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()
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

• Defined by us using Google’s protocol buffers
  – Protobuf
  – Language bindings generated by a compiler
  – Uses your definition of the data

• AVRO
Protobuf and AVRO

• These two approaches are interesting in that
  – They allow us to define complex types via a schema or IDL (Interface Definition Language)
  – They handle all the data marshalling/serialization
  – They create ”bindings” for various languages

• AVRO was designed for use with Hadoop
  – Writable interface implemented for us

• Protobufs require a Writable wrapper
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 Basics

• AVRO provides serialization of objects
  – RPC mechanism
  – Container file for storing objects (schema stored also)
  – Binary format as well as text format

• The schema language allows us to define complex objects
  – Schema language uses JSON syntax
  – Data structures containing primitive data types
  – Complex types: record, enum, array, map, union, fixed
  – Details: http://avro.apache.org/docs/1.7.4/spec.html
AVRO Example

```json
{
  "namespace": "com.refactorlabs.cs378.assign5",
  "type": "record",
  "name": "WordStatisticsData",
  "fields": [
    {"name": "document_count", "type": "long"},
    {"name": "total_count", "type": "long"},
    {"name": "sum_of_squares", "type": "long"},
    {"name": "mean", "type": ["double", "null"]},
    {"name": "variance", "type": ["double", "null"]}
  ]
}
```

- 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 from IDE)
AVRO Basics

• Primitive types
  – null
  – boolean
  – int, long
  – float, double
  – bytes, string

• Union: list of possible types
  – If null included, field can have no value
AVRO Basics

• Records
  – name, namespace
  – doc
  – aliases
  – fields
    • Name, doc, type, default, order, aliases

• Enums
  – name, namespace
  – aliases, doc
  – symbols
AVRO Basics

• Arrays
  – items
  
  {"type": "array", "items": "string"}

• Maps
  – values
  
  {"type": "map", "values": "string"}
  – Keys are assumed to be strings

• Fixed
  – Fixed number of bytes
AVRO Basics

• With a schema defined, we “compile” it to create “bindings” to a language

• Output is Java source code (Python available too)
  – Package and class name as we defined them

• So what does this Java class do for us?
  – Allows instance to be created and populated
  – Allows access to the data stored therein
  – Performs serialization
    • This is one main reason for using AVRO objects
    • AVRO objects implement `Writable` for use in Hadoop mapReduce
    • AVRO objects implement other stuff (`toString()`, parsing, …)
AVRO Generated Code

• Accessors for the internal data
  – Has methods
    • hasDocumentCount()
    • hasTotalCount()
    • …
  – Get methods
    • getDocumentCount()
    • getTotalCount()

• 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)
• 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 provided

• 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)
Schema Evolution

• As your data changes and you update the message definition

• In AVRO objects, the writer’s schema is included, and can be compared to the reader’s schema

• Comparison rules and rules for handling missing fields (in one schema but not the other) can be found here:
  – http://avro.apache.org/docs/1.7.4/spec.html#Schema+Resolution