Big Data Technologies

Agenda

- Hadoop 1.x vs 2.x
- Reading hadoop logs
- Job counters
- MR job configuration
- Uber job
- Input/Output format
- Cascaded MR jobs
- ~~MR execution with single reducer~~
- ~~Combiner~~
- ~~Partitioner~~
- ~~MR execution with multiple reducer~~
- ~~Map-only job~~
- ~~Reduce-only job~~
- ~~MR execution on YARN~~
- ~~Hadoop Performance~~
- ~~Hadoop streaming~~

Code Sharing

- http://172.18.4.63:4200

Hadoop 1.x vs 2.x

- Hadoop 1.x -- HDFS + MapReduce
  - HDFS: NameNode, DataNode, SecondaryNameNode
  - MapReduce: JobTracker, TaskTracker
    - JobTracker -- Keep track of progress of whole job (mapper(s) + reducer(s)).

        - One JobTracker in the cluster (Master).
    - TaskTracker -- Keep track of progress of individual tasks i.e. mapper/reducer.
        - Multiple TaskTrackers in the cluster (Workers).
    - MR job/processing is equally distributed across the cluster/TaskTrackers by the Master/JobTracker.
- Hadoop 2.x -- HDFS + MapReduce on YARN
    - HDFS: NameNode, DataNode, SecondaryNameNode (and/or StandbyNameNode, JournalNode - HA)
    - YARN: ResourceManager, NodeManager
        - ResourceManager -- Keep track of all computing resources in the cluster.
            - One ResourceManager in the cluster (Master) -- Multiple ResourceManagers can be configured for HA.
            - NodeManager -- Keep track of execution of tasks on individual worker node.
                - Multiple NodeManagers in the cluster (Workers).
            - MR job/processing is distributed across the cluster depending on availability of the resources.
        - Can use one of the following scheduler:
            - FIFO: Executes only one job at a time. Jobs executed in order of arrival.
            - Capacity: Can execute multiple jobs in parallel depending on availability of the computing resources.
            - Fair Scheduler: Can execute multiple jobs in parallel. If few resources are available, then schedule small jobs there.

Ncdc Job execution

- terminal> start-dfs.sh && start-yarn.sh
- terminal> mapred --daemon start historyserver
- browser> http://localhost:19888
- terminal> hadoop fs -mkdir -p /user/nilesh/ncdc/input
- terminal> hadoop fs -put * /user/nilesh/ncdc/input #command to be executed from data/ncdc directory
- terminal> hadoop jar mr3-ncdc-0.0.1-SNAPSHOT.jar /user/nilesh/ncdc/input /user/nilesh/ncdc/output1
- terminal> hadoop fs -ls /user/nilesh/ncdc/output1
- terminal> hadoop fs -head /user/nilesh/ncdc/output1/part-r-00000

Read hadoop logs

- Method 1: Access the worker nodes (over ssh/direct terminal)
    - $HADOOP_HOME/logs

        - Hadoop Daemon logs
        - --> userlogs --> Hadoop MR application logs
            - --> application directory
                - --> one directory for one task/container -- MrAppMaster, Mapper(s), Reducer(s).
- Method 2: Job historyserver
    - http://localhost:19888
    - Tools --> Local logs
        - Hadoop Daemon logs
        - --> userlogs --> Hadoop MR application logs
            - --> application directory
                - --> one directory for one task/container -- MrAppMaster, Mapper(s), Reducer(s).

Job Counters

- Hadoop maintains various system/pre-defined counters while executing MR jobs.
    - File System Counters
    - Job Counters -- Number of Mappers/Reducers, ...
    - Map-Reduce Framework -- Number of Mapper Input records, ...
    - Shuffle Errors
    - File Input/Output Format Counters
- To add custom job counter, create an enum.

```
enum NcdcJobCounter {
    VALID_READING, INVALID_READING, INVALID_RECORD
}
```

- To increment the counter

```
Counter cntr = context.getCounter(NcdcJobCounter.VALID_READING);
cntr.increment(1);
```

- See output after executing job on terminal or in job historyserver.

