Big data: architectures and data analytics

Hadoop implementation of MapReduce

MapReduce and Hadoop

- Designers/Developers focus on the definition of the Map and Reduce functions (i.e., m and r)
  - No need to manage the distributed execution of the map, shuffle and sort, and reduce phases
- The Hadoop framework coordinates the execution of the MapReduce program
  - Parallel execution of the map and reduce phases
  - Execution of the shuffle and sort phase
  - Scheduling of the subtasks
  - Synchronization

MapReduce programs

- The programming language is Java
- A Hadoop MapReduce program consists of three main parts
  - Driver
  - Mapper
  - Reducer
- Each part is "implemented" by means of a specific class

Terminology

- Driver class
  - The class containing the method/code that coordinates the configuration of the job and the "workflow" of the application
- Mapper class
  - A class "implementing" the map function
- Reducer class
  - A class "implementing" the reduce function
- Driver
  - Instance of the Driver class (i.e., an object)
- Mapper
  - Instance of the Mapper class (i.e., an object)
- Reducer
  - Instance of the Reducer class (i.e., an object)

Terminology

- (Hadoop) Job
  - Execution/run of a MapReduce code over a data set
- Task
  - Execution/run of a Mapper (Map task) or a Reducer (Reduce task) on a slice of data
  - Many tasks for each job
- Input split
  - Fixed-size piece of the input data
    - Usually each split as approximately the same size of a HDFS block/chuck

Driver

- The Driver
  - Is characterized by the main() method, which accepts arguments from the command line
    - i.e., it is the entry point of the application
  - Configures the job
  - Submits the job to the Hadoop Cluster
  - "Coordinates" the work flow of the application
  - Runs on the client machine
    - i.e., it does not run on the cluster

7

Mapper

- The Mapper
  - Is an instance of the Mapper class
  - "Implements" the map phase
  - Is characterized by the map(…) method
    - Processes the (key, value) pairs of the input file and emits (key, value) pairs
  - Runs on the cluster

8

Reducer

- The Reducer
  - Is an instance of the Reduce class
  - "Implements" the reduce phase
  - Is characterized by the reduce(…) method
    - Processes (key, [list of values]) pairs and emits (key, value) pairs
  - Runs on the cluster

9

Hadoop implementation of the MapReduce phases

- Input key-value pairs are read from the HDFS file system
- The map method of the Mapper
  - Is invoked over each input key-value pair
  - Emits a set of intermediate key-value pairs that are stored in the local file system of the computing server (they are not stored in HDFS)
- Intermediate results
  - Are aggregated by means of a shuffle and sort procedure
  - A set of <key, [list of values]> pairs are generated

10

Hadoop implementation of the MapReduce phases

- The reduce method of the Reduder
  - Is applied over each intermediate <key, [list of values]> pair
  - Emits a set of key-value pairs that are stored in HDFS (the final result of the MapReduce application)
- Intermediate key-value pairs are transient:
  - They are not stored on the distributed files system
  - They are stored locally to the node producing or processing them

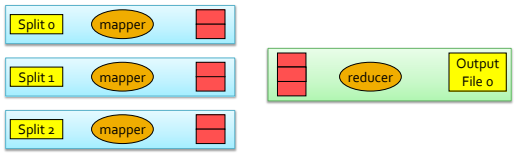
11

Hadoop implementation of the MapReduce phases

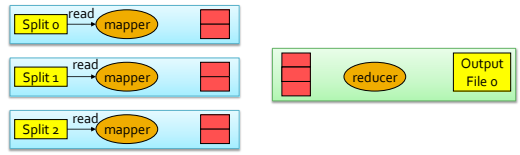
- In order to parallelize the work/the job, Hadoop executes a set of tasks in parallel
  - It instances one Mapper (Task) for each input split
  - And a user-specified number of Reducers
    - Each reducer is associated with a set of keys
      - It receives and processes all the key-value pairs associated with its set of keys
  - Mappers and Reducers are executed on the nodes/servers of the clusters

