CSE535 Asynchronous Systems

Hadoop I/O

YoungMin Kwon

Data Integrity

- Data integrity is an important issue in Hadoop
  - Every disk or network operation has a small chance of error
  - Due to large volume of data Hadoop is handling, there is a nontrivial chance of getting an error
- A common error detection technique
  - CRC-32 (Cyclic Redundancy Check)

Data Integrity

- Data integrity in HDFS
  - In Hadoop, CRC-32 checksum is computed for every 512 byte data (default)
  - Data nodes verify the checksum when receiving data over a network or reading data from a disk.
- Data integrity in LocalFileSystem
  - LocalFileSystem uses ChecksumFileSystem as follows
  - When writing to a file, called *filename*, its checksums are computed and are stored at *.filename.crc* file

```
FileSystem rawFs = ...
FileSystem checksummedFs = new ChecksumFileSystem(rawFs);
```

Compression

- Benefits
  - Reduce the storage space
  - Reduce the data transfer

Compression format	Tool	Algorithm	Filename extension	Splittable?
DEFLATE[a]	N/A	DEFLATE	*.deflate*	No
gzip	*gzip*	DEFLATE	*.gz*	No
bzip2	*bzip2*	bzip2	*.bz2*	Yes
LZO	*lzop*	LZO	*.lzo*	No[b]
LZ4	N/A	LZ4	*.lz4*	No
Snappy	N/A	Snappy	*.snappy*	No

Splittable: whether one can seek to any point and start reading from there on

Compression

- Codecs
  - Implement CompressionCodec interface

Compression format	Hadoop CompressionCodec
DEFLATE	org.apache.hadoop.io.compress.DefaultCodec
gzip	org.apache.hadoop.io.compress.GzipCodec
bzip2	org.apache.hadoop.io.compress.BZip2Codec
LZO	com.hadoop.compression.lzo.LzopCodec
LZ4	org.apache.hadoop.io.compress.Lz4Codec
Snappy	org.apache.hadoop.io.compress.SnappyCodec

CompressionCodec

```
import java.io.InputStream;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.compress.*;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.util.ReflectionUtils;

public class StreamCompressor {
    public static void main(String[] args) throws Exception {
        String codecClassName = args[0];
        Class<?> codecClass = Class.forName(codecClassName);
        Configuration conf = new Configuration();
        CompressionCodec codec = (CompressionCodec)
                        ReflectionUtils.newInstance(codecClass, conf);

        CompressionOutputStream out = codec.createOutputStream(System.out);
        IOUtils.copyBytes(System.in, out, 4096, false);
        out.finish();
    }
}
```

```
echo "Hello World" | hadoop jar All.jar StreamCompressor \
                     org.apache.hadoop.io.compress.GzipCodec | gunzip -
```

Compression Factory

```
import java.net.URI;
import java.io.InputStream;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.compress.*;

public class FileDecompressor {
    public static void main(String[] args) throws Exception {
        String uri = args[0];
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(URI.create(uri), conf);

        Path inputPath = new Path(uri);
        CompressionCodecFactory factory = new CompressionCodecFactory(conf);
        CompressionCodec codec = factory.getCodec(inputPath);
        if(codec == null) {
            System.err.println("No codec found for " + uri);
            System.exit(1);
        }
```

CompressionFactory

```
        InputStream in = null;
        try {
            in = codec.createInputStream(fs.open(inputPath));
            IOUtils.copyBytes(in, System.out, conf);
        } finally {
            IOUtils.closeStream(in);
        }
    }
}
```

```
echo "Hello World" | hadoop jar All.jar StreamCompressor \
                     org.apache.hadoop.io.compress.GzipCodec > hello.gz
hdfs dfs -put hello.gz /user/hadoop/
hadoop jar All.jar FileDecompressor /user/hadoop/hello.gz
```

Compression and Input Splits

- Input Splits
  - If an uncompressed file of 1GB is stored in HDFS of block size 64MB, the file will be stored as 16 blocks and MapReduce will create 16 input splits.

