Weather data analysis using MapReduce

Tushar B. Kute,
http://tusharkute.com

What is MapReduce?

- MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.
- MapReduce is a processing technique and a program model for distributed computing based on java.
- The MapReduce algorithm contains two important tasks, namely Map and Reduce.

Map and Reduce

- Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).
- Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

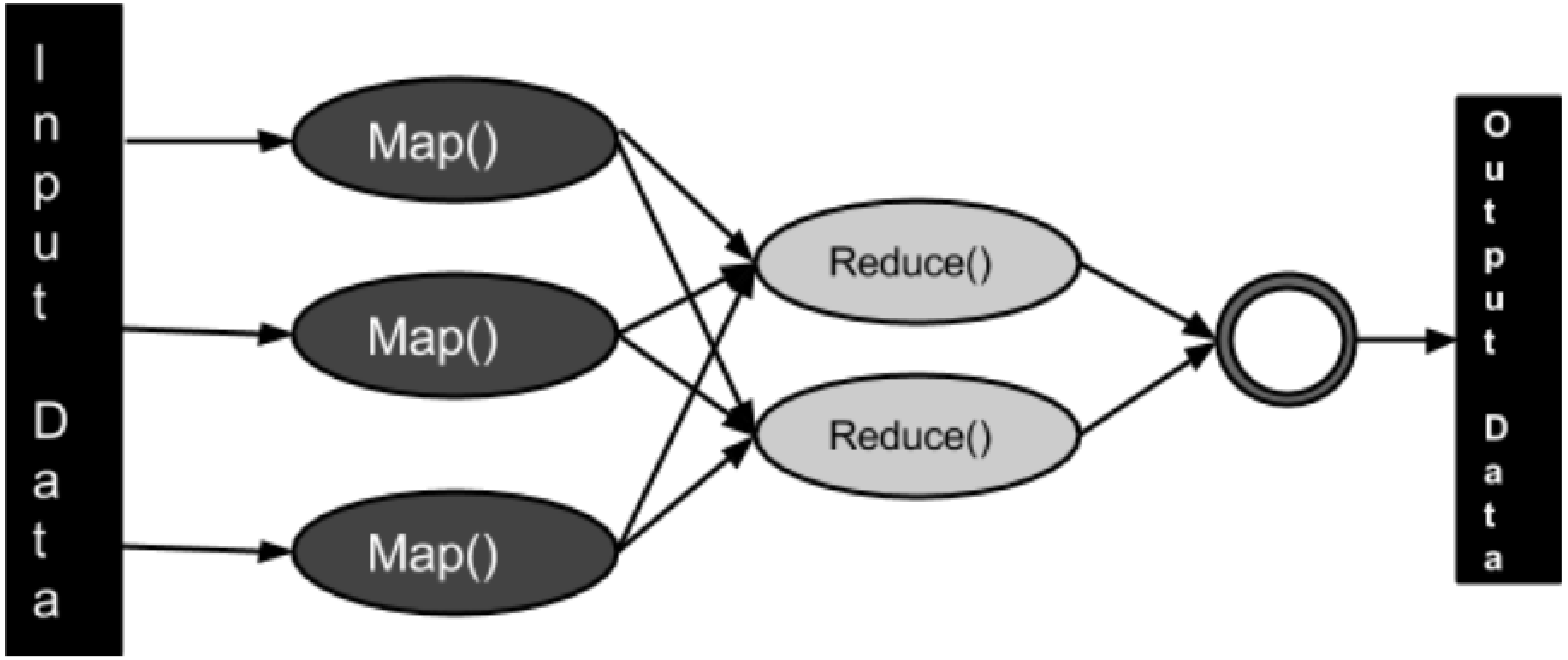
Map and Reduce

- The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes.
- Under the MapReduce model, the data processing primitives are called mappers and reducers.
- Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change.
- This simple scalability is what has attracted many programmers to use the MapReduce model.

The Algorithm

- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
- **Map stage**: The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
- **Reduce stage**: This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

The MapReduce



Inserting Data into HDFS

- The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.
- The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework.
- Input and Output types of a MapReduce job: (Input) <k1,v1> -> map -> <k2, v2>-> reduce -> <k3, v3> (Output).

