Flight Data Analysis

Authors:
David Apolinar
Shreena Mehta
Ted Moore

Introduction

Background [1]

In 2009, the ASA Data Expo presented an amalgamated data set encompassing the flight data of airlines arriving and departing from US airports from the years 1987-2008, based on the data from the Bureau of Transportation Statistics through the Research and Innovative Technology Administration. The Expo presented a **.csv** file for each of the years in the range, that included 29 different variable descriptions.

	Name	Description
1	Year	1987-2008
2	Month	1-12
3	DayofMonth	1-31
4	DayOfWeek	1 (Monday) - 7 (Sunday)
5	DepTime	actual departure time (local, hhmm)
6	CRSDepTime	scheduled departure time (local, hhmm)
7	ArrTime	actual arrival time (local, hhmm)
8	CRSArrTime	scheduled arrival time (local, hhmm)
9	UniqueCarrier	unique carrier code
10	FlightNum	flight number
11	TailNum	plane tail number
12	ActualElapsedTime	in minutes

13	CRSElapsedTime	in minutes
14	AirTime	in minutes
15	ArrDelay	arrival delay, in minutes
16	DepDelay	departure delay, in minutes
17	Origin	origin IATA airport code
18	Dest	destination IATA airport code
19	Distance	in miles
20	TaxiIn	taxi in time, in minutes
21	TaxiOut	taxi out time in minutes
22	Cancelled	was the flight cancelled?
23	CancellationCode	reason for cancellation (A = carrier, B = weather, C = NAS, D = security)
24	Diverted	1 = yes, 0 = no
25	CarrierDelay	in minutes
26	WeatherDelay	in minutes
27	NASDelay	in minutes
28	SecurityDelay	in minutes
29	LateAircraftDelay	in minutes

NOTE: A flight is considered **delayed** when it arrived 15 or more minutes than the schedule. **Delayed** minutes are calculated for delayed flights only. When multiple causes are assigned to one delayed flight, each cause is prorated based on delayed minutes it is responsible for. The displayed numbers are rounded and may not add up to the total.

Purpose [2]

Taking the Airline On-time Performance data set (flight data set) from the period of October 1987 to April 2008 on the Statistical Computing website, our goal was to design, implement, and run **MapReduce** jobs to find out a few key stats:

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

- the 3 airports with the longest and shortest average taxi time per flight (both in and out), respectively
- the most common reason for flight cancellations

In finding these three goal metrics, there were also three requirements we strove to meet:

- Utilize at least three **MapReduce** jobs
- Analyze the entire data set (total 22 years from 1987 to 2008) at one time and measure the execution time
- Analyze the data in a progressive manner with an increment of 1 year, i.e. the first year (1987), the first 2 years (1987-1988), the first 3 years (1987-1989), ..., and the total 22 years (1987-2008), and measure each corresponding execution time

All of this was to be done on Hadoop-installed virtual machines, hosted on some cloud platform of our choice. Our purpose in following these project specifications was to reinforce our understanding of **Hadoop** and **MapReduce** applications as they could apply to real-world data sets.

Summary of Materials

- `*.java` - MapReduce classes
- `njit-644-airlines.jar` - The jar archive, including all of the Map Reduce classes, on which the Hadoop jobs were executed
- `output.txt` - The results from each run of each Map Reduce job. Once for each incremental set of years (22 each) and one for all 22 years combined (1 each) for a total of 69.
- `commands.txt` - The setup commands to create a **fully distributed Hadoop cluster on Azure** and ultimately execute the jobs
- `mr_sorted.sh` - A shell script to run a Map Reduce job, then run a second Map Reduce `SortDescending` to sort the results.
- `incremental_yrs.sh` - A shell script to run the Map Reduce jobs on incrementally larger data by year.
- `timing*.txt` - Output logs from Hadoop runs including standard out from Hadoop and the results of the time operation for each year

Setup

Our cloud platform of choice was Microsoft Azure.

The Azure set-up consisted of three virtual machines:

A single master node, running the name node service and two workers nodes. All VMs were deployed within a single subnet in an Azure Virtual Network (VNET) and configured with in-bound connectivity for SSH access only to the team.

Device	Type	IP Address	Subnet
hadoop-m234	Network interface	10.22.0.7	hadoop
hadoop-w1663	Network interface	10.22.0.8	hadoop
hadoop-w249	Network interface	10.22.0.9	hadoop

Virtual Network Connected Devices

Inbound security rules

Priority	Name	Port	Protocol	Source	Destination	Action
300	SSH	22,9870,8088	TCP	50.49.205.242,8.109....	Any	Allow
310	Port_22	22,9870	Any	AzureCloud	Any	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

Outbound security rules

Priority	Name	Port	Protocol	Source	Destination	Action
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