MR job configuration

- MR job configuration can be given while executing MR on command line.
  - hadoop jar app.jar -fs hdfs://master:9000 -jt master:8032 -D dfs.replication=2 arg1 arg2 ...
- However it is common practice to put all required settings in a XML file and specify it using -conf generic option.
  - hadoop jar app.jar -conf job-config.xml arg1 arg2 ...
- Typically we can maintain different config files for executing same program in different environments.
  - local-config.xml -- execute on single machine (no HDFS, no YARN).
    - No need to start HDFS or YARN.
    - hadoop jar app.jar -conf job-config.xml /home/$USER/bigdata/data/ncdc /tmp/output1
    - ls /tmp/output1
  - pseudo-config.xml -- execute on single node hadoop cluster.
  - cluster-config.xml -- execute on multi node hadoop cluster.
- Usually config includes performance tunning settings for the job e.g. mapper output buffer size, number of reducers, jvm memory size, etc.

Uber job

- By default when a MR job executes on cluster, at least three containers/JVM processes are created i.e. MrAppMaster, Mapper, and Reducer.
- If job processing/computing is quite "small", we can execute all these tasks (MrAppMaster, Mappers, Reducers) in single JVM process/container, so that there is no IPC needed and execution is much faster. This will also save computing resources. Such exection is called as "uber" job.
- The small job can be defined in the config. The default config

```
<property>
    <name>mapreduce.job.ubertask.enable</name>
    <value>false</value>
</property>
<property>
    <name>mapreduce.job.ubertask.maxmaps</name>
    <value>9</value>
</property>
```

```
<property>
    <name>mapreduce.job.ubertask.maxreduces</name>
    <value>1</value>
</property>
```

- To enable uberization of task, we should modify the above config as per our requirement. Also programmer should keep computing resources of cluster in mind and configure them too if needed.
- Refer uber-config.xml.

```
<property>
    <name>mapreduce.job.ubertask.enable</name>
    <value>true</value>
</property>
```

- terminal> hadoop jar mr3-ncdc-0.0.1-SNAPSHOT.jar -conf ~/dbda/bigdata/day06/mr-config/uber-config.xml /user/nilesh/ncdc/input /user/nilesh/ncdc/output5

Input/Output format

- InputFormat
  - Used to read input data record by record from HDFS.
  - To read each record it internally uses "RecordReader" which is a nested class of InputFormat.
  - TextInputFormat and its RecordReader -- each record is one line (ended by '\n' or '\r').
    - Key=LongWritable -- line offset
    - Value=Text -- line
  - NLineInputFormat and its RecordReader -- each split is "n" lines.
    - Below example: one record/split = 3 lines.

```
name: Nilesh Ghule
email: nilesh@sunbeaminfo.com
```

```
mobile: 9527331338
name: Nitin Kudale
email: nitin@sunbeaminfo.com
mobile: 9881208115
name: Prashat Lad
email: prashant@sunbeaminfo.com
mobile: 9881208114
```

- KeyValueTextInputFormat and its RecordReader -- each record is one line with key and value separated by tab.
  - Key=Text before tab(separator)
  - Value=Text after tab(separator)

```
1   -74.49494451294697
2   -75.16562866684718
3   -44.24810668422786
4   9.304906241972771
5   67.9370272936534
6   126.61202558635395
7   160.3485770685968
8   137.20253584155134
9   91.14684224205146
10  36.454816285998014
11  -15.835389526356655
12  -56.066257566962435
```

- CombineTextInputFormat & its RecordReader -- treat multiple files as a single split (if size is smaller than block size).
  - e.g. If 20 files with total data size < 100mb, will be treated as single input split. So will create single mapper and execution will be faster.
- FixedLengthInputFormat & its RecordReader -- treat "n" bytes as one record.
  - Useful for text/binary file where each record is fixed number of bytes.
- SequenceFileInputFormat & its RecordReader -- used to read sequnce files.
  - These files have a definate structure. They are binary and can be compressed.
- DBInputFormat & its RecordReader -- used to read records from RDBMS.

- OutputFormat
  - Used to write output data record by record into HDFS.
  - To wriyte each record it internally uses "RecordWriter" which is a nested class of OutputFormat.
  - TextOutputFormat
    - each record: key "tab" value "\n"
  - SequenceFileOutputFormat
    - To write records in binary and compressed (optional) format
  - DBOutputFormat
    - To write output records into RDBMS.

