SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1347

Obrada podataka tehnologijom Apache Spark

Martin Matak

Zagreb, svibanj 2016.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`. Na ovoj stranici se

nalazi izvornik.

Zahvala - TODO . . . :)

SADRŽAJ

1. Uvod

1.1. Motivacija

Pametni mobilni uređaji postaju neizostavan dodatak svakog modernog čovjeka. Većina pametnih mobilnih uređaja u sebi sadrži sljedeće senzore:

akcelerometar - elektromehanička komponenta koja mjeri sile ubrzanja;

barometar - mehanički senzor za mjerenje atmosferskog pritiska (na trenutnoj lokaciji uređaja);

senzor svjetlosti - mjeri intenzitet, tj. jačinu svjetlosti i uglavnom se nalazi s prednje strane uređaja, iznad ekrana;

senzor blizine - u stanju prepoznati situacije kada mu neki objekt stoji u blizini - ovo omogućava automatske pozive prilikom primicanja telefona licu uz zaključavanje telefona da bi onemogućili slučajno prekidanje poziva uhom ili slično;

senzor gestikulacije - prepoznaje kretnje ruke tako što detektira infracrvene zrake koje se reflektiraju - omogućuje nam djelomično upravljanje telefonom bez dodirivanja ekrana;

žiroskop - uređaj koji se koristi za navigaciju i merenje kutne brzine;

geomagnetski senzor - mjeri okolno geomagnetsko polje za sve tri fizičke osi i u biti služi kao kompas na mobilnim uređajima i

Hall Sensor - magnetski senzor zadužen za prepoznavanje je li maska telefona zatvorena ili otvorena.

Pretpostavimo da je mobilni uređaj spojen na internet i da svakih nekoliko sekundi pošalje vrijednost koju u tom trenutku mjeri pojedini senzor. U samo jednom danu može se skupiti dosta podataka. A što kada to ne bi radili za jedan uređaj nego za sve izdane uređaje nekog modela? Količina podataka bi jako brzo narasla.

Kako količina podataka postaje sve veća, dolazimo do pojma *Velika količina podataka* (engl. *Big Data*). U današnje vrijeme imamo više podataka u digitalnom obliku nego što smo ikada imali. Jedan od zanimljivijih izazova je kako ih efektivno obraditi i zaključiti nešto iz toga. Kako od te velike količine podataka doći do nekih pametnih zaključaka iz kojih ćemo nešto novo naučiti.

Apache Spark je otvorena (engl. *open source*) tehnologija koja omogućava pisanje programa za obradu podataka u tri programskih jezika: *Java*, *Python* i *Scala*; a nudi i mogućnost interaktivnog rada.
U okviru ovog rada biti će proučene mogućnosti ove tehnologije, razrađeno nekoliko konkretnih primjera obrade podataka te ostvarena programska rješenja koja obavljaju tu obradu koristeći *Apache Spark*.

Svi primjeri će biti napisani u programskom jeziku *Java*.

1.2. Instalacija i izvorni kodovi

1.2.1. Instalacija

Apache Spark je pisan u programskom jeziku *Scala* i izvršava se na *Javinom virtualnom stroju* (engl. *Java Virtual Machine*) (JVM). Instalacija na osobno računalo je prilično jednostavna, a u ovdje će biti prikazana instalacija na operacijskom sustavu *Ubuntu 15.10*.
Za početak je potrebno imati instaliranu Javu, a je li Java instalirana na računalu se može provjeriti tako što se u naredbenom retku unese sljedeća naredba:
`java -version`. Kao rezultat bi trebali dobiti trenutno instaliranu verziju Jave. Ukoliko Java nije instalirana, potrebno ju je najprije instalirati. Taj dio neće biti objašnjen ovdje.

Jednom kada imamo instaliranu Javu, sve što treba napraviti je otići na službene stranice: `https://spark.apache.org/downloads.html`, odabrati najnoviju verziju (Za vrijeme pisanja ovog rada to je verzija *1.6.1 (Mar 09 2016*), izabrati odgovarajući paket te pokrenuti dohvaćanje odgovarajuće *.tgz* arhive. Najjednostavnije je odabrati neki *pre-built* paket, primjerice *Pre-built for Hadoop 2.6 and later* te će daljnji koraci instalacije biti napisani pod pretpostavkom da je korisnik dohvatio tu verziiju paketa. Ukoliko korisnik želi, moguće je instalirati i *Source code* varijantu paketa, ali taj postupak instalacije ovdje nije opisan.

