (tzv. Hadoop klaster) može se izvršavati mnoštvo *big data* tehnologija, sve po ranije objašnjenom principu zahtijevanja resursa i delegiranja poslova, a jedna od tih tehnologija je upravo Apache Spark. Pošto je Hadoop klaster najčešće okruženje u kojem se Spark koristi, za kasnije, detaljnije opise funkcionalnosti Sparka, bitno je spomenuti da u sklopu Hadoopa postoji distribuirani datotečni sustav HDFS (Hadoop Distributed File System) [8]. HDFS se sastoji od dvije komponente – Na-

meNodea i DataNodea. NameNode se najčešće instalira na master server i obavlja ulogu organizatora datotečnog sustava, a DataNodeovi na ostale servere te se tamo pohranjuju podaci.

Paralelno procesiranje

Paralelno izvršavanje zadataka ključ je brzog procesiranja velike količine podataka, a temelj tog paralelizma je MapReduce programski model ili uzorak.

Slika 1.1 prikazuje MapReduce algoritam za brojanje ponavljanja nekih riječi, koji se izvršava na Hadoopu. MapReduce iskorištava paralelizam pohrane podataka na HDFS-u tako što se izvršava na istim serverima na kojima su pohranjeni podaci. U fazi mapiranja broje se ponavljanja traženih riječi u nekom tekstu, na svakom serveru paralelno. Nakon toga, u fazi redukcije, ti rezultati se sažimaju tako da se dobije završni rezultat, tj. broj ponavljanja traženih riječi u svim podacima.



Slika 1.1: Prikaz MapReduce algoritma za brojanje ponavljanja riječi „apache”, „hadoop”, „class” i „track”. Po uzoru na sliku iz [10].

Prednosti ovakvih algoritama su očite – više procesa se izvršava istovremeno i smanjuje se mrežni promet podataka. No, MapReduce ima i svojih nedostataka. Međurezultati iz faze mapiranja zapisuju se na disk te čitaju s njega u fazi redukcije, što je spor proces. Taj problem je pogotovo izražen ako se više MapReduce izračuna odvija u nizu pa su pisanje i čitanje s diska česti. Također, zbog ključ-vrijednost (*key-value*) strukture međurezultata

i konačnih rezultata, kôd MapReduce algoritama zna često biti nezgrapan te kompliciran za korištenje. Dodatno, MapReduce model nije prikladan za procesiranje svih tipova podataka, na primjer za tokove (*streaming*) podataka, niti za sve tipove analiza, kao što je strojno učenje.

Sa ciljem stvaranja tehnologije koja bi se temeljila na ideji MapReducea, ali bila brža i jednostavnija za korištenje, znanstvenici UC Berkeleya 2009. godine pokrenuli su projekt pod nazivom *Spark*, a 2014. pušten je na korištenje Apache Spark 1.0. U trenutku pisanja ovog rada najnovija verzija Apache Sparka je 3.1.2.

Poglavlje 2

Apache Spark

Apache Spark je platforma osmišljena za distribuirano procesiranje velike količine podataka. Danas je jedna od najčešće korištenih *big data* tehnologija, a najviše ju koriste podatkovni inženjeri, znanstvenici i analitičari.

2.1 Glavne karakteristike

Glavne karakteristike Sparka koje ga čine uspješnom tehnologijom su brzina procesiranja podataka, opseg problema za koje se može koristiti, jednostavnost korištenja i povezivost s drugim *big data* tehnologijama.

Brzina

Primarni uvjet za uspješno procesiranje velike količine podataka svakako je brzina njihovog procesiranja, a glavni način kako se ona postiže kod Sparka je čuvanjem međurezultata u memoriji, tzv. *in-memory* pohranjivanje. To čini Spark puno bržim od Hadoop MapReducea jer je pisanje i čitanje s diska značajno sporiji proces. Sparkovo paralelno procesiranje podataka i pohranjivanje u memoriji oslanja se i maksimalno iskorištava današnja računala koja dolaze s puno memorije, više procesorskih jezgri i operacijskim sustavima za efikasno paralelno procesiranje.

Također, pri izvođenju koda, Spark najprije optimizira korisnikove upite te stvori plan izvedbe u obliku grafa, tzv. *directed acyclic graph* (DAG), koji se najčešće može podijeliti u manje dijelove za paralelno procesiranje.

Opsežnost i jednostavnost korištenja

Spark se može koristiti na podacima širokog spektra te se pomoću njega mogu provoditi razne analize. Za to su primarno zaslužne četiri biblioteke koje Spark nudi, koje su ujedno i funkcionalni „dijelovi" Sparka. Radi se o Spark SQL-u za rad sa strukturiranim podacima, Spark Streaming za obradu tokova podataka, Spark MLlib za strojno učenje te GraphX za obradu podataka strukturiranih kao graf. Sve ove biblioteke mogu se koristiti u istoj Spark aplikaciji.

U Spark aplikacijama mogu se koristiti programski jezici Scala, Java, Python i R, što uvelike olakšava posao korisnicima jer najčešće već znaju barem jedan od ovih programskih jezika prije nego što se krenu baviti Sparkom. Osim toga, Spark je jednostavan za korištenje zbog struktura podataka koje koristi, o čemu će biti riječ kasnije u radu.

Povezivost

Kako Spark nije platforma za pohranu podataka, nego za njihovo procesiranje, ona funkcionira u simbiozi s drugim tehnologijama, poput Apache Hadoopa, Apache Cassandre, MongoDB, Apache Hive, Apache Kafke i mnogim drugima, iz kojih čita podatke i kamo ih sprema.

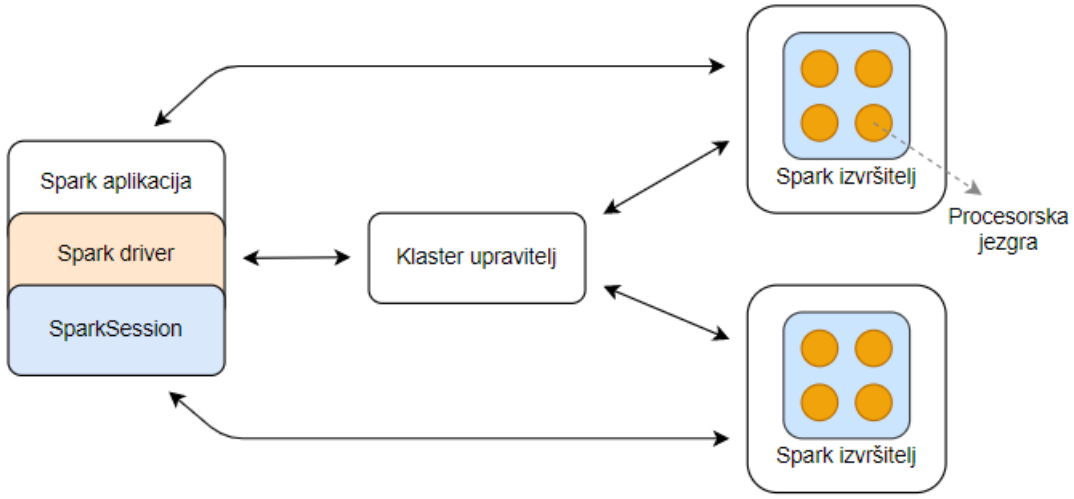
2.2 Distribuirano procesiranje

Sparkova arhitektura može se promatrati na dvije razine – na visokoj razini, koja obuhvaća pokretanje Spark aplikacije i interakciju između različitih Sparkovih komponenti, te na niskoj razini, koja obuhvaća samu organizaciju izvršavanja koda.

Organizacija Sparka na visokoj razini prikazana je na slici 2.1.

Spark driver ima nekoliko uloga. Odgovoran je za komunikaciju s klaster upraviteljem (*cluster manager*), organizacijskim dijelom klastera na kojem je instaliran, te zahtijeva od njega resurse za Sparkove izvršitelje (*Spark executors*). Također, Spark driver transformira sve Sparkove operacije u DAG-ove, određuje kada će se koji DAG izvršiti te raspoređuje njihovo izvršavanje na izvršiteljima.

SparkSession je objekt koji implementira osnovne Spark funkcionalnosti i prva je stvar koja se instancira u Spark aplikaciji. SparkSession ima ugrađene metode za učitavanje podataka, izvršavanje SQL upita, mijenjanje konfiguracije cijele Spark aplikacije i mnoge druge. Uobičajeni početak Spark aplikacije izgleda kao kod u nastavku, u kojemu je stvo-



Slika 2.1: Komponente i arhitektura Apache Sparka. Po uzoru na sliku iz [2].

ren SparkSession i učitani su podaci. Dodatno, napravljen je i SQL upit nad učitanim podacima, opet pomoću ugrađene metode SparkSessiona.

```
from pyspark.sql import SparkSession

# Pokretanje SparkSessiona
spark = SparkSession \
        .builder \
        .appName("ImeAplikacije") \
        .getOrCreate()

# Ucitavanje podataka
df1 = spark.read.json("datoteka.json")

df1.createOrReplaceTempView("sql_tablica")
# SQL upit nad DataFrameom df1
df2 = spark.sql(
    """
    SELECT stupac1, stupac2
    FROM sql_tablica
    WHERE uvjet = 1
    """)
```

Spark izvršitelji su izvedbeni dijelovi Spark aplikacije. To su Java virtualni strojevi (JVM-ovi), koji rade na čvorovima klastera te izvršavaju konkretne dijelove koda po uputi

Spark drivera. Svaki izvršitelj ima neki broj procesorskih jezgri te se na njima zadaci (*taskovi*) mogu izvoditi paralelno. Na primjer, ako Spark aplikacija ima tri izvršitelja, svaki sa četiri procesorske jezgre, najviše se dvanaest zadataka može izvršavati paralelno.

2.3 Particije podataka

Podaci koje Spark koristi mogu biti spremljeni na raznim lokacijama, najčešće raspoređeni u distribuiranom datotečnom sustavu. S druge strane, sam Spark može imati vlastitu particiju podataka, nevezanu za „fizičku" particioniranost podataka u datotečnom sustavu, na koju sam kreator Spark aplikacije može utjecati.

Kao što je već bilo spomenuto, u Spark aplikacijama teži se paralelnom procesiranju. Particioniranjem podataka Spark dijeli podatke na prikladniji broj dijelova, s obzirom na veličinu podataka i ukupni broj procesorskih jezgri s kojima raspolaže. Na broj particija korisnik može direktno utjecati i to je dio optimizacije Spark aplikacije.

