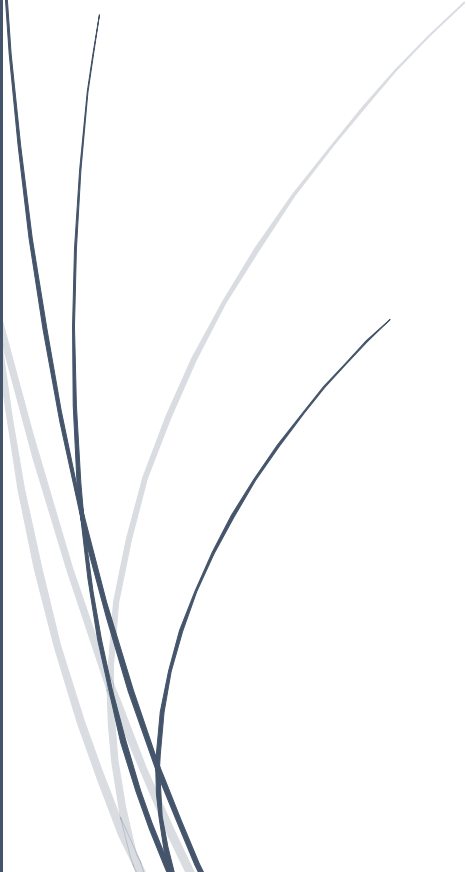


A dark blue vertical bar runs down the left side of the slide. A blue arrow points to the right from this bar, containing the date.

10/21/2016

Hadoop Map-Reduce Analysis

UK Crime Data

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Ravikumar, Roshan
RXR151330

Given

A sanitized crime database from the UK Police.

Objective

The objective of this project is to write a MapReduce program to compute the total crime incidents of each crime type in each region.

Further we are asked to analyse the following from the log files:

- How the files are distributed over the nodes
- How the mappers and reducers tasks are distributed over the nodes
- Analyse the shuffle and sorting phase.
- The execution time of each phase and the total execution time.
- Memory Usage.

Program and Output

The program and the output file generated by the reducer have been uploaded separately.

Hadoop Cluster

The cluster has 4 nodes. One node is the namenode (namenode1) and 3 datanodes namely: datanode1, datanode2, datanode3.

Analysis

1) Metrics and Resource Provisioning Stage

Once the MapReduce job is submitted at a node, Application Master validates the application's specifics. AppMaster Metrics systems starts and create a new job and assigns a job token to it. It calculates the number of splits (number of mappers) and number of reducers required for the job.

The AppMaster was initially started at datanode2.

In this project the metrics were as follows:

- Input size for job job_1476565537266_0034 = 2183794818. Number of splits = 17
 - The input file was 2.18GB. The block size was 128MB. Therefore 17 splits $(2183794818 / (128 * 1024 * 1024))$ were made.
- Number of reduces for job job_1476565537266_0034 = 1

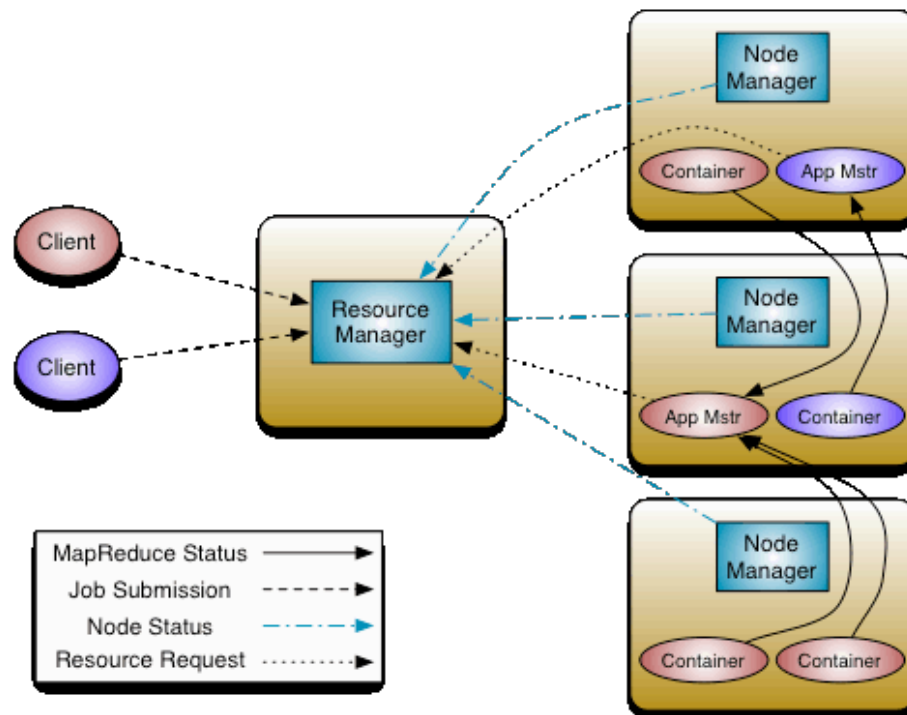
The job now transitioned from NEW to INITED state.