Nakon što smo dohvatili odgovarajuću arhivu, potrebno ju je raspakirati.
Raspakiranje arhive moguće je napraviti preko naredbe:

```
$ tar -xvf spark-1.6.1-bin-hadoop2.6.tgz
```

Nakon toga, dobra je praksa premjestiti instalaciju u neki prikladniji direktorij. Tako nešto može se napraviti na sljedeći način:

```
$ mv Downloads/spark-1.6.1-bin-hadoop2.6 faks/spark/
```

Službeno, sada je instalacija gotova. Idemo u sljedećem odlomku malo detaljnije pogledati što smo to zapravo dobili instalacijom. Koje datoteke se sada nalaze na našem računalu, a nije ih bilo ranije. Također, biti će objašnjena i uloga nekih datoteka i direktorija.

1.2.2. Izvorni kodovi

Ako izlistamo što nam se trenutno nalazi u novonastalom direktoriju, dobiti ćemo sljedeći ispis:

```
mmatak@martins-beast:~/faks/spark$ ls -l
total 1408
drwxr-xr-x 2 mmatak mmatak    4096 Feb 27 06:02 bin
-rw-r--r-- 1 mmatak mmatak 1343562 Feb 27 06:02 CHANGES.txt
drwxr-xr-x 2 mmatak mmatak    4096 Feb 27 06:02 conf
drwxr-xr-x 3 mmatak mmatak    4096 Feb 27 06:02 data
drwxr-xr-x 3 mmatak mmatak    4096 Feb 27 06:02 ec2
drwxr-xr-x 3 mmatak mmatak    4096 Feb 27 06:02 examples
drwxr-xr-x 2 mmatak mmatak    4096 Feb 27 06:02 lib
-rw-r--r-- 1 mmatak mmatak   17352 Feb 27 06:02 LICENSE
drwxr-xr-x 2 mmatak mmatak    4096 Feb 27 06:02 licenses
-rw-r--r-- 1 mmatak mmatak   23529 Feb 27 06:02 NOTICE
drwxr-xr-x 6 mmatak mmatak    4096 Feb 27 06:02 python
drwxr-xr-x 3 mmatak mmatak    4096 Feb 27 06:02 R
-rw-r--r-- 1 mmatak mmatak    3359 Feb 27 06:02 README.md
-rw-r--r-- 1 mmatak mmatak     120 Feb 27 06:02 RELEASE
drwxr-xr-x 2 mmatak mmatak    4096 Feb 27 06:02 sbin
```

README.md Sadrži kratke instrukcije za upoznavanje sa Spark-om.

bin Sadrži izvršive datoteke koje se koriste za interaktivni rad sa Spark-om. Zanimljivo je spomenuti da postoji interaktivna ljuska *Spark shell* isključivo za programske jezike *Python* i *Scala*. Datoteke u ovom direktoriju služe upravo za to. Budući da je ovaj rad ograničen isključivo na programski jezik *Java*, a u ovom trenutku takva interaktivna ljuska još ne postoji, ovaj dio neće biti detaljnije obrađen.

core, streaming, python, ... Sadrži glavne komponente projekta *Apache Spark*

examples Sastoji se od nekoliko jednostavnih primjera koje pomažu korisniku da se uhoda i što bezbolnije nauči koristiti odgovarajući API.

Kako bi bili sigurni da je instalacija uspješna, potrebno je pozicionirati se u *bin* direktorij te u terminalu upisati `./spark-shell`. Ispis bi trebao biti sličan ovome:

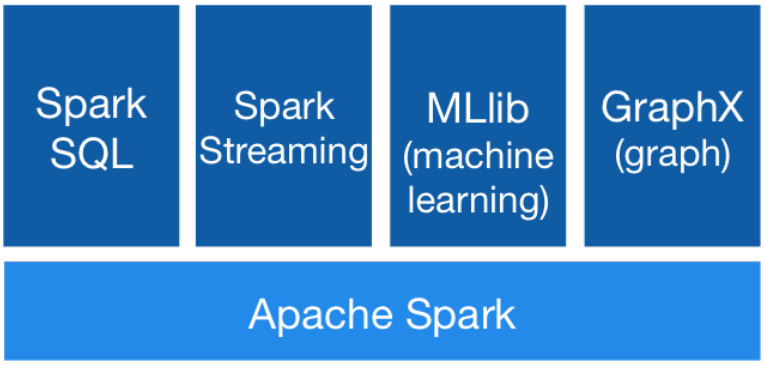
```
mmatak@martins-beast:~/faks/spark/bin$ ./spark-shell
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Scala version 2.10.5 (OpenJDK 64-Bit Server VM, Java 1.8.0_91)
Type in expressions to have them evaluated.
Type :help for more information.
scala>
```