Data input and output

	Input	Output
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

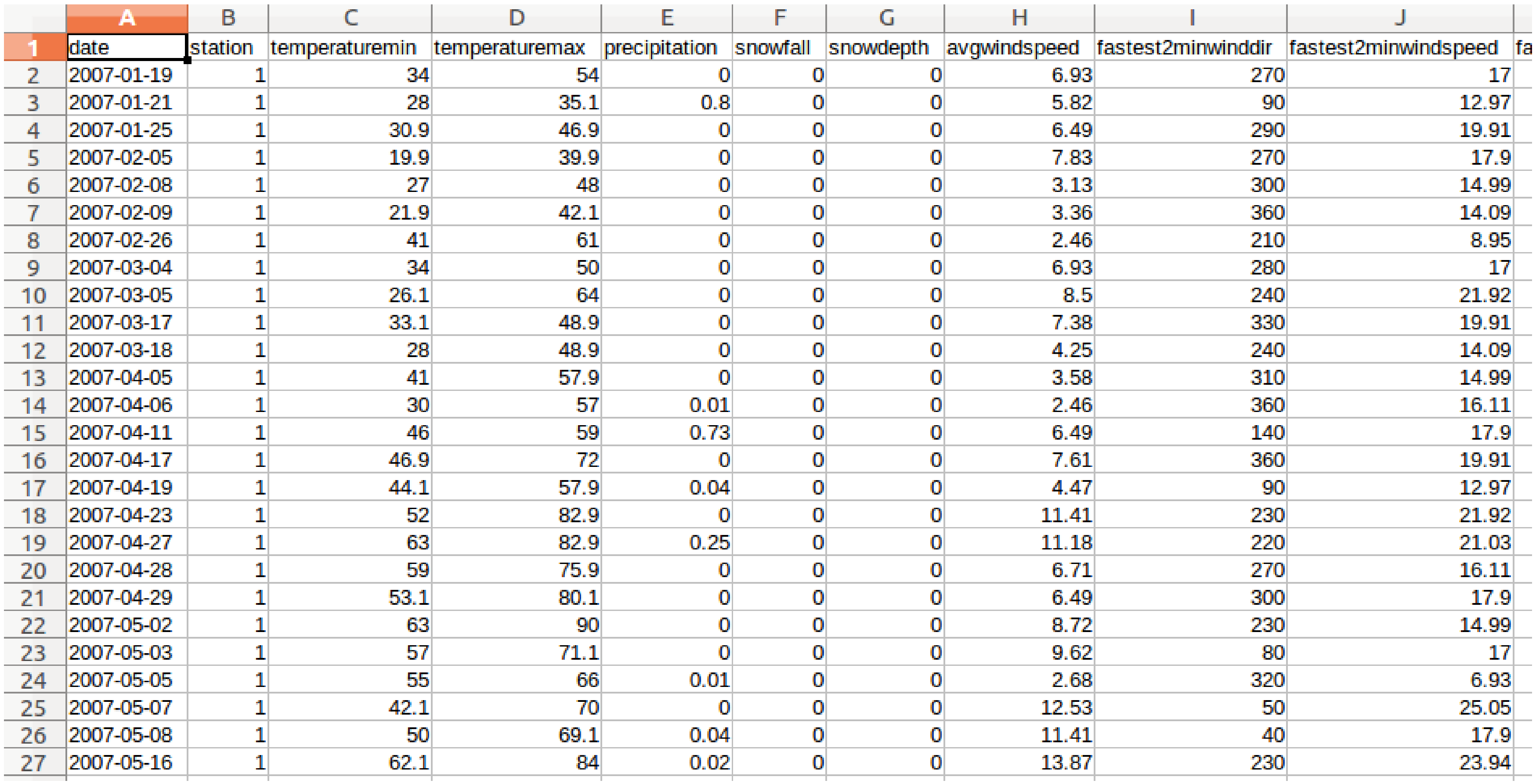
Terminologies

- Mapper - Mapper maps the input key/value pairs to a set of intermediate key/value pair.
- NamedNode - Node that manages the Hadoop Distributed File System (HDFS).
- DataNode - Node where data is presented in advance before any processing takes place.
- MasterNode - Node where JobTracker runs and which accepts job requests from clients.
- SlaveNode - Node where Map and Reduce program runs.
- JobTracker - Schedules jobs and tracks the assign jobs to Task tracker.
- Task Tracker - Tracks the task and reports status to JobTracker.
- Job - A program is an execution of a Mapper and Reducer across a dataset.
- Task - An execution of a Mapper or a Reducer on a slice of data.

Example:

- Write a program that interact with weather dataset. Find the day and station with maximum snowfall in year 2013.