12

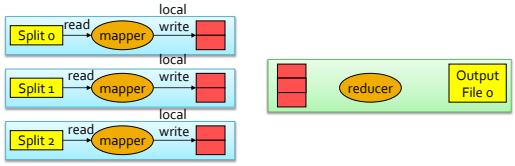
MapReduce data flow with a single reducer



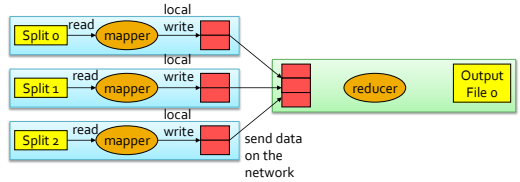
MapReduce data flow with a single reducer



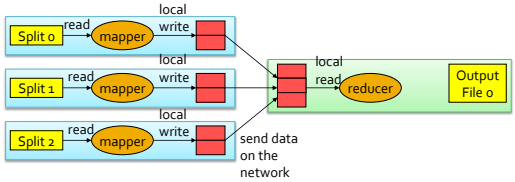
MapReduce data flow with a single reducer



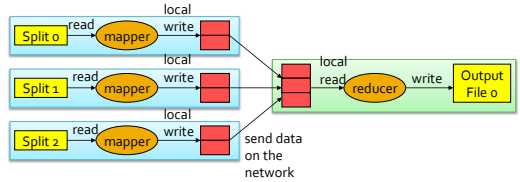
MapReduce data flow with a single reducer



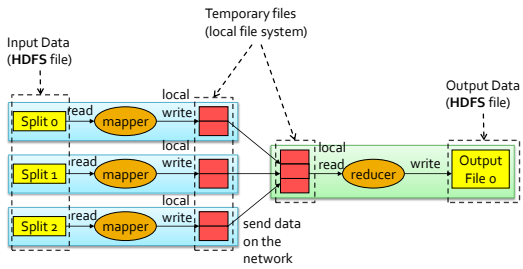
MapReduce data flow with a single reducer



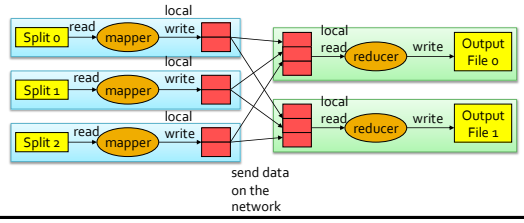
MapReduce data flow with a single reducer



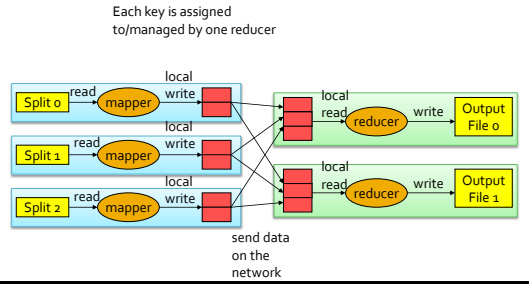
MapReduce data flow with a single reducer



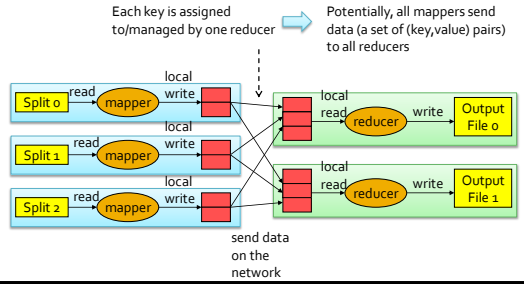
MapReduce data flow with multiple reducers



MapReduce data flow with multiple reducers



MapReduce data flow with multiple reducers



MapReduce programs - Driver

- The Driver class extends the org.apache.hadoop.conf.Configured class and implements the org.apache.hadoop.util.Tool interface
  - You can write a Driver class that does not extend Configured and does not implement Tool
    - However, you need to manage some low level details related to some command line parameters in that case
- The designer/developer implements the main(…) and run(…) methods