Ukoliko se prikaže greška vezana uz *sqlContext*, to je OK. U ovom trenutku to nije važno. Važno je napomenuti da su u gornjem ispisu (radi ljepšeg formata ovog rada) izbrisana upozorenja (engl. *warnings*) koja su normalna i očekivana.

1.3. Osnovni gradivni elementi

Osnovna programska apstrakcija s kojom *Apache Spark* radi naziva se *resilient distributed dataset* (RDD). RDD-ovi predstavljaju kolekcije podataka koje se nalaze na računalima (jednom ili više njih) i moguće ih je paralelno obrađivati. *Apache Spark* nudi bogati API za rad s RDD-ovima.

Apache Spark se sastoji od nekoliko ključnih elemenata koji će ovdje biti samo nabrojani i opisani rečenicom ili dvije, a u kasnijim poglavljima će biti detaljnije objašnjeni. Slika koja predstavlja osnovne elemente je sljedeća:



Slika 1.1: Osnovni elementi tehnologije *Apache Spark*.

Na slici su prikazana četiri osnovna elementa koja čine osnovu tehnologije *Apache Spark*, a to su:

Spark SQL - omogućuje rad s bilo kakvim strukturiranim podatcima, primjerice JSON. Nudi mogućnost izvršavanja SQL-a nad RDD-ovima;

Spark Streaming - komponenta zadužena za rad s tokovima podataka;

MLib - koristi se za postupak strojnog učenja i

GraphX - biblioteka za obradu grafova (npr. graf prijatelja na društvenoj mreži).

1.4. Kratki pregled

U ovom poglavlju je bila ideja da čitatelj dobije motivaciju i želju za upoznavanjem s tehnologijom *Apache Spark*. Detaljno je opisan postupak instalacije koji bi čitatelju trebao biti sasvim dovoljan za samostalnu instalaciju. Također, dan je pregled nekih najosnovnijih datoteka i direktorija koje dolaze s instalacijom. Zainteresiranog čitatelja se ohrabruje da samostalno prouči kako se koristi *Spark shell*. Na kraju, nabrojani su

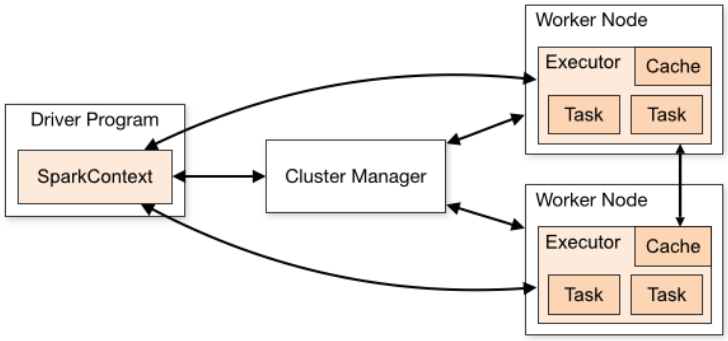
osnovni elementi koji čine jezgru tehnologije *Apache Spark* i uveden je pojam osnovne programske apstrakcije koja se koristi u ovoj tehnologiji, a to je *resilient distributed dataset* (RDD).

2. Prvi programi

2.1. Postavljanje temelja

2.1.1. Osnovni elementi aplikacije

Općenito govoreći, svaka Spark aplikacija sastoji se od nekoliko komponenata. Prva komponenta koju ćemo spomenuti je program koji se izvršava - onaj čija je `main` metoda pokrenuta, odnosno onaj koji pokreće obradu podataka. Taj program se naziva *driver*. On s podatcima priča kroz "tunel" između sebe i Sparka odnosno kroz *SparkContext*. Budući da je *Apache Spark* namijenjen za paralelnu obradu podataka, idemo se odmah upoznati i s još dvije komponente. Pojedino računalo u *grozdu* (engl. *cluster*) ćemo nazvati *radnim čvorom* (engl. *worker node*), a proces koji se izvršava na pojedinom računalu nazvati ćemo *radnik* (engl. *executor*). Dozvoljeno je da *radnici* međusobno komuniciraju. U cijeloj priči može, a i ne mora eksplicitno biti uključen *upravitelj grozdom* (engl. *cluster manager*).