The dataset: weather.csv



	A	B	C	D	E	F	G	H	I	J	
1	date	station	temperaturemin	temperaturemax	precipitation	snowfall	snowdepth	avgwindspeed	fastest2minwinddir	fastest2minwindspeed	fa
2	2007-01-19	1	34	54	0	0	0	6.93	270	17	
3	2007-01-21	1	28	35.1	0.8	0	0	5.82	90	12.97	
4	2007-01-25	1	30.9	46.9	0	0	0	6.49	290	19.91	
5	2007-02-05	1	19.9	39.9	0	0	0	7.83	270	17.9	
6	2007-02-08	1	27	48	0	0	0	3.13	300	14.99	
7	2007-02-09	1	21.9	42.1	0	0	0	3.36	360	14.09	
8	2007-02-26	1	41	61	0	0	0	2.46	210	8.95	
9	2007-03-04	1	34	50	0	0	0	6.93	280	17	
10	2007-03-05	1	26.1	64	0	0	0	8.5	240	21.92	
11	2007-03-17	1	33.1	48.9	0	0	0	7.38	330	19.91	
12	2007-03-18	1	28	48.9	0	0	0	4.25	240	14.09	
13	2007-04-05	1	41	57.9	0	0	0	3.58	310	14.99	
14	2007-04-06	1	30	57	0.01	0	0	2.46	360	16.11	
15	2007-04-11	1	46	59	0.73	0	0	6.49	140	17.9	
16	2007-04-17	1	46.9	72	0	0	0	7.61	360	19.91	
17	2007-04-19	1	44.1	57.9	0.04	0	0	4.47	90	12.97	
18	2007-04-23	1	52	82.9	0	0	0	11.41	230	21.92	
19	2007-04-27	1	63	82.9	0.25	0	0	11.18	220	21.03	
20	2007-04-28	1	59	75.9	0	0	0	6.71	270	16.11	
21	2007-04-29	1	53.1	80.1	0	0	0	6.49	300	17.9	
22	2007-05-02	1	63	90	0	0	0	8.72	230	14.99	
23	2007-05-03	1	57	71.1	0	0	0	9.62	80	17	
24	2007-05-05	1	55	66	0.01	0	0	2.68	320	6.93	
25	2007-05-07	1	42.1	70	0	0	0	12.53	50	25.05	
26	2007-05-08	1	50	69.1	0.04	0	0	11.41	40	17.9	
27	2007-05-16	1	62.1	84	0.02	0	0	13.87	230	23.94	

Example:

- Snow.java

Compilation and Execution

- Let us assume we are in the home directory of a Hadoop user (e.g. /home/rashmi).
- Follow the steps given below to compile and execute the above program.
- **Step 1**
  - The following command is to create a directory to store the compiled java classes.
  - `$ mkdir snow`

Compilation and Execution

- **Step 2**

  Download hadoop-core-1.2.1.jar, which is used to compile and execute the MapReduce program. Visit the following link

  http://mvnrepository.com/artifact/org.apache.hadoop/hadoop-core/1.2.1
  To download the jar. Let us assume the downloaded folder is /home/rashmi/snow.

- **Step 3**

  The following commands are used for compiling the wordcount.java program and creating a jar for the program.

  ```
  $ javac -classpath hadoop-core-1.2.1.jar snow/Snow.java
  $ jar -cvf  snow.jar   -C   snow/   .
  ```

Compilation and Execution

- **Step 4**
  - The following command is used to create an input directory in HDFS.
  - `$hadoop fs -mkdir /input`
- **Step 5**
  - The following command is used to copy input dataset file on HDFS.
  - `$hadoop fs -put fruits.txt /input`
- **Step 6**
  - The following command is used to verify the files in the input directory.
  - `$hadoop fs -ls /input`

Compilation and Execution

- **Step 7**
  - The following command is used to run the Snow application by taking the input files from the input directory.
  - `$hadoop  jar  snow.jar  Snow  /input  /output`
  - Wait for a while until the file is executed. After execution, the output will contain the number of input splits, the number of Map tasks, the number of reducer tasks, etc. The output directory must *not* be existing already.

Compilation and Execution

- **Step 8**
  - The following command is used to verify the resultant files in the output folder.
  - `$hadoop fs -ls /output`
- **Step 9**
  - The following command is used to see the output in part-r-00000 file. This file is generated by HDFS.
  - `$hadoop fs -cat /output/part-r-00000`

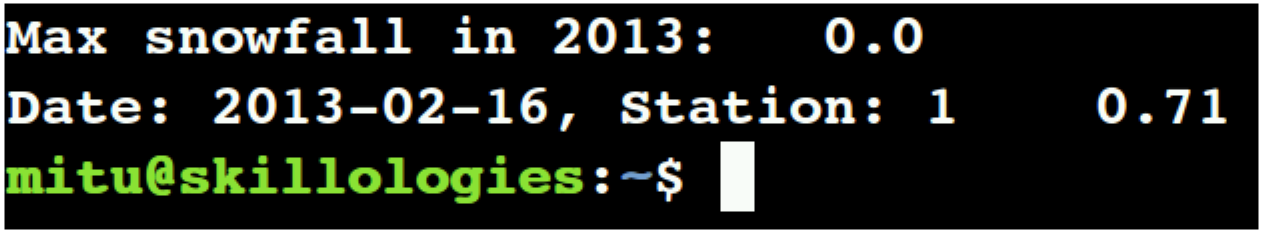
Compilation and Execution

- **Step 10**

  The following command is used to copy the output file from HDFS to the local file system for analyzing.

  - `$hadoop fs -get /output/part-r-00000`

Output:



```
Max snowfall in 2013:    0.0
Date: 2013-02-16, Station: 1     0.71
mitu@skillologies:~$
```

Thank you

@mitu_skillologies

/mITuSkillologies

@mitu_group

Web Resources
http://mitu.co.in
http://tusharkute.com

tushar@tusharkute.com

contact@mitu.co.in