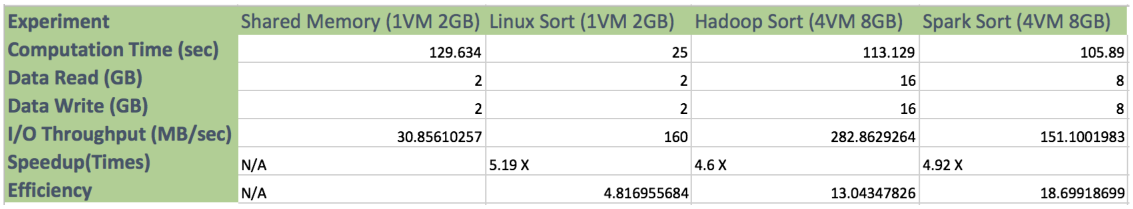
Solution to Homework 2b(CS 553)

Saptarshi Chatterjee
CWID: A20413922

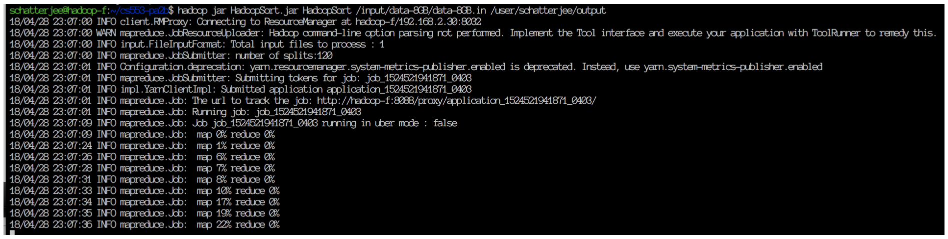
April 28, 2018

1 Table 1: Performance evaluation of sort (weak scaling-small dataset)



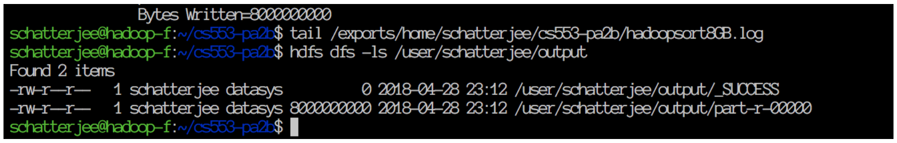
Experiment	Shared Memory (1VM 2GB)	Linux Sort (1VM 2GB)	Hadoop Sort (4VM 8GB)	Spark Sort (4VM 8GB)
Computation Time (sec)	129.634	25	113.129	105.89
Data Read (GB)	2	2	16	8
Data Write (GB)	2	2	16	8
I/O Throughput (MB/sec)	30.85610257	160	282.8629264	151.1001983
Speedup(Times)	N/A	5.19 X	4.6 X	4.92 X
Efficiency	N/A	4.816955684	13.04347826	18.69918699

Running Hadoop on Cluster



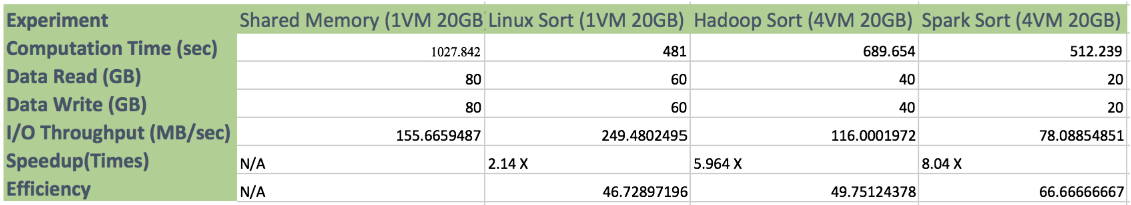
```
schatterjee@hadoop-f:~/cs553-pa2b$ hadoop jar HadoopSort.jar HadoopSort /input/data-8GB/data-8GB.in /user/schatterjee/output
18/04/28 23:07:00 INFO client.RMProxy: Connecting to ResourceManager at hadoop-f/192.168.2.30:8032
18/04/28 23:07:00 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
18/04/28 23:07:00 INFO input.FileInputFormat: Total input files to process : 1
18/04/28 23:07:00 INFO mapreduce.JobSubmitter: number of splits:120
18/04/28 23:07:01 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
18/04/28 23:07:01 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1524521941871_0403
18/04/28 23:07:01 INFO impl.YarnClientImpl: Submitted application application_1524521941871_0403
18/04/28 23:07:01 INFO mapreduce.Job: The url to track the job: http://hadoop-f:8088/proxy/application_1524521941871_0403/
18/04/28 23:07:01 INFO mapreduce.Job: Running job: job_1524521941871_0403
18/04/28 23:07:09 INFO mapreduce.Job: Job job_1524521941871_0403 running in uber mode : false
18/04/28 23:07:09 INFO mapreduce.Job:  map 0% reduce 0%
18/04/28 23:07:24 INFO mapreduce.Job:  map 1% reduce 0%
18/04/28 23:07:26 INFO mapreduce.Job:  map 6% reduce 0%
18/04/28 23:07:28 INFO mapreduce.Job:  map 7% reduce 0%
18/04/28 23:07:31 INFO mapreduce.Job:  map 8% reduce 0%
18/04/28 23:07:33 INFO mapreduce.Job:  map 10% reduce 0%
18/04/28 23:07:34 INFO mapreduce.Job:  map 17% reduce 0%
18/04/28 23:07:35 INFO mapreduce.Job:  map 19% reduce 0%
18/04/28 23:07:36 INFO mapreduce.Job:  map 22% reduce 0%
```