MapReduce programs - Driver

- The run(…) method
  - Configures the job
    - Name of the Job
    - Job Input format
    - Job Output format
    - Mapper class
      - Name of the class
      - Type of its input (key, value) pairs
      - Type of its output (key, value) pairs

MapReduce programs - Driver

- Reducer class
  - Name of the class
  - Type of its input (key, value) pairs
  - Type of its output (key, value) pairs
- Number of reducers

25

MapReduce programs - Mapper

- The Mapper class extends the org.apache.hadoop.mapreduce.Mapper class
  - The org.apache.hadoop.mapreduce.Mapper class
    - Is a generic type/generic class
    - With four type parameters: input key type, input value type, output key type, output value type
- The designer/developer implements the map(...) method
  - That is automatically called by the framework for each (key, value) pair of the input file

26

MapReduce programs - Mapper

- The map(...) method
  - Processes its input (key, value) pairs by using standard Java code
  - Emits (key, value) pairs by using the context.write(key, value) method

27

MapReduce programs - Reducer

- The Reducer class extends the org.apache.hadoop.mapreduce.Reducer class
  - The org.apache.hadoop.mapreduce.Reducer class
    - Is a generic type/generic class
    - With four type parameters: input key type, input value type, output key type, output value type
- The designer/developer implements the reduce(...) method
  - That is automatically called by the framework for each (key, [list of values]) pair obtained by aggregating the output of the mapper(s)

28

MapReduce programs - Reducer

- The reduce(...) method
  - Processes its input (key, [list of values]) pairs by using standard Java code
  - Emits (key, value) pairs by using the context.write(key, value) method

29

MapReduce Data Types

- Hadoop has its own basic data types
  - Optimized for network serialization
  - org.apache.hadoop.io.Text: like Java String
  - org.apache.hadoop.io.IntWritable: like Java Integer
  - org.apache.hadoop.io.LongWritable: like Java Long
  - org.apache.hadoop.io.FloatWritable : like Java Float
  - Etc

30

MapReduce Data Types

- The basic Hadoop data types implement the org.apache.hadoop.io.Writable and org.apache.hadoop.io.WritableComparable interfaces
- All classes (data types) used to represent keys are instances of WritableComparable
  - Keys must be "comparable" for supporting the sort and shuffle phase
- All classes (data types) used to represent values are instances of Writable
  - Usually, they are also instances of WritableComparable even if it is not indispensable

31

MapReduce Data Types

- Developers can define new data types by implementing the org.apache.hadoop.io.Writable and/or org.apache.hadoop.io.WritableComparable interfaces
  - It is useful for managing complex data types

32

InputFormat

- The input of the MapReduce program is a HDFS file
- While the input of the a Mapper is a set of (key, value) pairs
- The classes extending the org.apache.hadoop.mapreduce.InputFormat abstract class are used to read the input data and "logically transform" the input HDFS file in a set of (key, value) pairs

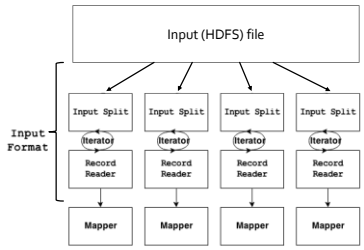
33

InputFormat

- InputFormat "describes" the input-format specification for a MapReduce application and processes the input file(s)
- The InputFormat class is used to
  - Read input data and validate the compliance of the input file with the expected input-format
  - Split the input file(s) into logical Input Splits
    - Each input split is then assigned to an individual Mapper
  - Provide the RecordReader implementation to be used to divide the logical input split in a set of (key,value) pairs (also called records) for the mapper

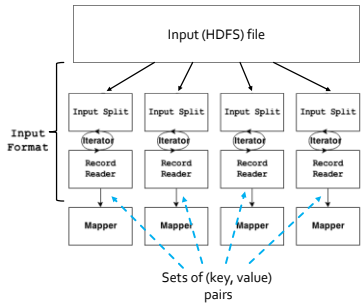
34

Getting Data to the Mapper



35

Getting Data to the Mapper



36

Reading Data

- InputFormat identifies partitions of the data that form an input split
  - Each input split is a (reference to a) part of the input file processed by a single mapper
  - Each split is divided into records, and the mapper processes one record (i.e., a (key,value) pair) at a time

37

InputFormat

- A set of predefined classes extending the InputFormat abstract class are available for standard input file formats
  - TextInputFormat
    - An InputFormat for plain text files
  - KeyValueTextInputFormat
    - Another InputFormat for plain text files
  - SequenceFileInputFormat
    - An InputFormat for sequential/binary files
  - ....