U Sparku se javlja više vrsta particioniranja podataka, a jedna od njih je za izmjenu podataka, tzv. *shuffle partitions*. O potrebi izmjene podataka bit će više riječi u odjeljku 2.6. Bez ulaženja u prevelike detalje optimizacije particija, ono što se pokušava je odabrati broj particija s kojim će se kôd izvest najbrže moguće. Na primjer, ako Spark aplikacija raspolaže s dvanaest procesorskih jezgri, nema smisla particionirati podatke na deset dijelova jer će uvijek dvije jezgre ostati neiskorištene. S druge strane, nema smisla niti podijeliti podatke na puno više dijelova od broja jezgara jer se troši vrijeme na pokretanje zadataka nad svakim od njih te se povećava mrežni promet. Određivanje optimalnog broja particija za danu Spark aplikaciju posao je podatkovnih inženjera i nije fokus ovog rada, ali osnovne upute inženjera su da bi broj particija uvijek trebao biti višekratnik broja jezgara te se najčešće odlučuje na

- isti broj particija kao što je broj jezgri, ili
- dva do četiri puta veći broj,

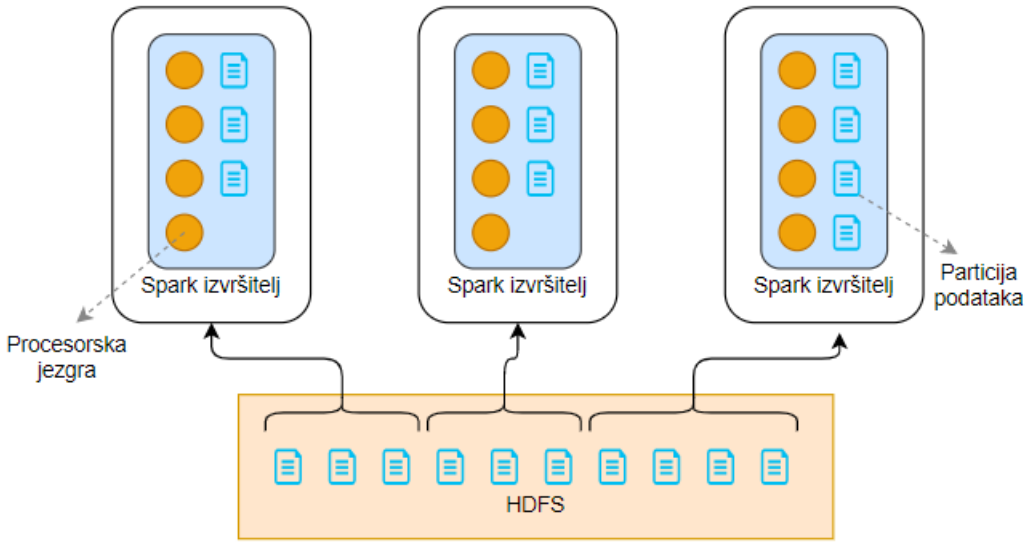
ovisno o količini podataka i resursa [15].

Sljedeći kod prikazuje postavku particije podataka na 24 dijela. Primijetimo, radi se o jos jednoj funkcionalnosti SparkSessiona.

```
1 spark.conf.set("spark.sql.shuffle.partitions", "24")
```

Pri particioniranju podataka, Spark pokušava što više ispuniti svojstvo lokalnosti. Odnosno, pokušava postići da su podaci koji se procesiraju na određenom izvršitelju pohra-

njeni na istom serveru, kako bi za njihovo učitavanje u memoriju bilo potrebno što manje mrežnog prometa. Slika 2.2 prikazuje dodjeljivanje particija podataka izvršiteljima.



Slika 2.2: Podjela podataka na Spark izvršitelje. Po uzoru na sliku iz [2].

2.4 Sparkove strukture podataka

Da bi paralelno procesiranje pomoću Sparka bilo moguće, podaci moraju biti spremljeni u strukture koje podržavaju particioniranost i paralelizam.

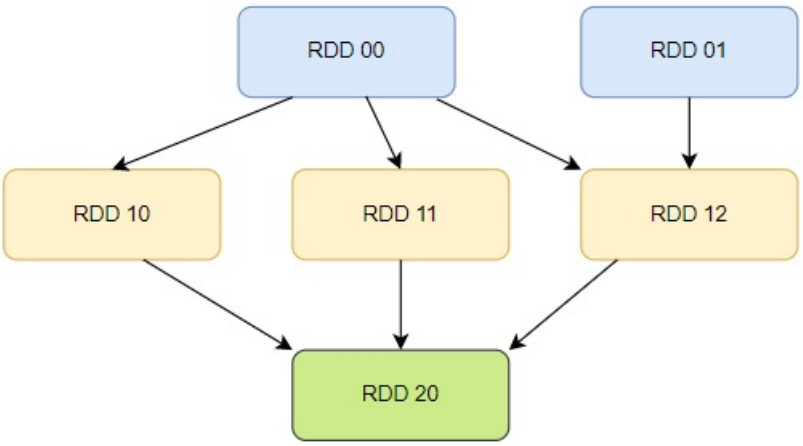
RDD

Resilient Distributed Datasets (RDD) osnovna je apstrakcija podataka u Sparku. To je nepromjenjiva distribuirana kolekcija objekata, pri čemu objekti mogu biti već implementirani objekti korištenog programskog jezika ili definirani od strane korisnika. Jedna je od najvažnijih karakteristika RDD-ova da se operacije nad njima mogu paralelizirati.

Također, RDD-jevi su otporni na greške (*fault-tolerant*), što znači da se mogu oporaviti ako dođe do gubitka neke particije zbog, na primjer, kvara servera. Svojstvo otpornosti na greške ostvaruje se tako što Spark pamti sve operacije koje su potrebne za stvaranje svakog RDD-ja. Na primjer, na slici 2.3 prikazan je proces stvaranja RDD-ja pod nazivom „RDD 20”. Za njegovo stvaranje potrebni su originalni RDD-jevi „RDD 00” i „RDD 01”, koji nisu

nastali iz nekog drugog RDD-ja, nego učitavanjem podataka iz datoteka ili direktnim unosom u Spark aplikaciji. Operacijama izvršenim nad tim RDD-jevima nastaju „RDD 10", „RDD 11" i „RDD 12". Napomenimo da izvršavanje operacija nad RDD-jevima njih ne mijenja, nego se izmijenjeni podaci ispisuju ili spremaju u novi RDD. Konačno, iz „RDD 10", „RDD 11" i „RDD 12" nastaje „RDD 20".

Za svaki RDD u aplikaciji, Spark pamti njegov put nastanka, koji naziva njegovom lozom (*lineage*). Slika 2.3 je shematski prikaz loze RDD-ja, a u stvarnosti se radi o nizu informacija o tome koji podaci se trebaju učitati te koje operacije se trebaju izvršiti nad RDD-jevima da bi se dobio traženi RDD. Dakle, loza sadrži potpunu uputu za stvaranje RDD-ja.



Slika 2.3: Shematski prikaz loze RDD-ja „RDD 20". Po uzoru na sliku iz [6].

Upravo su loze presudne za ostvarivanje svojstva otpornosti na greške. Ako dođe do kvara servera te na njemu prestane raditi Spark izvršitelj, gube se podaci iz memorije tog izvršitelja, na primjer neke particije RDD-jeva. U tom slučaju Spark će prebaciti posao na drugog izvršitelja te će se pomoću loza tih RDD-jeva oporaviti izgubljene particije. Bitno je napomenuti da zato svaki Spark izvršitelj mora imati pristup svim podacima koji se koriste u aplikaciji. Ako Spark aplikacija učitava podatke iz HDFS-a, ona zapravo u situaciji kvara koristi redundantnost HDFS-a za oporavak svojih RDD-jeva. Redundantnost HDFS-a znači da su podaci replicirani te raspoređeni tako da kvar jednog servera ne dovodi do stvarnog gubitka podataka. Dakle, unatoč tome što su možda izgubljeni podaci koji su se koristili za stvaranje particije RDD-ja koju se pokušava oporaviti, Spark će uspjeti oporaviti tu particiju jer će od HDFS-a imati lokaciju replikacije tih podataka.

Iako ima puno prednosti, rad s RDD-jevima nije ni elegantan ni jednostavan. Operacije nad njima pišu se u obliku lambda izraza, koji nisu lako čitljivi. Osim toga, takvim zadavanjem operacija korisnik određuje *kako* želi da se operacije naprave, a ne *što* želi da se napravi, što onemogućuje Spark da optimizira korisnikove izraze.

U nastavku je prikaz koda iz [12] u kojem se učitavaju podaci iz tekstualne datoteke u RDD i provodi operacija za računanje duljine te datoteke (ukupan broj simbola). Odmah se primijeti prisustvo `map()` i `reduce()` komponenti. Prvo se u fazi mapiranja u svakoj particiji RDD-ja izračuna broj simbola, a zatim u fazi redukcije zbrajaju ti rezultati. Iako je ovo skroz jednostavan primjer, najčešće nema potrebe da korisnik razmišlja o tome kako se željena operacija izvršava u konceptima MapReducea.

```
lines = sc.textFile("data.txt")
lineLengths = lines.map(lambda s: len(s))
totalLength = lineLengths.reduce(lambda a, b: a + b)
```

Strukturirani API-ji

U svrhu pisanja elegantnijeg i lakše čitljivog koda, kao i zbog mogućnosti optimizacije korisnikovih upita, u Sparku 2.0 uvedeni su strukturirani API-ji (Application Programming Interface), od kojih je najčešće korišten DataFrame.

Strukturirani API-ji mogu se shvatiti kao strukture više razine apstrakcije od RDD-jeva jer korisnik ne treba znati kako se njegovi upiti točno provode. Paralela se može povući s programskim jezicima poput Pythona ili Jave koji su više razine apstrakcije od C-a pa su korisniku jednostavniji za korištenje te on manje zna što njegov kod radi „u pozadini”. U nastavku slijedi primjer koda iz [2] za računanje prosjeka godina ljudi s istim imenom u kojem se prvo koristi RDD, a zatim DataFrame.

```
# Definiranje RDD-ja
dataRDD = sc.parallelize([("Brooke", 20), ("Denny", 31), ("Jules", 30),
    ("TD", 35), ("Brooke", 25)])
# Racunanje prosjeka godina ljudi s istim imenom pomocu map i reduce
   operacija
avgRDD = (dataRDD
    .map(lambda x: (x[0], (x[1], 1)))
    .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
    .map(lambda x: (x[0], x[1][0]/x[1][1])))


```

```
# Definiranje DataFramea
data_df = spark.createDataFrame([("Brooke", 20), ("Denny", 31),
    ("Jules", 30), ("TD", 35), ("Brooke", 25)], ["name", "age"])
# Grupiranje podataka po imenu i racunanje prosjeka njihovih godina
avg_df = data_df.groupBy("name").agg(avg("age"))
```

Računanje prosjeka godina pomoću RDD-ja zahtijeva razumijevanje MapReduce paradigme i lambda funkcija. Za razliku od toga, računanje istog pomoću DataFramea koristi puno intuitivniji i jednostavniji pristup – grupiranje i računanje prosjeka za svaku grupu.

DataFrame

Sparkov DataFrame je apstrakcija RDD-ja koja korisniku izgleda kao nepromjenjiva tablica podataka. Ta tablica ima imenovane stupce koji imaju definiran tip podataka. Radi se o istoj strukturi kao što je tablica u relacijskoj bazi podataka i *data frame* u programskim jezicima R i Python. Pošto je DataFrame u suštini RDD, particioniran je među Sparkovim izvršiteljima.

Na DataFrameu se mogu izvoditi razne operacije, kao što su grupiranje, filtriranje, agregiranje i mnoge druge. Prilikom izvršavanja upita nad DataFrameom, Spark prvo napravi analizu svih operacija koje treba izvršiti te ih zatim organizira i izmjenjuje da izvođenje bude najbrže moguće, odnosno provodi optimizaciju upita. Glavne komponente Sparkovog optimizatora su Catalyst optimizator i projekt Tungsten. Jednom kada je optimizacija završena, Spark zapisuje operacije u obliku lambda funkcija i izvršava ih na RDD-ju, a cijeli taj postupak ostaje skriven od korisnika.

