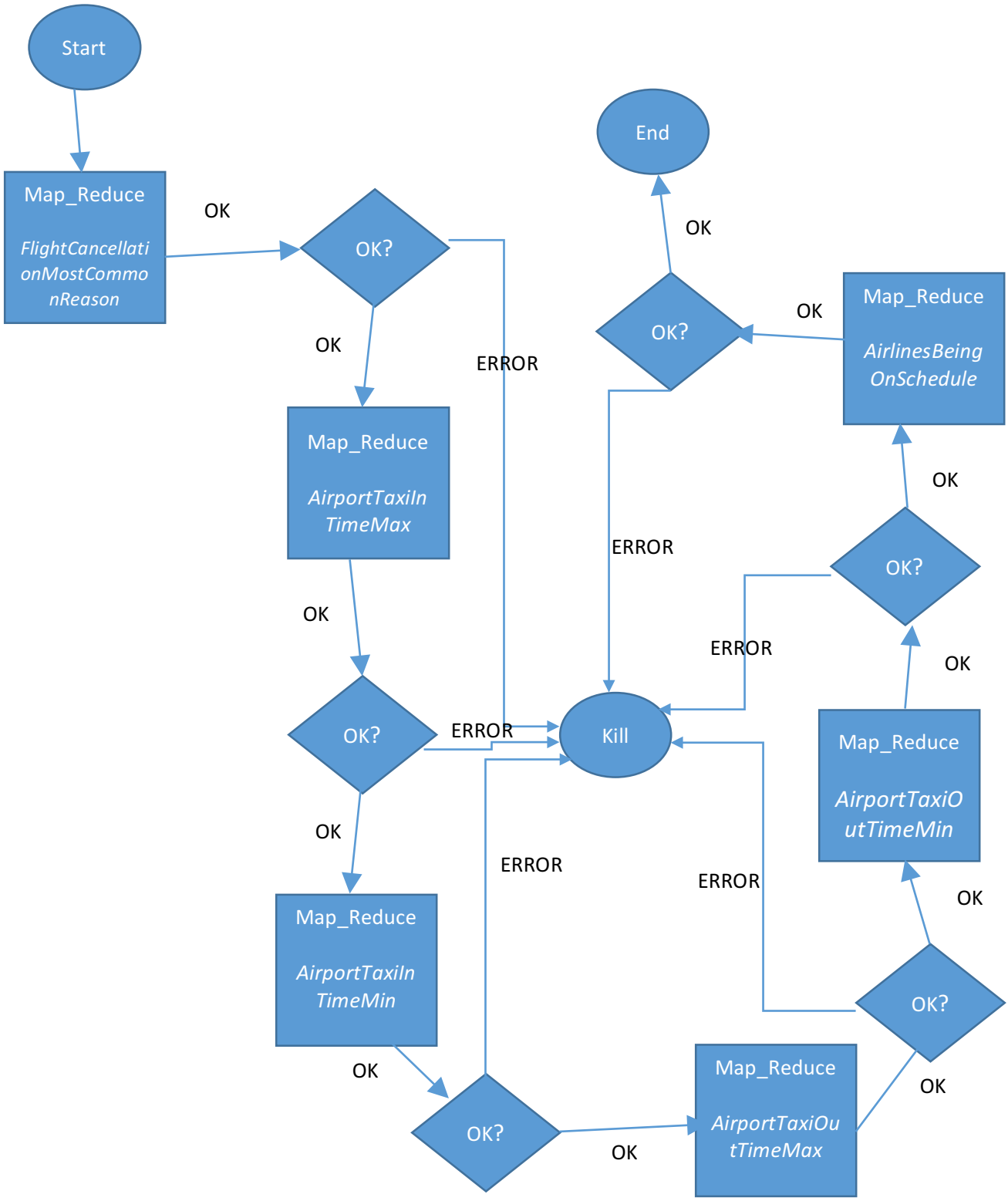
BIG DATA FINAL PROJECT REPORT

Bhaumik Patel (bkp43) and Paulius Mikalainis (pm56)

