COSC 6339

Fall 2018

Big Data Analytics

1st Assignment

Jaivardhan Singh Shekhawat

Problem Description:

Given a data set containing all flights which occurred between 2006 and 2008 in the US

- **~21 Million flights listed in the file**
- **small file for code development with 286 flights available in HDFS**
- **each line is one flight with information as listed on the next pages**

Name	Description
Year	1987-2008
Month	1-12
DayofMonth	1-31
DayOfWeek	1 (Monday) -7 (Sunday)
DepTime	actual departure time (local, hhmm)
CRSDepTime	scheduled departure time (local, hhmm)
ArrTime	actual arrival time (local, hhmm)
CRSArrTime	scheduled arrival time (local, hhmm)
UniqueCarrier	unique carrier code
FlightNum	flight number
TailNum	plane tail number
ActualElapsedTime	in minutes
CRSElapsedTime	in minutes
AirTime	in minutes
ArrDelayarrival	delay, in minutes
DepDelaydeparture	delay, in minutes
Origin	origin IATA airport code
Dest	destination IATA airport code
Distance	in miles
TaxiIn	taxi in time, in minutes
TaxiOut	taxi out time in minutes
Cancelled	was the flight cancelled?

CancellationCode	reason for cancellation (A = carrier, B = weather,C= NAS, D = security)
Diverted	1 = yes, 0 = no
CarrierDelay	in minutes
WeatherDelay	in minutes
NASDelay	in minutes
SecurityDelay	in minutes
LateAircraftDelay	in minutes

a. Implement a MapReduce job which determines the percentage of delayed flights per Origin Airport

Hadoop with python language combines to become Pydoop. Hadoop is a framework which supports the distributed execution of large scale data processing and is famous for the MapReduce parallel programming pattern. The cluster does have a HDFS file system that can be accessed from all nodes and should be used for MapReduce jobs. In MapReduce job, it contains mapper class and reducer class. Mapper class contains mapper function, which divides and sorts the data while reducer class contains reducer function, which combines the data for output.

Solution Strategy:

Mapper:

In mapper function, first we split each line of data through *split()* function, and storing to *words* variable. Then, using for loop, keeping in the variable *w* splitting words through ',' and storing into *line* variable. Now, using if-else condition, we need to check whether the flight is delayed or not, as per the question. Now, if *line[15]* (i.e. delay in minutes) is equal or more than '1', then the flight is delayed, then it emits airport code along with 11 (delayed)

through *context.emit* function, whereas if *line[15]* is having values other than defined above, then flight is not delayed, then it emits airport code along with 10 (not delayed).

Reducer:

In reducer function, first I introduced 3 variables; *s*, *quotient* and *remainder*. All the variables are initialized by 0.0, so that the output comes in float. Now, adding *for* loop having variable *i* in *context.values*, so to find total flights on that airport, I have used variable *quotient*, getting through *i* divided(/) by 10 and adding the value to itself to get sum. As to find the delayed flights on that airport, I have used *remainder* variable, getting through *i* divided(%) by 10 and adding the value to itself to get sum. Finally, to find the percentage (%) of delayed flights, I had used *s* variable, getting through *(remainder*100/quotient)*. To get final result, I used *context.emit(context.key, s)*. Table 1 and Figure 1.1 represents the execution time taken by code developed by this part.

How to run code:

```
pydoop submit --num-reducers 1 --upload-file-to-cache mapper3.py mapper3
/cosc6339_hw1/flights-longlist/allflights.csv/ output37
```

How to see output:

```
hdfs dfs -cat output37/part-r-00000
```

---

b. Implement a MapReduce job which determines the percentage delayed flights per Origin Airport and Month

Solution Strategy:

Mapper:

In mapper function, first we split each line of data through *split()* function, and storing to *words* variable. Then, using for loop, keeping in the variable *w* splitting words through ',' and storing into *line* variable. Now, using if-else condition, we need to check whether the flight is delayed or not, as per the question. Now, if *line[15]* (i.e. delay in minutes) is equal or more than '1', then the flight is delayed, and *pair* variable is used to keep values of airport code and month, then it emits *pair* along with 11 (delayed) through *context.emit* function, whereas if *line[15]* is having values other than defined above, then flight is not delayed, and again *pair* variable is used to keep values of airport code and month, then it emits *pair* along with 10 (not delayed).