Max Avg Temperature NCDC

- Input: Monthwise average Temperature. (Output of Avg Temperature job)

```
1   -74.49494451294697
2   -75.16562866684718
3   -44.24810668422786
4   9.304906241972771
5   67.9370272936534
6   126.61202558635395
7   160.3485770685968
8   137.20253584155134
9   91.14684224205146
10  36.454816285998014
11  -15.835389526356655
12  -56.066257566962435
```

- Output: Maximum Temperature and its Month

```
7   160.3485770685968
```

- Mapper:
  - Code/Logic

```
// if using KeyValueTextInputFormat -- Mapper input will be Text key, Text value
class MaxTemperatureMapper extends Mapper<Text,Text,NullWritable,Text> {
    @Override
    public void map(Text key, Text value, Mapper.Context context) {
        int month = Integer.parseInt(key.toString());
        double temperature = Double.parseDouble(value.toString());
        String monthTemperature = month + "," + temperature;
        context.write(NullWritable.get(),
                      new Text(monthTemperature));
    }
}
```

  - Mapper output

```
x    1,-74.49494451294697
x    2,-75.16562866684718
x    3,-44.24810668422786
x    4,9.304906241972771
x    5,67.9370272936534
x    6,126.61202558635395
x    7,160.3485770685968
x    8,137.20253584155134
x    9,91.14684224205146
x    10,36.454816285998014
x    11,-15.835389526356655
x    12,-56.066257566962435
```

- Reducer:

- Code/Logic

```
class MaxTemperatureReducer extends Reducer<NullWritable,Text,IntWritable,DoubleWritable> {
    @Override
    public void reduce(NullWritable key, Iterable<Text> values, Reducer.Context context) {
        double maxTemperature = -Double.MIN_VALUE;
        int maxMonth = 0;
        for(Text monthTemperatureWr: values) {
            String monthTemperature = monthTemperatureWr.toString();
            String[] parts = monthTemperature.split(",");
            int month = Integer.parseInt(parts[0]);
            double temperature = Double.parseDouble(parts[1]);
            if(temperature > maxTemperature) {
                maxTemperature = temperature;
                maxMonth = month;
            }
        }
        context.write(new IntWritable(maxMonth), new DoubleWritable(maxTemperature));
    }
}
```

- Output

```
7   160.3485770685968
```

Cascaded MR Job

- Input --> Job1 --> Aux Output --> Job2 --> Output
- Driver code needs to be modified creating and submitting second job.
- Second job must be submitted only if first job is successful.
- Typically, input directory of Second job is same as output directory of First job.

Hadoop Writables

- NullWritable -- for keeping key or value null.
- To send multiple fields in "key" or "value", we can do one of the following.
  - option 1: Use "Text" that keeps multiple values concatenated and separated using some symbol (e.g. ",").
  - option 2: To send multiple values of the same type, use ArrayWritable.
  - option 3: To send two value of different type, use MapWritable.
  - option 4: To send multiple value as "value", write custom writable class.

```
class CustomWritable extends Writable {
    private int month;
    private double temperature;
    // constructors
    // getters/setters
    public void write(DataOutput dout) {
        dout.write(month);
        dout.write(temperature);
    }
    public void readFields(DataInput din) {
        this.month = din.readInt();
        this.temperature = din.readDouble();
    }
}
```

  - option 5: To send multiple value as "key", write custom writable class.

```
class CustomWritable extends WritableComparable<CustomWritable> {
    private int month;
    private double temperature;
    // constructors
    // getters/setters
    public void write(DataOutput dout) {
```

```
            dout.write(month);
            dout.write(temperature);
        }
        public void readFields(DataInput din) {
            this.month = din.readInt();
            this.temperature = din.readDouble();
        }
        public int compareTo(CustomWritable other) {
            // ...
        }
        // equals() and hashCode()
    }
```

Assignment

- NCDC Data --> Job1 --> Yearly Avg Temperature --> Job2 --> Max & Min Temperature & Year.

Java Reflection Tutorial

- ClassName.class
- https://youtu.be/lAoNJ_7LD44