This is a summary of Project3 assignment.

Detailed steps can be found here:

<https://github.com/hansonzhao007/dis_sys/blob/project3/Hadoop_Installation.md>

<https://github.com/hansonzhao007/dis_sys/blob/project3/Hibench_Installation.md>

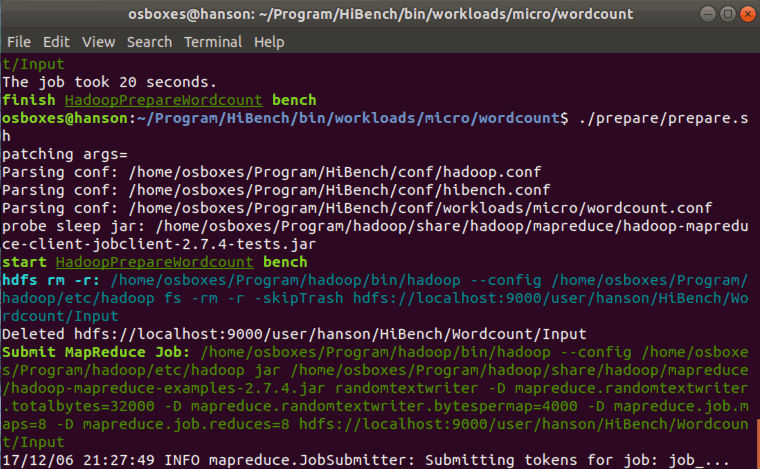
<https://github.com/hansonzhao007/dis_sys/blob/project3/MapReducer-in-python.md>

code can be found [here](https://github.com/hansonzhao007/dis_sys/tree/project3/example_code).

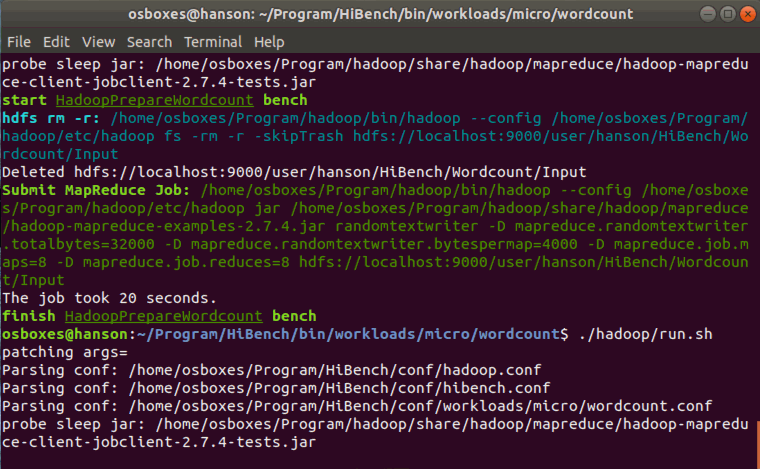
Assignment 1：

1. HiBench running results:

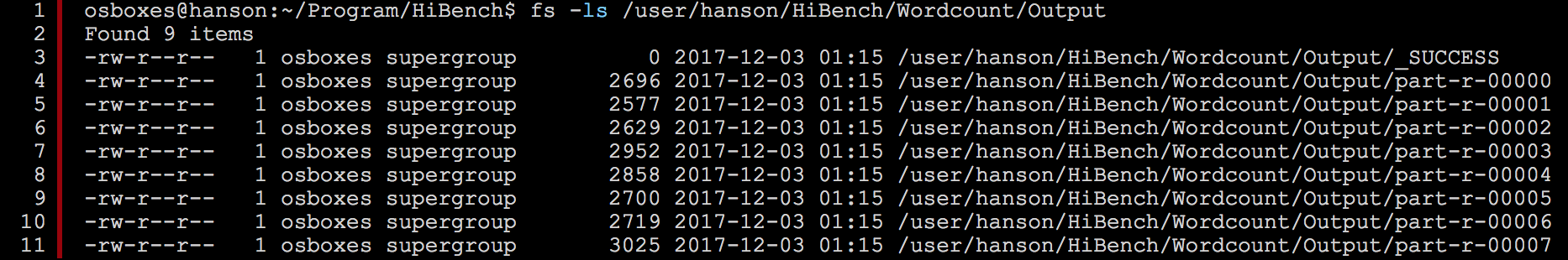
* Prepare running environments:



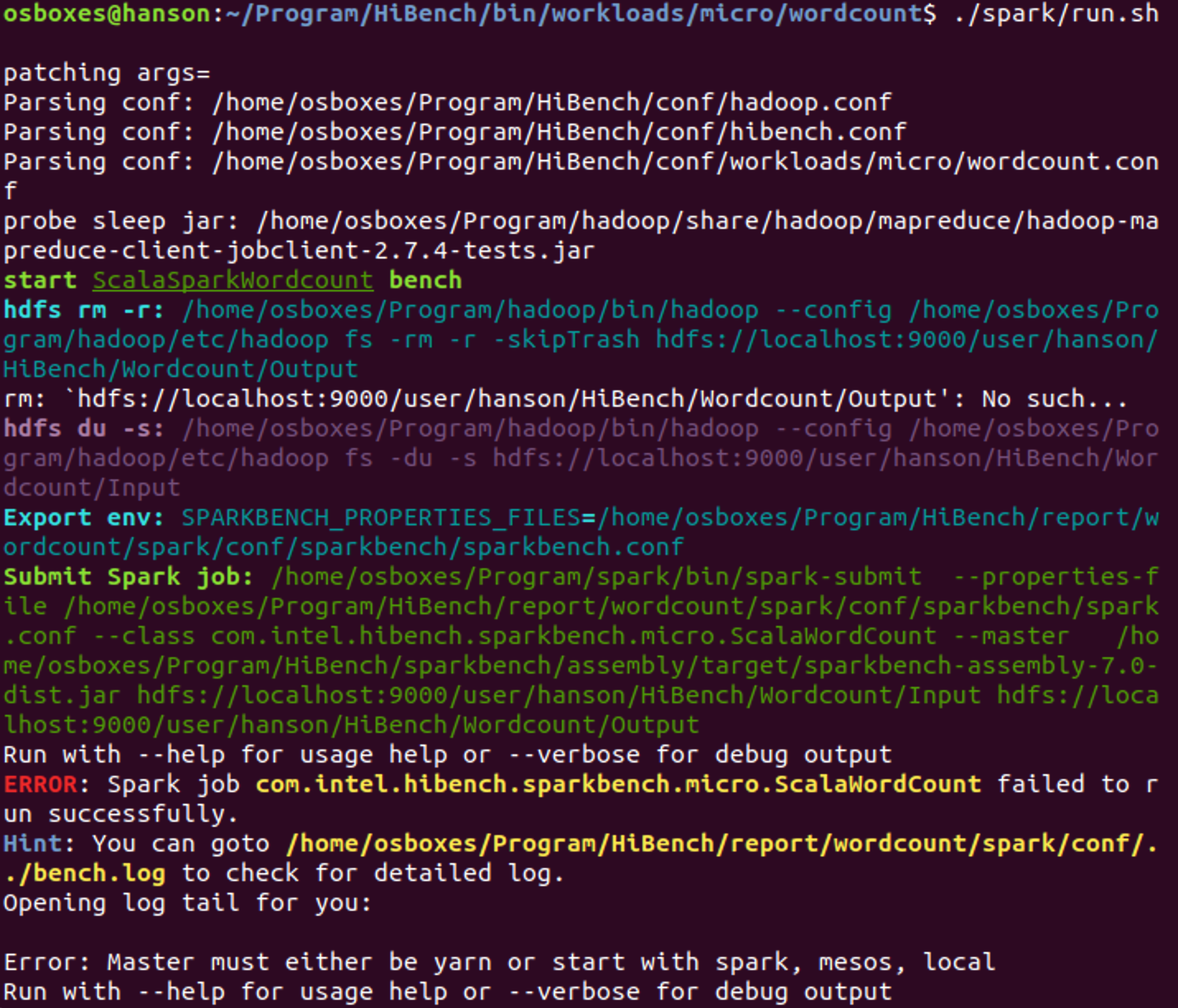
* Successfully running MapReduce benchmark:



This is the output results of HiBench Wordcount example in DFS:



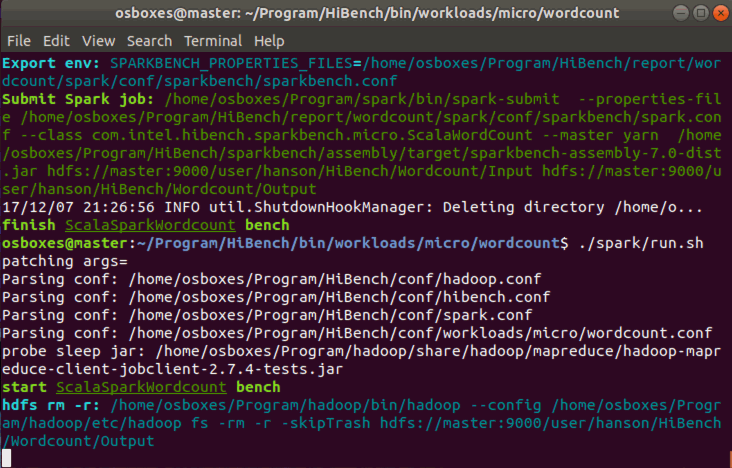
* Running spark benchmark



it reports an error: **Master must either be yarn or start with spark, mesos, local**

This is because default test command miss a configuration after `--master`

Then I add some configuration in `spark.conf` and re-run the spark Hibench.