Reducer:

In reducer function, first I introduced 3 variables; *s*, *quotient* and *remainder*. All the variables are initialized by 0.0, so that the output comes in float. Now, adding *for* loop having variable *i* in *context.values*, so to find total flights on that airport, I have used variable *quotient*, getting through *i* divided(/) by 10 and adding the value to itself to get sum. As to find the delayed flights on that airport, I have used *remainder* variable, getting through *i* divided (%) by 10 and adding the value to itself to get sum. Finally, to find the percentage (%) of delayed flights, I had used *s* variable, getting through *(remainder*100/quotient)*. To get final result, I used *context.emit(context.key, s)*. Table 2 and Figure 2.1 represents the execution time taken by code developed by this part.

How to run code:

```
pydoop submit --num-reducers 1 --upload-file-to-cache mapper_month.py
mapper_month /cosc6339_hw1/flights-longlist/allflights.csv/
code2_output16
```

How to see output:

```
hdfs dfs -cat code2_output16/part-r-00000
```

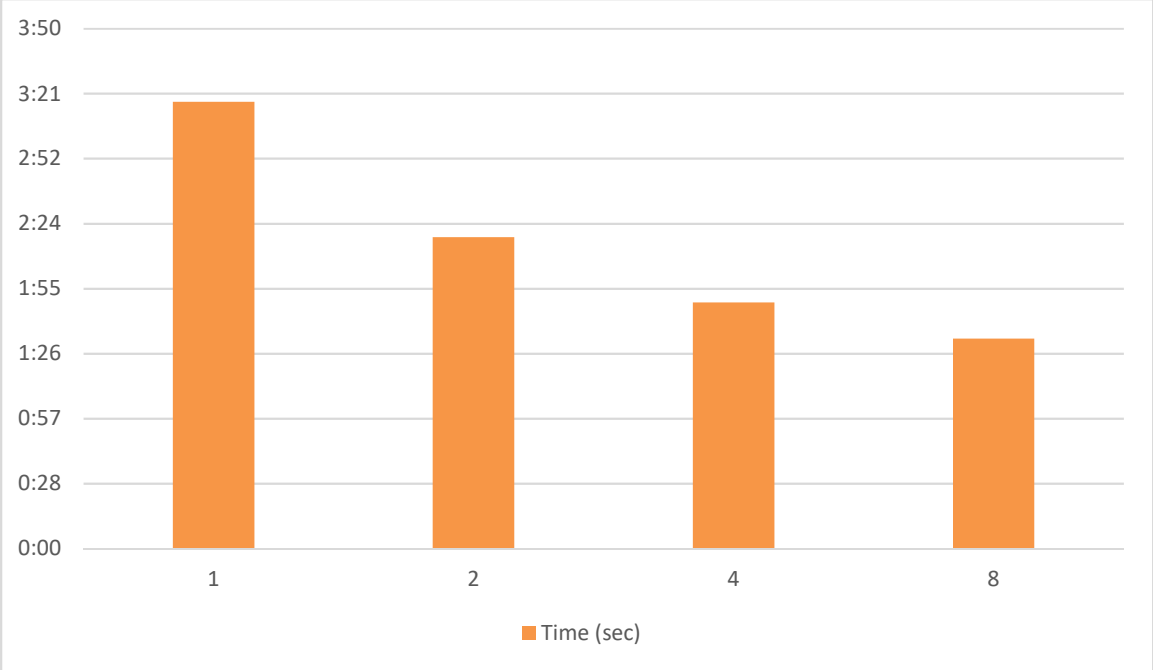
c. Determine the execution time of code developed in part a. and b. for the large data set using 1, 2, 4, and 8 reducers. Comment on the results.

Part a:

Table 1: Execution time of code developed from part a

Reducer(s) Used	Application ID	Time	Avg. Time
1	application_1532471318325_5752	3:16	3:18
	application_1532471318325_5753	3:21	
	application_1532471318325_5756	3:18	
2	application_1532471318325_5758	2:17	2:18
	application_1532471318325_5759	2:18	
	application_1532471318325_5762	2:18	
4	application_1532471318325_5765	1:49	1:49
	application_1532471318325_5767	1:49	
	application_1532471318325_5769	1:48	
8	application_1532471318325_5770	1:32	1:33
	application_1532471318325_5771	1:34	
	application_1532471318325_5772	1:31	

