CS 378 – Big Data Programming

Lecture 10
Complex “Writable” Types
AVRO
Review

• Assignment 4 – CustomWritable

• We’ll look at implementation details of:
  – Mapper
  – Combiner
  – Reducer
  – Supporting classes

• What’s being called where?
  – write(), readFields()
  – toString()
The number of reduce tasks is not governed by the size of the input, but instead is specified independently. In "The Definitive MapReduce Job" on page 227, you will see how to choose the number of reduce tasks for a given job.

When there are multiple reducers, the map tasks partition their output, each creating one partition for each reduce task. There can be many keys (and their associated values) in each partition, but the records for any given key are all in a single partition. The partitioning can be controlled by a user-defined partitioning function, but normally the default partitioner—which buckets keys using a hash function—works very well.

The data flow for the general case of multiple reduce tasks is illustrated in Figure 2-4. This diagram makes it clear why the data flow between map and reduce tasks is colloquially known as "the shuffle," as each reduce task is fed by many map tasks. The shuffle is more complicated than this diagram suggests, and tuning it can have a big impact on job execution time, as you will see in "Shuffle and Sort" on page 208.

Finally, it's also possible to have zero reduce tasks. This can be appropriate when you don't need the shuffle because the processing can be carried out entirely in parallel (a few examples are discussed in "NLineInputFormat" on page 247). In this case, the only off-node data transfer is when the map tasks write to HDFS (see Figure 2-5).

Combiner Functions

Many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays to minimize the data transferred between map and reduce tasks. Hadoop allows the user to specify a **combiner function** to be run on the map output, and the combiner...
Review

• Some changes in the code
• Our mapReduce job class
  – Extends Configured
  – Implements Tool
  – Preferred style
• Moved logic from `main()` to `run()`
• `printClassPath()` method
  – Useful when debugging classpath issues
  – Outputs the classpath to stdout (try it and see)
Custom Writables

• Last time we discussed custom Writables

• Provided by Hadoop
  – Coded for us in Java

• Google’s protocol buffers

• AVRO
  – Language bindings generated by a compiler
  – Uses your definition of the data
Custom Writables

• For our custom Writable

• We had to implement Writable interface
  - readFields()
  - write()

• We had to implement toString() for text output

• We had to be able to parse in the text representation

• AVRO will implement these things for us
AVRO Example

```json
{
   "namespace": "com.refactorlabs.cs378.assign5",
   "type": "record",
   "name": "WordCountData",
   "fields": [
      {
         "name": "word_count",
         "type": "long"
      }
   ]
}
```

- How does this get transformed to Java code?
  - Add the schema file to your project (`filename.avsc`)
  - Run maven to force AVRO compile
    - Or run maven target in your IDE
AVRO Generated Code

• Accessors for the internal data
  – Has methods
    • hasWordCount()
    • ...
  – Get methods
    • getWordCount()
    • ...

• Builder class for constructing instances
  – Above methods
  – Plus set and clear methods
AVRO I/O

• Text output
  – AVRO text representation is JSON

• Avro container files
  – Binary representation that we can read

• The particular format is determined by
  – The types of objects we output
  – The file output format
Assignment 5

• Bootstrap script (control classpath order)
  – We want a specific version of AVRO
  – This script will place your JAR file at the start of the classpath

• pom.xml provided
  – Use this one, as AVRO with Hadoop is version sensitive
  – Select AMI version 2.4.7 when defining your cluster

• Examples of WordCount using AVRO are provided
  – On Canvas / Files / Assignment 5
Assignment 5

• Implement an AVRO object for WordStatistics data
  – Call it WordStatisticsData
  – Mapper output:
    • Text, AvroValue<WordStatisticsData>
  – Reducer output:
    • AvroKey<Pair<CharSequence, WordStatisticsData>>
  – Output file format: TextOutputFormat (like WordCountD)