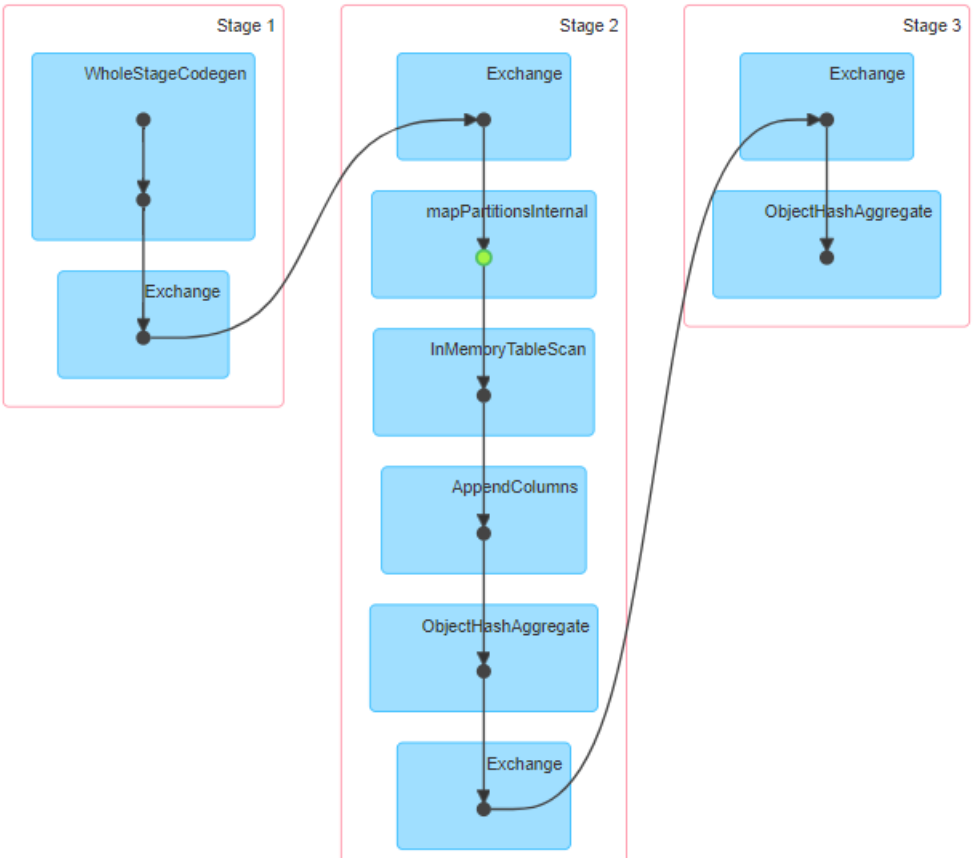
U nastavku slijedi primjer DataFramea s podacima iz [16].

```
+--------------------+-------------------+-----+--------------------+
|         business_id|               date|stars|                text|
+--------------------+-------------------+-----+--------------------+
|buF9druCkbuXLX526...|2014-10-11 03:34:02|  4.0|Apparently Prides...|
|RA4V8pr014UyUbDvI...|2015-07-03 20:38:25|  4.0|This store is pre...|
|_sS2LBIGNT5NQb6PD...|2013-05-28 20:38:06|  5.0|I called WVM on t...|
|0AzLzHfOJgL7ROwhd...|2010-01-08 02:29:15|  2.0|I've stayed at ma...|
|8zehGz9jnxPqXtOc7...|2011-07-28 18:05:01|  4.0|The food is alway...|
|xGXzsc-hzam-VArK6...|2018-01-21 04:41:03|  1.0|This place used t...|
|EXOsmAB1s71WePlQk...|2006-04-16 02:58:44|  2.0|The setting is pe...|
|DbXHNl890xSXNiyRc...|2017-12-02 18:16:13|  5.0|Probably one of t...|
|mD-A9KOWADXvfrZfw...|2012-05-28 15:00:47|  4.0|I am definitely a...|
|EEHhKSxUvJkoPSzeG...|2014-05-07 18:10:21|  5.0|I work in the Pru...|
|WQFn1A7-UAA4JT5YW...|2017-09-08 23:26:10|  1.0|They NEVER seem t...|
+--------------------+-------------------+-----+--------------------+
```

2.5 Sparkov plan izvedbe

Kôd Spark aplikacije driver dijeli na poslove (*job*), koji se izvršavaju sekvencijalno. Svaki posao ima strukturu DAG-a, a čvorovi unutar njega nazivaju se faze (*stage*). Faze se zatim dijele na zadatke (*task*), koji su konkretne operacije nad particijama. Oni se izvršavaju na Spark izvršiteljima te je njihovo izvršavanje paralelno, kada može biti.

Slika 2.4 prikazuje jedan posao, koji se sastoji od tri faze, a unutar svake faze vidi se niz zadataka. Podjela koda na ove dijelove dio je Sparkove optimizacije. Prikaz na slici je iz Sparkovog korisničkog sučelja (Spark UI), u kojemu korisnik može pronaći detalje izvođenja svoje aplikacije, kao što je organizacija zadataka, njihovo trajanje, gdje se izvršavaju i mnoge druge.



Slika 2.4: Plan izvedbe jednog Spark posla

2.6 Sparkove operacije i lijena evaluacija

Spark ima dvije vrste operacija koje izvodi na distribuiranim podacima – transformacije i akcije. Transformacije transformiraju Spark DataFrame u novi DataFrame, bez mijenjanja originalnih podataka. Na primjer, operacije poput

- `select()`, koja dohvaća određene stupce, ili
- `filter()`, koja dohvaća određene retke DataFramea,

neće promijeniti originalni DataFrame, već će iz originalnog stvoriti novi.

Transformacije su evaluirane lijeno (tzv. *lazy evaluation*), odnosno Spark ih pamti za kasnije provođenje, a ne provodi ih odmah. Lijena evaluacija je jedan od Sparkovih alata za optimizaciju izvođenja koda. Kada dođe vrijeme za provedbu zapamćenih transformacija, Spark im može mijenjati raspored izvedbe, spajati ih ili jednostavno optimizirati, kako bi izvedba svih tih transformacija bila efikasnija.

Okidač za provođenje zapamćenih transformacija je akcija. Akcije su operacije koje rezultiraju nekim prikazom korisniku, na primjer

- `show()`, koja ispisuje dio DataFramea, te
- `count()`, koja ispisuje broj redaka nekog DataFramea.

Također, akcije su operacije koje direktno pristupaju podacima, kao što su `load()` i `save()`.

Isječak koda u nastavku prikazuje tri Sparkove operacije, pri čemu će se učitavanje i filtriranje izvršiti tek pokretanjem akcije `show()`.

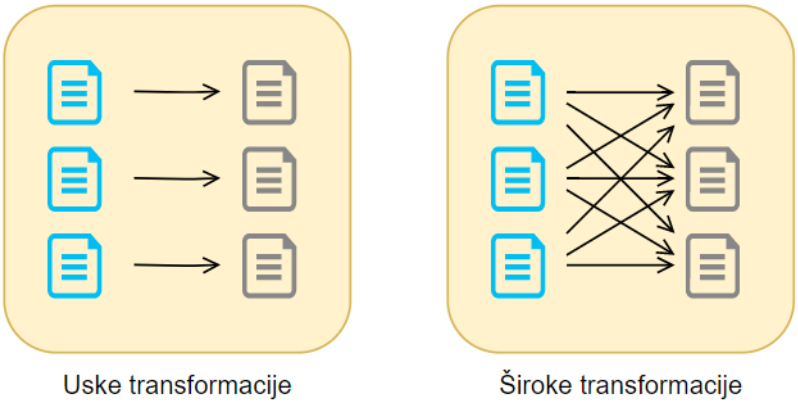
```
# Ucitavanje podataka -> transformacija
podaci = spark.read.text("datoteka.txt")

 # Filtriranje redaka -> transformacija
podaci_hr = podaci.filter("Drzava == 'Hrvatska'")
# Prikaz prvih 15 redaka -> akcija
podaci_hr.show(15)
```

Uske i široke transformacije

Operacije transformacija možemo podijeliti na dvije vrste – uske (*narrow*) i široke (*wide*). Transformacija je uska ako se rezultat te transformacije na određenoj particiji podataka može dobiti korištenjem samo te particije, a ne i ostalih particija podataka. Na primjer,

operacija filtriranja na jednoj particiji DataFramea vratit će sve retke koji zadovoljavaju dani uvjet i taj rezultat nikako ne ovisi o drugim particijama istog DataFramea. Dakle, za dobivanje konačnog rezultata filtriranja, na svakoj particiji se treba izvršiti ta transformacija te među njima nema izmjene podataka.

S druge strane, transformacije poput grupiranja ili sortiranja moraju kombinirati podatke iz svih particija da bi dale konačan rezultat. Na primjer, računanje broja redaka u svakoj grupi podataka zahtjeva prvo grupiranje i brojanje u svakoj particiji, ali nakon toga se rezultati moraju sažeti u jedinstveni rezultat. Transformacije tog tipa nazivaju se širokima. Slika 2.5 vizualno prikazuje razliku između uskih i širokih transformacija.



Slika 2.5: Korištenje particija za izvršenje uskih, odnosno širokih transformacija. Po uzoru na sliku iz [2].

Poglavlje 3

Analiza podataka u Apache Sparku

Kao demonstracija gradiva obrađenog u prošlom poglavlju, u sklopu rada napravljena je analiza podataka pomoću SparkSQL biblioteke.

Apache Spark nije bio pokrenut u sklopu Hadoopa ili nekog drugog *big data* sustava, već samostalno (u tzv. *standalone* verziji). Odnosno, instaliran je klaster samo za Spark. Konkretno, korišteni Spark klaster sastojao se od pet servera, od čega je jedan bio *master*. U Spark klasterima poput ovog, ostali serveri nazivaju se *workerima*. Resursi servera su:

- 8 GB radne memorije (RAM), od čega je 6,6 GB dostupno Spark izvršitelju,
- 50 GB prostora za trajnu pohranu na HDD ili SDD diskovima,
- 4 procesorske jezgre.

Korišten je Apache Spark 3.1.1, a kod napisan u jeziku Python.

3.1 Yelpov skup podataka

U analizi su korišteni javno dostupni podaci platforme Yelp [16]. Radi se o online platformi koja povezuje poduzeća uslužne djelatnosti, najčešće restorane, ali i dućane, servise i agencije, s njihovim korisnicima, koji ih ocjenjuju. Yelpov skup podataka sastoji se od podataka o poduzećima, korisnicima, recenzijama i savjetima koje korisnici ostavljaju o poduzećima, prijavama (*check-in*) koje korisnici rade u poduzećima te slikama koje objavljuju.

Ukupno ovi podaci čine 10,5 GB podataka, a u ovoj analizi korišteni su podaci o recenzijama, koji sačinjavaju 6,5 GB.

U nastavku slijedi primjer dijela podataka vezanih za jednu recenziju, relevantnih za ovu analizu.

```
{
# String, jedinstveni identifikator recenzije, sacinjen od 22 simbola
"review_id": "zdSx_SD6obEhz9VrW9uAWA",

# String, jedinstveni identifikator korisnika koji je napisao recenziju,
    sacinjen od 22 simbola
"user_id": "Ha3iJu77CxlrFm-vQRs_8g",

# String, jedinstveni identifikator poduzeca o kojem je napisana
   recenzija, sacinjen od 22 simbola
"business_id": "tnhfDv5Il8EaGSXZGiuQGg",

# String, datum i vrijeme kada je napisana recenzija u formatu YYYY-MM-
   DD hh:mm:ss
"date": "2014-10-11 03:34:02",

# String, tekst recenzije
"text": "Great place to hang out after work: the prices are decent,
and the ambience is fun. It's a bit loud, but very lively. The staff
is friendly, and the food is good. They have a good selection of
drinks.",

...
}
```

3.2 Analiza kvalitete podataka

Yelpov skup podataka se, zbog svoje veličine i dostupnosti, često koristi te je radi toga zanimljivo istražiti koliko je on kvalitetan. Pri kvaliteti skupa zapravo se misli na kvalitetu podataka o recenzijama, pošto su one suština Yelp platforme. U ovom radu fokus će biti na vremenskoj komponenti recenzija. Proučavat će se vrijeme koje korisniku protekne između pisanja dviju uzastopnih recenzija te će se na temelju toga pokušati odrediti kvaliteta skupa podataka.