Table of Contents

1 Oozie Workflow Diagram 3
2 Algorithms 4
2.1 The most common reason for flight cancellations. 4
2.2 The 3 airports with the longest and shortest average taxi time per flight (both in and out), respectively 4
2.3 The 3 airlines with the highest and lowest probability, respectively, for being on schedule 8
3 A performance measurement plot that compares the workflow execution time in response to an increasing number of VMs used 9
4 A performance measurement plot that compares the workflow execution time in response to an increasing data size 12

1 Oozie Workflow Diagram



2 Algorithms

2.1 The most common reason for flight cancellations.

This program uses sequence of 1 map-reduce.

map-reduce
It counts occurrence of each Flight Cancelled Code.
Here, we have counted occurrence of Flight Cancelled Code.

mapper - It generate key as Flight Cancelled Code to count occurrence of it.
And, it generates 1 as values for each of them.
So, mapper will generate --> key - A; value 1
--> key - B; value 1

reducer - It calculates occurrence of all Flight Cancelled Code, and also find which is Max.
So, it will produce max Flight Cancelled Code and its total count.
Example : A 54

2.2 The 3 airports with the longest and shortest average taxi time per flight (both in and out), respectively

Airport Taxi In time Max and Min
For Both, we need average Taxi In Time

Airport Average Taxi In time
map-reduce
It calculates average Taxi In time for each airport.

AirportTaxiInTimeMapper
mapper - It generate key as airport Code.
And, it generates airport Taxi In time as value.
So, mapper will generate --> key - A; value 2
--> key - A; value 4
--> key - B; value 8
--> key - B; value 6
AirportTaxiTimeAvgReducer
reducer - It calculates average Taxi In time for all the airports.
So, it will produce average Taxi In time for all airports.
Example : A 3
B 7

Airport Taxi In time Max
map-reduce

It finds top 3 average Taxi In time.

FindingTopKMapper

mapper - It generates key as average Taxi In time of airport
And, It generates value as average airport code.
Means it reveres key and value to perform sorting based on Taxi In time.
So, A 3 will be converted to --> key - 3; value - A

LongWritable.DecreasingComparator
Shorting - It shorts key-pairs in decreasing order.

FindingTop3Combiner

Combiner - Each combiner produces first 3 key-pairs by decreasing order of avg. taxi time
from all shorted key-pairs to make sure that reducers will have less records to process.

FindingTop3Reducer

reducer - It produces first 3 key-pairs by decreasing order of avg. taxi time , which is our output.

Example: B 7 (here B is airport code and 7 is its average Taxi In time)
A 3 (here A is airport code and 3 is its average Taxi In time)
C 2 (here C is airport code and 2 is its average Taxi In time)

Airport Taxi In time Min
map-reduce

It finds lowest 3 average Taxi In time.

FindingTopKMapper

mapper - It generates key as average Taxi In time of airport
And, It generates value as average airport code.
Means it reveres key and value to perform sorting based on Taxi In time.
So, A 3 will be converted to --> key - 3; value - A

DoubleWritable.Comparator
Shorting - It shorts key-pairs in increasing order.

FindingTop3Combiner

Combiner - Each combiner produces first 3 key-pairs by increasing order of avg. taxi time from all shorted key-pairs to make sure that reducers will have less records to process.

FindingTop3Reducer

reducer - It produces first 3 key-pairs by increasing order of avg. taxi time , which is our output.