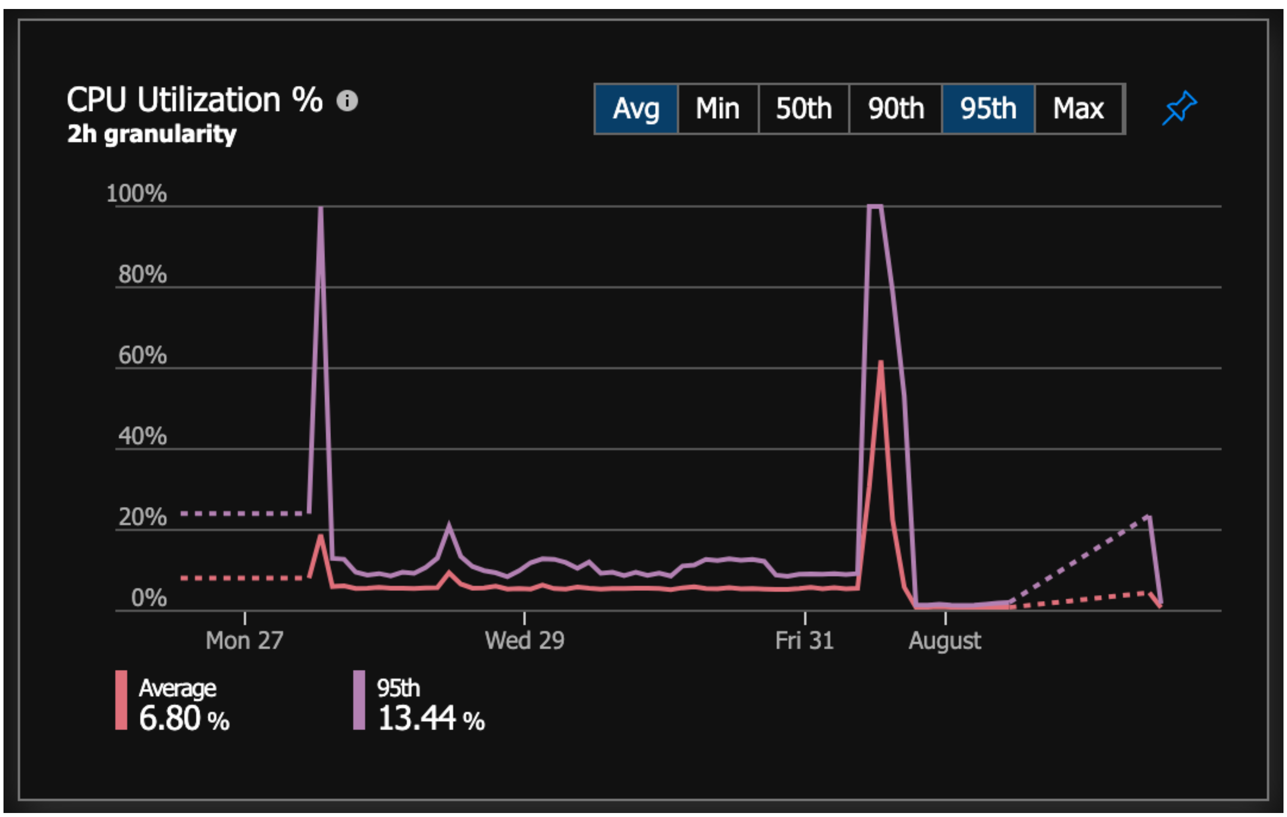
Virtual Machine Network Security Rules

For the initial configuration, we run the jobs with a D2s_v3 VM, which consistents of 2 vCPUs, and 8 GB of memory. The Dv3-series run on Intel® Xeon® Platinum 8272CL (Cascade Lake), Intel® Xeon® 8171M 2.1GHz (Skylake), Intel® Xeon® E5-2673 v4 2.3 GHz (Broadwell), or the Intel® Xeon® E5-2673 v3 2.4 GHz (Haswell) processors in a hyper-threaded configuration. The back-end storage consisted of a 128GB virtual disks, which provided 500 input/output operations per second (IOPS). While we had sufficient storage to run the full dataset, we noticed that the workload was CPU bound, meaning, that jobs were limited by the 2 vCPUs on the D2_v3 VMs. During the first runs, we noticed that the processors were hitting almost 99% CPU time, which was contributing to significant slow running times.

To improve run-time performance, we took advantage of the dynamic benefits that public cloud provides and resized the worker nodes to FSv2 Virtual Machines.

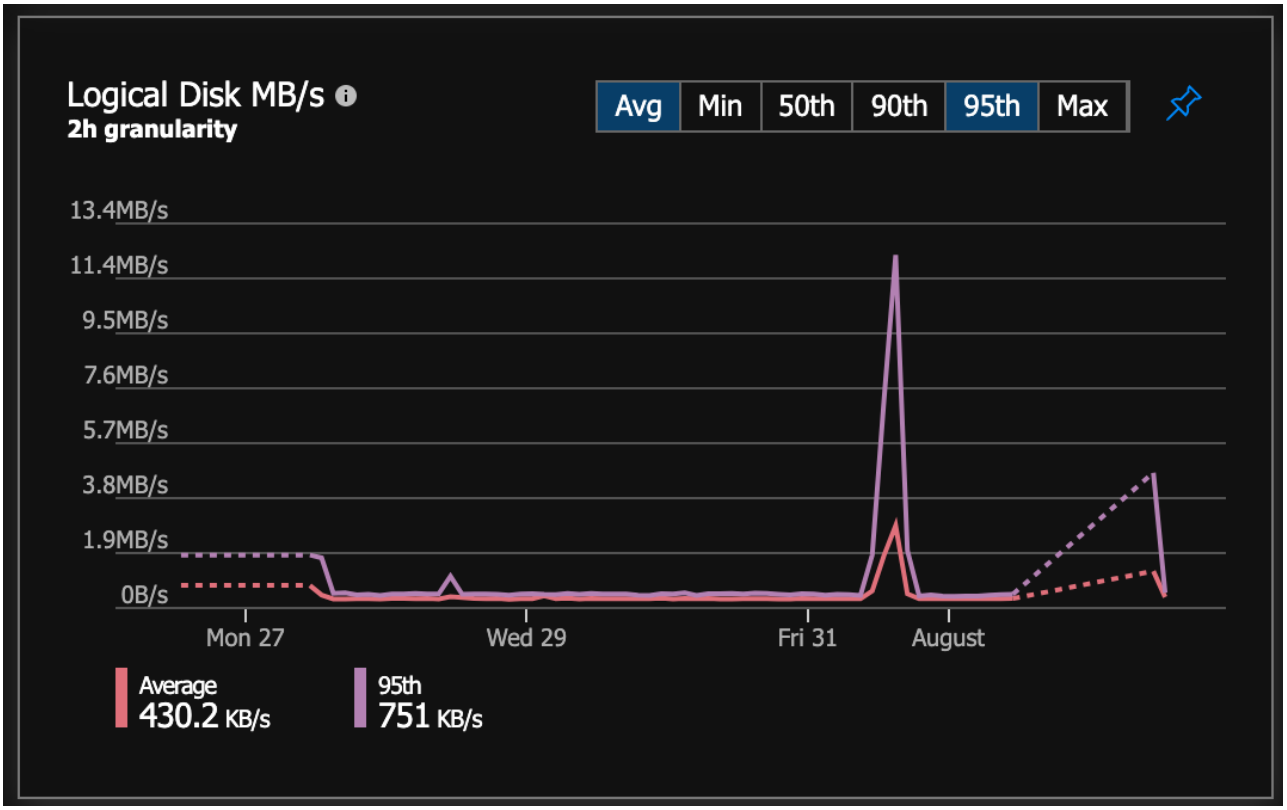
The Fsv2-series run on 2nd Generation Intel® Xeon® Platinum 8272CL (Cascade Lake) processors and Intel® Xeon® Platinum 8168 (Skylake) processors. It features a sustained all core Turbo clock speed of 3.4 GHz and a maximum single-core turbo frequency of 3.7 GHz. Intel® AVX-512 instructions are new on Intel Scalable Processors. These instructions provide up to a 2X performance boost to vector processing workloads on both single and double precision floating point operations.