Now the AppMaster contacts the Resource Manager running at namenode1 about this new job.

Resource Manager has two main components:

- Scheduler
- ApplicationsManager

The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The scheduler is also responsible for assigning containers for the individual Tasks. Container incorporates elements such as memory, cpu, disk, network etc.



After receiving the required resources and containers from the resource manager the job transitions from INITED to SETUP state. Immediately the job transitions to RUNNING state.

The job now releases new mapper tasks which start out in the NEW state. These tasks then transition to SCHEDULED state.

Each mapper task has task_attempts. The containers obtained from the resource manager are allocated to each task_attempt.

2) Mapper Task Attempts Phase at each Datanode

Each task_attempt is assigned to a datanode. The RackResolver finds out to which rack the datanode belongs to.

In this case we have only one rack (default-rack):

org.apache.hadoop.yarn.util.RackResolver: Resolved datanode2 to /default-rack

org.apache.hadoop.yarn.util.RackResolver: Resolved datanode1 to /default-rack

..... This will happen for each task.

At each datanode the task_attempts transition from NEW to UNASSIGNED and from UNASSIGNED to ASSIGNED state. In the ASSIGNED state a container with a container ID is allocated to the attempt and the task_attempt starts RUNNING. After the task_attempt completes running in the JVM of a datanode, the container is cleaned up and killed by the Application Master. The task attempt finally transitions to SUCCEEDED state.

3) Splits processed by each Datanode

Each data node gets 128MB splits to process. This can be observed from the logs at respective datanodes.

Each split is of 134217728 bytes in length. This translates to 128MB in size, which is same as the block size.

Datanode1

```
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:1879048192+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:2013265920+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:2147483648+36311170
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:536870912+134217728
```

Datanode2

```
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:1073741824+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:1207959552+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:1342177280+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:1476395008+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:1610612736+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:1744830464+134217728
```

Datanode3

```
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:0+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:134217728+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:268435456+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:402653184+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:536870912+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:671088640+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:805306368+134217728
Processing split: hdfs://10.176.128.92:8020/roshan_inp/csvFinal.csv:939524096+134217728
```

4) Output Collection of Mapper Output

OutputCollector is a generalization of the facility provided by the MapReduce framework to collect data output by the Mapper or the Reducer (either the intermediate outputs or the output of the job). The output of mapper tasks is collected by output collector and fed to the shuffle and sort phase. The disk spills are also taken care by Output Collector.

```
2016-10-20 21:41:21,982 INFO [main] org.apache.hadoop.mapred.MapTask: Map output collector class =
org.apache.hadoop.mapred.MapTask$MapOutputBuffer
2016-10-20 21:41:25,750 INFO [main] org.apache.hadoop.mapred.MapTask: Starting flush of map output
2016-10-20 21:41:25,750 INFO [main] org.apache.hadoop.mapred.MapTask: Spilling map output
2016-10-20 21:41:25,750 INFO [main] org.apache.hadoop.mapred.MapTask: bufstart = 0; bufend = 24344172; bufvoid = 104857600
2016-10-20 21:41:25,750 INFO [main] org.apache.hadoop.mapred.MapTask: kvstart = 26214396(104857584); kvend = 22879308(91517232);
length = 3335089/6553600
2016-10-20 21:41:27,045 INFO [main] org.apache.hadoop.mapred.MapTask: Finished spill 0
2016-10-20 21:41:27,134 INFO [main] org.apache.hadoop.mapred.Task: Task:attempt_1476565537266_0034_m_000014_0 is done. And is in
the process of committing
2016-10-20 21:41:27,182 INFO [main] org.apache.hadoop.mapred.Task: Task 'attempt_1476565537266_0034_m_000014_0' done.
```

In this case there was no spill as shown by “Finished spill 0”. This shows the output collection one mapper task. This might vary for other mapper tasks.

