CS 378 – Big Data Programming

Lecture 9
Complex “Writable” Types
Review

• Assignment 4 - CustomWritable

• Questions/issues?
Hadoop Provided Writables

• We’ve used several Hadoop Writable classes
  – Text
  – LongWritable
  – ArrayWritable
    • Extended as LongArrayWritable, DoubleArrayWritable

• 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
User Defined Data Types

• Where might we look for a solution?
  – How are *ad hoc* types transferred elsewhere?

• Web formats for data structures
  – XML, JSON
  – Plus: Human readable, self describing
  – Minus: verbose, serialization is slower

• Java serialization
  – We have to write the serialization code
  – Again tedious, as data types get complex
User Defined Data Types

• RPC mechanisms
  – Marshall data in objects to be transferred to a “remote” procedure (no shared memory)
  – Usually procedure calls share memory
• Java serialization is one such mechanism
• Some others we’ll look at:
  – Google protocol buffers (protobufs)
  – 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

• Protobufs require a Writable wrapper
  – May be provided now, wasn’t a few years ago
Protobuf Basics

• Protocol buffers (protobufs) used extensively at Google as the RPC mechanism
  – Multiple language support (Java, C++, Python)
  – Used in the Google map-reduce framework

• The schema language (IDL) defines “messages”
  – Data structures containing primitive data types
  – Required or optional
  – Repeated (array)
  – Embedded message
package stats;

option java_package = "com.refactorlabs.cs378.utils";
option java_outer_classname = "WordStatisticsProto";

message WordStatistics {
  required int64 document_count = 1;
  required int64 total_count = 2;
  required int64 sum_of_squares = 3;
  optional double mean = 4;
  optional double variance = 5;
}
Schema Evolution

• As your data changes and you update the message definition

• Old Java code can read and use data written under the new schema
  – It simply doesn’t see the new fields

• New Java code can read and use data written under the old schema
  – New fields added must be optional
  – The has() methods can be used to determine where new fields are unpopulated
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
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
  
  ```json
  {"type": "array", "items": "string"}
  ```

• Maps
  – values
  
  ```json
  {"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, ...)
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](http://avro.apache.org/docs/1.7.4/spec.html#Schema+Resolution)