Week 12
Hadoop MapReduce Using Java

Mastering Cloud Computing

Coleman Kane
(based on material by Paul Talaga)

UNIVERSITY OF Cincinnati

Basic MapReduce Recipe (from Week 11)

Most MapReduce applications operate in this manner:

- a “mapper” receives the raw data set
- a “reducer” receives the mapper output as input
- Reducer’s output yields the result set

Example:
https://github.com/ckane/CS6065-Cloud-CrimeStats

UNIVERSITY OF Cincinnati

Expand For Java Example

For the Java example, we are going to break the project up into three parts:

- Create a data model class to encapsulate the structured records we will work with
- Create a generator class that will be responsible for feeding new records to the mapper
- Create the Hadoop job classes to map the input records and perform the summary computation

UNIVERSITY OF Cincinnati

Data Models

Example Data Set:
https://data.cincinnati-oh.gov/Safer-Streets/Police-Crime-Incident-Data/w7vh-beui
(CSV)

CSV Data contains the following fields, in a "Quoted-CSV" layout, one line per record:

Report#, Offense, ORC#, Street, City, State, Neighborhood, Block start, Block end, Occurred Date, Reported Date, Police District, Beat, Area, Officer Name, Badge#

UNIVERSITY OF Cincinnati

Step 1: Implement Data Model

Data Model Class:
https://github.com/ckane/CS6065-Cloud-CrimeStats/blob/master/CrimeRow.java

Key features:

- Public fields for each data point
- “enum” id for each column name
- “setter” method updateField() to update records, as well as communicate column ordering to Hadoop classes
- For now, manage timestamps as String fields

UNIVERSITY OF Cincinnati

Step 2: Implement Input Class

Input class:
https://github.com/ckane/CS6065-Cloud-CrimeStats/blob/master/CrimeRecordInputFormat.java

Extends the “FileInputFormat” class, using a Key of Text and a Value of CrimeRow (our custom model from Step 1)

Contents:

- RecordReader factory method: createRecordReader(...)
- Custom RecordReader

UNIVERSITY OF Cincinnati

Step 2a: Record Reader

Extends RecordReader w/ same Key/Value as parent

For newline-divided records, include a LineRecordReader to simplify implementation. LRR cannot define custom key/value itself though.

Must override the following methods:

- `initialize, close, getCurrentKey, getCurrentValue, getProgress, nextKeyValue`

Acts as a tokenizer/lexer, stepping forward, extracting structured data, and returning control to caller program in step-wise or progressive fashion

UNIVERSITY OF Cincinnati

Step 3: Implement App/Job Class

Work class:
https://github.com/ckane/CS6065-Cloud-CrimeStats/blob/master/CrimeStats.java

Must implement main() like a normal Java application

Contains:

- Mapper class - input key/val, output key/val
- Reducer class - input key/val, output key/val
- Combiner class - input key/val, output key/val (may be same as Reducer)

UNIVERSITY OF Cincinnati

Step 3a: Mapper

The Mapper class we use is CrimeStats.CrimeRowMapper

Accepts in: key/value as Object/CrimeRow, from the CrimeRecordInputFormat's CrimeRecordReader

Outputs: key/value as Text/IntWritable - key is currently hardcoded as "offense"

Method map(...) analyzes each CrimeRow and uses context.write(k,v) to provide output

UNIVERSITY OF Cincinnati

Step 3a: Mapper

```
public static class CrimeRowMapper extends Mapper<Object,CrimeRow,Text,IntWritable> {
  public void map(Object key, CrimeRow value, Context context)
      throws IOException, InterruptedException {
    context.write(new Text(value.offense), new IntWritable(1));
  };
};
```

UNIVERSITY OF Cincinnati

Step 3a: Reducer

The Reducer class we use is CrimeStats.CrimeRowReducer

Accepts in: key/value as Text/IntWritable

Outputs: key/value as Text/IntWritable

Method reduce(...) sums the array of each input value list and yields key/value that consists of same key as input plus the sum result

UNIVERSITY OF Cincinnati

Step 3a: Reducer

```
public static class CrimeRowReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
    int sum = 0;
    for(IntWritable i : values) {
      sum += i.get();
    };
    context.write(key, new IntWritable(sum));
  };
};
```

UNIVERSITY OF Cincinnati

Some Observations

- Choosing different column values to summarize only involves modifying CrimeRecordMapper class
- Hadoop program works very similarly to a normal Java application.
- Changing computation algorithm involves modification to the CrimeRecordReducer class

UNIVERSITY OF Cincinnati

Worker Job Initialization

1. New Configuration instance
2. Name the application/job
3. Set the JAR class
4. Set Mapper/Reducer/Combiner classes
5. Define data types for output key/value
6. Set input file path
7. Set input file format processing class
8. Set output file path
9. Wait for completion (or not!)

Worker Init Code

```
public static void main(String args[]) throws IOException, InterruptedException, ClassNotFoundException {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "Crime Stats");
  job.setJarByClass(CrimeStats.class);
  job.setMapperClass(CrimeRowMapper.class);
  job.setCombinerClass(CrimeRowReducer.class);
  job.setReducerClass(CrimeRowReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  job.setInputFormatClass(CrimeRecordInputFormat.class);
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
};
```

UNIVERSITY OF Cincinnati