Priprema podataka

Pošto su podaci već strukturirani u JSON formatu, lako ih je učitati u Sparkov DataFrame pomoću `spark.read.json()` naredbe. U podacima je za svaku recenziju spremljen jedinstveni identifikator (ID) korisnika koji ju je napisao pa se lako mogu dobiti sve recenzije jednog korisnika. Ako se te recenzije poredaju kronološki, može se izračunati vrijeme koje

je prošlo između pisanja dviju uzastopnih recenzija.

Za povezivanje jedne recenzije sa sljedećom od istog korisnika, najjednostavnije je bilo iskoristiti Sparkovu opciju izvršavanja SQL upita nad DataFrameom, pošto SQL ima ugrađene funkcije za dobivanje takvih rezultata.

```
df_reviews = spark.read.json("yelp_academic_dataset_review.json")

# Kreiranje pogleda na DataFrame df_reviews
df_reviews.createOrReplaceTempView("reviews")

# Spremanje podataka o uzastopnim recenzijama u novi DataFrame
df_reviews_next = spark.sql(
    """
    SELECT
        user_id, review_id, date, business_id,
        LAG(review_id, -1, '-') OVER(PARTITION BY user_id ORDER BY date)
            AS next_review_id,
        LAG(date, -1, '-') OVER(PARTITION BY user_id ORDER BY date)
            AS next_date
    FROM reviews
    """)
```

U skupu podataka postoji 8.635.403 recenzija, od čega 6.445.946 onih koje imaju recenziju sljedbenika. To znači da ima 2.189.457 recenzija koje su zadnje napisane od strane njihovog autora. Pošto takve recenzije nisu od interesa u ovoj analizi, u nastavku će se koristiti recenzije koje imaju sljedbenika.

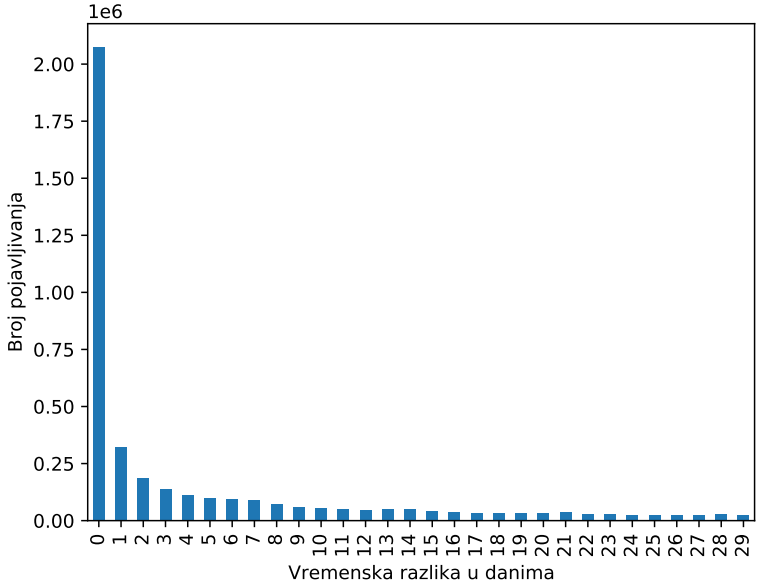
```
df_successive_reviews = df_reviews_next.where("next_date != '-'")
```

Distribucija vremenskih perioda

Za dobivanje distribucije vremenskih perioda između dviju uzastopnih recenzija, dodani su stupci u DataFrame s vremenskom razlikama u sekundama, minutama, satima i danima.

```
df_successive_reviews = df_successive_reviews\
        .withColumn("date_diff_sec",
            to_timestamp(col("next_date")).cast(LongType()) -
            to_timestamp(col("date")).cast(LongType()))
        .withColumn("date_diff_min",
            round(col("date_diff_sec") / 60))\
        .withColumn("date_diff_hours",
            round(col("date_diff_sec") / 3600))\
        .withColumn("date_diff_days",
            round(col("date_diff_sec") / 86400))
```

Slutnja je na početku analize bila da većina korisnika ne piše često recenzije pa je distribucija prvo prikazana u odnosu na dnevne vremenske razmake. Na slici 3.1 prikazan je broj pojavljivanja dana manjih od 30 pošto za veće vremenske razlike postoji mali broj pojavljivanja (relativno na broj pojavljivanja manjih vremenskih razlika), a i njihovo dodavanje čini graf vrlo nepreglednim.

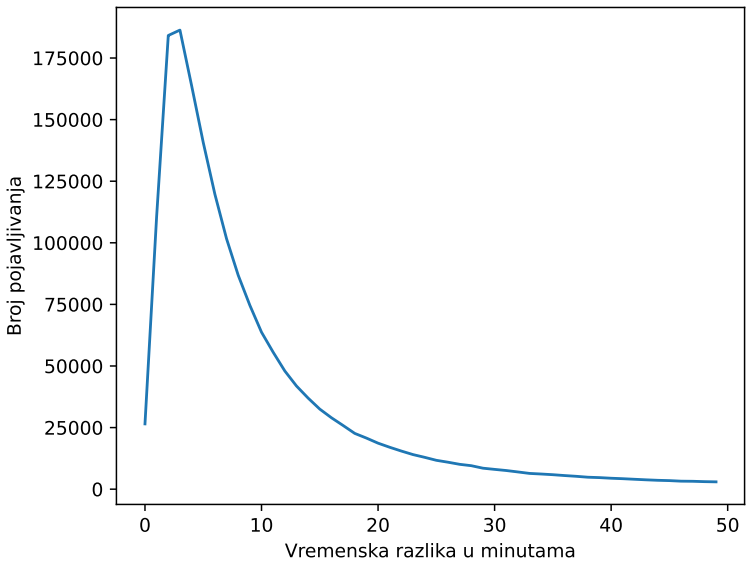


Slika 3.1: Distribucija vremenskih razlika u danima

Ono što je svakako iznenađujuće je količina recenzija koje su napisane isti dan kao i prethodna recenzija. Radi toga, slika 3.2 prikazuje distribuciju vremenskih razlika u minutama. Ponovno radi preglednosti grafa, prikazane su samo vremenske razlike manje od 50 minuta.

Opet iznenađujuće, najveći broj pojavljivanja imaju vremenske razlike u trajanju od oko 3 minute. Također, u oči upada velik broj pojavljivanja vrijednosti 0 minuta. Dakle, po ovim podacima postoji više od 250.000 slučajeva u kojima su korisnici napisali dvije recenzije unutar jedne minute. Taj podatak je vrlo neočekivan te je potaknuo na detaljniji pogled manjih vremenskih razlika, u sekundama, što je prikazano na slici 3.3.

Vidljiva je velika količina vrijednosti od 0 sekundi, koja se vizualno ne uklapa u ostatak grafa. Vrijednost od 0 sekundi znači da su dvije različite recenzije napisane iste sekunde, što ne djeluje moguće. No, u podacima zaista postoje takve vrijednosti, dapače više od

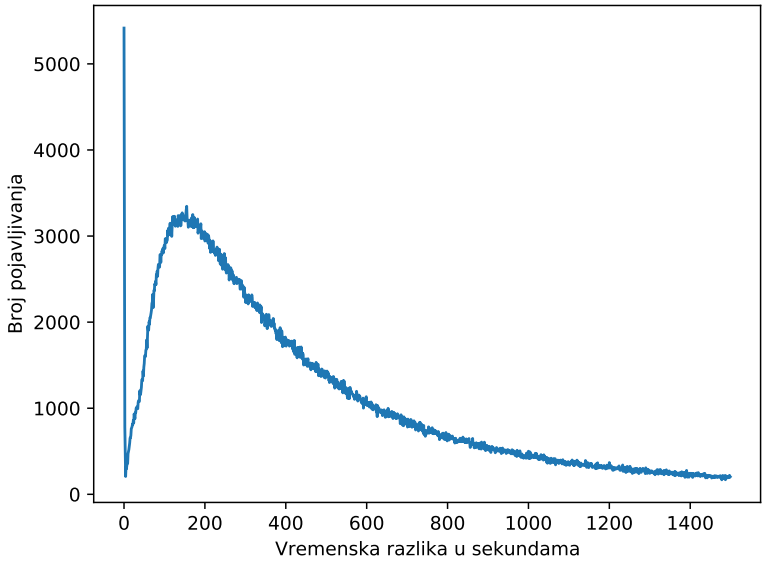


Slika 3.2: Distribucija vremenskih razlika u minutama

5.400 takvih. U nastavku slijedi prikaz recenzija napisanih od strane jednog korisnika, koje sve imaju u sekundu isto vrijeme pisanja.

```
df_successive_reviews\
    .where("date_diff_sec == 0 and
            user_id == '-32Lun4LUnhTKIaKcd9ulw'")\
    .select("review_id", "date", "business_id")\
    .show()

+--------------------+-------------------+--------------------+
|           review_id|               date|         business_id|
+--------------------+-------------------+--------------------+
|4ZhNMDspaVaABUeRI...|2014-06-19 16:49:10|uSz3be1o76OhMUa9t...|
|smWh82Dm_vaseCym_...|2014-06-19 16:49:10|8TJj1Wtf8xbUtHaIg...|
|emoUQBW61zNsndgtq...|2014-06-19 16:49:10|drMEqusiIKvxwH9K0...|
+--------------------+-------------------+--------------------+
```

Jasno je da ovi podaci ne mogu biti točni. Također, na slici 3.3 se vidi velik broj pojavljivanja malih vremenskih razlika općenito, a ne samo onih koje imaju vrijednost 0 sekundi.



Slika 3.3: Distribucija vremenskih razlika u sekundama

Valja napomenuti da se na temelju ovih podataka ne može zaključiti da korisnici *većinom* pišu recenzije u razmaku od nekoliko minuta, pošto postoji još mnoštvo recenzija koje su napisane u puno većim vremenskim razmacima, poput jedne godine. No, svi ti podaci nisu prikazani na grafovima radi njihove preglednosti.

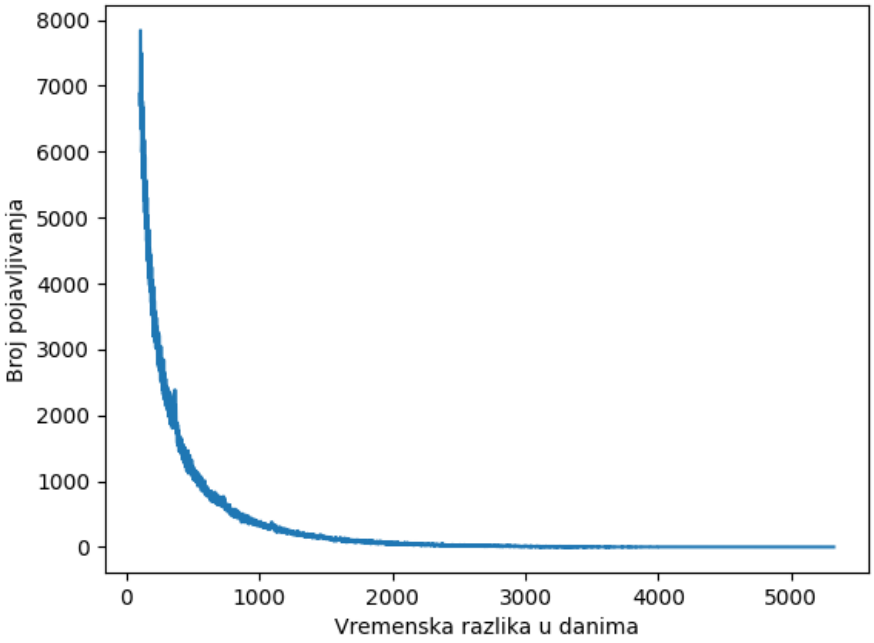
Međutim, od koje god veličine vremenskog razmaka se pogledaju podaci, uvijek se vidi nagli pad broja ponavljanja, sličan kao što se vidi na slici 3.1. Na primjer, na slici 3.4 prikazana je distribucija vremenskih razmaka od 100 dana nadalje te se vidi nagli pad, samo što su brojevi ponavljanja 10^3 puta manjeg reda veličine nego na slici 3.1. Također, sličan pad, samo malo manje strm, vidi se na donjem grafu iste slike, s vremenskim razmacima od 1000 dana nadalje.