Success Output



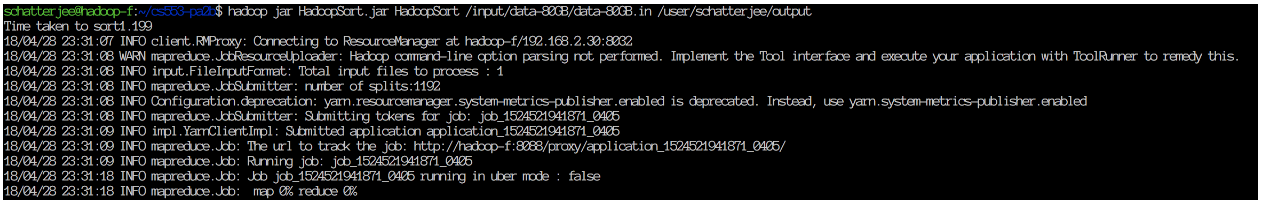
```
                Bytes Written=8000000000
schatterjee@hadoop-f:~/cs553-pa2b$ tail /exports/home/schatterjee/cs553-pa2b/hadoopsort8GB.log
schatterjee@hadoop-f:~/cs553-pa2b$ hdfs dfs -ls /user/schatterjee/output
Found 2 items
-rw-r--r--   1 schatterjee datasys          0 2018-04-28 23:12 /user/schatterjee/output/_SUCCESS
-rw-r--r--   1 schatterjee datasys 8000000000 2018-04-28 23:12 /user/schatterjee/output/part-r-00000
schatterjee@hadoop-f:~/cs553-pa2b$
```

2 Table 2: Performance evaluation of sort (strong scaling-large dataset)



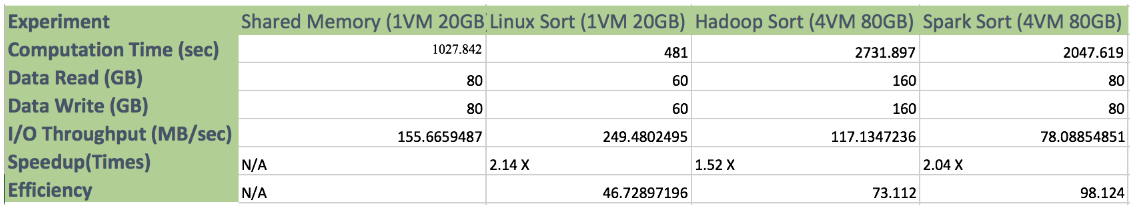
Experiment	Shared Memory (1VM 20GB	Linux Sort (1VM 20GB)	Hadoop Sort (4VM 20GB)	Spark Sort (4VM 20GB)
Computation Time (sec)	1027.842	481	689.654	512.239
Data Read (GB)	80	60	40	20
Data Write (GB)	80	60	40	20
I/O Throughput (MB/sec)	155.6659487	249.4802495	116.0001972	78.08854851
Speedup(Times)	N/A	2.14 X	5.964 X	8.04 X
Efficiency	N/A	46.72897196	49.75124378	66.66666667