- Will input split work for compressed files?
  - gzip files cannot be decompressed starting from an arbitrary place
  - bzip2 files and preprocessed LZO files can be decompressed starting from an arbitrary location. Hence they can be used for input splits.

Compression in MapReduce

- Compressed input files
  - They will be automatically decompressed using CompressionCodecFactory on the filename.
- Compressed output files
  - Set mapred.output.compress property to true
    Set mapred.output.compression.codec to the class name of compression codec
  - Or use FileOutputFormat

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.compress.GzipCodec;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperatureWithCompression {

    public static void main(String[] args)
        throws Exception {

        if(args.length != 2) {
            System.err.println("Usage: MaxTemperature " +
                               "<input path> <output path>");
            System.exit(-1);
        }

        Job job = new Job();
        job.setJarByClass(MaxTemperatureWithCompression.class);
        job.setJobName("Max Temperature");
```

```
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        FileOutputFormat.setCompressOutput(job, true);
        FileOutputFormat.setOutputCompressorClass(job, GzipCodec.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);

        job.setMapperClass(MaxTemperatureMapper.class);
        job.setCombinerClass(MaxTemperatureReducer.class);
        job.setReducerClass(MaxTemperatureReducer.class);


        System.exit(job.waitForCompletion(true) ? 0: 1);
    }
}
```

```
hadoop jar MaxTemperature.jar MaxTemperatureWithCompression \
                              /input/sample2.txt /output/sample2
hdfs dfs -cat /output/sample2/part-r-00000.gz | gunzip -
```

Compressing Map Output

- Example to enable gzip map output

```
...
Configuration conf = new Configuration();
conf.setBoolean("mapred.compress.map.out", true);
conf.setClass("mapred.map.output.compression.codec",
              GzipCodec.class,
              CompressionCodec.class);

Job job = new Job(conf);
...
```

Serialization

- Serialization
  - Turning structured object into a byte stream
  - The reverse process is called Deserialization
  - Used in RPC (remote process call), persistent store

Serialization

- Writable Interface

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public interface Writable {
    // Serialize object into out
    public void write(DataOutput out) throws IOException;

    // Constructing object from in
    public void readFields(DataInput in) throws IOException;
}
```

MaxTemperature Example

- Parse each line into TemperatureData object
- TemperatureData has
  - int year, int month, and double temperature fields
  - They need to be serialized to DataOut (using write method)
  - TemperatureData object needs to be constructed from DataIn by deserialization (using readFields method)

```
import java.io.*;
import org.apache.hadoop.io.*;

public class TemperatureData implements Writable {
    private int year;
    private int month;
    private double temperature;

    public TemperatureData() {
        set(0, 0, 0.0);
    }

    public TemperatureData(int year, int month, double temperature) {
        set(year, month, temperature);
    }

    public void set(int year, int month, double temperature) {
        this.year = year;
        this.month = month;
        this.temperature = temperature;
    }

    public int getYear()            { return year; }
    public int getMonth()           { return month; }
    public double getTemperature()  { return temperature; }
```

```
    @Override
    public void write(DataOutput out) throws IOException {
        out.writeInt(year);
        out.writeInt(month);
        out.writeDouble(temperature);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        year = in.readInt();
        month = in.readInt();
        temperature = in.readDouble();
    }
}
```

```
import java.io.IOException;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class MaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, TemperatureData> {
    private static final double MISSING = -9999;

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        String line = value.toString();
        String strYear = line.substring(6,10);
        String strMonth = line.substring(10,12);
        int year = Integer.parseInt(strYear);
        int month = Integer.parseInt(strMonth);
        double temperature;
        temperature = Double.parseDouble(line.substring(80, 86).trim());

        if(temperature != MISSING) {
            context.write(new Text(strMonth),
                          new TemperatureData(year, month, temperature));
        }
    }
}
```

```
import java.io.IOException;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;


public class MaxTemperatureReducer
    extends Reducer<Text, TemperatureData, Text, DoubleWritable> {

    @Override
    public void reduce(Text key,
                       Iterable<TemperatureData> values,
                       Context context)
        throws IOException, InterruptedException {

        double maxValue = Double.MIN_VALUE;
        for(TemperatureData value: values) {
            maxValue = Math.max(maxValue, value.getTemperature());
        }
        context.write(key, new DoubleWritable(maxValue));
    }
}
```