Since we were hitting a cap with CPU and computing several calculations, it made logical sense to leverage Virtual Machines that could process large scale calculates much faster. This change resulted in reducing the average run times as indicated by the graph below:



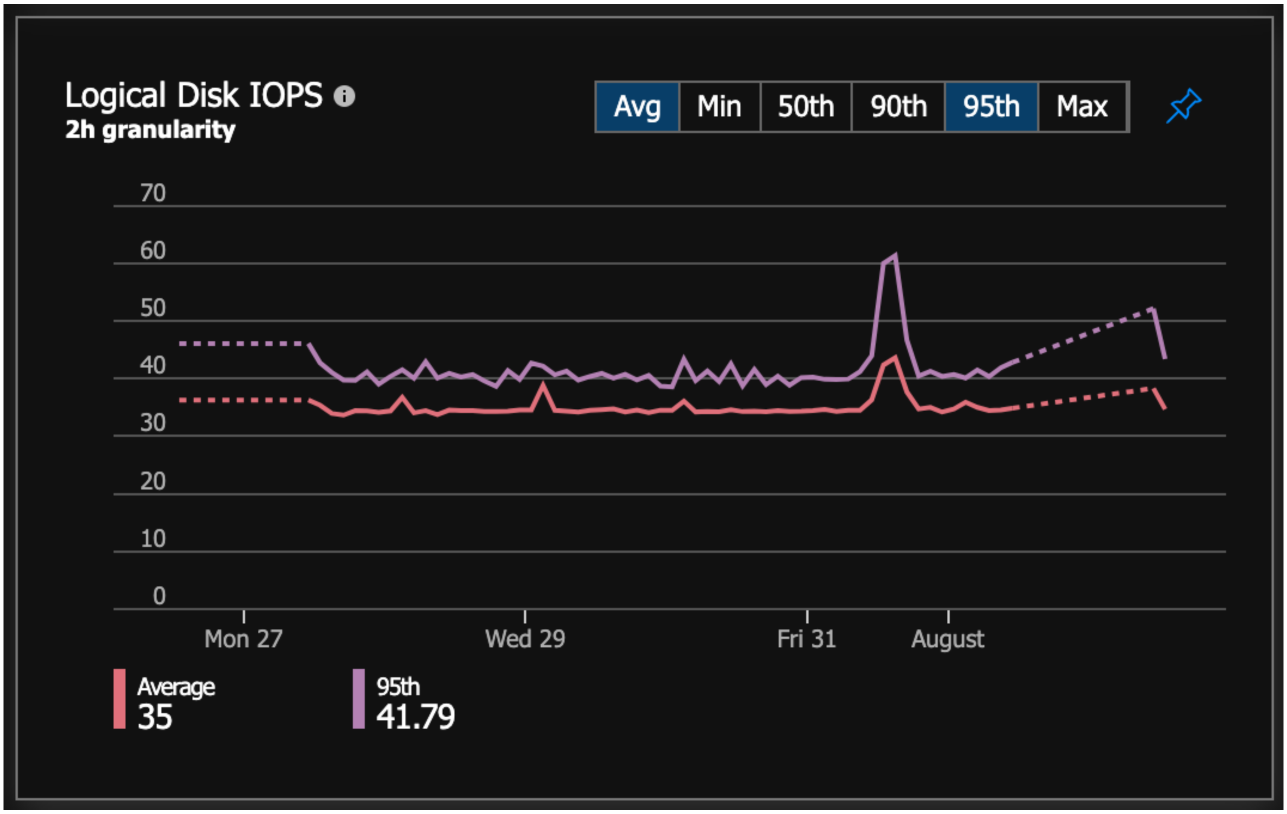
VM CPU Usages

Memory usage was fairly low as the dataset is not a typical Big Data sized set:



VM Memory Usage

Disk IOPS was also fairly low and never peaked towards the upper limit of the VM disk:



Azure Managed Disk Performance (IOPS)

One of the benefits of cloud infrastructure is the ability to deploy Infrastructure as Code. Azure provides the ability to export existing configurations for easy reuse via modifications. The configurations for the Master and Worker nodes are documented below. Using these configuration files, it is easy to deploy additional worker nodes with the JSON templates below:

An Azure VM can be deployed using the following command with the provided CLI commands:

```
group deployment create --resource-group hadoop-project-rg-name --template-uri master.json
group deployment create --resource-group hadoop-project-rg-name --template-uri worker1.json
group deployment create --resource-group hadoop-project-rg-name --template-uri worker2.json
```

Master Node ARM Template:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "virtualMachines_hadoop_m_name": {
            "defaultValue": "hadoop-m",
            "type": "String"
        },
        "disks_hadoop_m_disk1_194b28f811da4fbd9ea792f69d1ac9c4_externalid": {
            "defaultValue": "/subscriptions/f0586e39-f152-4e45-aa39-6acf1fba98ad/resourceGroups/HADOOP-NAMENODE_GROUP/providers/Microsoft.Compute/disks/hadoop-m_disk1_194b28f811da4fbd9ea792f69d1ac9c4",
            "type": "String"
        },
        "networkInterfaces_hadoop_m234_externalid": {
            "defaultValue": "/subscriptions/f0586e39-f152-4e45-aa39-6acf1fba98ad/resourceGroups/hadoop-namenode_group/providers/Microsoft.Network/networkInterfaces/hadoop-m234",
            "type": "String"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Compute/virtualMachines",
            "apiVersion": "2019-07-01",
            "name": "[parameters('virtualMachines_hadoop_m_name')]",
```

```
            "location": "eastus2",
            "tags": {
                "environment": "MSDN"
            },
            "identity": {
                "principalId": "a1bc4551-a490-4410-8b50-fdf167392abf",
                "tenantId": "fb3a99b4-c0f5-4c61-9f05-0fcbf22eaa63",
                "type": "SystemAssigned"
            },
            "properties": {
                "hardwareProfile": {
                    "vmSize": "Standard_D2s_v3"
                },
                "storageProfile": {
                    "imageReference": {
                        "publisher": "Canonical",
                        "offer": "UbuntuServer",
                        "sku": "18.04-LTS",
                        "version": "latest"
                    },
                    "osDisk": {
                        "osType": "Linux",
                        "name": "[concat(parameters('virtualMachines_hadoop_m_name'), '_disk1_194b28f811da4fbd9ea792f69d1ac9c4')]",
                        "createOption": "FromImage",
                        "caching": "ReadWrite",
                        "managedDisk": {
```

