Bach Khoa University Of Technology
Faculty of Computer Science & Engineering

Course: Parallel Processing
Lab #7 - Spark & Data Analysis

Minh Thanh Chung

November 13, 2018

Goal: This lab helps student to get familiar with Spark on the server - SuperNode-XP. Then, student can learn basic skills with Data Analysis based on Spark (referencing http://xd-deng.com/).

Content: The lab introduces how to use Spark for re-processing data as well as analyze data with basic commands. In detail, the lab shows steps from installation, configuration and operations with Spark.

Result: After this Lab, student can understand and write a program with Spark to re-process or analyze data.

Contents

1 Introduction

1.1 Contents and Grading

1.2 Spark Tutorial: Why Spark when Hadoop is already there?

To answer this, we have to look at the concept of batch and real-time processing. Hadoop is based on the concept of batch processing where the processing happens of blocks of data that have already been stored over a period of time. At the time, Hadoop broke all the expectations with the revolutionary MapReduce framework in 2005. Hadoop MapReduce is the best framework for processing data in batches. This went on until 2014, till Spark overtook Hadoop. The USP for Spark was that it could process data in real time and was about 100 times faster than Hadoop MapReduce in batch processing large data sets.

The following figure gives a detailed explanation of the differences between processing in Spark and Hadoop.

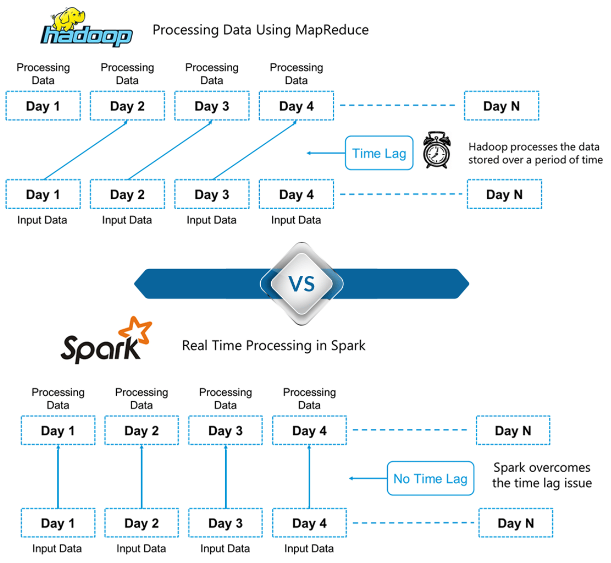

hadoop
Processing Data Using MapReduce
Processing Data
Day 1
Day 2
Day 3
Day 4
Day N
Time Lag
Hadoop processes the data stored over a period of time
Input Data
VS
Spark
Real Time Processing in Spark
No Time Lag
Spark overcomes the time lag issue

Figure 1: Differences between Hadoop and Spark

1.3 Download, Install, and Run PySpark

1. Make Sure Java is Installed Properly

```
$ java -version
    java version "1.8.0_72"
    Java(TM) SE Runtime Environment (build 1.8.0_72-b15)
    Java HotSpot(TM) 64-Bit Server VM (build 25.72-b15, mixed mode)
```

2. Download the latest binary Spark from the following URL

```
https://spark.apache.org/downloads.html
```

3. Open the downloaded file and untar

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

4. Start the Spark server

```
# cd to the path containing Spark source code
# (e.g., spark-1.6.1-bin-hadoop2.6/)
$ ./sbin/start-all.sh
```

5. Make sure that Master and Worker processes are running

```
$ jps
    1347 Master
    1390 Worker
```

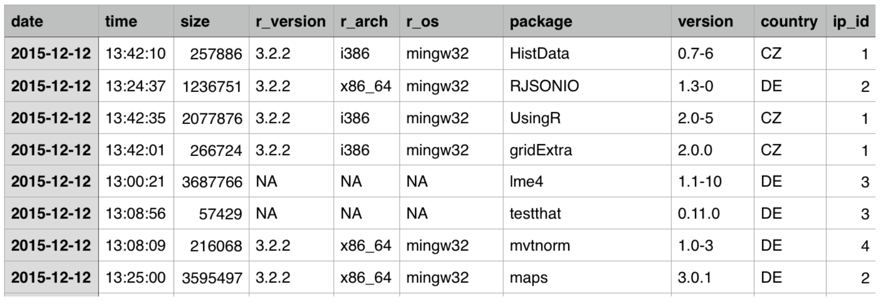
6. Start and run PySpark

```
$ ./bin/pyspark
    Python 2.7.10 (default, Oct 23 2015, 19:19:21)
    [GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on
        darwin
    ...
```

2 Practice with basic commands and functions in Spark

2.1 Sample data

The sample data we use is from http://cran-logs.rstudio.com/. It is the full downloads log of R packages from Rstudio's CRAN mirror on December 12 2015.



date	time	size	r_version	r_arch	r_os	package	version	country	ip_id
2015-12-12	13:42:10	257886	3.2.2	i386	mingw32	HistData	0.7-6	CZ	1
2015-12-12	13:24:37	1236751	3.2.2	x86_64	mingw32	RJSONIO	1.3-0	DE	2
2015-12-12	13:42:35	2077876	3.2.2	i386	mingw32	UsingR	2.0-5	CZ	1
2015-12-12	13:42:01	266724	3.2.2	i386	mingw32	gridExtra	2.0.0	CZ	1
2015-12-12	13:00:21	3687766	NA	NA	NA	lme4	1.1-10	DE	3
2015-12-12	13:08:56	57429	NA	NA	NA	testthat	0.11.0	DE	3
2015-12-12	13:08:09	216068	3.2.2	x86_64	mingw32	mvtnorm	1.0-3	DE	4
2015-12-12	13:25:00	3595497	3.2.2	x86_64	mingw32	maps	3.0.1	DE	2

Figure 2: Sample Data from Rstudio's CRAN

2.2 How we use Spark (PySpark) interactively

After Spark is started, a default SparkContext will be created (usually named as "sc").