Figure 1.1: Graph representation of table above

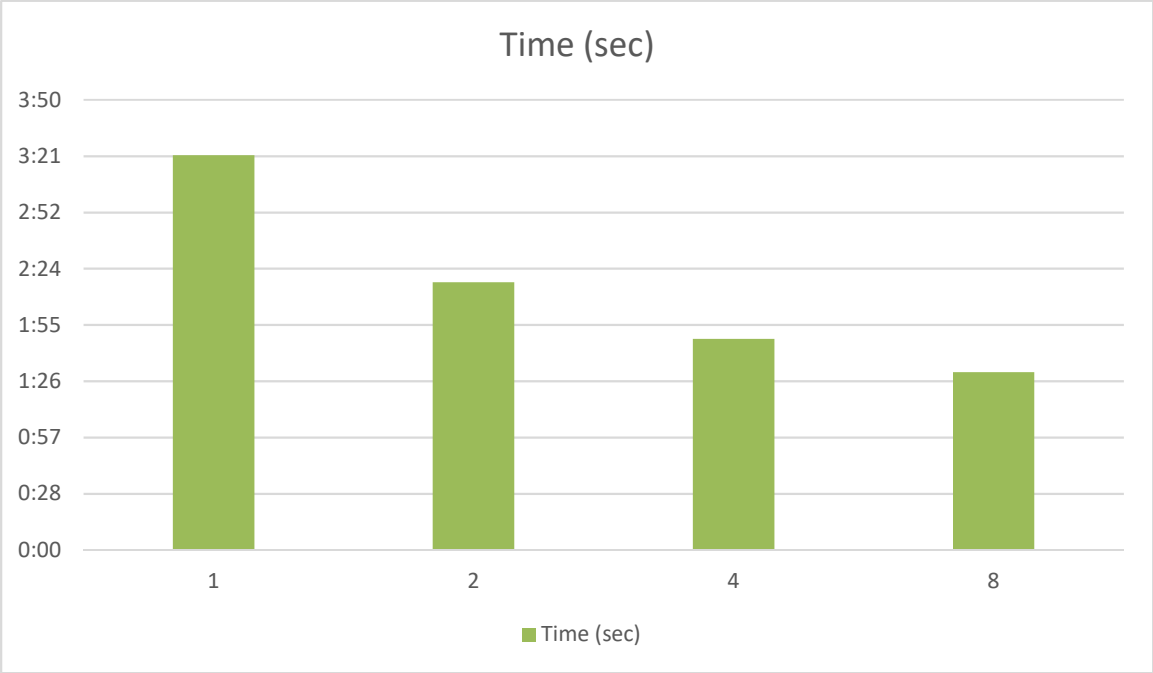


Part b:

Table 2: Execution time of code developed from part b

Reducer(s) Used	Application ID	Time	Avg. Time
1	application_1532471318325_5805	3:24	3:22
	application_1532471318325_5821	3:22	
	application_1532471318325_5825	3:21	
2	application_1532471318325_5827	2:16	2:17
	application_1532471318325_5829	2:17	
	application_1532471318325_5832	2:17	
4	application_1532471318325_5838	1:46	1:48
	application_1532471318325_5841	1:51	
	application_1532471318325_5844	1:46	
8	application_1532471318325_5848	1:34	1:31
	application_1532471318325_5858	1:31	
	application_1532471318325_5859	1:28	

Figure 2.1: Graph representation of table above



Resources Used:

Cluster:

50 Appro 1522H nodes (whale-001 to whale-057), each node with

- two 2.2 GHz quad-core AMD Opteron processor (8 cores total)
- 16 GB main memory
- Gigabit Ehternet
- 4xDDR InfiniBand HCAs (not used at the moment)

Network Interconnect

- 144 port 4xInfiniBand DDR Voltaire Grid Director ISR 2012 switch (donation from TOTAL)
- two 48 port HP GE switch

Storage

- 4 TB NFS /home file system (shared with crill)
- ~7 TB HDFS file system (using triple replication)

Analysis:

After execution of code in cluster and table (and graph) I derived from that, I can surely say that as the number of reducers increases on a particular job, the execution time reduces. This statement is derived from Amdahl's law. This law states that there is (small) sequential fraction, which limits the speedup. There will be a limit on number of reducers, when execution time is not decreasing as it need to be.