Slika 2.1: Prikaz elemenata aplikacije.

2.1.2. Programiranje u Javi

Kako bi mogli pisati Spark programe u programskom jeziku *Java*, potrebno je povezati svoju aplikaciju s `spark-core` artifaktom s Mavena. Za postavljanje i instalaciju

Mavena konzultirati `https://maven.apache.org/install.html` i knjigu (Programiranje u Javi, Čupić). Za vrijeme pisanja ovog rada, najnovija verzija Sparka je 1.6.1, a odgovarajuće Maven koordinate su:

```
groupId = org.apache.spark
artifactId = spark-core_2.10
version = 1.6.1
```

Odgovarajuća `pom.xml` datoteka nalazi se u dodatku A. Jednom kada smo namjestili `pom.xml` datoteku i povezali se s `spark-core`, sve što trebamo još napraviti je inicijalizirati *SparkContext* i napisati prvu aplikaciju. U nastavku slijedi jednostavna aplikacija koja jedino što radi je broji koliko puta se pojedina riječ pojavljuje u tekstualnoj datoteci.

```
1  /**
2   *
3   */
4  package hr.fer.zemris;
5
6  import java.util.Arrays;
7
8  import org.apache.spark.SparkConf;
9  import org.apache.spark.api.java.JavaPairRDD;
10 import org.apache.spark.api.java.JavaRDD;
11 import org.apache.spark.api.java.JavaSparkContext;
12 import org.apache.spark.api.java.function.FlatMapFunction;
13 import org.apache.spark.api.java.function.Function2;
14 import org.apache.spark.api.java.function.PairFunction;
15
16 import scala.Tuple2;
17
18 /**
19  * Razred ima svrhu prikazati osnovnu funkcionalnost Spark tehnologije na
20  * primjeru prebrojavanja riječi u tekstualnoj datoteci. Kao rezultat program će
21  * zapisati u tekstualnu datoteku koja riječ se koliko puta ponavlja. Očekuje se
22  * dva argumenta kroz naredbeni redak, a to su putanja do tekstualne datoteke u
23  * kojoj treba izbrojati riječi i putanja do direktorija u koji će se zapisati
24  * rezultat.
25  *
26  * @author mmatak
27  *
28  */
29 public class BrojanjeRijeci {
30         /**
31          * Metoda koja se pokrene kada se pokrene program. Očekuje putanju do
32          * datoteke s riječima i putanju do direktorija gdje će zapisati rezultat
33          * izvođenja programa.
34          *
35          * @param args
36          *            Argumenti naredbenog retka.
37          */
38         @SuppressWarnings("serial")
39         public static void main(String[] args) {
40                 String ulaznaDatoteka = args[0];
41                 String izlazniDirektorij = args[1];
42
43                 // inicijalizacija SparkContext-a
44                 SparkConf conf = new SparkConf().setMaster("local")
45                 .setAppName("Brojanje rijeci");
46                 JavaSparkContext sc = new JavaSparkContext(conf);
47
48                 // učitavanje podataka
49                 JavaRDD<String> ulaz = sc.textFile(ulaznaDatoteka);
50
51                 // razmak se koristi da bi razdvojio dvije riječi
52                 JavaRDD<String> rijeci = ulaz.flatMap(new FlatMapFunction<String, String>() {
53                         public Iterable<String> call(String redak) {
```

```
54                              return Arrays.asList(redak.split(" "));
55                      }
56              });
57
58              // transformiraj u parove (rijec,1) i broji
59              JavaPairRDD<String, Integer> brojRijeci = rijeci
60              .mapToPair(new PairFunction<String, String, Integer>() {
61                      public Tuple2<String, Integer> call(String rijec) {
62                              return new Tuple2<String, Integer>(rijec, 1);
63                      }
64              }).reduceByKey(new Function2<Integer, Integer, Integer>() {
65                      public Integer call(Integer x, Integer y) {
66                              return x + y;
67                      }
68              });
69
70              // spremi rezultat u izlaznu datoteku
71              brojRijeci.saveAsTextFile(izlazniDirektorij);
72              // zatvori "tunel" odnosno SparkContext
73              sc.close();
74      }
75  }
```

Primjer 2.1: Program koji broji koliko se puta koja riječ pojavljuje u datoteci.