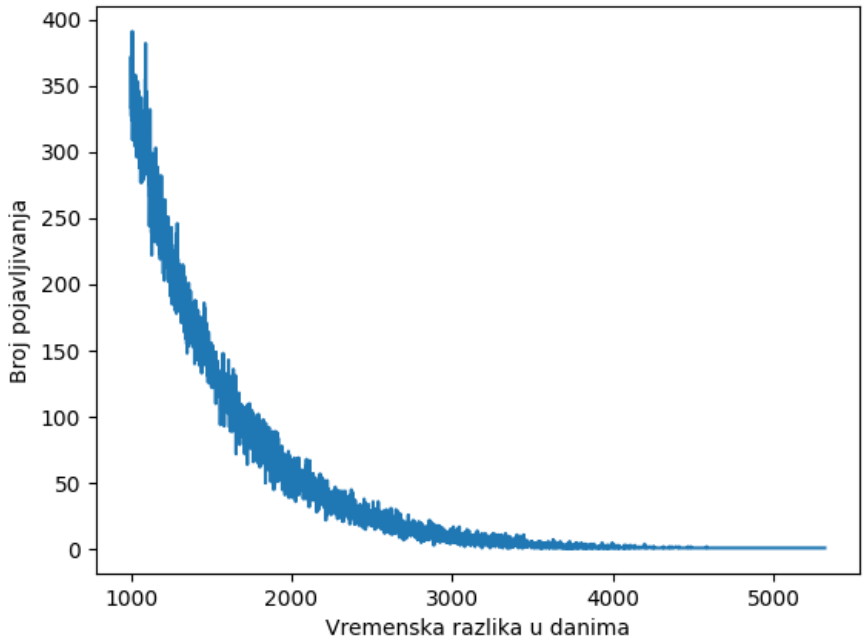
Gornje opservacije potiču na kategorizaciju vremenskih razmaka. Naime, osim što je nezgrapno raditi s podacima ovako širokog raspona, nepotrebno je. Općenito, napiše li korisnik recenziju za godinu dana ili godinu dana i jedan dan, ne pravi nikakvu razliku.

U nastavku je naredbom `summary()` ispisan sažetak podataka o vremenskim razlikama. Konkretno, prikazani su razni percentili podataka, na temelju kojih su odabrane kategorije koje se mogu vidjeti na slici 3.5. Pri tome, kategorija ne uključuje podatke iz kategorija

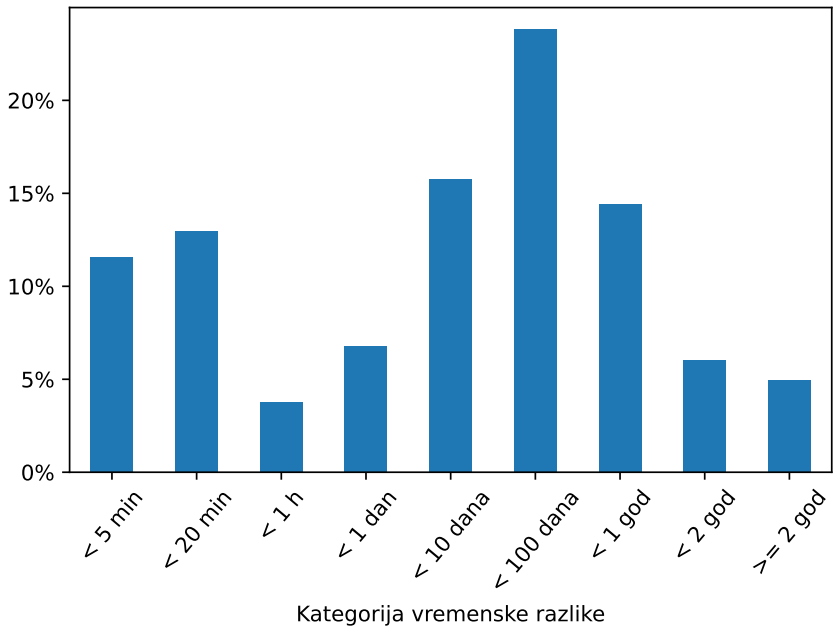
manje vremenske razlike. Na primjer, kategorija `< 1 h` uključuje vremenske razmake koji su manji od jedan sat, ali ne one koji su upali u kategorije `< 5 min` i `< 20 min`.

```
+-------+-------------+----------+--------+-------+------+
|summary|date_diff_sec|    minute|    sati|   dani|godine|
+-------+-------------+----------+--------+-------+------+
|    min|            0|       0.0|     0.0|    0.0|   0.0|
|    10%|          259|      4.32|    0.07|    0.0|   0.0|
|    25%|         1304|     21.73|    0.36|   0.02|   0.0|
|    50%|       778780|  12979.67|  216.33|   9.01|  0.02|
|    75%|      8923032|  148717.2| 2478.62| 103.28|  0.28|
|    85%|     21627914| 360465.23| 6007.75| 250.32|  0.69|
|    95%|     62484283|1041404.72|17356.75|  723.2|  1.98|
|    max|    459582464|7659707.73|127661.8|5319.24| 14.57|
+-------+-------------+----------+--------+-------+------+
```





Slika 3.4: Distribucija vremenskih razlika u danima, od 100 (gore) i 1000 (dolje) dana nadalje



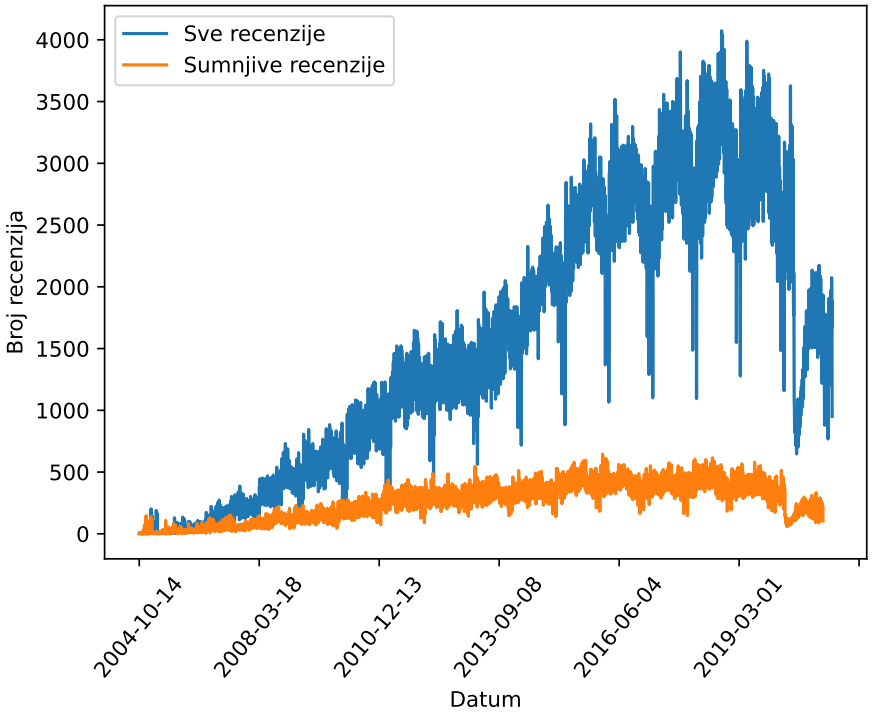
Slika 3.5: Distribucija vremenskih razlika među kategorijama

Na slici 3.5 vidi se da je vrijeme između pisanja uzastopnih dviju recenzija u skoro 25% slučajeva u kategoriji `< 100 dana`, odnosno između 10 i 100 dana. Općenito, veliki postotci recenzija u kategorijama `< 10 dana`, `< 100 dana` i `< 1 god` su za očekivati s obzirom na prirodu podataka. S druge strane, vidi se da je oko 25% recenzija napisano u manje od 20 minuta, što nije iznenađujuće s obzirom na prethodne grafove, ali svakako odskače od ostatka podataka na slici 3.5.

Zaključak ovog pododjeljka je da postoji neočekivano velik broj recenzija nakon kojih je u nekom kratkom periodu napisana sljedeća recenzija od strane istog korisnika. Da bi se detaljnije proučile takve recenzije, potrebno je točno odrediti što znači da je period između dvije recenzije „mali”. Temeljem slika 3.5 i 3.3, određena je granica od 900 sekundi, odnosno 15 minuta. Dakle, recenzije nakon kojih je napisana sljedeća recenzija u razmaku manjem od 15 minuta smatraju se sumnjivim. Takvih recenzija ima 1.457.946, odnosno skoro 17% svih recenzija te skoro 23% recenzija koje imaju sljedbenika. U nastavku ovog poglavlja detaljnije će se proučiti takve recenzije, kako bi se zaključilo jesu li ispravne.

Vrijeme pisanja recenzija

Jedno od mogućih objašnjenja malih vremenskih perioda između recenzija je naknadna manipulacija podataka od strane Yelpa. Na primjer, možda se radi o starim recenzijama koje su migrirane iz prijašnje baze podataka, u kojoj nije bilo podataka o vremenu pisanja recenzije, pa je naknadno stavljeno vrijeme migracije ili uređivanja podataka. Time bi puno recenzija imalo isto ili slično vrijeme pisanja pa ne bi bilo neobično vidjeti uzastopne recenzije u vremenskom periodu od nekoliko minuta.



Slika 3.6: Broj napisanih recenzija u danu

Ako se radi o ovom ili sličnom scenariju, tada postoji neki manji vremenski period u kojemu je napisan veći broj recenzija nego što je uobičajeno. Na slici 3.6 plavom je bojom za svaki datum prikazan broj recenzija napisanih taj dan, gledajući pritom sve recenzije u skupu podataka. Vidi se da broj dnevno napisanih recenzija ima uzlazni trend te se primjećuje i sezonalnost podataka, ali se ne vidi iznenadan porast broja recenzija u nekom vremenskom periodu. Primjećuje se također da noviji podaci odskaču od ostalih jer ih ima manje. U kontekstu ove analize to nije bitno, a pošto se radi o novijim podacima, uzrok je

vjerojatno nepotpunost podataka.

Na istoj slici narančastom je bojom prikazan dnevni broj recenzija nakon kojih je napisana sljedeća unutar 15 minuta. Također se niti jedan dan ne vidi iznenadni porast tih recenzija. Dakle, ovaj graf ne upućuje na to da je naknadna manipulacija podacima uzrokovala pojavu sumnjivih recenzija.

