# تمرین رایانش ابری

*درس مبانی رایانش توزیع شده*


**سید پارسا حسینی‌نژاد**

99/9/24

810196604

## سوال اول

**تعریف Apache Hadoop:** یک نرم افزار منبع باز برای محاسبات توزیع شده قابل اعتماد ، مقیاس پذیر و قابل توسعه است. کتابخانه Apache Hadoop چارچوبی است که امکان پردازش توزیع شده مجموعه داده های بزرگ را در cluster های مختلف با استفاده از مدل های ساده برنامه نویسی فراهم می کند. این کتابخانه طوری طراحی شده است تا از یک سرور تا هزاران ماشین قابلیت scale داشته باشد و در هر کدام محاسبات و ذخیره سازی محلی ارائه می شود. این کتابخانه به جای تکیه بر سخت افزار برای ارائه دسترسی بالا، برای شناسایی و کنترل خرابی های لایه ی application طراحی شده است، بنابراین یک سرویس کاملاً در دسترس را در بالای یک cluster از رایانه ارائه می دهد.

**تعریف Apache Spark:** یک سیستم پردازش منبع باز و توزیع شده است که برای big data استفاده می شود. این برنامه از حافظه پنهان در حافظه و اجرای بهینه سازی پرس و جو برای پرس و جوهای سریع در برابر داده های هر اندازه استفاده می کند. به زبان ساده ، Spark یک موتور سریع و عمومی برای پردازش داده های در مقیاس بزرگ است.

تفاوت اصلی بین Hadoop MapReduce و Spark در رویکرد پردازش نهفته است: Spark می تواند این کار را با استفاده از حافظه انجام دهد، در حالی که Hadoop MapReduce باید از روی دیسک بخواند و روی آن بنویسد. در نتیجه ، سرعت پردازش به طور قابل توجهی متفاوت است - Spark ممکن است تا 100 برابر سریعتر باشد. با این حال ، حجم داده های پردازش شده نیز متفاوت است: Hadoop MapReduce قادر است با مجموعه داده های بسیار بزرگتری نسبت به Spark کار کند.

## سوال دوم

سیستم پرونده توزیع شده ی هدوپ یا HDFS سیستم اصلی ذخیره اطلاعات است که توسط برنامه های Hadoop استفاده می شود. این سیستم یک معماری NameNode و DataNode را برای پیاده سازی یک سیستم فایل توزیع شده فراهم می کند که دسترسی با کارایی بالا به داده ها را در خوشه های Hadoop بسیار مقیاس پذیر فراهم می کند.

با استفاده از دستور hdfs dfs -ls /user/username/app1/subdir میتوان لیست فایل های موجود در هر مسیر از HDFS را گرفت.

## سوال سوم

ابتدا با استفاده از این دستور یک پوشه در HDFS می سازیم:

```
hdfs dfs -mkdir /myfolder
```

سپس با دستور استفاده از این دستور فایل ورودی را در hdfs وارد میکنیم:

```
hdfs dfs -put /path/to/file/mygraph.txt   /myfolder
```

## سوال چهارم

ابتدا مطابق شکل های زیر mapper و reducer را پیاده سازی می کنیم. متود mapper به ازای هر یال، شماره‌ی نود ورودی و خروجی را به همراه میزان تاثیر وزن یال بر آن را یادداشت میکند. مثلا این عبارت زیر به این معناست که از نود 100 یک یال با وزن 5 خارج شده است: 5 100

```
public void map(Object key, Text value, Context context
) throws IOException, InterruptedException {
    String inNodeNum, outNodeNum, edgeValue;
    IntWritable edgeValueWritable = new IntWritable( value: 0);
    String[] splited = value.toString().split( s: "\\s+");

    for (int i = 0; i < splited.length; i += 3) {
        outNodeNum = splited[i];
        inNodeNum = splited[i + 1];
        edgeValue = splited[i + 2];

        edgeValueWritable.set(Integer.parseInt(edgeValue));
        word.set(inNodeNum);
        context.write(word, edgeValueWritable);

        edgeValueWritable.set(-1 * Integer.parseInt(edgeValue));
        word.set(outNodeNum);
        context.write(word, edgeValueWritable);
    }
}
```

حال reducer اعداد مربوط به یک نود را جمع می زند و آنهایی که مجموع فرد دارند را گزارش میکند.

```
public void reduce(Text key, Iterable<IntWritable> values,
                   Context context
) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
        sum += val.get();
    }
    sum = Math.abs(sum);
    if (sum % 2 == 1) {
        result.set(sum);
        context.write(key, result);
    }
}
```