38

TextInputFormat

- TextInputFormat
  - An InputFormat for plain text files
  - Files are broken into lines
    - Either linefeed or carriage-return are used to signal end of line
  - One pair (key, value) is emitted for each line of the file
    - Key is the position (offset) of the line in the file
    - Value is the content of the line

39

TextInputFormat example

Input HDFS file

Toy example file for Hadoop.\n
Hadoop running example.\n
TextinputFormat is used to split data.\n

(key, value) pairs generated by using TextInputFormat

(0, "Toy example file for Hadoop.")
(31, "Hadoop running example.")
(56, "TextinputFormat is used to split data.")

40

KeyValueTextInputFormat

- KeyValueTextInputFormat
  - An InputFormat for plain text files
    - Each line of the file must have the format
      key<separator>value
    - The default separator is tab (\t)
  - Files are broken into lines
    - Either linefeed or carriage-return are used to signal end of line
    - Each line is split into key and value parts by considering the separator symbol/character
  - One pair (key, value) is emitted for each line of the file
    - Key is the text preceding the separator
    - Value is the text following the separator

41

KeyValueTextInputFormat

Input HDFS file

10125\tMister John\n
10236\tMiss Jenny\n
1\tMister Donald Duck\n

(key, value) pairs generated by using KeyValueTextInputFormat

(10125, "Mister John")
(10236, "Miss Jenny")
(1, "Mister Donald Duck")

42

OutputFormat

- The classes extending the org.apache.hadoop.mapreduce.OutputFormat abstract class are used to write the output of the MapReduce program in a HDFS file(s)

43

OutputFormat

- A set of predefined classes extending the OutputFormat abstract class are available for standard output file formats
  - TextOutputFormat
    - An OutputFormat for plain text files
  - SequenceFileOutputFormat
    - An OutputFormat for sequential/binary files
  - ....

44

TextOutputFormat

- TextOutputFormat
  - An OutputFormat for plain text files
  - For each output (key, value) pair TextOutputFormat writes one line in the output file
  - The format of each output line is

    key\tvalue\n

45

Structure of a MapReduce program in Hadoop

46

Basic structure of a MapReduce program - Driver (1)

```
/* Set package */
package it.polito.bigdata.hadoop.mypackage;

/* Import libraries */
import java.io.IOException;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
Import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
.....
```

47

Basic structure of a MapReduce program - Driver (2)

```
/* Driver class */
public class MapReduceAppDriver extends Configured implements
    Tool {
  @Override
  public int run(String[] args) throws Exception {
    /* variables */
    int exitCode;
    .....

    // Parse parameters
    numberOfReducers = Integer.parseInt(args[0]);
    inputPath = new Path(args[1]);
    outputDir = new Path(args[2]);
    ....
```

48

Basic structure of a MapReduce program - Driver (3)

```
// Define and configure a new job
Configuration conf = this.getConf();
Job job = Job.getInstance(conf);

// Assign a name to the job
job.setJobName("My First MapReduce program");
```

49

Basic structure of a MapReduce program - Driver (4)

```
// Set path of the input file/folder (if it is a folder, the job reads all
the files in the specified folder) for this job
FileInputFormat.addInputPath(job, inputPath);

// Set path of the output folder for this job
FileOutputFormat.setOutputPath(job, outputDir);

// Set input format
// TextInputFormat = textual files
job.setInputFormatClass(TextInputFormat.class);

// Set job output format
job.setOutputFormatClass(TextOutputFormat.class);
```