Example:
C 2 (here C is airport code and 2 is its average Taxi In time)
A 3 (here A is airport code and 3 is its average Taxi In time)
B 7 (here B is airport code and 7 is its average Taxi In time)

~~~~~~

Airport Taxi Out time Max and Min
For Both, we need average Taxi Out Time

Airport Average Taxi Out time
map-reduce

It calculates average Taxi Out time for each airport.

AirportTaxiOutTimeMapper

mapper - It generate key as airport Code.
And, it generates airport Taxi Out time as value.
So, mapper will generate --> key - A; value 2
--> key - A; value 4
--> key - B; value 8
--> key - B; value 6

AirportTaxiTimeAvgReducer

reducer - It calculates average Taxi Out time for all the airports.
So, it will produce average Taxi Out time for all airports.
Example : A 3
B 7

Airport Taxi Out time Max
map-reduce

It finds top 3 average Taxi Out time.

FindingTopKMapper
~~~~~~

mapper - It generates key as average Taxi Out time of airport
And, It generates value as average airport code.
Means it reveres key and value to perform sorting based on Taxi Out time.
So, A 3 will be converted to --> key - 3; value - A

LongWritable.DecreasingComparator
Shorting - It shorts key-pairs in decreasing order.

FindingTop3Combiner
Combiner - Each combiner produces first 3 key-pairs by decreasing order of avg. taxi time
from all shorted key-pairs to make sure that reducers will have less records to process.

FindingTop3Reducer
reducer - It produces first 3 key-pairs by decreasing order of avg. taxi time , which is our output.

Example: B 7 (here B is airport code and 7 is its average Taxi Out time)
A 3 (here A is airport code and 3 is its average Taxi Out time)
C 2 (here C is airport code and 2 is its average Taxi Out time)

Airport Taxi Out time Min
map-reduce
It finds lowest 3 average Taxi Out time.

FindingTopKMapper
mapper - It generates key as average Taxi Out time of airport
And, It generates value as average airport code.
Means it reveres key and value to perform sorting based on Taxi Out time.
So, A 3 will be converted to --> key - 3; value - A

DoubleWritable.Comparator
Shorting - It shorts key-pairs in increasing order.

FindingTop3Combiner
Combiner - Each combiner produces first 3 key-pairs by increasing order of avg. taxi time
from all shorted key-pairs to make sure that reducers will have less records to process.

FindingTop3Reducer
reducer - It produces first 3 key-pairs by increasing order of avg. taxi time , which is our output.

Example: C 2 (here C is airport code and 2 is its average Taxi Out time)

A 3 (here A is airport code and 3 is its average Taxi Out time)
B 7 (here B is airport code and 7 is its average Taxi Out time)

2.3 The 3 airlines with the highest and lowest probability, respectively, for being on schedule

Airlines reliability
This program uses sequence of 1 map-reduce.

FlightArrDelayMapper1
mapper - For each data row, it generate key as Carrier Code with prefix appended "a-".
And, it also generate key as Carrier Code with prefix appended "a-"
if that data row has Air Delay > 10 mins
And, it generates 1 as values for each of them
So, for data row with Carrier Code "A", and Air Delay 11
mapper will generate --> key - a-A; value 1
--> key - b-A; value 1
So, for data row with Carrier Code "B", and Air Delay 4
mapper will generate --> key - a-B; value 1

FlightArrDelayReducer1
reducer - It calculates probabilities for each carrier(flight) being late.
It finds top 3 most reliable and less reliable flights from the all flights.

So, it will produce most reliable flight codes as keys and their probabilities of being late as values.

