CS 378 – Big Data Programming

Lecture 4

Summarization Patterns

Review

- Assignment 1 – Issues
  - AWS Error: Moved Permamently Service: Amazon S3
    - Make regions for S3 bucket and EMR cluster the same

- Other questions?

Simple Debugging

- Counters
  - controller
  - syslog

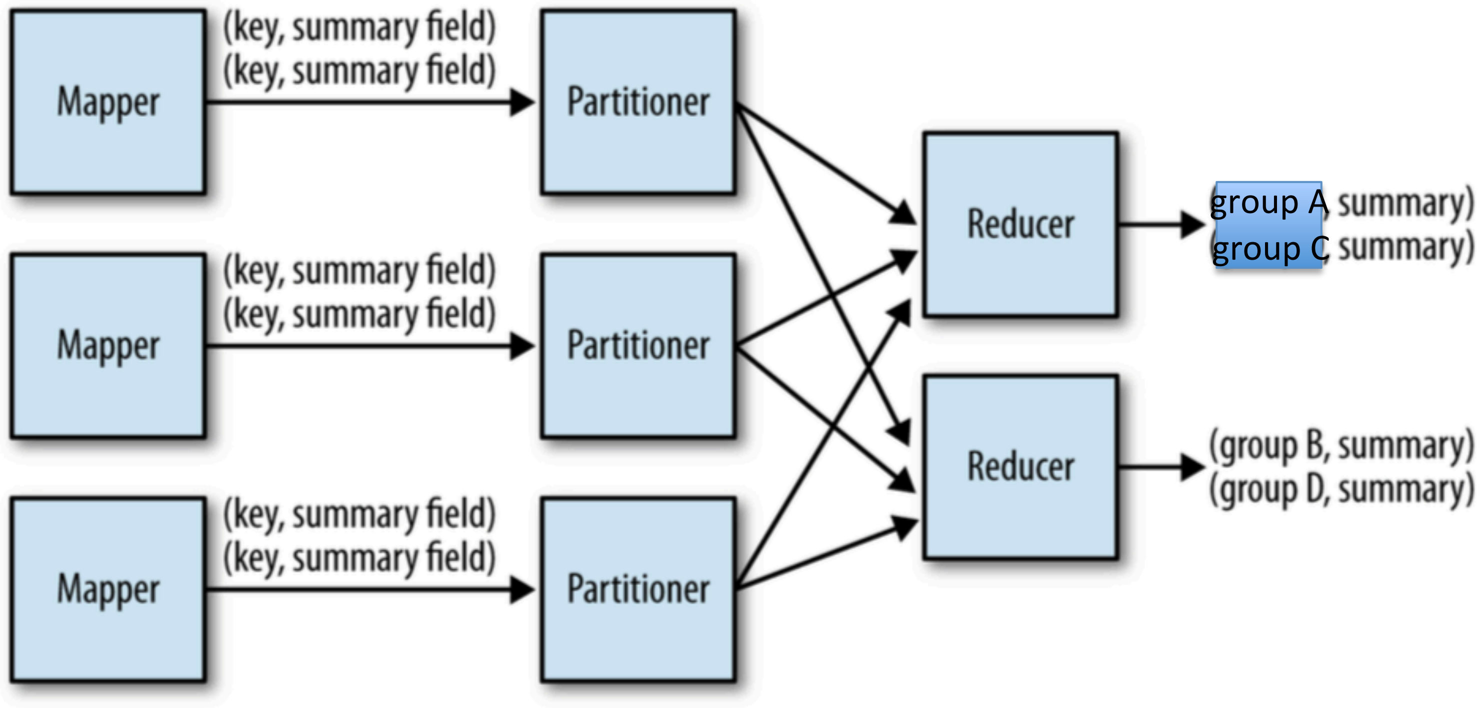
- Custom counters
  - `context.getCounter(group, counter).increment(1L);`
  - *group* and *counter* are strings