50

Basic structure of a MapReduce program - Driver (5)

```
// Specify the class of the Driver for this job
job.setJarByClass(MapReduceAppDriver .class);

// Set mapper class
job.setMapperClass(MyMapperClass.class);

// Set map output key and value classes
job.setMapOutputKeyClass(output key type.class);
job.setMapOutputValueClass(output value type.class);
```

51

Basic structure of a MapReduce program - Driver (6)

```
// Set reduce class
job.setReducerClass(MyReducerClass.class);

// Set reduce output key and value classes
job.setOutputKeyClass(output key type.class);
job.setOutputValueClass(output value type.class);

// Set number of reducers
job.setNumReduceTasks(numberOfReducers);
```

52

Basic structure of a MapReduce program - Driver (7)

```
    // Execute the job and wait for completion
    if (job.waitForCompletion(true)==true)
        exitCode=0;
    else
        exitCode=1;
    return exitCode;
}   // End of the run method
```

53

Basic structure of a MapReduce program - Driver (8)

```
/* main method of the driver class   */
  public static void main(String args[]) throws Exception {
   /* Exploit the ToolRunner class to "configure" and run the
      Hadoop application */

   int res = ToolRunner.run(new Configuration(),
                            new MapReduceAppDriver(), args);

   System.exit(res);
 } // End of the main method

} // End of public class MapReduceAppDriver
```

54

Basic structure of a MapReduce program - Mapper (1)

```
/* Set package */
package it.polito.bigdata.hadoop.mypackage;

/* Import libraries */
import java.io.IOException;

import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.io.*;
.....
```

55

Basic structure of a MapReduce program - Mapper (2)

```
/* Mapper Class */
class myMapperClass extends Mapper<
            MapperInputKeyType,     // Input key type (must be
  consistent with the InputFormat class specified in the Driver)
            MapperInputValueType,   // Input value type (must be
  consistent with the InputFormat class specified in the Driver)
            MapperOutputKeyType,    // Output key type
            MapperOutputValueType> // Output value type
{
```

56

Basic structure of a MapReduce program - Mapper (3)

```
  /* Implementation of the map method */
  protected void map(
      MapperInputKeyType key,         // Input key
      MapperInputValueType value,     // Input value
      Context context) throws IOException, InterruptedException {

      /* Process the input (key, value) pair and
         emit a set of (key,value) pairs.
         context.write(..) is used to emit (key, value) pairs
         context.write(new outputkey, new outputvalue); */
  } // End of the map method

} // End of class myMapperClass
```

57

Basic structure of a MapReduce program - Reducer (1)

```
/* Set package */
package it.polito.bigdata.hadoop.mypackage;

/* Import libraries */
import java.io.IOException;

import org.apache.hadoop.mapreduce.Reducer;
Import org.apache.hadoop.io.*;
.....
```

58

Basic structure of a MapReduce program - Reducer (2)

```
/* Reducer Class */
class myReducerClass extends Reducer<
            ReducerInputKeyType,    // Input key type (must be
  consistent with the OutputKeyType of the Mapper)
            ReducerInputValueType,  // Input value type (must be
  consistent with the OutputValueType of the Mapper)
            ReducerOutputKeyType,   // Output key type (must be
  consistent with the OutputFormat class specified in the Driver)
            ReducerOutputValueType> // Output value type (must
  be consistent with the OutputFormat class specified in the Driver)
{
```

59

Basic structure of a MapReduce program - Reducer (3)

```
  /* Implementation of the reduce method */
  protected void reduce(
      ReducerInputKeyType key,                // Input key
      Iterable<ReducerInputValueType> values, // Input values (list of
  values)
      Context context) throws IOException, InterruptedException {

      /* Process the input (key, [list of values]) pair and
         emit a set of (key,value) pairs.
         context.write(..) is used to emit (key, value) pairs
         context.write(new outputkey, new outputvalue); */
  } // End of the reduce method

} // End of class myReducerClass
```

60