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

    public static void main(String[] args)
        throws Exception {

        if(args.length != 2) {
            System.err.println("Usage: MaxTemperature " +
                               "<input path> <output path>");
            System.exit(-1);
        }

        Job job = new Job();
        job.setJarByClass(MaxTemperature.class);
        job.setJobName("Max Temperature");
```

```
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setMapperClass(MaxTemperatureMapper.class);
        job.setReducerClass(MaxTemperatureReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(TemperatureData.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);

        System.exit(job.waitForCompletion(true) ? 0: 1);
    }
}
```

Serialization Framework

- Pluggable Serialization Framework
  - MapReduce uses Writable key and value types, but this is not mandatory
  - WritableSerialization is the implementation of Serialization for Writable types
  - Serialization defines a mapping from types to Serializer instances and Deserializer instances

Avro

- Apache Avro is a language-neutral data serialization system
- Language portability for Hadoop
- Avro Schema
  - Avro data is defined in a language-independent schema
  - The schema is usually written in JSON and data is usually encoded using a binary format
- Avro datafile
  - Has a metadata section where the schema is stored
  - Datafiles support compression and are splittable

Avro Data Types and Schemas

Avro Primitive Types

Type	Description	Schema
`null`	The absence of a value	`"null"`
`boolean`	A binary value	`"boolean"`
`int`	32-bit signed integer	`"int"`
`long`	64-bit signed integer	`"long"`
`float`	Single-precision (32-bit) IEEE 754 floating-point number	`"float"`
`double`	Double-precision (64-bit) IEEE 754 floating-point number	`"double"`
`bytes`	Sequence of 8-bit unsigned bytes	`"bytes"`
`string`	Sequence of Unicode characters	`"string"`

Avro Complex Types

Type	Description	Schema example
`array`	An ordered collection of objects. All objects in a particular array must have the same schema.	`{ "type": "array", "items": "long" }`
`map`	An unordered collection of key-value pairs. Keys must be strings and values may be any type, although within a particular map, all values must have the same schema.	`{ "type": "map", "values": "string" }`
`record`	A collection of named fields of any type.	`{ "type": "record", "name": "WeatherRecord", "doc": "A weather reading.", "fields": [ {"name": "year", "type": "int"}, {"name": "temperature", "type": "int"}, {"name": "stationId", "type": "string"} ] }`
`enum`	A set of named values.	`{ "type": "enum", "name": "Cutlery", "doc": "An eating utensil.", "symbols": ["KNIFE", "FORK", "SPOON"] }`

Avro Complex Types (continued)

Type	Description	Schema example
`fixed`	A fixed number of 8-bit unsigned bytes.	`{ "type": "fixed", "name": "Md5Hash", "size": 16 }`
`union`	A union of schemas. A union is represented by a JSON array, where each element in the array is a schema. Data represented by a union must match one of the schemas in the union.	`[ "null", "string", {"type": "map", "values": "string"} ]`

In-Memory (De)Serialization

- Create a Schema instance using Schema.Parse() method
- Create an Avro instance using the Generic API
- Serialize the object to an output stream
  - DatumWriter: translates data object into the type understood by Encoder
  - Encoder: writes to the output stream
- Deserialize: reverse the process using
  - DatumReader and Decoder

Generic mapping

Avro type	Generic Java mapping	Avro type	Generic Java mapping
boolean	boolean	map	java.util.Map
int	int	record	org.apache.avro.generic.GenericRecord
long	long		
float	float		
double	double	enum	java.lang.String
bytes	java.nio.ByteBuffer	fixed	org.apache.avro.generic.GenericFixed
string	org.apache.avro.util.Utf8 or java.lang.String	union	java.lang.Object
array	org.apache.avro.generic.GenericArray	null	null type

