Big data: architectures and data analytics

MapReduce Programming Paradigm and Hadoop – Part 2

Combiner

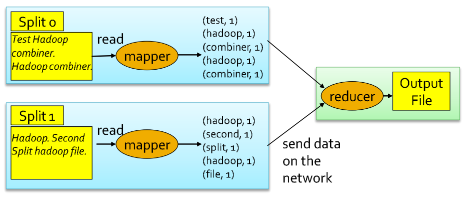
Combiner

- "Standard" MapReduce applications
  - The (key,value) pairs emitted by the Mappers are sent to the Reducers through the network
- Some "pre-aggregations" could be performed to limit the amount of network data

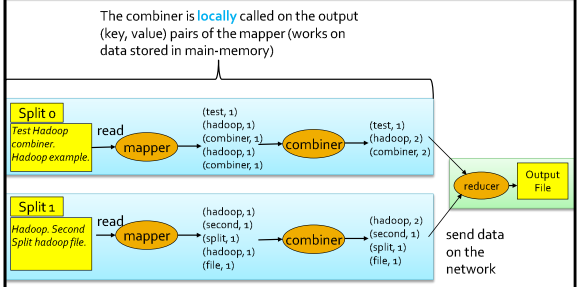
Combiner – Word count example

- Consider the standard word count problem
- Suppose the input file is split in two Input Splits
  - Hence, two Mappers are instanced (one for each split)

Word count example without combiner



Word count example with combiner



Combiner

- MapReduce applications with combiners
  - The (key,value) pairs emitted by the Mappers are analyzed in main-memory and aggregated by the Combiners
  - The combiner perform some "pre-aggregations" to limit the amount of network data
    - Pre-aggregate the values associated with the same key emitted by a Mapper
- Works only if the reduce function is commutative and associative

7

Combiner

- The Combiner
  - Is an instance of the org.apache.hadoop.mapreduce.Reducer class
    - There is not a specific combiner-template class
  - "Implements" a pre-reduce phase
  - Is characterized by the reduce(…) method
    - Processes (key, [list of values]) pairs and emits (key, value) pairs
  - Runs on the cluster

8

MapReduce programs - Combiner

- The Combiner 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 a mapper
- i.e., Combiners and Reducers extend the same class

9

MapReduce programs - Combiner

- The Combiner class is specified by using the job.setCombinerClass() method in the run method of the Driver
  - i.e., in the job configuration part of the code

10

MapReduce Programming Paradigm and Hadoop – Part 2

Personalized Data Types

11

Personalized Data Types and Values

- Personalized Data Types are useful when the value of a key-value pair is a complex data type
- Personalized Data Types are defined by implementing the org.apache.hadoop.io.Writable interface
  - The following methods must be implemented
    - public void readFields(DataInput in)
    - public void write(DataOutput out)
  - To properly format the output of the job usually also the following method is "redefined"
    - public String toString()

12

Personalized Data Types - Example

- Suppose to be interested in “complex” values composed of two parts:
  - a counter (int)
  - a sum (float)
- An ad-hoc Data Type can be used to implement this complex data type in Hadoop

Personalized Data Types - Example (1)

```
package it.polito.bigdata.hadoop.combinerexample;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class SumAndCountWritable implements
    org.apache.hadoop.io.Writable {
    /* Private variables */
    private float sum = 0;
    private int count = 0;
```

Personalized Data Types - Example (2)

```
/* Methods to get and set private variables of the class */
public float getSum() {
      return sum;
}

public void setSum(float sumValue) {
      sum=sumValue;
}

public int getCount() {
      return count;
}

public void setCount(int countValue) {
      count=countValue;
}
```

Personalized Data Types - Example (3)

```
/* Methods to serialize and deserialize the contents of the
instances of this class */
@Override  /* Serialize the fields of this object to out */
public void write(DataOutput out) throws IOException {
      out.writeFloat(sum);
      out.writeInt(count);
}

@Override /* Deserialize the fields of this object from in */
public void readFields(DataInput in) throws IOException {
      sum=in.readFloat();
      count=in.readInt();
}
```

Personalized Data Types - Example (4)