Analizirajmo što smo napravili. Inicijalizirali smo *SparkContext* tako što smo rekli da se odvija na lokalnom računalu i zadali smo ime aplikacije. Zatim smo kreirali RDD iz tekstualne datoteke koja je predana kao argument naredbenog retka. Taj RDD nam je poslužio za kreiranje novog RDD-a koji je nastao tako što smo svaki redak razdvojili po razmaku i kreirali uređeni par (*riječ*, 1). U tom uređenom paru nam *riječ* predstavlja ključ, a 1 predstavlja vrijednost. Odnosno, uređeni par je oblika (*ključ*, *vrijednost*). Zatim smo iz tako uređenih parova, one koji imaju jednaki ključ zbrojili po vrijednostima i u tom trenutku[1] nije više postojalo dva uređena para koji imaju jednake ključeve. Na kraju smo rezultat zapisali i eksplicitno zatvorili *SparkContext*. Kako je ovo bio početni primjer, *lambda* izrazi koji su uobičajeni za programski jezik *Java 8* nisu korišteni iz razloga da bi čitatelj mogao lakše shvatiti što se sve treba implementirati. Odgovarajući kod koristeći lambda izraze bi izgledao ovako:

```
// razmak se koristi da bi razdvojio dvije riječi
JavaRDD<String> rijeci = ulaz.flatMap(redak -> Arrays.asList(" "));

// transformiraj u parove (rijec,1) i broji
                JavaPairRDD<String, Integer> brojRijeci = rijeci
                                .mapToPair(rijec -> new Tuple2<String, Integer>(rijec, 1))
                                .reduceByKey((x, y) -> x + y);
```

Primjer 2.2: Brojanje riječi koristeći lambda izraze.

[1]Nije zapravo u tom trenutku se ništa dogodilo nego tek nakon poziva metode `saveAsTextFile()`, ali o *lijenoj evaluaciji* (engl. *lazy evaluation*) će biti govora tek kasnije.

2.2. Programiranje s RDD-ovima

Resilient distributed dataset (RDD) je osnovna podatkovna struktura Spark-a. To je *nepromjenjiva* (engl. *immutable*) kolekcija podataka. To znači da se iz jednog RDD-a može jedino napraviti drugi RDD, a ne može se promijeniti postojeći RDD. U prethodnom primjeru to vidimo pri pozivu metode `flatMap`. Ta metoda transformira postojeći RDD `ulaz` u novi RDD `rijeci`. Sličnu transformaciju radi i metoda `mapToPair`. Drugim riječima, *transformacija* (engl. *transformation*) je svaka metoda koja iz jednog RDD-a kreira drugi RDD. Uz transformacije, postoje i *akcije* (engl. *actions*). Akcije se razlikuju od transformacija po tome što ne vraćaju novi RDD nego vraćaju jedan ili više elemenata iz nekog RDD-a, ili kao u našem primjeru s brojanjem riječi, zapisuju RDD u obliku tekstualne datoteke - metoda `saveAsTextFile`.
U tablici 2.1 mogu se naći neke od mogućih transformacija i njihovi opisi, a u tablici 2.2 nalaze se akcije koje je moguće pozvati zajedno s odgovarajućim opisima.

Tablica 2.1: Neke od transformacija

Transformacija	Opis
`map(`*`func`*`)`	Vraća novi RDD nastao tako što je svaki element orginalnog RDD-a predan funckiji *func*.
`filter(`*`func`*`)`	Vraća novi RDD nastao tako što su iz orginalnog RDD-a preuzeti samo oni elementi za koje funkcija *func* vraća `true`.
`flatMap(`*`func`*`)`	Slično kao `map(`*`func`*`)`, ali jedan ulazni element kreira 0 ili više izlaznih elemenata.
`union(`*`drugiRDD`*`)`	Vraća uniju između RDD-a nad kojim je akcija pozana i RDD-a koji je predan kao argument.
`intersection(`*`drugiRDD`*`)`	Vraća presjek između RDD-a nad kojim je akcija pozana i RDD-a koji je predan kao argument.

Za opis ostalih transformacija, pogledati `http://spark.apache.org/docs/latest/programming-guide.html#transformations`.

