Big data: architectures and data analytics

MapReduce Programming Paradigm and Hadoop – Part 3

Counters

Counters

- Hadoop provides a set of basic, built-in, counters to store some statistics about jobs, mappers, reducers
  - E.g., number of input and output records
  - E.g., number of transmitted bytes
- Ad-hoc, user-defined, counters can be defined to compute global "statistics" associated with the goal of the application

3

User-defined Counters

- User-defined counters
  - Are defined by means of Java enum
    - Each application can define an arbitrary number of enums
  - Are incremented in the Mappers and Reducers
  - The global/final value of each counter is available a the end of the job
    - It is stored/printed by the Driver (at the end of the execution of the job)

4

User-defined Counters

- The name of the enum is the group name
  - Each enum as a number of "fields"
- The enum's fields are the counter name
- In mappers and/or reduces counters are incremented by using the increment() method
  - context.getCounter(countername).increment(value);

5

User-defined Counters

- The getCounters() and findCounter() methods are used by the Driver to retrieve the final values of the counters

6

User-defined Dynamic Counters

- User-defined counters can be also defined on the fly
  - By using the method incrCounter("group name", "counter name", value)
- Dynamic counters are useful when the set of counters is unknown at design time

7

Example of user-defined counters

- In the driver

```
public static enum COUNTERS {
  ERROR_COUNT,
  MISSING_FIELDS_RECORD_COUNT
}
```

- This enum defines two counters
  - COUNTERS.ERROR_COUNT
  - COUNTERS.MISSING_FIELDS_RECORD_COUNT

8

Example of user-defined counters

- This example increments the COUNTERS.ERROR_COUNT counter
- In the mapper or the reducer

```
context.getCounter(COUNTERS.ERROR_COUNT).i
ncrement(1);
```

9

Example of user-defined counters

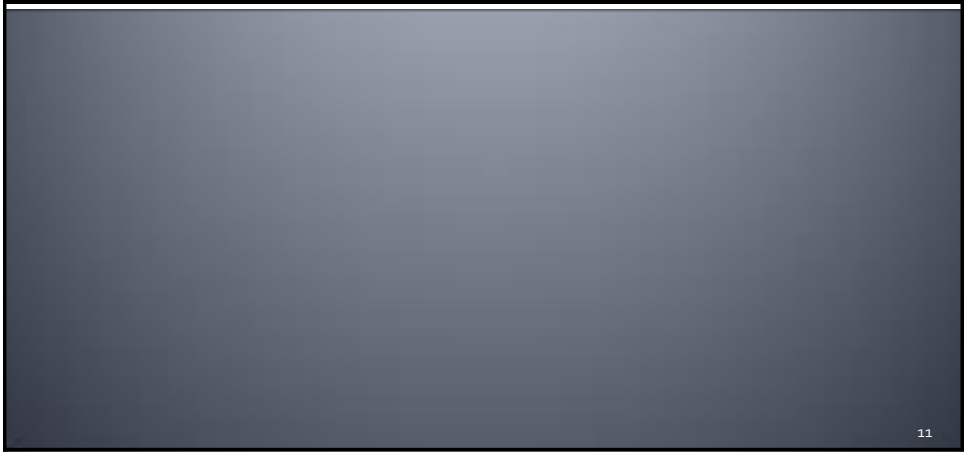
- This example retrieves the final value of the COUNTERS.ERROR_COUNT counter
- In the driver

```
Counter errorCounter =
job.getCounters().findCounter(COUNTERS.ERROR
_COUNT);
```

10

MapReduce Programming Paradigm and Hadoop – Part 3

Map-only job



11

Map-only job

- In some applications all the work can be performed by the mapper(s)
  - E.g., record filtering applications
- Hadoop allows executing Map-only jobs
  - The reduce phase is avoided
  - Also the shuffle and sort phase is not executed
  - The output of the map job is directly stored in HDFS
    - i.e., the set of pairs emitted by the map phase is already the final output

12

Map-only job

- Implementation of a Map-only job
  - Implement the map method
  - Set the number of reducers to 0 during the configuration of the job (in the driver)
    - job.setNumReduceTasks(0);

13

MapReduce Programming Paradigm and Hadoop – Part 3

In-Mapper combiner

14

Setup and cleanup method

- Mapper classes are characterized also by a setup and a cleanup method
  - They are empty if they are not override
- The setup method is called once for each mapper prior to the many calls to the map method
  - It can be used to set the values of in-mapper variables
  - In-mapper variables are used to maintain in-mapper statistics and preserve the state (locally for each mapper) within and across calls to the map method

15

Setup and cleanup method

- The map method, invoked many times, updates the value of the in-mapper variables
  - Each mapper (each instance of the mapper class) has its how copy of the in-mapper variables
- The **cleanup** method **is called once for each mapper** after the many calls to the map method
  - It can be used to emit (key,value) pairs based on the values of the in-mapper variables/statistics

16

In-Mapper Combiners

- In-Mapper Combiners, a possible improvement over "standard" Combiners
  - Initialize a set of in-mapper variables during the instance of the Mapper
    - Initialize them in the setup method of the mapper
  - Update the in-mapper variables/statistics in the map method
    - Usually, no (key,value) pairs are emitted in the map method of an in-mapper combiner

17

In-Mapper Combiners

- After all the input records (input (key, value) pairs) of a mapper have been analyzed by the map method, emit the output (key, value) pairs of the mapper
  - (key, value) pairs are emitted in the cleanup method of the mapper based on the values of the in-mapper variables

18

In-Mapper Combiners

- The in-mapper variables are used to perform the work of the combiner in the mapper
  - It can allow improving the overall performance of the application
  - But **pay attention** to the amount of **used main memory**
    - Each mapper can use a limited amount of main-memory
    - Hence, **in-mapper variables** should be **"small"** (at least smaller than the maximum amount of memory assigned to each mapper)

19

In-Mapper Combiner – Word count Pseudo code

```
class MAPPER
    method setup
        A ← new AssociativeArray

    method map(offset key, line l)
        for all word w ∈ line l do
            A{w} ← A{w} + 1

    method cleanup
    for all word w ∈ A do
        EMIT(term w , count A{w})
```

20

In-Mapper Combiner – Word count Pseudo code

```
class MAPPER
    method setup
        A ← new AssociativeArray

    method map(offset key, line l)
        for all word w ∈ line l do
            A{w} ← A{w} + 1

    method cleanup
    for all word w ∈ A do
        EMIT(term w , count A{w})
```

Invoked **one time** for each mapper

Invoked **one time** for each mapper

21

In-Mapper Combiner – Word count Pseudo code

```
class MAPPER
    method setup
        A ← new AssociativeArray

    method map(offset key, line l)
        for all word w ∈ line l do
            A{w} ← A{w} + 1

    method cleanup
    for all word w ∈ A do
        EMIT(term w , count A{w})
```

Invoked **one time** for each mapper

Invoked **multiple times** for each mapper

Invoked **one time** for each mapper

22