```
                            "storageAccountType": "Premium_LR
S",
                            "id": "[parameters('disks_hadoop_m
_disk1_194b28f811da4fbd9ea792f69d1ac9c4_externalid')]"
                        },
                        "diskSizeGB": 30
                    },
                    "dataDisks": []
                },
                "osProfile": {
                    "computerName": "[parameters('virtualMachi
nes_hadoop_m_name')]",
                    "adminUsername": "hadoop",
                    "linuxConfiguration": {
                        "disablePasswordAuthentication": fals
e,
                        "provisionVMAgent": true
                    },
                    "secrets": [],
                    "allowExtensionOperations": true,
                    "requireGuestProvisionSignal": true
                },
                "networkProfile": {
                    "networkInterfaces": [
                        {
                            "id": "[parameters('networkInterfa
ces_hadoop_m234_externalid')]"
                        }
                    ]
                }
            }
```

```
        },
        {
            "type": "Microsoft.Compute/virtualMachines/extensi
ons",
            "apiVersion": "2019-07-01",
            "name": "[concat(parameters('virtualMachines_hadoo
p_m_name'), '/AzureNetworkWatcherExtension')]",
            "location": "eastus2",
            "dependsOn": [
                "[resourceId('Microsoft.Compute/virtualMachine
s', parameters('virtualMachines_hadoop_m_name'))]"
            ],
            "tags": {
                "environment": "MSDN"
            },
            "properties": {
                "autoUpgradeMinorVersion": true,
                "publisher": "Microsoft.Azure.NetworkWatcher",
                "type": "NetworkWatcherAgentLinux",
                "typeHandlerVersion": "1.4"
            }
        }
    ]
}
```

Worker 1 ARM Template:

```
{
    "$schema": "https://schema.management.azure.com/schemas/20
15-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
```

```
    "parameters": {
        "virtualMachines_hadoop_w1_name": {
            "defaultValue": "hadoop-w1",
            "type": "String"
        },
        "disks_hadoop_w1_disk1_65a5e0b7441c43659354eac2dae909b6_externalid": {
            "defaultValue": "/subscriptions/f0586e39-f152-4e45-aa39-6acf1fba98ad/resourceGroups/HADOOP-NAMENODE_GROUP/providers/Microsoft.Compute/disks/hadoop-w1_disk1_65a5e0b7441c43659354eac2dae909b6",
            "type": "String"
        },
        "networkInterfaces_hadoop_w1663_externalid": {
            "defaultValue": "/subscriptions/f0586e39-f152-4e45-aa39-6acf1fba98ad/resourceGroups/hadoop-namenode_group/providers/Microsoft.Network/networkInterfaces/hadoop-w1663",
            "type": "String"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Compute/virtualMachines",
            "apiVersion": "2019-07-01",
            "name": "[parameters('virtualMachines_hadoop_w1_name')]",
            "location": "eastus2",
            "tags": {
                "environment": "MSDN"
            },
```

```
            "properties": {
                "hardwareProfile": {
                    "vmSize": "Standard_D2s_v3"
                },
                "storageProfile": {
                    "imageReference": {
                        "publisher": "Canonical",
                        "offer": "UbuntuServer",
                        "sku": "18.04-LTS",
                        "version": "latest"
                    },
                    "osDisk": {
                        "osType": "Linux",
                        "name": "[concat(parameters('virtualMachines_hadoop_w1_name'), '_disk1_65a5e0b7441c43659354eac2dae909b6')]",
                        "createOption": "FromImage",
                        "caching": "ReadWrite",
                        "managedDisk": {
                            "storageAccountType": "Premium_LRS",
                            "id": "[parameters('disks_hadoop_w1_disk1_65a5e0b7441c43659354eac2dae909b6_externalid')]"
                        },
                        "diskSizeGB": 30
                    },
                    "dataDisks": []
                },
                "osProfile": {
                    "computerName": "[parameters('virtualMachines_hadoop_w1_name')]",
```

```
                    "adminUsername": "hadoop",
                    "linuxConfiguration": {
                        "disablePasswordAuthentication": fals
e,
                        "provisionVMAgent": true
                    },
                    "secrets": [],
                    "allowExtensionOperations": true,
                    "requireGuestProvisionSignal": true
                },
                "networkProfile": {
                    "networkInterfaces": [
                        {
                            "id": "[parameters('networkInterfa
ces_hadoop_w1663_externalid')]"
                        }
                    ]
                }
            }
        }
    ]
}
```

Worker 2 ARM Template:

```
{
    "$schema": "https://schema.management.azure.com/schemas/20
15-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "virtualMachines_hadoop_w2_name": {
```

```
            "defaultValue": "hadoop-w2",
            "type": "String"
        },
        "disks_hadoop_w2_disk1_f188c95537ab4ddc85085f1f9c54adf
1_externalid": {
            "defaultValue": "/subscriptions/f0586e39-f152-4e45
-aa39-6acf1fba98ad/resourceGroups/HADOOP-NAMENODE_GROUP/provid
ers/Microsoft.Compute/disks/hadoop-w2_disk1_f188c95537ab4ddc85
085f1f9c54adf1",
            "type": "String"
        },
        "networkInterfaces_hadoop_w249_externalid": {
            "defaultValue": "/subscriptions/f0586e39-f152-4e45
-aa39-6acf1fba98ad/resourceGroups/hadoop-namenode_group/provid
ers/Microsoft.Network/networkInterfaces/hadoop-w249",
            "type": "String"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Compute/virtualMachines",
            "apiVersion": "2019-07-01",
            "name": "[parameters('virtualMachines_hadoop_w2_na
me')]",
            "location": "eastus2",
            "tags": {
                "environment": "MSDN"
            },
            "properties": {
                "hardwareProfile": {
```