Example : X 0.10
Y 0.15
Z 0.20

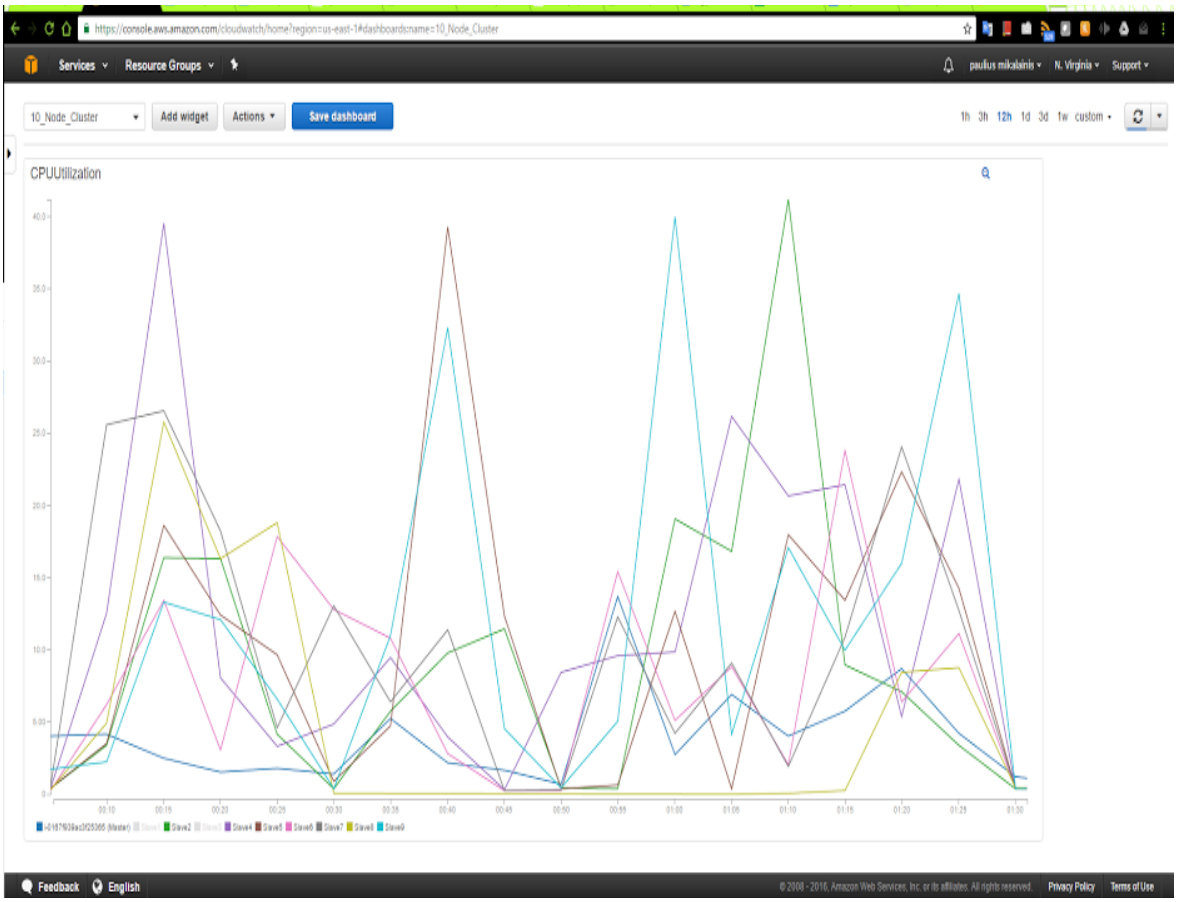
So, it will produce least reliable flight codes as keys and their probabilities of being late as values .

Example : A 0.40
B 0.35
C 0.30

3 A performance measurement plot that compares the workflow execution time in response to an increasing number of VMs used