5) Shuffle Phase

After the map attempt completes, the output of each map task attempt goes through a shuffle and sort phase. Below, a sample log is shown indicating the output of map tasks entering the shuffle stage.

```
2016-10-20 21:41:29,964 INFO [fetcher#5] org.apache.hadoop.mapreduce.task.reduce.Fetcher: for url=13562/mapOutput?job=job_1476565537266_0034&reduce=0&map=attempt_1476565537266_0034_m_000016_0,attempt_1476565537266_0034_m_000015_0,attempt_1476565537266_0034_m_000014_0 sent hash and received reply
```

```
2016-10-20 21:41:29,967 INFO [fetcher#5] org.apache.hadoop.mapreduce.task.reduce.Fetcher: fetcher#5 about to shuffle output of map attempt_1476565537266_0034_m_000016_0 decomp: 1977176 len: 1977180 to MEMORY
```

```
2016-10-20 21:41:29,970 INFO [fetcher#5] org.apache.hadoop.mapreduce.task.reduce.InMemoryMapOutput: Read 1977176 bytes from map-output for attempt_1476565537266_0034_m_000016_0
```

```
2016-10-20 21:41:29,971 INFO [fetcher#5] org.apache.hadoop.mapreduce.task.reduce.MergeManagerImpl: closeInMemoryFile -> map-output of size: 1977176, inMemoryMapOutputs.size() -> 1, commitMemory -> 0, usedMemory ->1977176
```

The above log shows us that an output of size 1977180 bytes from the mapper attempt attempt_1476565537266_0034_m_000016_0 entering shuffle stage. The same will happen for the outputs of all other map tasks. The size in bytes might vary based on the number of keys assigned to each mapper.

6) Reducer Phase

Shuffle is where the data is collected by the reducer from each mapper. This can happen while mappers are generating data since it is only a data transfer.

On the other hand, sort and reduce can only start once all the mappers are done.

Below was the output of the map reduce application:

```
16/10/20 21:41:33 INFO mapreduce.Job: map 0% reduce 0%
16/10/20 21:41:43 INFO mapreduce.Job: map 6% reduce 0%
16/10/20 21:41:45 INFO mapreduce.Job: map 18% reduce 0%
16/10/20 21:41:48 INFO mapreduce.Job: map 19% reduce 0%
16/10/20 21:41:49 INFO mapreduce.Job: map 20% reduce 0%
16/10/20 21:41:51 INFO mapreduce.Job: map 21% reduce 0%
16/10/20 21:41:52 INFO mapreduce.Job: map 25% reduce 0%
16/10/20 21:41:53 INFO mapreduce.Job: map 28% reduce 0%
16/10/20 21:41:54 INFO mapreduce.Job: map 30% reduce 6%
16/10/20 21:41:55 INFO mapreduce.Job: map 34% reduce 6%
16/10/20 21:41:56 INFO mapreduce.Job: map 38% reduce 6%
16/10/20 21:41:57 INFO mapreduce.Job: map 42% reduce 6%
16/10/20 21:41:58 INFO mapreduce.Job: map 45% reduce 6%
16/10/20 21:42:00 INFO mapreduce.Job: map 48% reduce 6%
16/10/20 21:42:02 INFO mapreduce.Job: map 56% reduce 6%
16/10/20 21:42:05 INFO mapreduce.Job: map 66% reduce 6%
16/10/20 21:42:06 INFO mapreduce.Job: map 76% reduce 6%
16/10/20 21:42:07 INFO mapreduce.Job: map 78% reduce 6%
16/10/20 21:42:08 INFO mapreduce.Job: map 88% reduce 6%
16/10/20 21:42:09 INFO mapreduce.Job: map 88% reduce 20%
16/10/20 21:42:10 INFO mapreduce.Job: map 90% reduce 20%
16/10/20 21:42:11 INFO mapreduce.Job: map 94% reduce 20%
16/10/20 21:42:12 INFO mapreduce.Job: map 100% reduce 25%
16/10/20 21:42:15 INFO mapreduce.Job: map 100% reduce 67%
16/10/20 21:42:17 INFO mapreduce.Job: map 100% reduce 100%
16/10/20 21:42:17 INFO mapreduce.Job: Job job_1476565537266_0034 completed successfully
```