```
                    "vmSize": "Standard_D2s_v3"
                },
                "storageProfile": {
                    "imageReference": {
                        "publisher": "Canonical",
                        "offer": "UbuntuServer",
                        "sku": "18.04-LTS",
                        "version": "latest"
                    },
                    "osDisk": {
                        "osType": "Linux",
                        "name": "[concat(parameters('virtualMachines_hadoop_w2_name'), '_disk1_f188c95537ab4ddc85085f1f9c54adf1')]",
                        "createOption": "FromImage",
                        "caching": "ReadWrite",
                        "managedDisk": {
                            "storageAccountType": "Premium_LRS",
                            "id": "[parameters('disks_hadoop_w2_disk1_f188c95537ab4ddc85085f1f9c54adf1_externalid')]"
                        },
                        "diskSizeGB": 30
                    },
                    "dataDisks": []
                },
                "osProfile": {
                    "computerName": "[parameters('virtualMachines_hadoop_w2_name')]",
                    "adminUsername": "hadoop",
                    "linuxConfiguration": {
```

```
                        "disablePasswordAuthentication": fals
e,
                        "provisionVMAgent": true
                    },
                    "secrets": [],
                    "allowExtensionOperations": true,
                    "requireGuestProvisionSignal": true
                },
                "networkProfile": {
                    "networkInterfaces": [
                        {
                            "id": "[parameters('networkInterfa
ces_hadoop_w249_externalid')]"
                        }
                    ]
                }
            }
        }
    ]
}
```

Once we had these instances configured, our next step was to install Java, Hadoop, and Yarn. as well as set up our working environments and HDFS using the following commands:

```
# Head Node Set-up
ssh-keygen -t rsa -P ""
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
ssh localhost

# Copy SSH ID to Worker 1 and Worker 2
ssh-copy-id hadoopuser@hadoop-m
```

```
ssh-copy-id hadoopuser@hadoop-w1
ssh-copy-id hadoopuser@hadoop-w2

# Java Install (Run on master, workers)
install openjdk-8-jdk
java -version
sudo wget -P ~ https://mirrors.sonic.net/apache/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz
tar xzf hadoop-3.2.1.tar.gz
mv hadoop-3.2.1 hadoop

# Add export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/ to hadoop-env.sh
vim ~/hadoop/etc/hadoop/hadoop-env.sh

# move to /usr/loca/hadoop directory (Run on master, workers)
mv hadoop /usr/local/hadoop (Run on master, workers)

# Modify Environment variable configuration and add the following line to /etc/environment
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/local/hadoop/bin:/usr/local/hadoop/sbin"JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64/jre"
vim /etc/environment

# modify hosts files vim /etc/hosts as root (Run on master, workers)
echo "10.22.0.7 hadoop-m" >> /etc/hosts
echo "10.22.0.8 hadoop-w1" >> /etc/hosts
echo "10.22.0.9 hadoop-w2" >> /etc/hosts
```

```
# Modify core-site.xml
vim /usr/local/hadoop/etc/hadoop/core-site.xml

# Add master server in config
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://hadoop-m:9000</value>
</property>
</configuration>

# Modify HDFS-Site.xml
vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml
<configuration>
<property>
<name>dfs.namenode.name.dir</name><value>/usr/local/hadoop/data/nameNode</value>
</property>
<property>
<name>dfs.datanode.data.dir</name><value>/usr/local/hadoop/data/dataNode</value>
</property>
<property>
<name>dfs.replication</name>
<value>2</value>
</property>
</configuration>

# Add Workers
```

```
vim /usr/local/hadoop/etc/hadoop/workers
echo "hadoop-w1" >> /usr/local/hadoop/etc/hadoop/workers
echo "hadoop-w2" >> /usr/local/hadoop/etc/hadoop/workers

# Copy config to worker nodes
scp /usr/local/hadoop/etc/hadoop/* hadoop-w1:/usr/local/hadoo
p/etc/hadoop/
scp /usr/local/hadoop/etc/hadoop/* hadoop-w2:/usr/local/hadoo
p/etc/hadoop/

# Format namenode
source /etc/environment
hdfs namenode -format

# Start HDFS
start-dfs.sh

# Configure Yarn
vim /usr/local/hadoop/etc/hadoop/yarn-site.xml
<property>
<name>yarn.resourcemanager.hostname</name>
<value>hadoop-m</value>
</property>

# Copy config to worker nodes
scp /usr/local/hadoop/etc/hadoop/* hadoop-w1:/usr/local/hadoo
p/etc/hadoop/
scp /usr/local/hadoop/etc/hadoop/* hadoop-w2:/usr/local/hadoo
p/etc/hadoop/

# Start-Yarn
```

```
start-yarn.sh
```

Algorithmic Approaches

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

> We approached this task by writing a binary `onTime` variable (really we created two `IntWritable` variables: `one` and `zero`) to `context` in the map phase, `1` for on schedule, `0` for delayed. A flight that was delayed 15 minutes or more upon departure and/or on arrival were considered **not** on time. The keys used were the codes for the airline carriers.
> In the reduce phase, we kept a counter to track the total number of flights by carriers as the denominator, and summed up the binary `onTime` as the numerator, providing the on time probability of each carrier.
> To sort, we created a second **MapReduce** function called `SortDescending`. This simply swaps the key and the values (`value * -1`, to sort from greatest to least) so that the automatic sorting achieves the desired goal, then swaps back the key and value in the reduce phase, returning a sorted list.

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

> This we approached by writing to `context` twice per flight record in the map phase: once for the `TaxiOut` time and once for the `TaxiIn` time. Because a flight doesn't (usually) land at the airport it took off from, this was a necessary step to parse out the relevant data points from the raw data. Since the taxi time calculation is "both in and out" we simply added up all the taxi records for each airport key, and kept a simple counter to create the denominator. The division of these is the final result.
> The sorting again was handled by the `SortDescending` method from the previous task.

The most common reason for flight cancellations

This function was the simplest of the three. We looked at the cancellation code, a binary `0` or `1`, and if it was a canceled flight, we parsed the corresponding reason from the record. As we're tabulating cancellation reasons, the keys used in the mapping phase were the **4 cancellation codes** (`A` = carrier, `B` = weather, `C` = NAS, `D` = security), as well as an additional value of `N/A`. It appears as if these codes were not collected for several years, so a large number of earlier flights are encoded as `N/A`.

Running the jobs

To return each job sorted, we implemented the following shell script `mr_sorted.sh` to run the jobs in sequence, where `$1` is the JavaClass:

```
#!/bin/bash


yarn jar njit-644-airlines.jar $1 /$2 /unsort_temp
yarn jar njit-644-airlines.jar SortDescending /unsort_temp /$3
hdfs dfs -rm -r /unsort_temp
```

In order to run on incrementally larger sets of years, we used the Hadoop convention for specifying files within a directory to include, for example:

```
yarn jar a-jar-file.jar JavaClass input/{x,1987.csv.bz2,1988.csv.bz2,1989.csv.bz2,1990.csv.bz2} output
```

will run only on the files for the years 1987-1990. To generate this programmatically, we implemented another script that called `mr_sorted.sh` for each set of years in turn and stored the time results to an output file:

```
#!/bin/bash


for i in $(seq 1987 2008);
do
  match="{x"
```

```
  for ((j=1987; j<=$i; j++));
  do
    match=$match","$j.csv.bz2
  done
  match=$match"}"
  j="$((j-1))"
  echo "Running through year: $j"
  { time bash mr_sorted.sh $1 input/$match out_$1_$j ; } 2>> timing_$1.txt
done
```

This allowed us to recover the runtimes for each incremental set of years from 1987-2008.

Results

Goal Metrics (1987-2008)

Below are the direct answers to the questions in the prompt. The full output of the **MapReduce** jobs can be seen in `output.txt`

The 3 airlines with the highest and lowest probability, respectively, for being on schedule.
Highest Probability: Hawaiian Airlines, Aloha Airlines, Midway Airlines Inc.

HA	0.9326062
AQ	0.90286916
ML (1)	0.84682935

Lowest Probability: Piedmont Aviation, JetBlue Airways, Atlantic Southeast Airlines

PI	0.74301124
B6	0.7230906
EV	0.71170384

The 3 airports with the longest and shortest average taxi time per flight (both in and out), respectively.
NOTE: This data has been recorded since 1995.
There are several outliers in this dateset, and for which the taxi time averages don't move year-over-year, indicating very few or no flights. Another job should be programmed to use the total number of records and exclude from the list if it is less than a given threshold (e.g. < 25).

Longest Taxi Time (both in/out): Valdosta Regional, Florence Regional, Gainesville Regional

CKB	197.85715 (exclude)
MKK	45.778164 (exclude)
VLD	22.28021
FLO	21.845243
GNV	21.60531

Shortest Taxi Time (both in/out): Cheyenne, Scotts Bluff County, Provo Muni (caveat: these probably would not make the cut over a threshold)