```
import java.io.*;
import org.apache.avro.*;
import org.apache.avro.generic.*;
import org.apache.avro.io.*;

public class AvroInMemory {
    public static void main(String[] args) throws IOException {
        String strSchema = "{"+
        "   \"type\": \"record\"," +
        "   \"name\": \"StringPair\"," +
        "   \"doc\":  \"A pair of strings.\"," +
        "   \"fields\": [" +
        "       {\"name\": \"left\",  \"type\": \"string\"}," +
        "       {\"name\": \"right\", \"type\": \"string\"}" +
        "   ]"+
        "}";

        // Create a schema instance
        Schema schema = Schema.parse(strSchema);

        // Create an instance of Avro record
        GenericRecord datum = new GenericData.Record(schema);
        datum.put("left", "L");
        datum.put("right", "R");
```

```
        // Serialize the object to ByteArrayOutputStream
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        DatumWriter<GenericRecord> writer =
                                new GenericDatumWriter<GenericRecord>(schema);
        Encoder encoder = EncoderFactory.get().binaryEncoder(out, null);
        writer.write(datum, encoder);
        encoder.flush();
        out.close();

        // Deserialize the object from ByteArrayOutputStream
        DatumReader<GenericRecord> reader =
                                 new GenericDatumReader<GenericRecord>(schema);
        Decoder decoder = DecoderFactory.get().binaryDecoder(
                                                    out.toByteArray(), null);
        GenericRecord result = reader.read(null, decoder);

        System.out.println(result.get("left"));
        System.out.println(result.get("right"));
    }
}
```

Avro Datafiles

- Avro's object container file format is for storing sequences of Avro objects
  - Header contains metadata including a schema
  - Blocks contain the serialized Avro objects
  - Sync marker is a unique sequence to the file in between the blocks => Make datafiles splittable
- Use DataFileWriter/DataFileReader instead of Encoder/Decoder

```
import java.io.*;
import org.apache.avro.*;
import org.apache.avro.file.*;
import org.apache.avro.generic.*;
import org.apache.avro.io.*;

public class AvroDataFile {
    public static void main(String[] args) throws IOException {
        String strSchema = "{"+
        "   \"type\": \"record\"," +
        "   \"name\": \"StringPair\"," +
        "   \"doc\":  \"A pair of strings.\"," +
        "   \"fields\": [" +
        "       {\"name\": \"left\",  \"type\": \"string\"}," +
        "       {\"name\": \"right\", \"type\": \"string\"}" +
        "   ]"+
        "}";

        // Create the schema
        Schema schema = Schema.parse(strSchema);

        // Create an instance of Avro record
        GenericRecord datum = new GenericData.Record(schema);
        datum.put("left", "L");
        datum.put("right", "R");
```

```
        // Serialize the object to a file
        File file = new File("data.avro");
        DatumWriter<GenericRecord> writer =
                                new GenericDatumWriter<GenericRecord>(schema);
        DataFileWriter<GenericRecord> dataFileWriter =
                                    new DataFileWriter<GenericRecord>(writer);
        dataFileWriter.create(schema, file);
        dataFileWriter.append(datum);
        dataFileWriter.close();

        // Deserialize the object from the file
        DatumReader<GenericRecord> reader =
                new GenericDatumReader<GenericRecord>(/*without schema*/);
        DataFileReader<GenericRecord> dataFileReader =
                            new DataFileReader<GenericRecord>(file, reader);
        GenericRecord result = dataFileReader.next();

