CS 378 – Big Data Programming

Lecture 8

Complex “Writable” Types

AVRO

Review

- Assignment 3 - InvertedIndex
- We'll look at implementation details of:
  - Mapper
  - Combiner
  - Reducer
  - Supporting classes
- Other questions/issues?

Hadoop Provided Writables

- We've discussed some Hadoop Writable classes
  - `Text`
  - `LongWritable`
  - `DoubleWritable`
- Hadoop provides many other classes
  - Wrappers for all Java primitive types
  - Some for Hadoop usage (`TaskTrackerStatus`)
  - Others for us to extend (`MapWritable`)

User Defined Writables

- Hadoop provided classes cover commonly used types and data structures
- But we're likely to need more application specific data structures/types
  - For example, `WordStatistics`
- We can define these one by one
  - Must implement the `Writable` interface
  - This will become tedious

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

```
{“namespace”: “com.refactorlabs.cs378.assign4”,
 “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

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 – Builder Classes

- Why construct instances using the Builder class?
- You AVRO schema contains constraints
  - Value types: enforced by accessors
  - Required vs. optional values (union): checked by build
- Incremental construction
  - For arrays and maps, data can be added incrementally

AVRO I/O

- Text output
  - AVRO text representation is JSON
- Avro container files
  - Binary representation that we can read as input
- The particular format is determined by
  - The types of objects we output
  - The file output format

Assignment 4

- pom.xml provided
  - Added AVRO version, dependency
  - Instructions to compile AVRO schema definitions

- Example use of AVRO: WordCountA.java
  - AvroJob versus Job
  - AvroValue<>

- All files on Canvas / Files / Assignment 4

Assignment 4

- Implement WordStatistics using an Avro schema
  - Use the Avro schema in `wordCountData.avsc` as an example

- For each word calculate (like assignment 2):
  - *Mean, variance, min, max* of the occurrences of the word in those paragraphs where the word appears
  - Also compute these stats for paragraph length
  - All in one pass over the data

Assignment 4

- Implement an AVRO object for WordStatistics data
  - Call it `WordStatisticsData`
  - Mapper output:
    - `Text, AvroValue<WordStatisticsData>`
  - Reducer output:
    - `Text, AvroValue<WordStatisticsData>`

- See code in WordCountA
  - Output file format: `TextOutputFormat`
  - Calls using `AvroJob` to define the Avro schema