PVU	1.7692307
BFF	1.3333334
CYS	1.3333334
SKA	0.0 (exclude)
RCA	0.0 (exclude)
LBF	0.0 (exclude)
LAR	0.0 (exclude)

The most common reason for flight cancellations.
NOTE: This data has been recorded since 2003.
Most Common Cancellation Reason: Carrier (A)

A	317868.0
B	267000.0
C	149060.0

D	601.0

Performance Measurement Plots

On-Schedule Probability

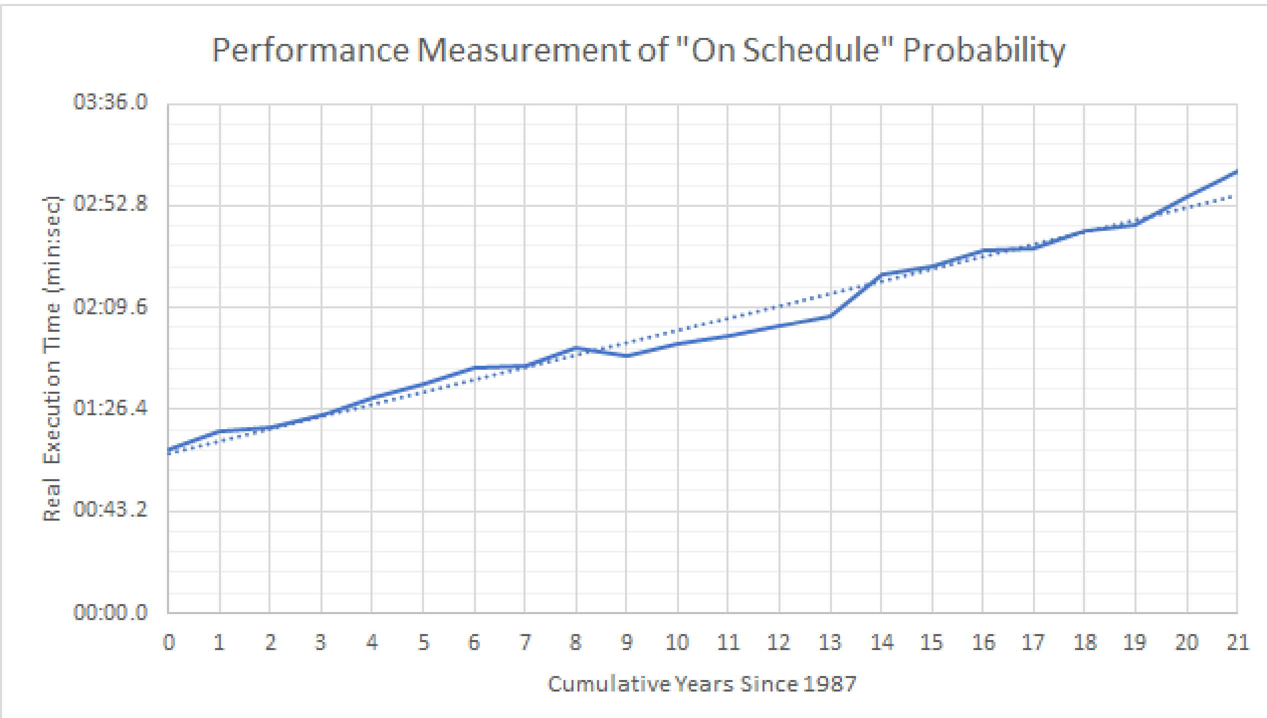
Isolated 22 years execution:

Probability -> 2:25

Sort Descending -> 0:12

Total ---> 2:37

Incremental years execution:



Incremential Execution Time (Years)

In the graph, we have shown the number of cumulative data sets on the X axis as representing the increasing data size (from 1 year to 22 years), and the time it took in minutes and seconds 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 approximately 3 minutes using 8 core nodes.

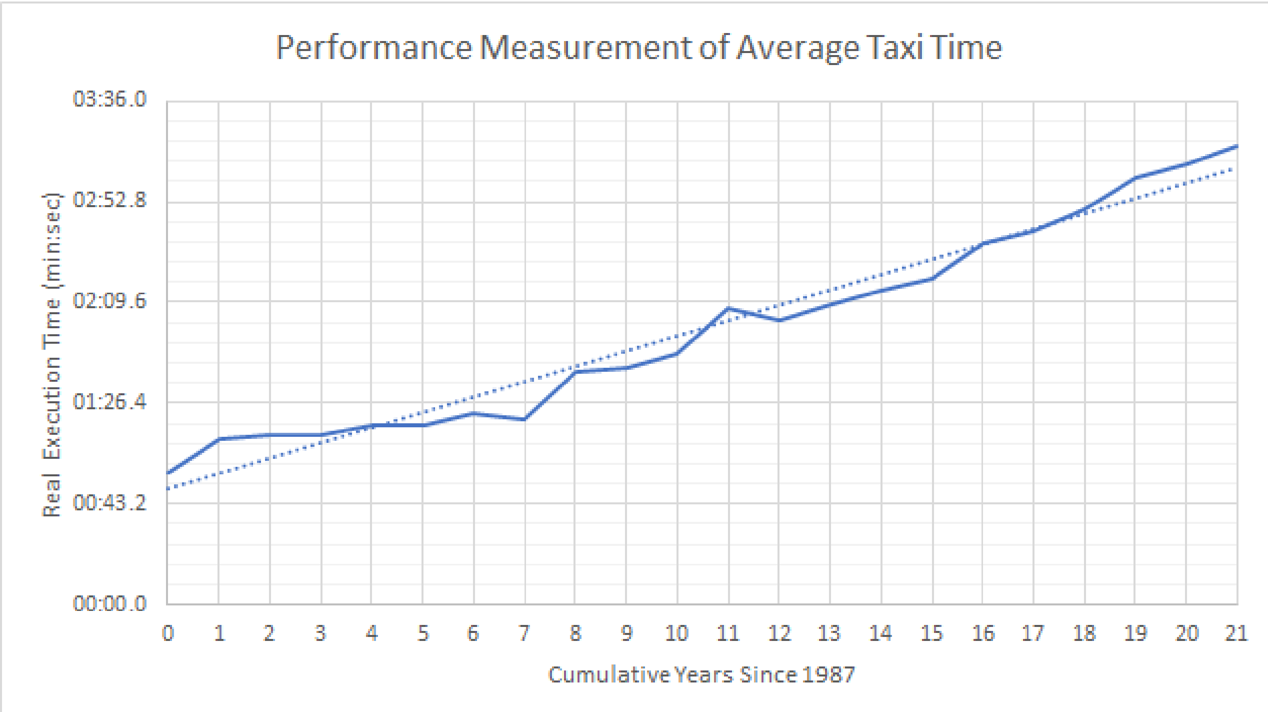
Average Taxi Time per Flight

Isolated 22 years execution:

TaxiTime -> 2:37

Sort Descending -> 0:11
Total ---> 2:48

Incremental years execution:

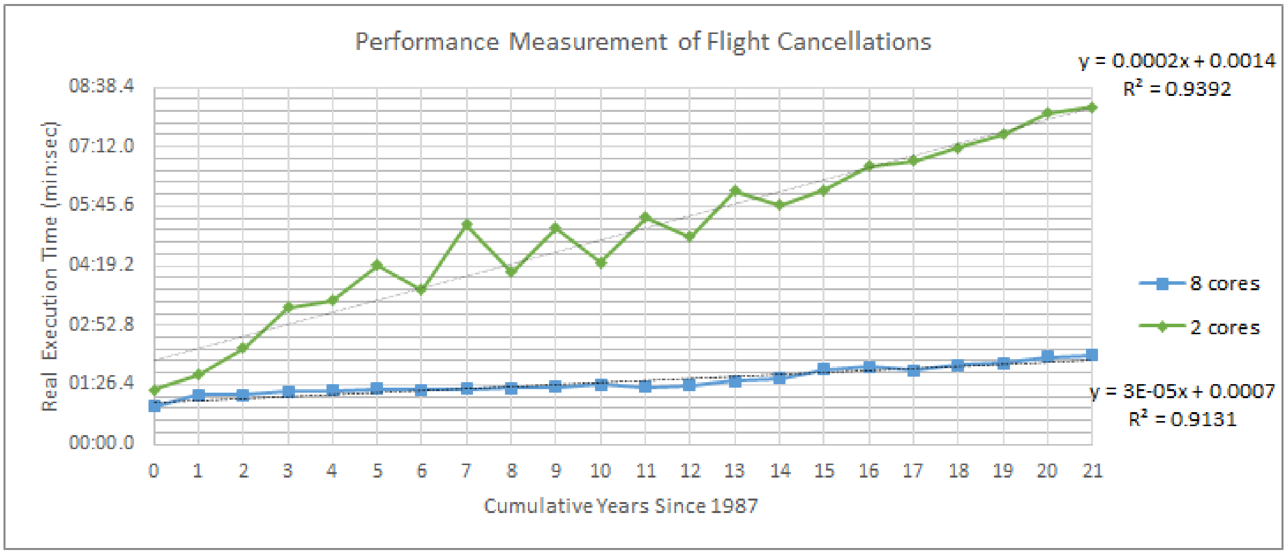


Incremental Job Execution Time

In the graph, we have shown the number of cumulative data sets on the X axis as representing the increasing data size (from 1 year to 22 years), and the time it took in minutes and seconds 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 approximately 3.5 minutes using 8 core nodes.

Most Common Reason for Flight Cancellations
Isolated 22 years execution:
AirlineCancellation -> 1:27
Sort Descending -> 0:11
Total ---> 1:38

Incremental years execution:



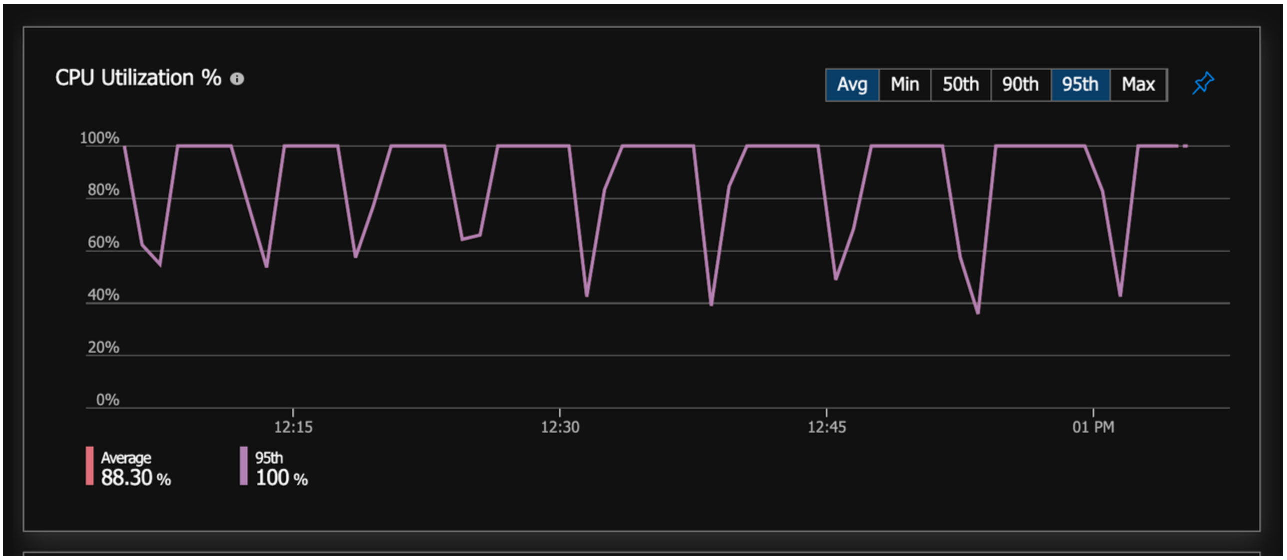
In the graph, we have shown the number of cumulative data sets on the X axis as representing the increasing data size (from 1 year to 22 years), and the time it took in minutes and seconds on the Y axis. We can see from the graph that as the number of datasets is increasing the time it takes to processed them is also increasing. We were able to process all 22 datasets within approximately 8.5 minutes using 2 cores, and 2 minutes using 8 cores.

Conclusion