      Running time comparison

| ID | Name | Application Type | StartTime | FinishTime | FinalStatus |
| --- | --- | --- | --- | --- | --- |
| application..0022 | ScalaWordCount | SPARK | Thu Dec 7 21:54:21 | Thu Dec 7 21:54:35 | SUCCEEDED |
| application..0021 | word count | MAPREDUCE | Thu Dec 7 21:53:22 | Thu Dec 7 21:53:48 | SUCCEEDED |

From the results above, Hadoop MapReduce used **26 seconds** to finish the Wordcount job, spark used **14 seconds** to finish the job. Spark is faster than Hadoop. And all of these jobs are running in the default configuration in `hibench.conf` file. (8 Mappers and 8 Reducers for Hadoop, 8 Partitions and 8 Shuffle Partitions in spark)

This is the execution time line of Hadoop and spark running HiBench Wordcount example. Number represent how many Mapper and Reducer each run has.

This is the execution time line of Hadoop and spark running HiBench Wordcount example. Here reducers number is fixed to 4.

Assignment 2: MapReduce Version

Here I use python to implement Intsum program.

**Mapper:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #!/usr/bin/python    import sys  num\_cnts = dict()  # input comes from STDIN (standard input)  for line in sys.stdin:      #for each document create dictionary of numbers      line = line.strip()      nums = line.split()      for num in nums:          if num not in num\_cnts.keys(): num\_cnts[num] = 1          else: num\_cnts[num] += 1    # emit key-value pairs only for distinct numbers per document  for w in num\_cnts.keys():      print '%s\t%s' % (w,num\_cnts[w]) |

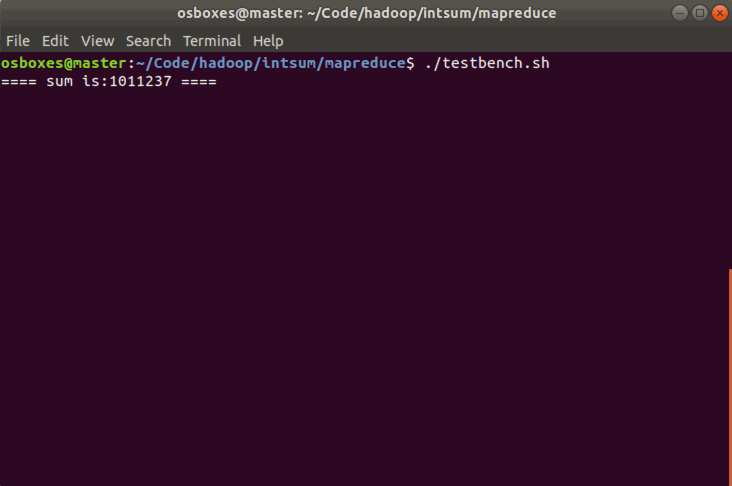
**Reducer:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | #!/usr/bin/python    from operator import itemgetter  import sys    current\_number = None  current\_count = 0  num = None  tsum = 0  # input comes from STDIN  for line in sys.stdin:      # remove leading and trailing whitespace      line = line.strip()        # parse the input we got from mapper.py      num, count = line.split('\t', 1)        # convert count (currently a string) to int      try:          count = int(count)      except ValueError:          # count was not a number, so silently          # ignore/discard this line          continue        tsum = tsum + int(num) \* count    print '==== sum is:%s ====' %(tsum) |

**testbench:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | cat intsum\*.txt | python mapper.py | python reducer.py  hadoop fs -rm -R /user/output  hadoop \  jar /home/osboxes/Program/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.4.jar \  -mapper mapper.py \  -reducer reducer.py \  -input /user/data/intsum\*.txt \  -output /user/output \  -file mapper.py \  -file reducer.py    hadoop fs -cat /user/output/\* |

**Running log:** You can see the output '==== sum is:1011237 ====' in **the beginning** and **the end** after the testbench completion.