Tablica 2.2: Akcije

Akcija	Opis
`reduce(`*`func`*`)`	Agregacija elemenata iz RDD-a tako što se nad dva elementa pozove funkcije *func*, a ta funkcija vrati jedan element. Funkcija *func* bi trebala biti komutativna i asocijativna kako bi se pravilno izvršavala u paralelnoj obradi podataka.
`collect()`	Vraća polje svih elemenata iz RDD-a direktno u *driver* program. Budući da je tih elemenata puno, u praksi se poziva nakon transformacije `filter()`.
`count()`	Vraća broj elemenata u RDD-u
`take(`*`n`*`)`	Vraća polje prvih *n* elemenata iz RDD-a.
`first()`	Vraća prvi element iz RDD-a. Može se ostvariti i pozivom metode `take(`*`1`*`)`

Za opis ostalih akcija, pogledati `http://spark.apache.org/docs/latest/programming-guide.html#actions`.

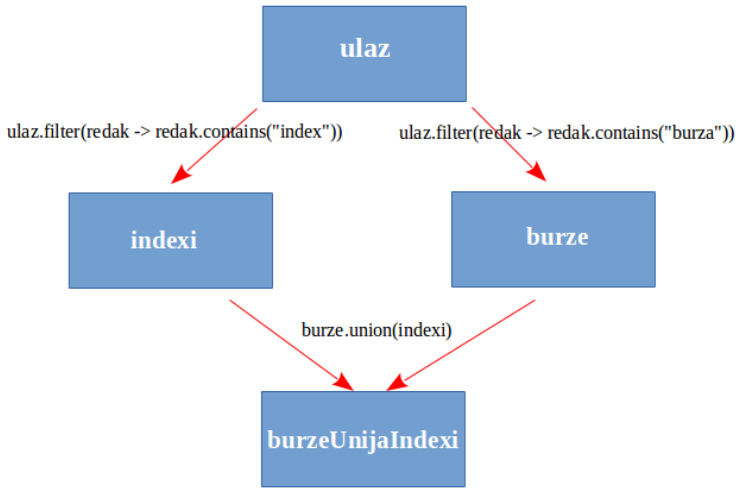
Budući da se radi o velikoj količini podataka, evaluacija transformacija je *lijena* (engl. *lazy evaluation*). To znači da Spark samo pamti koje sve transformacije treba napraviti nad RDD-ovima, ali ne i da ih odmah odradi. Sve potrebne transformacije budu napravljene tek pozivom prve akcije.
U nastavku slijedi primjer koji iz datoteke `logfile.txt` broji koliko puta je zahtjev bio na URL koji u sebi sadrži riječ "burza" i koliko je puta došao zahtjev na URL koji u sebi sadrži riječ "index". Njihova unija je također izračunata.

```
package hr.fer.zemris;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

public class Primjer3 {
        public static void main(String[] args) {
                // inicijalizacija SparkContext-a
                SparkConf conf = new SparkConf().setMaster("local")
                                .setAppName("Brojanje rijeci");
                JavaSparkContext sc = new JavaSparkContext(conf);

                // učitavanje podataka
                JavaRDD<String> ulaz = sc.textFile("logfile.txt");

                // transformacije
                JavaRDD<String> burze = ulaz.filter(redak -> redak.contains("burza"));
```

```
                JavaRDD<String> indexi = ulaz.filter(redak -> redak.contains("index"));
                JavaRDD<String> burzeUnijaIndexi = burze.union(indexi);

                // akcije
                long brojLinija = burzeUnijaIndexi.count();
                long ukupanBrojLinija = ulaz.count();
                System.out.println(
                                "Broj linija koje sadrže riječ 'burza' ili 'index' je: "
                                                + brojLinija + ", odnosno "
                                                + (double) 100 * brojLinija / ukupanBrojLinija + "%.");
                System.out.println(
                                "Prva linija koja sadrži riječ 'burza' ili 'index' je: \n"
                                                + burzeUnijaIndexi.first());
                sc.close();
        }
}
```

Primjer 2.3: Korištenje transformacija i akcija.

Ovdje imamo 4 RDD-a: `ulaz`, `burze`, `indexi` i `burzeUnijaIndexi`. RDD `burze` i RDD `indexi` su kreirani na temelju RDD-a `ulaz`. RDD `burzeUnijaIndexi` je kreiran na temelju RDD-a `burze` i na temelju RDD-a `indexi`. U nastavku je graf koji to opisuje.



Slika 2.2: Transformacije nad RDD-ovima.