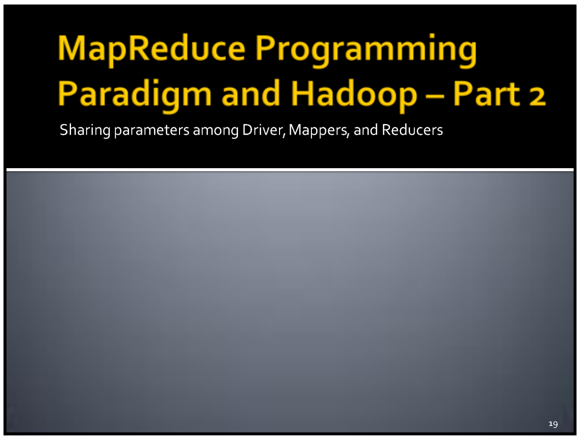
```
    /* Specify how to convert the contents of the instances of this
    class to a String
    * Useful to specify how to store/write the content of this class
    * in a textual file */
    public String toString()
    {
          String formattedString=
                      new String(“sum="+sum+",count="+count);

          return formattedString;
    }

}
```

Personalized Data Types and Keys

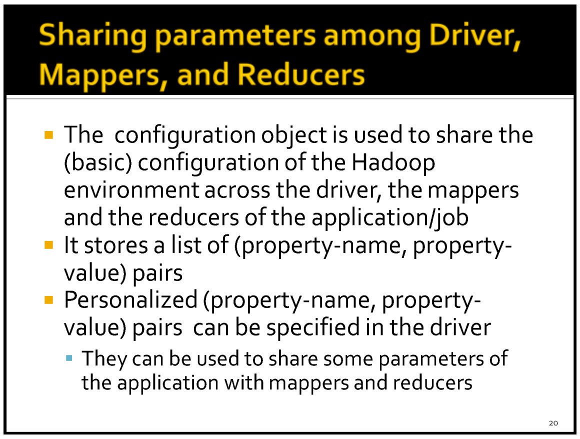
- Personalized Data Types can be used also to manage complex keys
- In that case the Personalized Data Type must implement the org.apache.hadoop.io. WritableComparable interface
  - Because keys must be comparables



MapReduce Programming Paradigm and Hadoop – Part 2

Sharing parameters among Driver, Mappers, and Reducers

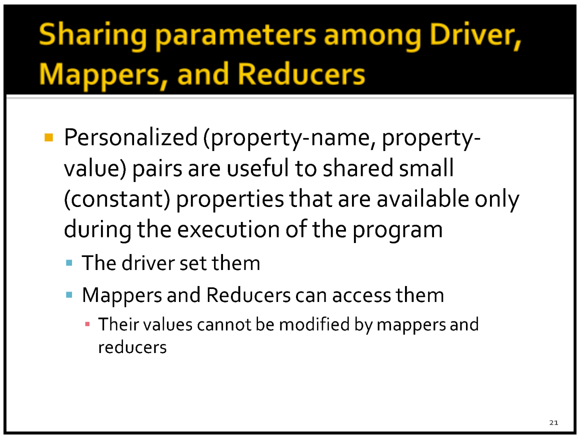
19



Sharing parameters among Driver, Mappers, and Reducers

- The configuration object is used to share the (basic) configuration of the Hadoop environment across the driver, the mappers and the reducers of the application/job
- It stores a list of (property-name, property-value) pairs
- Personalized (property-name, property-value) pairs can be specified in the driver
  - They can be used to share some parameters of the application with mappers and reducers

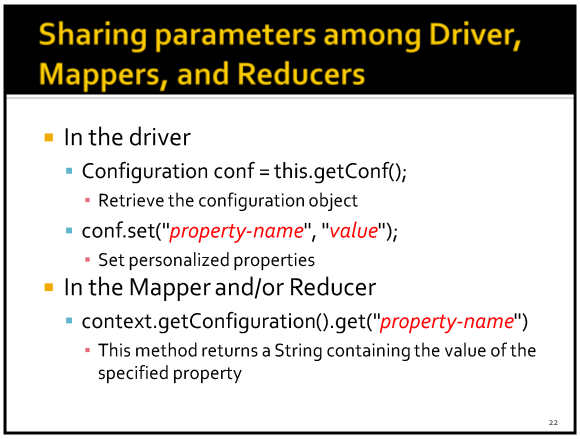
20



Sharing parameters among Driver, Mappers, and Reducers

- Personalized (property-name, property-value) pairs are useful to shared small (constant) properties that are available only during the execution of the program
  - The driver set them
  - Mappers and Reducers can access them
    - Their values cannot be modified by mappers and reducers

21



Sharing parameters among Driver, Mappers, and Reducers

- In the driver
  - Configuration conf = this.getConf();
    - Retrieve the configuration object
  - conf.set("*property-name*", "*value*");
    - Set personalized properties
- In the Mapper and/or Reducer
  - context.getConfiguration().get("*property-name*")
    - This method returns a String containing the value of the specified property

22