1. Load data
   The most common method used to load data is sc.textFile. This method takes an URI for the file (local file or other URI like hdfs://), and will read the data as a collections of lines.

```
# Load the data
>>> raw_content = sc.textFile("/path/to/file/2015-12-12.csv")

# Print the type of the object
>>> type(raw_content)
```

```
          <class 'pyspark.rdd.RDD'>

      # Print the number of lines
      >>> raw_content.count()
          421970
```

2. Show the Head (first n rows)
   We can use take method to return first n rows.

```
>>> raw_content.take(5)
[u'"date","time","size","r_version","r_arch","r_os","package","version","
    country","ip_id"',
u'"2015-12-12","13:42:10",257886,"3.2.2","i386","mingw32","HistData","0.7-6",
    "CZ",1',
...
```

3. Transformation (map & flatMap)
   We may need to note that each row of the data is a character string, and it would be more convenient to have an array in some scenarios. We can use map to transform them and use take method to get the first a few rows to check how the results look like.

```
>>> content = raw_content.map(lambda x: x.split(','))
>>> content.take(3)
[
[u'"date"', u'"time"', u'"size"', u'"r_version"', u'"r_arch"', u'"r_os"', u'"
    package"', u'"version"', u'"country"', u'"ip_id"'],
[u'"2015-12-12"', u'"13:42:10"', u'257886', u'"3.2.2"', u'"i386"', u'"mingw32
    "', u'"HistData"', u'"0.7-6"', u'"CZ"', u'1'],
...
]
```

4. Reduce and counting
   In the case, we want to know how many downloading records each package has.

```
# Note here x[6] is just the 7th element of each row, that is the package
    name.
>>> package_count = content.map(lambda x: (x[6], 1)).reduceByKey(lambda a,b:
    a+b)
>>> type(package_count)
     <class 'pyspark.rdd.PipelinedRDD'>
>>> package_count.count()
     8660
>>> package_count.take(3)
     [(u'SIS', 24),
     (u'StatMethRank', 15),
     (u'dbmss', 54)]
```

   To achive the same purpose, we can also use countByKey method. The result returned by it is in hashmap (like dictionary) structure.

5. Sorting
   After counting using reduce method, we can know the rankings of these packages based on how many downloads they have. Then we need to use **sortByKey** method. Please note:

- The ’Key’ here refers to the first element of each tuple.
- The argument of sortByKey (0 or 1) will determine if we’re sorting decreasingly (’0’, or False) or increasingly (’1’, or True).

```
    # Sort decreasingly and get the first 10
    >>> package_count.map(lambda x: (x[1], x[0])).sortByKey(0).take(10)
    [(4783, u’Rcpp’),
    (3913, u’ggplot2’),
    (3748, u’stringi’),
    ...]
```

6. Filtering
   We can consider filter as the *SELECT * from TABLE WHERE ???* statement in SQL. It can help return a new dataset formed by selecting those elements on which the function specified by user returns **True**.

   For example, we would like to obtain these downloading records of R package ”Rtts” from China (CN), then the condition is *package == ’Rtts’ AND country = ’CN’*.

```
    >>> content.filter(lambda x: x[6] == ’"Rtts"’ and x[8] == ’"CN"’).count()
         1
    >>> content.filter(lambda x: x[6] == ’Rtts’ and x[8] == ’CN’).take(1)
         [[u’2015-12-12’, u’20:15:24’, u’23820’, u’3.2.2’, u’x86_64’, u’mingw32’,
              u’Rtts’, u’0.3.3’, u’CN’, u’41’]]
```

7. Set operation
   Like the set operators in vanilla SQL, we can do set operations in Spark. Here we would introduce *union*, *intersection*, and *distinct*. We can make intuitive interpretations as below.

   - union of A and B: return elements of A AND elements of B.
   - intersection of A and B: return these elements existing in both A and B.
   - distinct of A: return the distinct values in A. That is, if element a appears more than once, it will only appear once in the result returned.

```
    >>> raw_content.count()
         421970

    # one set’s union with itself equals to its "double"
    >>> raw_content.union(raw_content).count()
         843940

    # one set’s intersection with itself equals to its disctinct value set
    >>> raw_content.intersection(raw_content).count()
         421553

    >>> raw_content.distinct().count()
         421553
```

8. Submitting application
   All examples showed above were implemented interactively. To automate things, we may need to

prepare scripts (applications) in advance and call them, instead of entering line by line. The *spark-submit* script in Sparks *bin* directory is just used to figure out this problem, i.e. launch applications. It can use all of Sparks supported cluster managers (Scala, Java, Python, and R) through a uniform interface so you dont have to configure your application specially for each one. This means that we only need to prepare and call the scripts while we don't need to tell Spark which driver we're using.

```
# submit application written with Python
$ ./bin/spark-submit  examples/src/main/python/pi.py
```

3 Exercises

4 Submission

4.1 Source code