Slijedi prikaz dijela koda za dobivanje gornjih rezultata. Pomoću stupca `"date"`, koji sadrži datum i vrijeme pisanja recenzije, napravljen je novi stupac `"date_"` koji sadrži samo datum. Zatim su sumnjive recenzije grupirane po datumu i izbrojane. Na kraju su agregirani podaci spremljeni u DataFrame Pythonove biblioteke Pandas[1], metodom `toPandas()`. To je zato što Spark kao takav nema mogućnost vizualizacije podataka. No, uz korištenje Sparka svakako se mogu koristiti i sve druge biblioteke dostupne u korištenom programskom jeziku, tako da Spark radi s njima u tandemu. Priprema podataka i izračuni rade se pomoću Sparka, a za vizualizaciju se može pozvati metoda `plot()` nad Pandas DataFrameom. Pritom se sama vizualizacija zapravo izvršava pomoću Pythonove biblioteke Matplotlib[2].

```
# Dodavanje stupca s datumom
df_successive_reviews = df_successive_reviews\
    .withColumn("date_", substring(col("date"), 0, 10))

# Agregiranje i spremanje rezultata u Pandas DataFrame
df_successive_reviews_date_agg = df_successive_reviews\
    .where("date_diff_sec < 900")
    .groupby("date_")\
    .count()\
    .toPandas()

# Vizualizacija
df_successive_reviews_date_agg.plot(x="date_", y="count", ...)
```

Ponavljajuće recenzije

Drugo moguće objašnjenje recenzija koje su napisane u malom vremenskom razmaku je da su napisane od strane robota. Najjednostavniji tip robota bio bi onaj koji piše identične recenzije istim ili različitim poduzećima.

[1] https://pandas.pydata.org/
[2] https://matplotlib.org/

Među parovima uzastopnih recenzija pronađeno je 11.134 njih koji imaju identičan tekst recenzije, od čega je 7.446 njih napisano istom poduzeću. Jedan dio parova identičnih uzastopnih recenzija koje su napisane istom poduzeću sigurno su greške jer se na platformama tog tipa znaju vidjeti ponovljeni komentari, recenzije, *postovi* i sl., zbog korisnikove greške ili greške u sustavu. S druge strane, identične recenzije napisane različitim poduzećima mogle bi biti napisane od strane robota. Takvih recenzija ukupno ima 3.688, a čak njih 3.137 (85%) su označene kao sumnjive jer su napisane u razmaku manjem od 15 minuta. Međutim, 3.137 recenzija čini samo 0,22% sumnjivih recenzija pa ne objašnjava veliku većinu njih.

Drugi način gledanja na ove podatke je da se korisnike koji su napisali identične uzastopne recenzije različitim poduzećima označi kao robote te da se sve njihove recenzije smatraju neispravnima. Kod u nastavku prikazuje dobivanje takvih korisnika te zbrajanje svih njihovih recenzija. Opet je korištena mogućnost Sparka da se s DataFrameovima radi kao s tablicama u relacijskoj bazi te se zbroj uzastopnih i sumnjivih recenzija dobiva pomoću SQL upita.

```
df_duplicates_users = df_successive_reviews\
        .where("text == next_text and business_id != next_business_id")\
        .select("user_id")\
        .distinct()

df_duplicates_users.createOrReplaceTempView("duplicates_users")
df_successive_reviews.createOrReplaceTempView("successive_reviews")

df_count_reviews_duplicates = spark.sql(
    """
    SELECT
        COUNT(*) AS count1,
        COUNT(CASE WHEN date_diff_sec < 900 THEN review_id END) AS
            count2
    FROM successive_reviews a
    INNER JOIN duplicates_users b
    ON a.user_id = b.user_id
    """
)
```

Tim postupkom dobiva se 93.791 recenzija, od čega 30.723 sumnjivih (2,1% svih sumnjivih recenzija). Dakle, i ovom metodom može se objasniti samo jako mali udio recenzija.

Do sada su se proučavale samo *uzastopne* recenzije koje imaju identičan tekst. Ali, moguće je da postoje roboti koji pišu identične tekstove u recenzijama, ali ne uzastopno. Na primjer, oni mogu imati nekoliko različitih tekstova koje koriste ciklički pa se na taj način ni jedan neće ponoviti uzastopno. Ako se korisnici koji su barem jednom napisali

dvije iste recenzije označe kao roboti te se zbroje sve njihove recenzije, dobiva se 109.423 sumnjivih recenzija, odnosno 7,5%.

Zaključak proučavanja ponavljajućih recenzija je da njihova prisutnost, pogotovo onih različitim poduzećima, daje indikaciju prisutnosti robota, ali pomoću njih se ne može objasniti velik dio sumnjivih recenzija.

Poduzeća koja dobivaju sumnjive recenzije

Kada postoji sumnja da roboti pišu recenzije, postavlja se pitanje kojim poduzećima se one sumnjive pišu te jesu li kupljene.

Podaci pokazuju da čak 77% svih poduzeća ima barem jednu sumnjivu recenziju. Kako je to stvarno velik postotak, moguće je da poduzeća pri dolasku na platformu dobiju recenzije od robota. To bi mogao biti Yelpov mehanizam zadržavanja poduzeća na platformi, kao i poticanja korisnika na ostavljanje recenzija tim poduzećima. Tome u prilog ide činjenica da u Yelpovom skupu podataka ne postoji ni jedno poduzeće koje nema niti jednu recenziju. Međutim, je li to zaista točan podatak ili su poduzeća bez recenzija uklonjena iz skupa podataka, ne zna se.

Ako se recenzije sa sljedbenikom grupiraju po poduzeću o kojem su napisane te poredaju kronološki, dobije se redoslijed recenzija napisanih o nekom poduzeću. U nastavku je kod za dobivanje poretka (*rank*) recenzija u poduzeću te primjer dvaju poduzeća i njihovih recenzija.

```
df_business_reviews = spark.sql(
    """
    SELECT
        business_id, review_id, date,
        RANK() OVER(PARTITION BY business_id ORDER BY date) AS rank
    FROM reviews
    """)
```

```
+--------------------+--------------------+-------------------+----+
|         business_id|           review_id|               date|rank|
+--------------------+--------------------+-------------------+----+
|--0DF12EMHYI8XIgo...|WQrgE4DqTwUEDCBT-...|2014-11-25 20:35:03|   1|
|--0DF12EMHYI8XIgo...|GufMa-RHVjLGK8dKz...|2015-06-17 17:25:19|   2|
|--0DF12EMHYI8XIgo...|S53YoW4aOvZDR2YkF...|2015-09-03 13:43:13|   3|
|--0DF12EMHYI8XIgo...|2tCUJo_x4MuEtPyQc...|2017-06-07 01:59:58|   4|
...
|--0r8K_AQ4FZfLsX3...|i7W0pSfyae8pgMwpA...|2017-09-03 17:15:48|   1|
```

```
|--0r8K_AQ4FZfLsX3...|7uAEYWIEeSJgC9EhR...|2017-11-15 20:19:50|   2|
|--0r8K_AQ4FZfLsX3...|o-0eZF-xeR8GeodqR...|2019-10-26 23:08:46|   3|
|--0r8K_AQ4FZfLsX3...|PgTEptriK23jopn-O...|2019-11-06 20:20:03|   4|
|--0r8K_AQ4FZfLsX3...|oqckRGgF2La7Izo8J...|2020-04-15 13:33:08|   5|
+--------------------+--------------------+-------------------+----+
```

Ako je istina da su sumnjive reakcije napisane od strane robota o poduzećima koja su tek pristupili Yelp platformi, tada bi njihovi rangovi trebali većinom biti male vrijednosti. Odnosno, općenito bi rangovi tih recenzija trebali biti manji od rangova svih recenzija. Međutim, sljedeći sažetak pokazuje da to nije slučaj. Dapače, sumnjive recenzije imaju veću srednju vrijednost rangova.

```
+-------+-------+-----------------------+
|summary|   rank|rank sumnjive recenzije|
+-------+-------+-----------------------+
|   mean| 193.64|                 237.15|
| stddev| 437.61|                  527.6|
|    min|      1|                      1|
|    25%|     18|                     22|
|    50%|     63|                     78|
|    75%|    192|                    232|
|    max|   9295|                   9291|
+-------+-------+-----------------------+
```

Također, nije pronađena poveznica između sumnjivih recenzija i nekih određenih poduzeća, za koja bi se tada moglo zaključiti da kupuju svoje recenzije. Za svako poduzeće, nađen je udio sumnjivih recenzija u odnosu na ukupan broj recenzija napisanih tom poduzeću. Ona poduzeća koja imaju velik udio sumnjivih recenzija uvijek imaju ukupno mali broj recenzija. Također, takvih poduzeća nema puno pa se niti ovom metodom ne može objasniti velika većina sumnjivih recenzija.

Korisnici koji pišu sumnjive recenzije

Jednako kao što je u prošlom pododjeljku svakoj recenziji dodijeljen redni broj u odnosu na poduzeće o kojem je napisana, isto tako se može dodijeliti i redni broj u odnosu na korisnika koji ju je napisao. Motivacija za to je vidjeti mogu li se sumnjive recenzije povezati s rednim brojem njihovog pisanja, u odnosu na ostale recenzije istog korisnika. Na primjer, jedno objašnjenje uzastopnih recenzija napisanih u razmaku manjim od 15 minuta moglo bi biti da su to prve dvije recenzije korisnika. Naime, moguće je da je korisnik u trenutku pristupanja Yelp platformi zainteresiraniji za ostavljanje recenzija nego inače pa u kratkom

vremenu ostavi više od jedne recenzije.

No, u podacima nema nikakvih naznaka takvog ponašanja, kao što prikazuje sljedeći ispis rangova sumnjivih i svih recenzija.

```
+-------+------------------------+----+
|summary|rank sumnjive recenzije|rank|
+-------+------------------------+----+
|    min|                       1|   1|
|     5%|                       1|   1|
|    10%|                       1|   1|
|    25%|                       3|   2|
|    50%|                       8|   7|
|    75%|                      29|  26|
|    80%|                      40|  37|
|    90%|                      89|  85|
|    max|                    6071|6072|
+-------+------------------------+----+
```

Sljedeći pristup je koristiti isključivo sumnjive recenzije te pogledati koliko ih je koji korisnik napisao. Korisnika koji su napisali barem jednu sumnjivu recenziju ima 393.709, što od ukupnog broja korisnika čini 18%. U nastavku je prikazan kod za zbrajanje sumnjivih recenzija po korisniku i ispis sažetka podataka pomoću različitih percentila, koje korisnik može sam definirati.

```
user_review_count = df_successive_reviews\
    .where("date_diff_sec < 900")\
    .groupby("user_id")\
    .count()

user_review_count.select("count")\
    .summary("min", "25%", "50%", "75%", "80%", "90%", "95%", "99%",
        "max")\
    .show()
```