Summarization

- Counting things is a common map-reduce task
  - Word count was a simple example
  - Min, max, mean, median, variance, …
- By making the "things" being counted keys, MapReduce is doing much of the work for us
  - Hadoop sorts and groups data by key
- In WordCount, the words counted are the keys

Summarization

Figure 2.4, Map Reduce Design Patterns (edited)



Summarization

- Simple and useful pattern
- Mappers do local counts, reducers sum up
- Combiners are very useful here
- Usually collecting multiple statistics

Assignment 2 – Word Statistics

- Input:
  - Each input record/value is a paragraph of a document

- Output (similar to word count, but more numbers):
  - For each word in the document, output:
  - Number of paragraphs containing the word
  - Mean
    - In paragraphs where the word appears, what is the average number of times it appears
  - Variance
    - In paragraphs where the word appears, what is the variance

Word Statistics

- What do we need to calculate mean, variance?

- Mean is straightforward
  - Total number of occurrences of the word
  - Number of paragraphs containing the word

- Variance is less obvious
  - We can get there with a little algebra
  - "Mean of square minus square of mean"

Designing a Map-Reduce App

- We need to answer these questions:
  - What are the map input key and value types?
  - What does the mapper do?
  - What are the map output key and value types?
  - Can we use a combiner?
  - What does the reducer do?
  - What are the reduce output key and value types?

- And: What are the file formats?
  - For now we are using text files, we'll expand our options later

Multiple Output Values

- If we are to output multiple values for each key
  - How do we do that?
  - WordCount output a single number as the value
- Remember, our object containing the values needs to implement the **`Writable`** interface
- We could use **`Text`**
  - Value is a string of comma separated values
  - Have to convert our counts to strings, build the full string
  - Have to parse the string on input (not hard)

Multiple Output Values

- Suppose we wanted to implement a custom class
- Call it: **`WordStatisticsWritable`**
  - How would we implement this class?
  - Needs to implement the **`Writable`** interface
  - **`write()`** method:
    - Output the values needed for mean, variance
  - **`readFields()`** method:
    - Read the values needed for mean, variance

Custom Writable

- Approach 1 for **`WordStatisticsWritable`**:
  - Include instance variables of type **`LongWritable`** and **`DoubleWritable`**

- Required methods:
  - **`write(DataOutput out)`**
  - Writes the instance variable values (call **`write()`**)
  - **`readFields(DataInput in)`**
  - Reads the instance variable values
  - Create instances, call **`readFields()`**

Multiple Output Values

- Approach 2: **`ArrayWritable`**
  - Class provided by Hadoop
- In addition to **`write()`** and **`readFields()`**:
  - **`Writable[] get()`**
  - **`Class getValueClass()`**
  - **`void setWritable(Writable[] values)`**
  - **`Object toArray()`**
  - **`String[] toStrings()`**

Custom Writable

- Approach 3 for **`WordStatisticsWritable`**:
  - Use primitive Java types (**`long`**, **`double`**)

- Required methods:
  - **`write(DataOutput out)`**
  - Write primitive values to **`DataOutput`** instance
    - **`writeLong(), writeDouble()`**
  - **`readFields(DataInput in)`**
  - Read primitive values from **`DataInput`** instance
    - **`readLong(), readDouble()`**

Custom Writable

- What other methods might we want/need for **`WordStatisticsWritable`**?

- For output to text file:
  - **`toString()`**
- For reading in from text:
  - **`parse(String input)`**
- For MRUnit tests:
  - **`equals()`**

Word Statistics

- Mapper
  - What are the input key/value types?
  - What are the output key/value types?