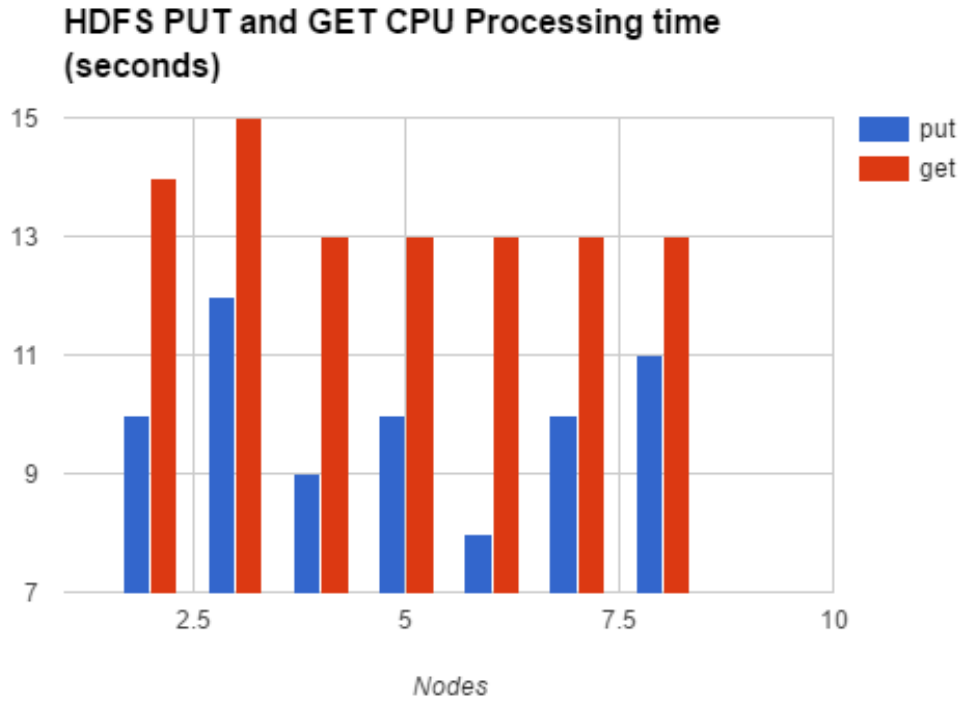
We used 2008 flight data to perform first part of the analysis. We measured three time deltas for each run: time to put data on the HDFS, time time to process the data and time to get data from HDFS.

Below an AWS Cloudwatch dashboard displaying CPU Utilization percent on x-axis and time on y-axis for 8 node cluster while processing 2008 data.



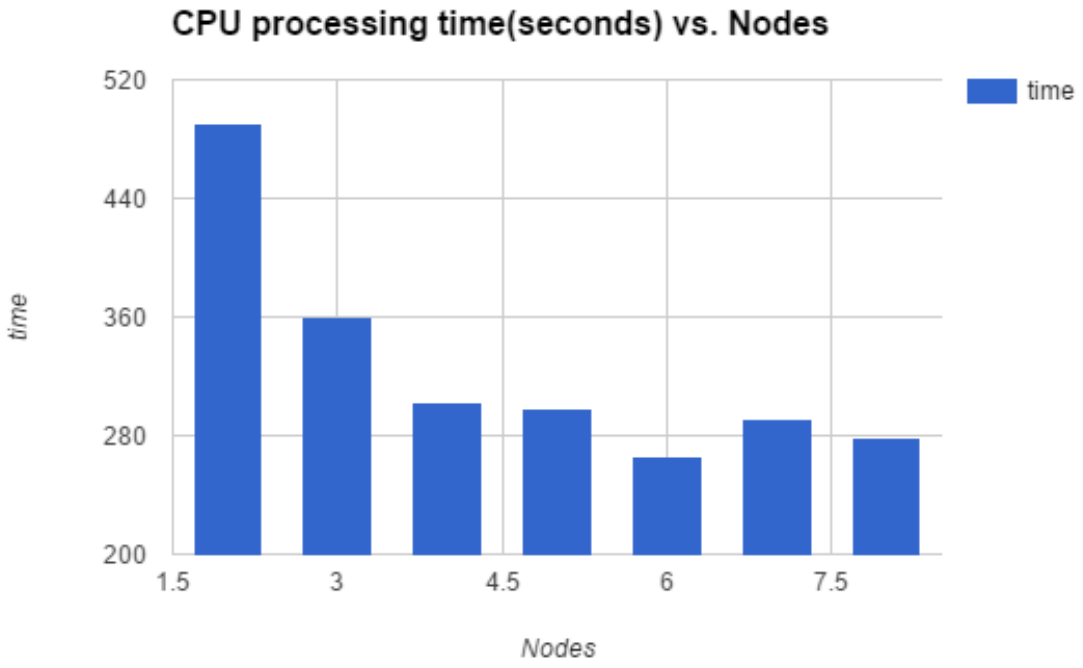
We used the following HDFS commands to put and get data:
$HADOOP_HOME/bin/hdfs dfs -put
$HADOOP_HOME/bin/hdfs dfs –get

Below is a chart displaying the times that took to put and get data from HDFS using 2,3,4,5,6,7 and 8 nodes. Surprisingly enough, it the time it took to distribute data onto HDFS was longer than retrieving data from HDFS. The time differences of few seconds between put and get is very insignificant because and could be due to the NameNode processing overhead.