        System.out.println(result.get("left"));
        System.out.println(result.get("right"));
    }
}
```

Avro MapReduce

- Avro provides a number of helper classes
  - AvroMapper
  - AvroReducer
- org.apache.avro.mapred.Pair to wrap the map output key and value
  - Map-only jobs can emit just values
- AvroJob is a helper class for specifying Avro schemas for the input, map output, and final output data

```
public class AvroMaxTemperature extends Configured implements Tool {

    private static final Schema schema = new Schema.Parser().parse(
        "{"+
        "   \"type\": \"record\"," +
        "   \"name\": \"WeatherRecord\"," +
        "   \"doc\":  \"A weather reading.\"," +
        "   \"fields\": [" +
        "       {\"name\": \"year\",        \"type\": \"int\"}," +
        "       {\"name\": \"month\",       \"type\": \"int\"}," +
        "       {\"name\": \"temperature\", \"type\": \"double\"}" +
        "   ]"+
        "}"
    );
```

```
public static class MaxTemperatureMapper
        extends AvroMapper<Utf8, Pair<Integer, GenericRecord>> {

        private GenericRecord record = new GenericData.Record(schema);

        @Override
        public void map(Utf8 utf8Line,
            AvroCollector<Pair<Integer, GenericRecord>> collector,
            Reporter reporter) throws IOException {

            String line = utf8Line.toString();
            int year = Integer.parseInt(line.substring(6,10));
            int month = Integer.parseInt(line.substring(10,12));
            double temperature = Double.parseDouble(line.substring(80,86));
            record.put("year",        year);
            record.put("month",       month);
            record.put("temperature", temperature);
            collector.collect(
                          new Pair<Integer, GenericRecord>(month, record));
        }
    }
```

```
public static class MaxTemperatureReducer
        extends AvroReducer<Integer, GenericRecord, GenericRecord> {
        @Override
        public void reduce(Integer key, Iterable<GenericRecord> values,
            AvroCollector<GenericRecord> collector,
            Reporter reporter) throws IOException {
            double maxValue = Double.MIN_VALUE;
            GenericRecord maxRecord = null;
            for(GenericRecord value : values) {
                double temp = (
                          (Double)value.get("temperature")).doubleValue();
                if(temp > maxValue) {
                    maxValue = temp;
                    maxRecord = new GenericData.Record(schema);
                    maxRecord.put("month",       value.get("month"));
                    maxRecord.put("year",        value.get("year"));
                    maxRecord.put("temperature", value.get("temperature"));
                }
            }
            collector.collect(maxRecord);
        }
    }
```

```
@Override
    public int run(String[] args) throws Exception {
        if(args.length != 2) {
            System.err.printf("Usage %s [generic options] <input> <output>\n",
                getClass().getSimpleName());
            ToolRunner.printGenericCommandUsage(System.err);
            return -1;
        }

        JobConf conf = new JobConf(getConf(), getClass());
        conf.setJobName("Max temperature");

        FileInputFormat.addInputPath(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        AvroJob.setInputSchema(conf, Schema.create(Schema.Type.STRING));
        AvroJob.setMapOutputSchema(conf,
            Pair.getPairSchema(Schema.create(Schema.Type.INT), schema));
        AvroJob.setOutputSchema(conf, schema);
        conf.setInputFormat(AvroUtf8InputFormat.class);

        AvroJob.setMapperClass(conf, MaxTemperatureMapper.class);
        AvroJob.setReducerClass(conf, MaxTemperatureReducer.class);

        JobClient.runJob(conf);
        return 0;
    }
```

```
 public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new AvroMaxTemperature(), args);
        System.exit(exitCode);
    }
}
```

To compile

```
    javac *.java
    jar cf MaxTemperature.jar *.class
```

To execute

```
 hadoop dfs -put sample2.txt /input/
 hadoop jar MaxTemperature.jar AvroMaxTemperature
                                    /input/sample2.txt /output/sample2
```