****

Assignment 3: Spark Version

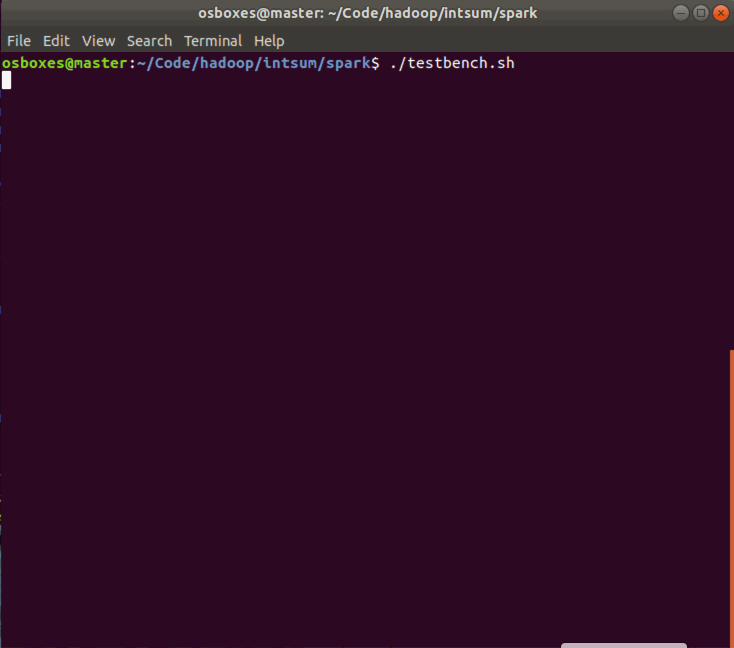
**Code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | from \_\_future\_\_ import print\_function  import sys  reload(sys)  sys.setdefaultencoding('utf-8')  from operator import add    from pyspark.sql import SparkSession    if \_\_name\_\_ == "\_\_main\_\_":      if len(sys.argv) != 2:          print("Usage: intsum <file>", file=sys.stderr)      exit(-1)        spark = SparkSession\              .builder\              .appName("PythonIntSum")\              .getOrCreate()        lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])      counts = lines.flatMap(lambda x: x.split(' ')) \              .map(lambda x: (x, 1)) \              .reduceByKey(add)      output = counts.collect()      int\_sum = 0      for (num, count) in output:          int\_sum = int\_sum + int(num) \* int(count)          print("%s: %i" % (num, count)) # every number's count      print("==== sum is: %d =====" % int\_sum) # final result      spark.stop() |

**Testbench:**

|  |  |
| --- | --- |
| 1 | spark-submit --master yarn intsum.py /user/data/intsum\*.txt |

**Running log:** You can see output "==== sum is: 1011237 =====" in the end after testbench completion.

****

Performance comparison:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | User | Name | Application Type | Queue | StartTime | FinishTime |
| [application\_1512872313403\_0034](http://master:8088/cluster/app/application_1512872313403_0034) | osboxes | streamjob5178667293119064775.jar | MAPREDUCE | default | Sat Dec 9 22:02:24 -0600 2017 | Sat Dec 9 22:02:38 -0600 2017 |
| [application\_1512872313403\_0033](http://master:8088/cluster/app/application_1512872313403_0033) | osboxes | streamjob2652047070281976099.jar | MAPREDUCE | default | Sat Dec 9 22:00:36 -0600 2017 | Sat Dec 9 22:00:51 -0600 2017 |
| [application\_1512872313403\_0032](http://master:8088/cluster/app/application_1512872313403_0032) | osboxes | PythonIntSum | SPARK | default | Sat Dec 9 21:59:44 -0600 2017 | Sat Dec 9 21:59:58 -0600 2017 |
| [application\_1512872313403\_0031](http://master:8088/cluster/app/application_1512872313403_0031) | osboxes | PythonIntSum | SPARK | default | Sat Dec 9 21:58:28 -0600 2017 | Sat Dec 9 21:58:42 -0600 2017 |

Both program’s inputs consist of 2 files, each has 10,000 number ranging from 1 to 100.

From the table above we see little difference considering running time, spark on yarn is 1 or 2 seconds faster than hadoop.

Assignment 4:

1. What are the key differences between Hadoop and Spark, and their respective advantages?

|  |  |
| --- | --- |
| Hadoop | Spark |
| Fast | 100X faster than MapReduce |
| Batch processing | Real-time processing |
| Store data on disk | Store data in memory |
| Written in Java | Written in Scala |

2. Discuss how to recover a failed task in Hadoop and Spark, respectively.

**Hadoop:** Master will ping every worker periodically to check if any of the workers fail. If true, the worker will be marked as failed and all the task completed by this failed worker will be reset back to their initial idle state, and will be scheduled to other workers. If master fails, MapReduce computation will abort.

**Spark:** Spark operates on data in fault-tolerant file systems like HDFS or S3. So all the RDDs generated from fault tolerant data is fault tolerant. But this does not set true for streaming/live data (data over the network). If the driver node fails, all the data that was received and replicated in memory will be lost. This will affect the result of the state of transformation. To avoid the loss of data, Spark 1.2 introduced write ahead logs, which save received data to fault-tolerant storage. All the data received is written to write ahead logs before it can be processed to Spark Streaming.