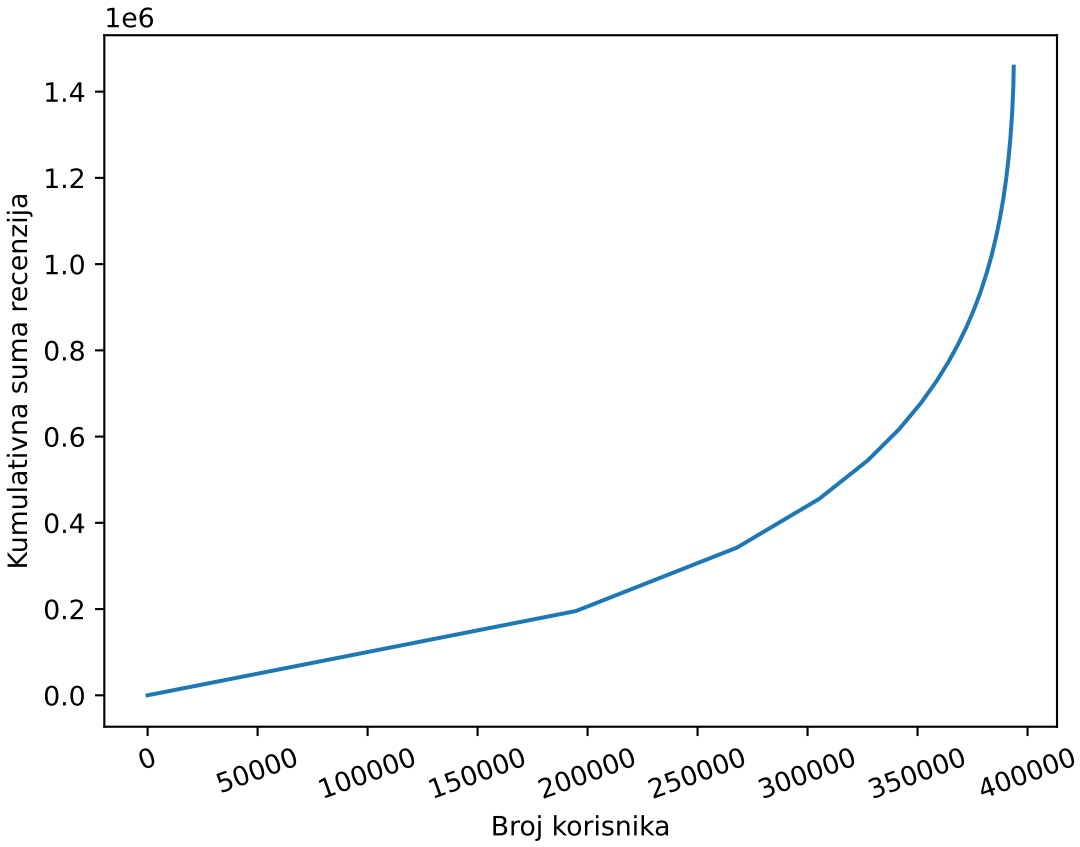
```
+-------+-----+
|summary|count|
+-------+-----+
|    min|    1|
|    25%|    1|
|    50%|    2|
|    75%|    3|
|    80%|    4|
|    90%|    7|
|    95%|   12|
```

```
|     99%|   36|
|     max| 3789|
+-------+-----+
```

Očekivano, najveći broj tih korisnika napisao je svega nekoliko sumnjivih recenzija, ali zanimljivo je da ima i onih koji su ih napisali preko 3.000. Primjećuje se da najmanje 25% korisnika ima samo jednu sumnjivu recenziju, a preko 90% njih ih ima manje od 10. Naravno, posebno su zanimljivi korisnici koji su napisali tisuće sumnjivih recenzija. Sljedeći ispis prikazuje dio recenzija korisnika s najvećim brojem recenzija, poredanih kronološki. Primjećuje se velika količina gusto raspoređenih recenzija koje ne mogu biti napisane od strane stvarne osobe, dakle vrlo je izgledno da se radi o robotu.

```
+--------------------+-------------------+--------------------+
|           review_id|               date|         business_id|
+--------------------+-------------------+--------------------+
|dkuSApVuOqFZAJXQU...|2018-03-30 17:33:56|Ofsa_M0q6oZ7twsgG...|
|F2OAB3uMoEVmuop4_...|2018-03-30 17:33:59|OrQjG2Wg7hBdVQGxT...|
|K9nLYkes7KF6aDU9c...|2018-03-30 17:35:58|DjHUxKQLBN3kFaZaW...|
|9a1d5aMJTWOhFmSAS...|2018-03-30 17:36:00|YKmwKQoWIeoLyStYr...|
|oIjAxg0L6OPdJ6BrW...|2018-03-30 17:36:02|eo_Ygo_85TPEwN0dH...|
|_eokwOlkm78blb2j_...|2018-03-30 17:36:04|HKtlZvPSL26NsII-a...|
|2FK9u2HvgTtn_nUD_...|2018-03-30 17:36:08|RMMVHnjtD_5iBhfQs...|
|i39Z9pFt83gpcjBFx...|2018-03-30 17:36:10|_JKL3wBoY5wpTeHhN...|
|gRWR-dGDLWJIaFE8H...|2018-03-30 17:36:13|pSburEkD_dG8gIcP3...|
|y1gJhS_TeNMuGzz54...|2018-03-30 17:36:15|gCuXXHOXocujsgFKs...|
|285x9nvLYA0lyPfYl...|2018-03-30 17:36:18|hidVKxL7xOU8ArNKD...|
|mxWk3cET5gumvbg2T...|2018-03-30 17:36:21|zGdzlxzg9qyMLIFCA...|
|xHxYB-GiyGLDf_Drc...|2018-03-30 18:10:55|qmrXkq79qIqahh6jN...|
|x9AlVerxdNw1JAgEk...|2018-03-30 18:10:57|wQX_TZqgV-xJH6aNN...|
|nuAoF9HFPH4-UD5Rv...|2018-03-30 18:10:58|ZzrPFl_E0Fa5s8kQo...|
|ySbh-dAXWiR7Kmjuh...|2018-03-30 18:11:00|CIUfKOF5bfkyWRrx5...|
|F6oZVN0V8fz_AI2HC...|2018-03-30 18:11:05|rmGsHhx-Dn7z9i59V...|
|RQvkyiky16TSnnYI8...|2018-03-30 18:11:07|PmL0q8IC4cxxFAyUB...|
|P2pMY3t5pNG0_8-Z5...|2018-03-30 18:11:10|oax3-jBdnCX3liH4A...|
|syp6CLhD0x8W5NRb7...|2018-03-30 18:11:13|DqthDA7duTDynAuVn...|
|0IInW4EtIrPUn0Ree...|2018-03-30 18:11:16|WCyc2DPB3WnF1amV1...|
+--------------------+-------------------+--------------------+
```

Sljedeće je pitanje može li se temeljem broja sumnjivih recenzija odrediti koji od ovih korisnika su roboti, a koji nisu. Radi toga je napravljena slika 3.7 koja prikazuje kumulativnu sumu broja sumnjivih recenzija. Dakle, korisnici su poredani po broju sumnjivih recenzija te su ti brojevi postepeno zbrojeni. Na slici se vidi da suma raste prilično glatko te je teško odrediti količinu sumnjivih recenzija koja čini korisnika robotom.



Slika 3.7: Kumulativna suma broja sumnjivih recenzija

Iz sažetka broja recenzija na strani 30 vidi se da ima manje od 10%, a više od 5% korisnika koji imaju više od 10 sumnjivih recenzija. Točnije, postoji 23.437 korisnika, odnosno njih 6% koji su napisali više od 10 takvih recenzija. S druge strane, broj svih sumnjivih recenzija od tih korisnika iznosi 658.510, odnosno čak 45%. Dakle, 6% svih korisnika koji su barem jednom napisali sumnjivu recenziju je odgovorno za 45% takvih recenzija.

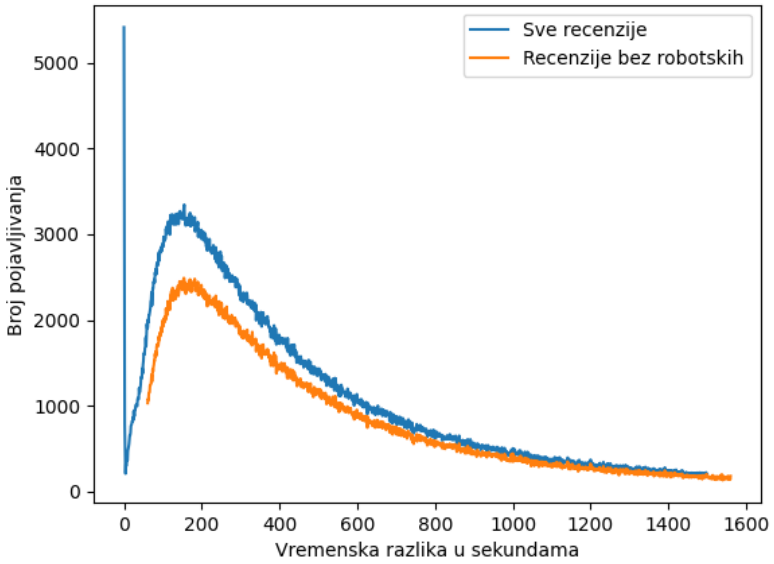
Detekcija robota

Na ispisu dijela recenzija jednog od najaktivnijih korisnika na strani 31 vidi se da nisu sve uzastopne recenzije napisane u razmaku manjem od 15 minuta (crveno označeno). To

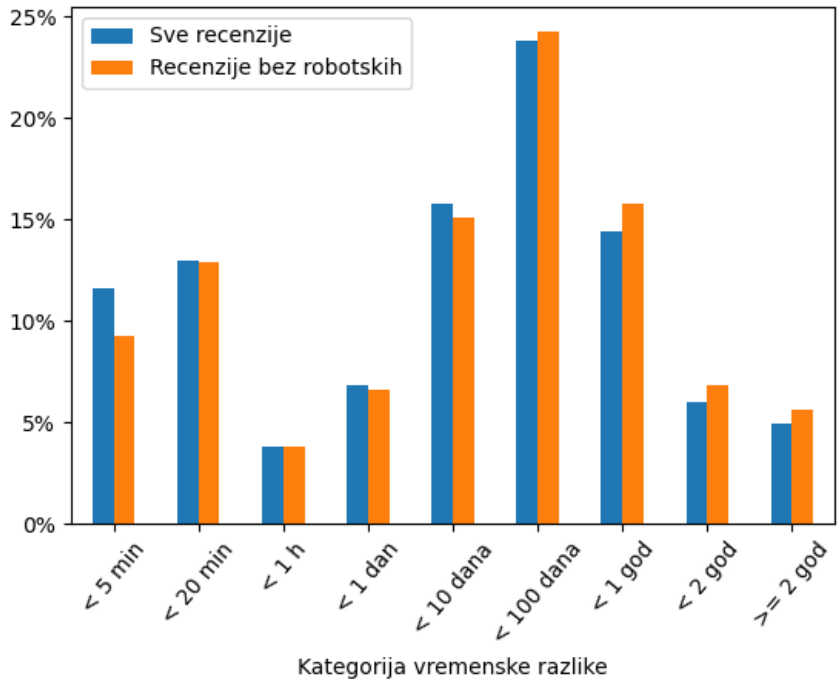
znači da iako je vrlo vjerojatno korisnik robot, dosadašnjim načinom određivanja recenzija kao sumnjivima propuštene su neke recenzije tog korisnika. To saznanje potiče na novi pristup detekcije robota i njihovih recenzija.

Neka su kao roboti označeni svi korisnici koji su barem jednom napisali više od jedne recenzije unutar jedne minute. Takvih korisnika ima 37.349, odnosno 1,7% svih korisnika. Od svih recenzija koje imaju sljedbenika, takvi korisnici napisali su ih čak 1.027.262, odnosno 16%. Od tih recenzija, 364.241 njih su ranije bile označene kao sumnjive, što čini 25%.

Na slici 3.8 prikazana je usporedba distribucije vremenskih razlika u sekundama. Plavom bojom prikazane su sve uzastopne recenzije, a narančastom one od korisnika koji ovom metodom nisu označeni kao roboti. Dakle, narančasti graf prikazuje recenzije nakon uklanjanja onih koje su napisali roboti. Vidi se da su uklonjene recenzije koje su imale sljedbenika unutar iste minute te da se općenito broj manjih vremenskih razlika smanjio. Također, na slici 3.9 prikazana je usporedba distribucija kategoriziranih vremenskih razlika. Najviše se primjećuje smanjenje udjela kategorije `< 5 min`. Isto tako, malo se smanjio udio kategorije `< 10 dana`, dok su se povećali udjeli svih kategorija s vremenskim periodom dužim od 10 dana.



Slika 3.8: Distribucija vremenskih razlika u sekundama prije i nakon micanja robota



Slika 3.9: Distribucija vremenskih razlika među kategorijama

Rezultati

Zaključak ove analize kvalitete podataka je da među podacima postoje recenzije koje nisu ispravne te postoji opravdana slutnja da su ih napisali roboti, a ne ljudi. No, u prošlim pododjeljcima pokazano je da nije lako detektirati sumnjive korisnike.