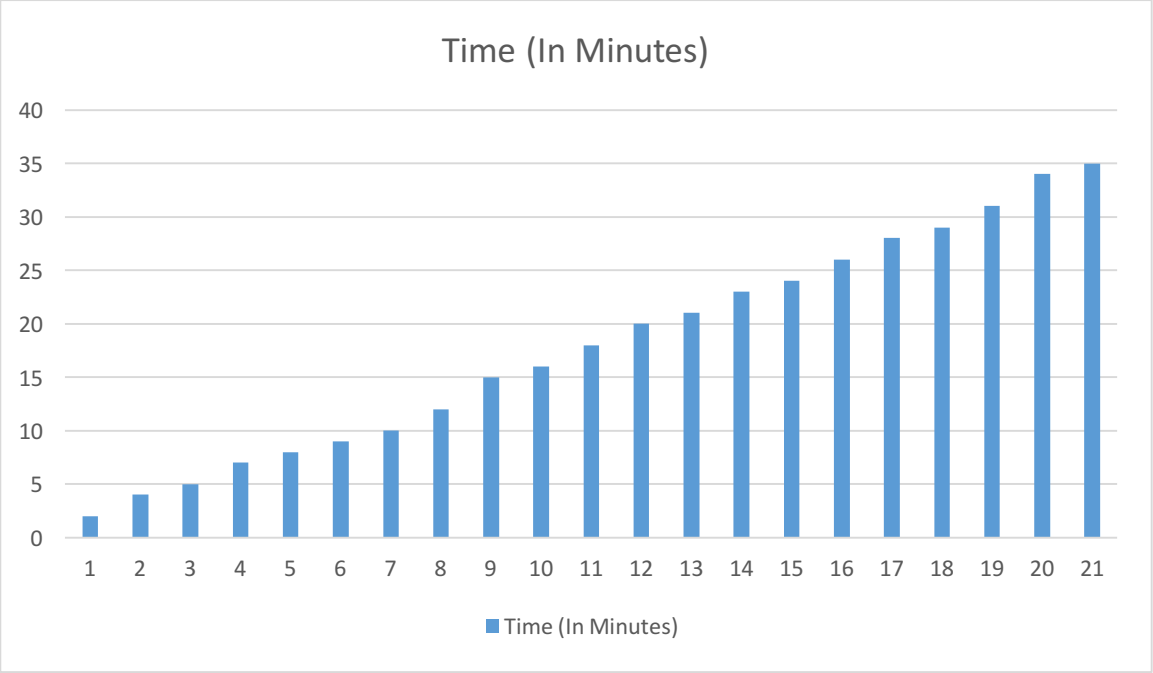


Similarly, we benchmarked processing time by number of nodes in the cluster. It took the cluster on average 8 minutes to process 2008 data on 8 nodes. The largest processing time

decreases were realized when moving from 2 to 3 node cluster (-27%) and from 3 to 4 node cluster (-17%). The rest of node increase in clusters saw about 2% improvement in time. The leveling off in processing most likely due to “NameNode” and “ResourceManager” increased overhead in extra node management and little improvement in time from actual added nodes.



4 A performance measurement plot that compares the workflow execution time in response to an increasing data size



In the graph, we have shown a number of datasets on X axis, and the time it took to process these datasets is shown on the Y axis.

We can see from the graph that as the number of datasets is increasing the time it takes to processed them is also increasing.

We were able to process all 22 datasets within 35 minutes on the 8 node cluster.