Generally, one can tell which phase MapReduce is in by looking at the reducer completion percentage: 0-33% means its doing shuffle, 34-66% is sort, 67%-100% is

reduce. In the above case reducer is still at 25% when mapper has completed 100%. This shows that sorting and reduce phase is waiting for all mappers to complete.

Below are some logs showing the reducers receiving the completed map jobs to sort and reduce.

```
org.apache.hadoop.mapreduce.task.reduce.EventFetcher: attempt_1476565537266_0034_r_000000_0 Thread started: EventFetcher for fetching Map Completion Events
org.apache.hadoop.mapreduce.task.reduce.EventFetcher: attempt_1476565537266_0034_r_000000_0: Got 3 new map-outputs
org.apache.hadoop.mapreduce.task.reduce.EventFetcher: attempt_1476565537266_0034_r_000000_0: Got 6 new map-outputs
org.apache.hadoop.mapreduce.task.reduce.EventFetcher: attempt_1476565537266_0034_r_000000_0: Got 1 new map-outputs
org.apache.hadoop.mapreduce.task.reduce.EventFetcher: attempt_1476565537266_0034_r_000000_0: Got 1 new map-outputs
org.apache.hadoop.mapreduce.task.reduce.EventFetcher: attempt_1476565537266_0034_r_000000_0: Got 2 new map-outputs
org.apache.hadoop.mapreduce.task.reduce.EventFetcher: attempt_1476565537266_0034_r_000000_0: Got 3 new map-outputs
```

After the reducer completes, SUCCESS and part-r-* files are written to the specified output directory in HDFS.

7) Performance

Total time spent by all maps in occupied slots (ms)=510205
Total time spent by all reduces in occupied slots (ms)=31011
Total time spent by all map tasks (ms)=510205
Total time spent by all reduce tasks (ms)=31011

CPU time spent (ms)=92770
Physical memory (bytes) snapshot=5055033344
Virtual memory (bytes) snapshot=16093519872
Total committed heap usage (bytes)=3720347648

8) Difference between Region Definition 1 and Region Definition 2

Region definition 1 use only the first digit of coordinates to define a region.
Region definition 2 use only the first 3 digits of coordinates to define a region.

If we use region definition 1 we will get lesser number of output keys, since many crime records will be merged into one very large region.

If we use region definition 2 we will get way more output keys, since one region will be small and there will be many region keys in the Map-Reduce output.

Region definition 1 gives 694 output regions to crime type combinations.
Region definition 2 gives 480389 output regions to crime type combinations.

9) Number of Reducers

The number of reducers for a map reduce application can be modified with the following line of code.

```
JobConf.setNumReduceTasks(int)
```

10) Error handling in Map-Reduce

During map phase if one of the mapper comes across a line that is not conforming to the requirements in the code, exception would be thrown.

There are two options:

- 1) We could surround our code in a try-catch block and the exception would be consumed and handled appropriately. But the record would be skipped from our computation.
- 2) Or we can have the exception be thrown. We can decide how many times an exception can be thrown, before the entire job gets stopped is as below.

For Map tasks: mapreduce.map.maxattempts property

For reducer tasks: mapreduce.reduce.maxattempts

The default for both these properties is 4.

11) Steps to run the program

Please find the jars named 'CrimeOne.jar' (For Region Definition 1) and 'Crime.jar'(For Region Definition 2) in the submission folder.

Source code can be viewed in CrimeCount.java and CrimeCountOne.java.

The command to run the map reduce program:

```
hadoop jar Crime.jar /hadoop_inp/ /hadoop_out/
```

```
hadoop jar CrimeOne.jar /hadoop_inp/input.csv /hadoop_out/
```