Očito se ipak radi o sofisticiranim robotima koji „znaju" kako da se što bolje uklope među ljude pa ih je teško uočiti. Također, Yelp sigurno ima mnoštvo mehanizama kako se bori s robotima. Lako je moguće da su neke recenzije napisane od strane robota već uklonjene s njihove strane, koje bi možda olakšale detekciju robota.

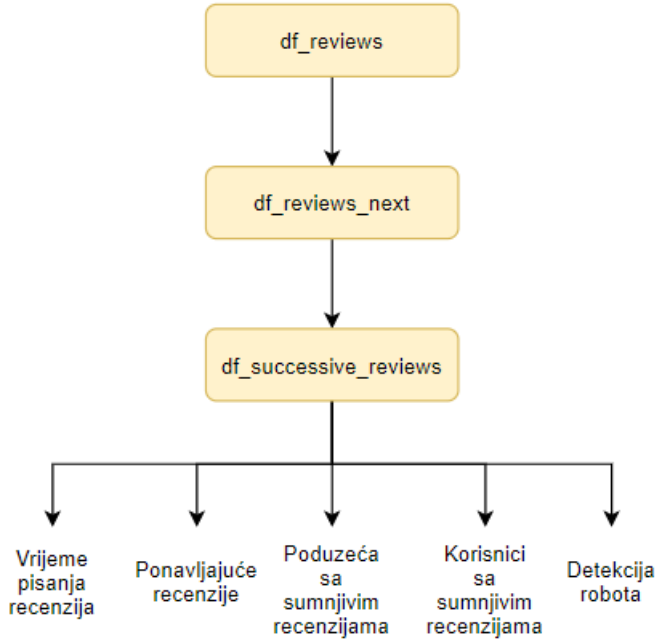
3.3 Performanse

U Spark aplikaciji koja je napravljena u sklopu ovog rada podaci su učitani i pripremljeni te su brojne analize i ispisi izvršeni. Također, napravljeni su svi dijagrami iz prošlog odjeljka,

kao i neki dodatni.

Prilikom pokretanja svoje Spark aplikacije korisnik ima mogućnost mijenjanja resursa koje ona koristi. Sa zadanim postavkama od 4 Spark izvršitelja, od kojih svaki ima 1 GB radne memorije i 4 procesorske jezgre, prosječno vrijeme izvođenja aplikacije bilo je 9 minuta. Smanjenje broja izvršitelja, s nepromijenjenim ostalim postavkama, imalo je veće vrijeme izvršavanje, dakle nisu se mogle dobiti jednake performanse i s manjim brojem izvršitelja.

Detaljniji pregled izvršavanja aplikacije na Spark korisničkom sučelju otkrio je da se podaci iznova učitavaju otprilike 10 puta tijekom aplikacije, što svakako nije dobro. Problem je u prirodi analize i Sparkovom načinu računanja DataFrameova. Naime, u aplikaciji su prvo učitani podaci u DataFrame `df_reviews`, zatim pripremljene dodatne informacije i spremljene u `df_reviews_next`, a iz toga je dobiven DataFrame `df_successive_reviews` s uzastopnim recenzijama, kao što je prikazano na slici 3.10. Od tamo je analiza krenula u različitim smjerovima, a za veliku većinu daljnjih izračuna i ispisa korišten je DataFrame `df_successive_reviews`.



Slika 3.10: Shema analize kvalitete Yelp podataka iz prošlog odjeljka

Kao što je objašnjeno u odjeljcima 2.4 i 2.6, Spark sprema sve informacije potrebne za stvaranje nekog DataFramea te ih izvršava tek kad korisnik zada neku akciju nad tim

DataFrameom, kao što je ispis rezultata. To znači da je za sve rezultate u analizi Spark iznova učitavao podatke i prolazio isti postupak pripreme. Odnosno, svaki put su se iznova izvršavala prva tri koraka prikazana na shemi 3.10, što je svakako suvišno.

Da bi se riješio ovaj problem, Spark ima mogućnost ranije evaluacije DataFramea, nakon čega ga sprema u memoriju i/ili na disk, ovisno o resursima. Na taj način se ne ponavljaju koraci za stvaranje tog DataFramea, već je on direktno dostupan. Opisani postupak ostvaruje se operacijom `cache()`, a u ovoj analizi primijenjen je na DataFrameu `df_successive_reviews`, pošto se najviše koristi.

```
1 df_successive_reviews.cache()
```

Nakon dodavanja ovog koraka trebalo je povećati količinu radne memorije dostupne izvršiteljima, ali izvršavanje se tada jako ubrzalo. Sa 4 GB radne memorije svakog izvršitelja, vrijeme izvršavanja u prosjeku se smanjilo na 6,2 minute, a sa 6 GB na 5,7 minuta. Valja napomenuti da samo povećanje radne memorije bez dodavanja operacije `cache()` nije poboljšalo vrijeme izvršavanja.

Zaključak

U ovom radu objašnjena je potreba za naprednim tehnologijama za obradu podataka, prikazana je arhitektura i način rada Apache Spark platforme te je pomoću nje provedena analiza velike količine podataka.

U teorijskom dijelu prvo je objašnjeno kako se Spark uklapa u *big data* ekosustav kao što je Hadoop. Zatim je objašnjeno Sparkovo particioniranje podataka koje mu omogućuje paralelizam pri izvršavanju. Na kraju je dan pregled Sparkovih struktura podataka i različitih operacija koje se provode nad njima.

Kao demonstracija Sparkovih mogućnosti i prednosti napravljena je Spark aplikacija u kojoj se analizira kvaliteta javno dostupnog skupa podataka. Prilikom rada sa Sparkom, odmah se primjećuje da ga nije teško koristiti. Sparkova mogućnost korištenja četiri programska jezika čini ga vrlo dostupnim njegovim korisnicima. Apstrakcije podataka koje se koriste u Sparku, kao što je DataFrame, imaju jako logičan način korištenja. Rad s njima nalikuje na rad s Pandas strukturama u Pythonu, tablicama podataka u R-u ili s tablicama u relacijskoj bazi podataka, zbog čega korisnik često ne mora iznova učiti sintaksu.

Osim brzine izvođenja, svakako treba napomenuti da je velika prednost Spark aplikacije to što omogućuje cjelovito rješenje problema. Na primjer, u aplikaciji napravljenoj u sklopu ovog rada, prvo su učitani i očišćeni podaci, zatim provedene razne analize te nacrtani brojni grafovi, pri čemu je korišten Spark i razne Pythonove biblioteke. Aplikacija tog tipa napisana u istom programskom jeziku, ali bez korištenja Sparka, ne bi se mogla efikasno nositi s količinom podataka koja je korištena, iako Python ima prilično dobre biblioteke za analizu podataka, kao što su Pandas i Numpy[3].

Nakon rada sa Sparkom, jasno je zašto je jedna od najkorištenijih tehnologija za rad s velikom količinom podataka.

[3] https://numpy.org/

Bibliografija

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes i R. E. Gruber, *Bigtable: A Distributed Storage System for Structured Data*, https://storage.googleapis.com/pub-tools-public-publication-data/pdf/68a74a85e1662fe02ff3967497f31fda7f32225c.pdf.

[2] J. S. Damji, B. Wenig, T. Das i D. Lee, *Learning Spark*, O'Reilly, 2020.

[3] J. Damji, *A Tale of Three Apache Spark APIs: RDDs vs DataFrames and Datasets*, 2016, https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html.

[4] J. Dean i S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, https://storage.googleapis.com/pub-tools-public-publication-data/pdf/16cb30b4b92fd4989b8619a61752a2387c6dd474.pdf.

[5] Data Flair, *Fault tolerance in Apache Spark – Reliable Spark Streaming*, https://data-flair.training/blogs/fault-tolerance-in-apache-spark.

[6] ———, *RDD lineage in Spark: ToDebugString Method*, https://data-flair.training/blogs/rdd-lineage.

[7] S. Ghemawat, H. Gobioff i S. Leung, *The Google File System*, (2003), https://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf.

[8] Apache Hadoop, *HDFS Architecture Guide*, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[9] IBM, *What is distributed computing*, https://www.ibm.com/docs/en/txseries/8.1.0?topic=overview-what-is-distributed-computing.

[10] G. Jevtic, *What is Hadoop Mapreduce and How Does it Work*, 2020, https://phoenixnap.com/kb/hadoop-mapreduce.

[11] T. M. Porter, *Probability and statistics*, https://www.britannica.com/science/probability.

[12] Apache Spark, *RDD Programming Guide*, https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds.

[13] ———, *Spark SQL, DataFrames and Datasets Guide*, https://spark.apache.org/docs/latest/sql-programming-guide.html.

[14] Dr. M. van Rijmenam, *A Short History Of Big Data*, 2013, https://datafloq.com/read/big-data-history/239.

[15] X. Xu, *Databricks Spark jobs optimization techniques: Shuffle partition technique (Part 1)*, 2020, https://nealanalytics.com/blog/databricks-spark-jobs-optimization-techniques-shuffle-partition-technique-p

[16] Yelp, *Yelp Open Dataset*, https://www.yelp.com/dataset.

Sažetak

U današnjem digitalnom svijetu potrebno je analizirati ogromne količine podataka. Da bi to bilo moguće, najprije su se razvili distribuirani računalni sustavi i paralelno procesiranje podataka, a s vremenom i mnoštvo tehnologija za obradu i analizu podataka, jedna od kojih je Apache Spark.

Apache Spark je platforma osmišljena za distribuirano procesiranje velike količine podataka. Ona se odlikuje svojom brzinom, širinom primjene i jednostavnosti korištenja. Brzina izvođenja ostvaruje se čuvanjem podataka u memoriji i internom optimizacijom koda. U Sparku se može raditi sa strukturiranim i graf podacima te tokovima podataka, a mogu se provoditi razne analize, uključujući strojno učenje.

U praktičnom dijelu radu napravljena je analiza javno dostupnog skupa podataka Yelp platforme koristeći Apache Spark. Radi se o podacima o poduzećima, njihovim korisnicima i recenzijama koje oni ostavljaju. Analizirala se kvaliteta tih podataka tako što su se proučavale vremenske razlike između dvije uzastopne recenzije istog korisnika. Fokus analize bila je detekcija recenzija koje su napisane od strane robota.

Summary

In today's digital world there is a need for analysing vast amounts of data. To achieve that, distributed computer systems and parallel processing were invented. In the following years, a number of technologies for processing and analysing data were created, with Apache Spark being one of them.

Apache Spark is a platform designed for large-scale distributed data processing. Its main characteristics are speed, broad application and ease of use. Its speed is achieved with in-memory storage and code optimisation. Spark can be used on structured, graph and streaming data to perform different types of analyses, including machine learning.

As part of this thesis, data quality analysis was performed on Yelp's open dataset using Apache Spark. The data consisted of information about businesses, their customers and the reviews that they write. The focus of the analysis was on bot detection, which was performed by exploring the time between two consecutive reviews of a customer.

Životopis

Rođena sam u Zagrebu 18. prosinca 1995. Osnovnu školu pohađala sam u OŠ Gustava Krkleca u naselju Travnom, u Zagrebu. Nakon toga pohađala sam XV. gimnaziju u Zagrebu te Školu suvremenog plesa Ane Maletić, gdje sam zaradila diplomu plesnog edukatora. Godine 2014. upisala sam Medicinski fakultet Sveučilišta u Zagrebu te položila dvije godine studija, ali potom odlučila prekinuti studij i baviti se matematikom. 2016. godine upisala sam preddiplomski studij Matematike na Prirodoslovno matematičkom fakultetu u Zagrebu. Nakon završetka preddiplomskog studija, 2019. godine, upisala sam diplomski studij Matematičke statistike na istom fakultetu.