These results yielded a couple of takeaways. First, we observed that the "Flight Cancellation Reasons" job operates much faster than the others, largely because we were able to skip writing to context in the map phase for all non-canceled flights. This reduced the amount of time needed for writing/reading to disk and reduced the work of the reducer, as there were less key-values pairs sent through.

A second observation is that in all cases, the time required appears to increase linearly with the amount of data. Linear is typically a good thing in the world of run-times, but our intuition was that it might start to taper off once the cost of the overhead was amortized over many years of data.

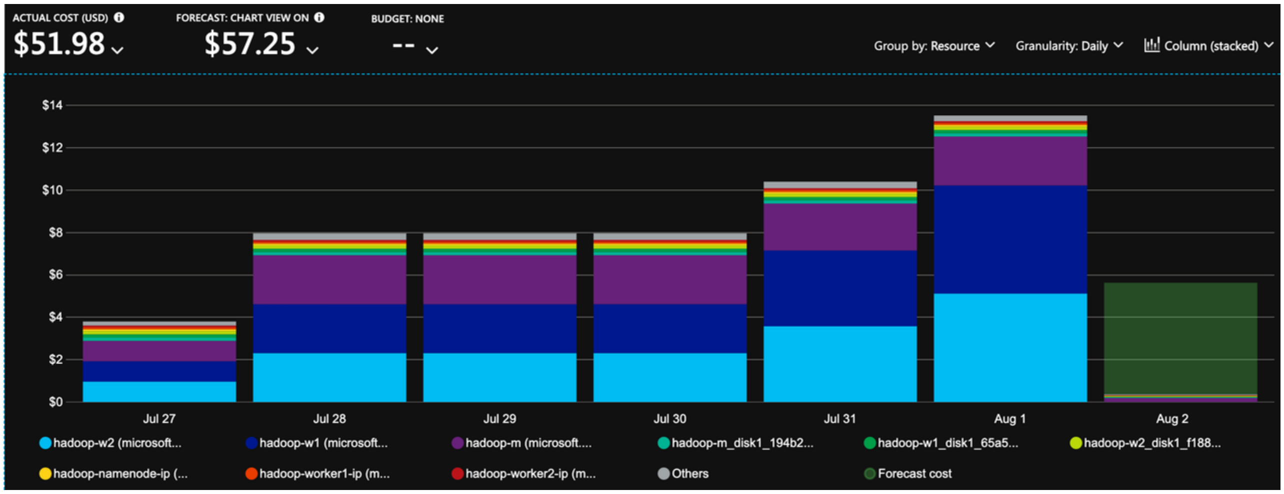
Finally, our A/B node comparison yielded fascinating results. For the Flight Cancellation job we first ran on two less powerful nodes, notably with 2 CPU cores each. In observing the performance charts in Azure Portal, we discovered that the task was CPU-bound, and we were maxing out intermittently (see image below). To account for this we rebooted our workers with much faster, 8-core CPUs, and observed a tremendous increase in performance. While the results still increase linearly, the slope was at about 1/15th of the smaller nodes (see image above).



CPU Performance

Cost Information

While it would have been fairly easy to add additional worker nodes to reduce the run time, we felt that leveraging two nodes with 8 vCPUs was good cost/performance ratio. Our total cost for this job ended up totalling roughly $50 to process all of the data. This includes both the original SKU size (2 vCPU by 8 GB) and the compute optimized SKUs (8 vCPU by 16GB). We were able to gain more than 4X performance gains by only paying less than double the cost. This flexibility allowed our to experiment and run the Hadoop jobs on all of the data.



Azure Daily Cost

Citations

1. http://stat-computing.org/dataexpo/2009/the-data.html
2. https://njit.instructure.com/courses/11885/assignments/59290
3. https://www.bts.dot.gov/topics/airlines-and-airports/understanding-reporting-causes-flight-delays-and-cancellations
4. https://web.archive.org/web/20070930184124/http://www.transtats.bts.gov/OT_Delay/OT_DelayCause1.asp
5. https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-compute?toc=/azure/virtual-machines/linux/toc.json&bc=/azure/virtual-machines/linux/breadcrumb/toc.json