Running with 80GB dataSet



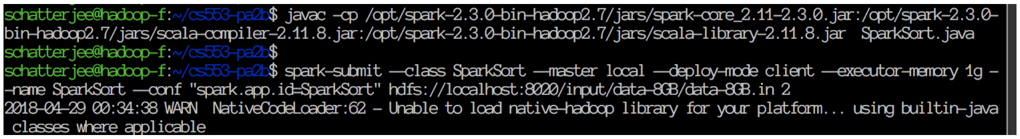
```
schatterjee@hadoop-f:~/cs553-pa2b$ hadoop jar HadoopSort.jar HadoopSort /input/data-80GB/data-80GB.in /user/schatterjee/output
Time taken to sort1.199
18/04/28 23:31:07 INFO client.RMProxy: Connecting to ResourceManager at hadoop-f/192.168.2.30:8032
18/04/28 23:31:08 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
18/04/28 23:31:08 INFO input.FileInputFormat: Total input files to process : 1
18/04/28 23:31:08 INFO mapreduce.JobSubmitter: number of splits:1192
18/04/28 23:31:08 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
18/04/28 23:31:08 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1524521941871_0405
18/04/28 23:31:09 INFO impl.YarnClientImpl: Submitted application application_1524521941871_0405
18/04/28 23:31:09 INFO mapreduce.Job: The url to track the job: http://hadoop-f:8088/proxy/application_1524521941871_0405/
18/04/28 23:31:09 INFO mapreduce.Job: Running job: job_1524521941871_0405
18/04/28 23:31:18 INFO mapreduce.Job: Job job_1524521941871_0405 running in uber mode : false
18/04/28 23:31:18 INFO mapreduce.Job:  map 0% reduce 0%
```

3 Table 3: Performance evaluation of sort (weak scaling-large dataset)



Experiment	Shared Memory (1VM 20GB	Linux Sort (1VM 20GB)	Hadoop Sort (4VM 80GB)	Spark Sort (4VM 80GB)
Computation Time (sec)	1027.842	481	2731.897	2047.619
Data Read (GB)	80	60	160	80
Data Write (GB)	80	60	160	80
I/O Throughput (MB/sec)	155.6659487	249.4802495	117.1347236	78.08854851
Speedup(Times)	N/A	2.14 X	1.52 X	2.04 X
Efficiency	N/A	46.72897196	73.112	98.124

Spark Run



```
schatterjee@hadoop-f:~/cs553-pa2b$ javac -cp /opt/spark-2.3.0-bin-hadoop2.7/jars/spark-core_2.11-2.3.0.jar:/opt/spark-2.3.0-bin-hadoop2.7/jars/scala-compiler-2.11.8.jar:/opt/spark-2.3.0-bin-hadoop2.7/jars/scala-library-2.11.8.jar  SparkSort.java
schatterjee@hadoop-f:~/cs553-pa2b$
schatterjee@hadoop-f:~/cs553-pa2b$ spark-submit --class SparkSort --master local --deploy-mode client --executor-memory 1g --name SparkSort --conf "spark.app.id=SparkSort" hdfs://localhost:8020/input/data-8GB/data-8GB.in 2
2018-04-29 00:34:38 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

4 What conclusions can you draw? Which seems to be best at 1 node scale? How about 4 nodes? Can you predict which would be best at 100 node scale? How about 1000 node scales?

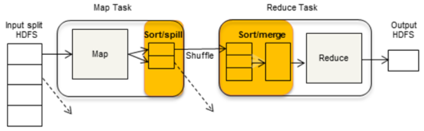
In the context of high performance computing there are two common notions of scalability:

- The first is strong scaling, which is defined as how the solution time varies with the number of processors for a fixed total problem size.

-The second is weak scaling, which is defined as how the solution time varies with the number of processors for a fixed problem size per processor

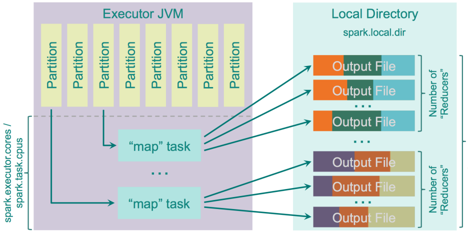
Synopsis

- Hadoop is stores intermediate result in HDFS , so it involves intermediate storage in Disk



Sort in MapReduce

- Spark uses in-memory computation and Uses RDD to speed up execution



- Spark uses in-memory computation and Uses RDD to speed up execution
- For small 2GB data set 1VM linux sort provides best performance , as Hadoop and Spark involves overhead for Scheduling and Tracking .
- For Single node Hadoop and Spark has almost similar efficiency
- For large DataSets Spark is significantly faster than Hadoop
- As we increase number of machines the efficiency doesn't increase linearly .
- For 100 node scale we should get 50-60X speed-up , for 1000 nodes speed up should be 300-400X.