در آخر نیز بدنه ی تابع main را پیاده سازی می کنیم و توابعی که نوشتیم را به عنوان conf ست میکنیم. دقت شود تابع main دو ورودی دارد که یکی آدرس فایل ورودی و بعدی آدرسی است که خروجی باید در آن نوشته شود.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, jobName: "is in out diff odd");
    job.setJarByClass(GraphHadoop.class);
    job.setMapperClass(GraphInOutMapper.class);
    job.setCombinerClass(IsDiffOddReducer.class);
    job.setReducerClass(IsDiffOddReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
}
```

حال jar مربوط به برنامه ی نوشته شده را ساخته و در آخر نیز با استفاده از دستور زیر برنامه را اجرا میکنیم.

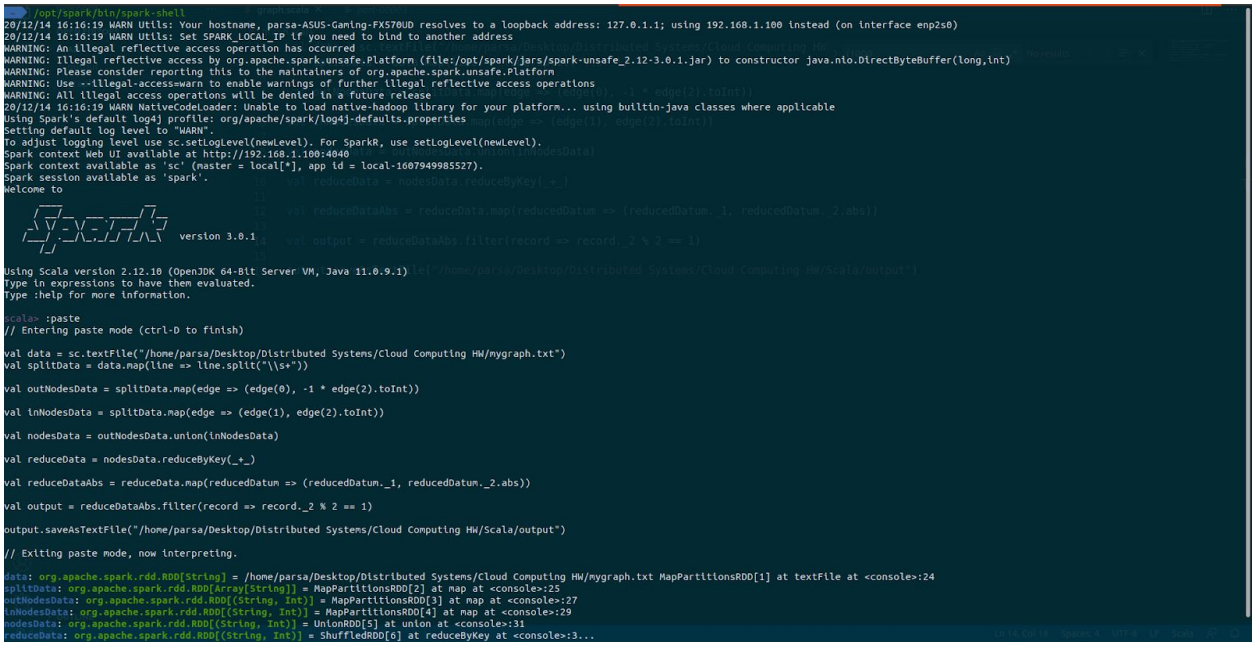
```
/usr/local/hadoop/bin/hadoop jar /home/parsa/Desktop/Distributed\ Systems/Cloud\ Computing\ HW/graphHadoop/out/artifacts/graphHadoop_jar/graphHadoop.jar  ~/input ~/graph_example
```

## سوال پنجم

برای اینکار، ابتدا فایل مربوطه را خوانده و آن را بر اساس whitespace جدا میکنیم. سپس به ازای هر یال، میزان اثر وزن آن را یک بار برای نود ورودی و یک بار برای نود خروجی حساب کرده و بعد آنها را در یک RDD میریزیم. سپس بر اساس شماره ی نود این مقادیر را جمع میزنیم تا عمل reduce انجام پذیرد. حال برای هر خانه ی این RDD جدید قدر مطلق را حساب کرده و سپس اگر این عدد فرد بود در یک متغیر دیگر میریزیم. در انتها نیز این متغیر را در فایل مربوطه چاپ میکنیم.

```
val data = sc.textFile("/home/parsa/Desktop/Distributed Systems/Cloud Computing HW/mygraph.txt")
val splitData = data.map(line => line.split("\\s+"))

val outNodesData = splitData.map(edge => (edge(0), -1 * edge(2).toInt))

val inNodesData = splitData.map(edge => (edge(1), edge(2).toInt))

val nodesData = outNodesData.union(inNodesData)

val reduceData = nodesData.reduceByKey(_+_)

val reduceDataAbs = reduceData.map(reducedDatum => (reducedDatum._1, reducedDatum._2.abs))

val output = reduceDataAbs.filter(record => record._2 % 2 == 1)

output.saveAsTextFile("/home/parsa/Desktop/Distributed Systems/Cloud Computing HW/Scala/output")
```

این کد در spark shell اجرا شده است که در عکس زیر قابل مشاهده است.



## سوال ششم

همانطور که قابل مشاهده است، خروجی هر دو برنامه دقیقا یکی هستند. در پروژه‌ی java یک فایل خروجی به همراه 4988 داده تولید شد. در پروژه ی scala چهار فایل خروجی تولید شد که جمع تعداد داده‌های آن در زیر آمده است.

1218 + 1281 + 1229 + 1260 = 4988

همانطور که مشاهده می شود تعداد این دو یکی هستند و در واقع جواب های هر دو نیز دقیقا یکسان میباشند. اما Hadoop تمام خروجی ها را در یک فایل ریخته در حالی که Spark در چهار فایل مجزا ریخته است. از لحاظ performance هم تغییر آن چنانی احساس نشد (با توجه به این که یک میلیون داده داریم که میزان خیلی زیادی نیست که تغییر آنچنانی به وجود آورد) اما حس کردم در Spark تولید خروجی اندکی سریع تر است.

علاوه بر آن، نوشتن این برنامه در Spark و Scala با توجه به این که API های بهتری در دست میگذارند، راحت تر و بهتر بود.

پی نوشت: تمامی خروجی ها به همراه کد دو برنامه ضمیمه شده است.