Big Data Technologies

Agenda

- Commissioning/Decommissioning
- Map-Reduce
- Map-Reduce Programming

Commissioning/Decommissioning

- Adding new nodes into Hadoop cluster (while cluster is still running) is called as "Commissioning".
- Removing existing nodes from Hadoop cluster (while cluster is still running) is called as "Decommissioning".

Map-Reduce

- Deptwise total salary
- HDFS --> record=line --> Mapper --> (key=deptno,value=sal) --> Sort (by key) --> Grouping (by key) --> (key=deptno, values=[sal, sal, sal, ...]) --> Reducer --> (key=deptno,value=total_sal) --> HDFS
- Input file: emp.csv
    - empno,ename,job,mgr,hire,sal,comm,deptno
- Mapper
    - Input: Key=Line offset (long), Value=One record=One line (String)
    - Output: Key=Deptno (int), Value=Sal (double)
    - Steps:
        - Split line by comma.
        - Key=(int)parts[7], Value=(double)parts[5]
- Reducer
    - Input: Key=deptno (int), Values=[sal, sal, sal, ...] (Iterable[double])
    - Output: Key=deptno (int), Values=total_sal (double)
    - Steps:
        - Sum all sal values.

    - Key=(int) deptno, Value=(double)total_sal

Hadoop IO types

- Hadoop doesn't use serialization (ObjectOutputStream/ObjectInputStream) while sending data across the nodes (like mapper to reducer), because it consumes more bandwidth (includes data and metadata both).
- Hadoop created special IO types optimised for sending data over the network. It internally uses DataOutputStream/DataInputStream to transfer the data.
- Hadoop IO types
    - byte -- ByteWritable, short -- ShortWritable, int -- IntWritable, long -- LongWritable
    - float -- FloatWriable, double -- DoubleWritable
    - boolean -- BooleanWritable, null -- NullWritable
    - String -- Text
    - Array -- ArrayWritable, Map -- MapWritable
- All these IO classes inherited from org.apache.hadoop.io.Writable interface.
    - void write(DataOutput out);
    - void readFields(DataInput in)
- Almost all IO classes has set() and get() method to access primitive data value wrapped in it.

Hadoop Map-Reduce -- Demo 01

- class EmpMapper extends `Mapper<LongWritable,Text,IntWritable,DoubleWritable>` -- @Override map()
    - Input: Key=Line offset (LongWritable), Value=One record=One line (Text)
    - Output: Key=Deptno (IntWritable), Value=Sal (DoubleWritable)
    - Steps: void map(LongWritable key, Text value, Mapper.Context ctx)
        - Get the String line from the Value (Text).
        - Split line by comma.
        - Key=(IntWritable)parts[7], Value=(DoubleWritable)parts[5]
- class EmpReducer extends `Reducer<IntWritable,DoubleWritable,IntWritable,DoubleWritable>` -- @Override reduce()
    - Input: Key=deptno (IntWritable), Values=[sal, sal, sal, ...] (Iterable[DoubleWritable])
    - Output: Key=deptno (IntWritable), Values=total_sal (DoubleWritable)
    - Steps: void reduce(IntWritable key, `Iterable<DoubleWritable>` values, Reducer.Context ctx)