- Reducer will calculate mean, variance
  - What are the input key/value types?
  - Make the output key/value types be:
    - **`Text`**, **`WordStatisticsWritable`**

Word Statistics

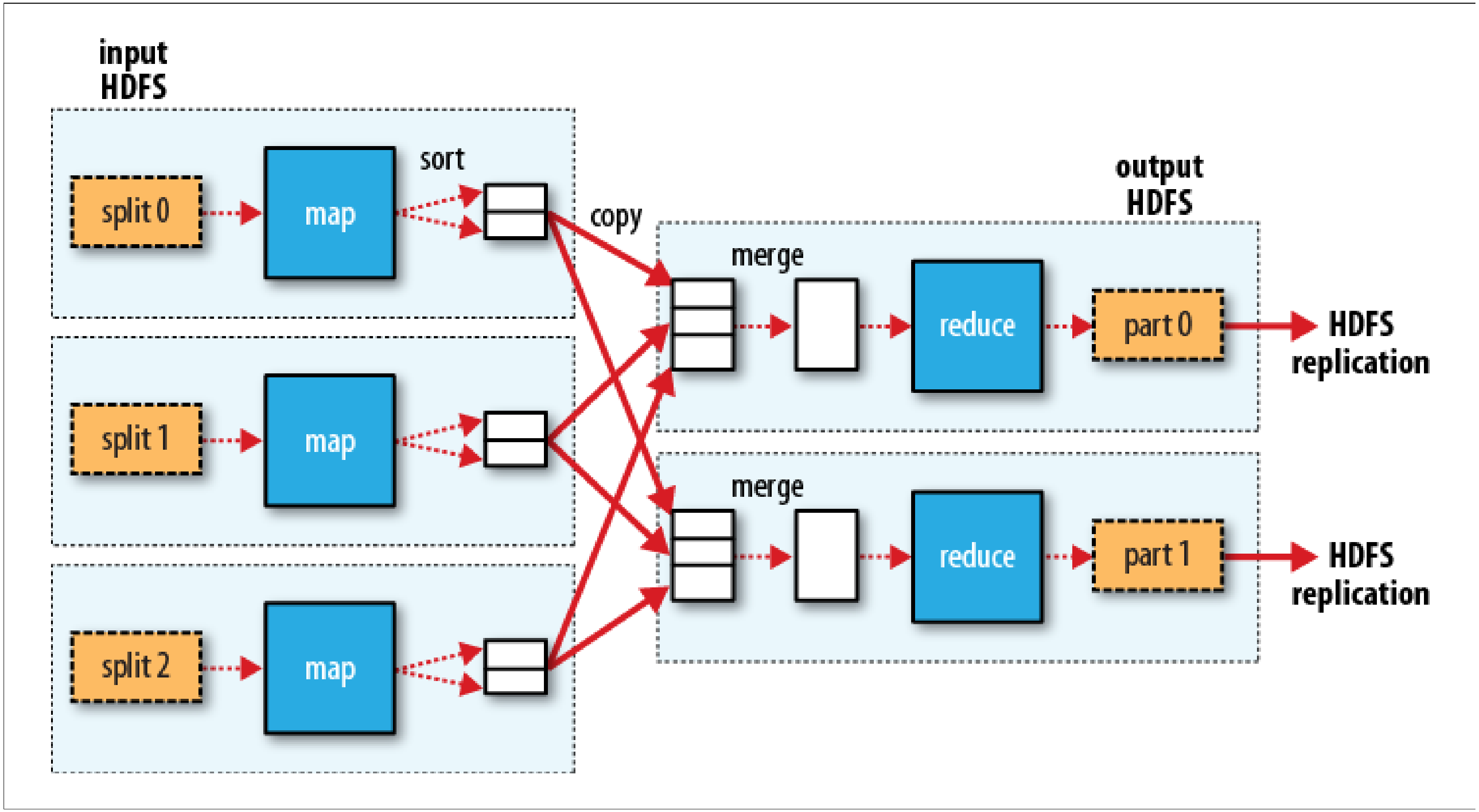
- Combiner will be useful for computing word statistics

- Why?

- Can we reuse the reducer class for the combiner?
  - What are the combiner input key/value types?
  - What are the combiner output key/input types?

MapReduce in Hadoop

Figure 2.4, Hadoop - The Definitive Guide


input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
output
HDFS
merge
reduce
part 0
HDFS
replication
merge
reduce
part 1
HDFS
replication