Performance Benchmarks

There are multiple methods for manipulating data with Hadoop. This document summarizes our findings from comparing the performance of different methods of processing data.

The following query was executed using different technologies and methods available on the Hadoop cluster. The results recorded are the average of running each scenario 10 times.

```
select round(datediff(donation_dt, birth_dt)/365) as age, count(*)
from all_states1_2000_2016 where round(datediff(donation_dt,
birth_dt)/365) >= 0
group by round(datediff(donation_dt, birth_dt)/365) order by age;
```

The following chart shows the execution time when the data is accessed using different methods.

Execution Time (seconds)

Hive Command Line HiveQL	PySpark using Hive Context and Spark Dataframe	PySpark using SQL Context, Dataframe and Parquet-local
34	73	35

The following chart shows the execution time when processing occurs on the head node. The data is accessed from parquet files stored in HDFS using pyspark SQL Context with execution distributed across an increasing number of cores. The number of cores is specified in the local[n] where n is the number of cores being used on the head node.

Execution Time (seconds)

PySpark using SQL Context, Dataframe and Parquet-local[2]	PySpark using SQL Context, Dataframe and Parquet-local[3]	PySpark using SQL Context, Dataframe and Parquet-local[4]	PySpark using SQL Context, Dataframe and Parquet-local[8]	PySpark using SQL Context, Dataframe and Parquet-local[16]	PySpark using SQL Context, Dataframe and Parquet-local[32]
23	21	17	16	14	14

As can be seen, performance generally improves as more cores are utilized. However, a point of diminishing returns is eventually reached.

YARN

We also investigated using multiple nodes in the cluster to process the same query. We were unable to get this work from Jupyter notebook. However, we were able to invoke YARN from the command line and execute the query there.

The pyspark command line environment has to be invoked with the --master yarn option as shown below:

```
$ pyspark --master yarn

Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.5 (default, Nov 20 2015 02:00:19)
SparkContext available as sc, HiveContext available as sqlContext.
```

The query is saved to a .py file and executed from within the pyspark command line environment as shown below. Execution is distributed across multiple computing nodes in the cluster. This can be seen in the highlighted portions of the execution log shown below.

```
>>> execfile('/home/data/MySpark.py')

16/12/01 18:45:34 INFO TaskSetManager: Starting task 111.0 in stage 5.0
(TID 696, backend-0-2.insight.gsu.edu, partition 111,NODE_LOCAL, 1999
bytes)
16/12/01 18:45:34 INFO TaskSetManager: Finished task 109.0 in stage 5.0
(TID 694) in 5 ms on backend-0-2.insight.gsu.edu (110/126)
16/12/01 18:45:34 INFO TaskSetManager: Starting task 112.0 in stage 5.0
(TID 697, backend-0-1.insight.gsu.edu, partition 112,NODE_LOCAL, 1999
bytes)
16/12/01 18:45:34 INFO TaskSetManager: Finished task 110.0 in stage 5.0
(TID 695) in 6 ms on backend-0-1.insight.gsu.edu (111/126)
…
16/12/01 18:45:34 INFO TaskSetManager: Finished task 123.0 in stage 5.0
(TID 708) in 6 ms on backend-0-1.insight.gsu.edu (124/126)
16/12/01 18:45:34 INFO TaskSetManager: Finished task 124.0 in stage 5.0
(TID 709) in 4 ms on backend-0-2.insight.gsu.edu (125/126)
16/12/01 18:45:34 INFO TaskSetManager: Finished task 125.0 in stage 5.0
(TID 710) in 5 ms on backend-0-1.insight.gsu.edu (126/126)
16/12/01 18:45:34 INFO YarnScheduler: Removed TaskSet 5.0, whose tasks have
all completed, from pool
```

```
16/12/01 18:45:34 INFO DAGScheduler: ResultStage 5 (collect at
/home/data/MySpark.py:39) finished in 0.476 s
16/12/01 18:45:34 INFO DAGScheduler: Job 2 finished: collect at
/home/data/MySpark.py:39, took 5.371944 s
```

YARN Results

The results of distributing execution of the query to multiple nodes are shown below. The total time to execute the query from the MySpark.py file took a long time.

Total Time	31 seconds

We believe that this relatively long time is due to the overhead required to distribute the data and collect the results. On a smaller dataset such as ours that consists of approximately 39 million rows, the gains associated with distributing execution to multiple nodes is offset by the overhead incurred. On a much larger dataset, the gains associated with distributing the data will offset the overhead incurred.