        - Get double sal from DoubleWritable Iterable.
        - Sum sal values.
        - Key=(IntWritable) deptno, Value=(DoubleWritable)total_sal
- Execution (Non-Runnable Jar):
    - Ensure that HDFS and YARN both are started.
    - terminal> jps
    - terminal> hadoop fs -mkdir -p /user/nilesh/emp/input
    - terminal> hadoop fs -put emp.csv /user/nilesh/emp/input
    - Eclipse -- Project -- Run as -- Maven build -- Goals: package -- Ok
    - terminal> hadoop jar mr1-emp-deptsaltotal-0.0.1-SNAPSHOT.jar com.sunbeam.EmpDriver
- Execution (Runnable Jar):
    - pom.xml -- add maven-jar-plugin with your main class.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>2.4</version>
            <configuration>
                <archive>
                    <index>true</index>
                    <manifest>
                        <mainClass>com.sunbeam.EmpDriver</mainClass>
                    </manifest>
                </archive>
            </configuration>
        </plugin>
    </plugins>
</build>
```

    - Eclipse -- Project -- Run as -- Maven build -- Goals: package -- Ok

    - terminal> hadoop jar mr1-emp-deptsaltotal-0.0.1-SNAPSHOT.jar

Hadoop Map-Reduce -- Demo 02

- Input: emp10.csv, emp20.csv, emp30.csv --> HDFS /user/nilesh/deptemp/input
    - terminal> hadoop fs -mkdir -p /user/nilesh/deptemp/input
    - terminal> hadoop fs -put data/emp*0.csv /user/nilesh/deptemp/input
    - terminal> hadoop fs -ls /user/nilesh/deptemp/input
- Requirement: Job wise average salary.
- EmpMapper extends `Mapper<LongWritable,Text,Text,DoubleWritable>`
    - Input: Key=Line offset (LongWritable), Value=One record=One line (Text)
    - Output: Key=Job (Text), Value=Sal (DoubleWritable)
- EmpReducer extends `Reducer<Text,DoubleWritable,Text,DoubleWritable>`
    - Input: Key=Job (Text), Values=[sal,sal,...] (`Iterable<DoubleWritable>`)
    - Output: Key=Job (Text), Value=Avg Sal (DoubleWritable)
- EmpDriver
- Execution (Runnable Jar):
    - Eclipse -- Project -- Run as -- Maven build -- Goals: package -- Ok
    - terminal> hadoop jar mr2-emp-jobsalavg-0.0.1-SNAPSHOT.jar /user/nilesh/deptemp/input /user/nilesh/deptemp/output1
    - terminal> hadoop fs -head /user/nilesh/deptemp/output1/part-r-00000

GenericOptionsParser class

- If hadoop job is executed with command line args, then main() method "args" may have generic options + user defined args.
- GenericOptionsParser is used to handle generic options (e.g. -fs, -jt, -files, -conf, etc.) and create Configuration object.

```
GenericOptionsParser parser = new GenericOptionsParser(args);
Configuration conf = parser.getConfiguration();
```

Configured class

- Typically our driver class is inherited from Configured class.
- This class provides (inherits) two methods -- setConf() and getConf().
- Due to this we can associate a Configuration with our driver object.

```
class EmpDriver extends Configured ... {
    // ...
}
```

Tool interface and ToolRunner class

- Tool interface is standard of implementing hadoop processing/utilities.
- All hadoop processing (like HDFS file access, map-reduce task submit, etc) must be done in implemented run() method.

```
class EmpDriver extends Configured implements Tool {
    // ...

    public void run(String[] args) throws Exception {
        Configuration conf = this.getConf();
        // hadoop processing
    }
}
```

- Such tool must be executed using ToolRunner class of hadoop.

```
EmpDriver driver = new EmpDriver();
ret = ToolRunner.run(conf, driver, args);
```

- This ToolRunner.run() method internally
  - driver.setConf(conf);
  - driver.run(userArgs);

Data-Local Map Tasks

- If mapper is executing on the same data node on which data block is present, then mapper is said to be data local.
- Since no network transfer is invokved, data local mappers executes faster.

Mappers

- Mapper process individual records and produce key-value output.
- Number of mappers = Number of input splits
- Number of input splits = Number of HDFS blocks (approx)
- One input split contains multiple logically completed records. If last record in HDFS block is incomplete, then few bytes from next block will be added to complete the input split.

Assignment

- NCDC Assignment
  - Mapper class
    - Input: LongWritable offset, Text line
    - Output: IntWritable month, DoubleWritable temperature

```
int month = Integer.parseInt(line.substring(19, 21));
double temp = Double.parseDouble(line.substring(87, 92));
int quality = Integer.parseInt(line.substring(92, 93));
if((quality == 0 || quality == 1 || quality == 4 || quality == 5 || quality == 9) && ((int)temp !=
9999)) {
```

```
        // ...
        context.Write(monthWr, tempWr);
    }
```

- Reducer class
  - Input: IntWritable month, DoubleWritable temperatures
  - Output: IntWritable month, DoubleWritable avgTemperature