Tek prilikom poziva akcije `burzaUnijaIndexi.count()` se zapravo kreira RDD `ulaz`, na temelju njega RDD `burza` i RDD `indexi` i onda tek na temelju njih se kreira RDD `burzaUnijaIndexi` te se tek onda računa ukupni broj linija u tom RDD-u. Nakon što se to izračuna, svi RDD-ovi nestaju iz memorije. Prilikom poziva akcije `burzeUnijaIndexi.first()` **ponovno** se kreiraju prethodno navedeni RDD-i i sve ide iz početka.

Na prvi pogled ovo izgleda kao loša implementacija, ali budući da se ovdje radi o velikoj količini podataka i ne možemo ih nikako sve pohraniti u memoriju, ovo je

zapravo logična implementacija. Ukoliko ne želimo svaki puta iz početka računati i kreirati RDD `burzaUnijaIndexi`, trebamo ga `perzistirati` pozivom metode `persist()` i predavanjem odgovarajućeg parametra (pogledati tablicu 2.3), odnosno spremiti ga u memoriju ili na disk. Ako ga imamo spremljenog, ne treba ga ponovno računati. Razine perzistencije dane su u tablici u 2.3.

Tablica 2.3: Razine perzistencije

Razina	Prostorno zauzeće	Procesorsko vrijeme	U memoriji	Na disku
`MEMORY_ONLY`	Visoko	Nisko	Da	Ne
`MEMORY_ONLY_SER`	Nisko	Visoko	Da	Ne
`MEMORY_AND_DISK`	Visoko	Srednje	Dio	Dio
`MEMORY_AND_DISK_SER`	Nisko	Visoko	Dio	Dio
`DISK_ONLY`	Nisko	Visoko	Ne	Da

2.3. Kratki pregled

U ovom poglavlju smo postavili temelje nad kojima ćemo kasnije graditi složenije aplikacije. Vrlo je važno da čitatelj jasno razumije što je to RDD jer bez toga nema smisla dalje pratiti ovaj rad. Dan je pregled osnovnih transformacija i akcija koje zapravo, uz RDD, čine jezgru tehnologije *Apache Spark*. Također je uveden i pojam perzistencije za efektivniji rad s RDD-ovima.

3. Napredno programiranje

3.1. Primjer: Analiza PageRank algoritma

3.1.1. Algoritam

Zanimljivo je pitanje kako je *Google* toliko dobar u rezultatima koje dobijemo na naš upit, točnije, kako ispravno sortira po relevantnosti stranice od bolje prema lošijoj? Odgovor na to daje *PageRank* algoritam koji je dobio ime po jednom od osnivača *Google*-a, Larry Pageu. Ovo potpoglavlje ćemo započeti kroz analizu *PageRank* algoritma. Ovdje će biti iznesena pojednostavljena verzija algoritma[1] koja je sasvim dovoljna za pobliže upoznavanje sa Sparkom.
Algoritam mjeri koliko je koja stranica važna po tome koliko drugih stranica upućuje na nju. Ideja je jednostavna: što više stranica ima poveznicu na neku stranicu `N`, to je stranica `N` važnija i time bolje rangirana. U obzir se uzima i koja stranica pokazuje na stranicu `N` (je li to stranica na koju sve ostale stranice pokazuju ili je to neka stranica na koju nitko ne pokazuje) kao i na koliko ostalih stranica ta stranica sadrži poveznice (nije isto ako je na cijeloj stranici samo jedna poveznica i ako na cijeloj stranici ima 1000 poveznica). Stranice na koje neka stranica `M` ima linkove nazivamo *susjedima*.
Pojednostavljena izvedba algoritma je sljedeća:

1. Inicijalizira se rang svake stranice na 1.0.
2. Svaka stranica *n* svim svojim susjedima šalje doprinos `rang(`*n*`)/brojSusjeda(`*n*`)`.
3. Postavi ukupni rang stranice prema formuli: $0.15 + 0.85 *$ *ukupan primljeni doprinos*.

Posljednja dva koraka se odrade nekoliko puta kako bi se dobio što precizniji rezultat, u praksi je dovoljno desetak puta.

[1] Zainteresiranog čitatelja se upućuje na `https://en.wikipedia.org/wiki/PageRank`.

Implementacija algoritma dana je u primjeru ispod.

```
1  package hr.fer.zemris.naprednoProgramiranje;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import org.apache.spark.SparkConf;
7  import org.apache.spark.api.java.JavaPairRDD;
8  import org.apache.spark.api.java.JavaRDD;
9  import org.apache.spark.api.java.JavaSparkContext;
10
11 import com.google.common.collect.Iterables;
12
13 import scala.Tuple2;
14
15 public class PageRankAlgoritam {
16         private static final String ULAZNA_DATOTEKA = "pageRankInput.txt";
17         private static final int BROJ_ITERACIJA = 10;
18
19         public static void main(String[] args) {
20
21                 // Inicijalizacija SparkContext-a.
22                 SparkConf conf = new SparkConf().setMaster("local")
23                                 .setAppName("Brojanje rijeci");
24                 JavaSparkContext sc = new JavaSparkContext(conf);
25
26                 // Učitavanje podataka.
27                 JavaRDD<String> ulaz = sc.textFile(ULAZNA_DATOTEKA);
28
29                 // Pročitaj sve ulazne URL-e i inicijaliziraj njihove susjede.
30                 JavaPairRDD<String, Iterable<String>> linkovi = ulaz
31                                 .mapToPair(redak -> {
32                                         String[] elementi = redak.split("\\s+");
33                                         return new Tuple2<String, String>(elementi[0], elementi[1]);
34                                 }).distinct().groupByKey().cache();
35                 // Svaku vrijednost u orginalnom RDDu zamijeni s 1.0 i vrati novi RDD.
36                 JavaPairRDD<String, Double> rangovi = linkovi.mapValues(value -> 1.0);
37
38                 // iteracija 2. i 3. koraka algoritma
39                 for (int i = 0; i < BROJ_ITERACIJA; i++) {
40                         // za svaki link izracunaj njegovu doprinos drugim linkovima
41                         JavaPairRDD<String, Double> doprinosi = linkovi.join(rangovi)
42                                         .values().flatMapToPair(linkoviRang -> {
43                                                 int brojSusjeda = Iterables.size(linkoviRang._1);
44                                                 List<Tuple2<String, Double>> rezultati = new ArrayList<Tuple2<
                                                         String, Double>>();
45                                                 for (String susjed : linkoviRang._1) {
46                                                         rezultati.add(new Tuple2<String, Double>(susjed,
47                                                                         linkoviRang._2 / brojSusjeda));
48                                                 }
49                                                 return rezultati;
50                                         });
51                         rangovi = doprinosi.reduceByKey((v1, v2) -> v1 + v2)
52                                         .mapValues(suma -> 0.15 + suma * 0.85);
53                 }
54                 // spremi u memoriju
55                 List<Tuple2<String, Double>> rangoviFinal = rangovi.collect();
56                 // ispis
57                 for (Tuple2<String, Double> entry : rangoviFinal) {
58                         System.out.println(
59                                         "URL " + entry._1 + "\tima vrijednost " + entry._2);
60                 }
61                 sc.close();
62         }
63 }
```

Primjer 3.1: PageRank algoritam

3.1.2. Analiza algoritma

3.2. RDD-i kao uređeni parovi

3.3. Čitanje i spremanje podataka

3.4. Globalne varijable

4. Zaključak

Zaključak.

LITERATURA

[1] Službena dokumentacija projekta *Apache Spark* `http://spark.apache.org/docs/latest/`

[2] Holden Karau, Andy Konwinski, Patrick Wendell i Matei Zaharia *Learning Spark, lighting-fast*

[3] Marko Čupić, *Programiranje u Javi*, verzija 0.3.26

[4] Ivan Krišto, Boran Car, Mateja Čuljak, Vedrana Janković, Hrvoje Bandov *Upute za korištenje LaTeXpredloška za Završni i Diplomski rad te seminar*, godina 2010.

Appendices

A. Postavljanje datoteke pom.xml

Datoteka koja je potrebna kako bi Maven ispravno dohvatio artifakt `spark-core` bi trebala izgledati slično kao što je dano u nastavku. Za ispravno funkcioniranje, trebala bi se zvati `pom.xml`.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>hr.fer.zemris</groupId>
  <artifactId>Prvi-programi</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Prvi programi</name>
  <description>Prvi programi u Spark-u</description>
  <dependencies>
        <dependency>
                <groupId>org.apache.spark</groupId>
                <artifactId>spark-core_2.10</artifactId>
                <version>1.6.1</version>
        </dependency>
  </dependencies>
</project>
```

Obrada podataka tehnologijom Apache Spark

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Title

Abstract

Abstract.

Keywords: Keywords.