Big Data Analytics with MapReduce

Phil Grim

HARRISBURG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Overview

- MapReduce Model Review
- Writing a MapReduce Program
- Use Case Description
- Solution
  - Classification
  - Scoring
  - Clustering

MapReduce Model

1

MAP

- Split input amongst nodes
- Each node processes local data set & emit key-value pairs

2

SHUFFLE

- Transfer results from mappers to reducers
- Merge results in each partition by key

3

REDUCE

- Read all results for each key
- Perform application-specific logic on data

Writing a MapReduce Program

- Summarize programming problem
- Design and implement
  - Mapper class
  - Reducer class
  - Driver class
- Build and execute code

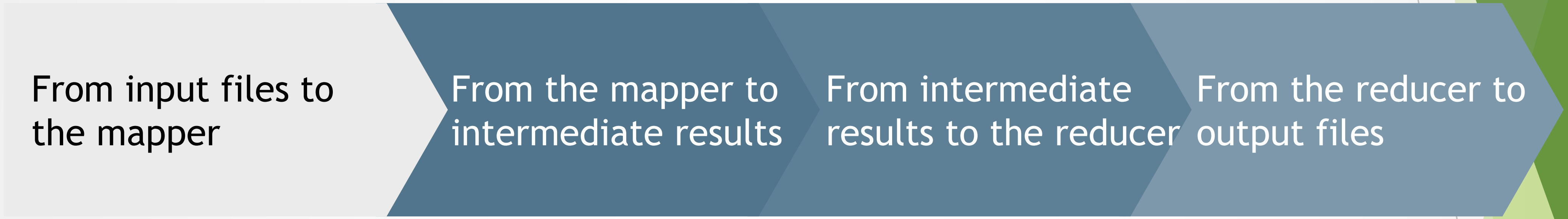
Summarize the Problem

- MapReduce Design Patterns

Pattern			
Summarizing data	Filtering data	Organizing data	Joining data

Design and Implement

- **Understand the flow and transformation of data:**


From input files to the mapper
From the mapper to intermediate results
From intermediate results to the reducer
From the reducer to output files

- **Identify appropriate types for keys and values**
- **Start with a template for the driver, mapper, and reducer classes**
- **Modify the template to suit the needs of your application**

Use Case Description

- Scenario:

  As part of an expert system, you must implement an automated method of assigning computer resources (mostly laptops) to users in an enterprise based on computing needs.

  If a user's computer is underpowered, productivity is lost, but if the user's computer is overpowered, money is wasted.

  Computing requirements have been assigned based on job descriptions, for example, engineers require high-end computers, while administrative personnel require low-end computers sufficient for word processing and e-mail.

Use Case Description

- Data:

The source data consists of catalog entries for various models of computers. The make and model of the computer are listed along with processor type and speed, memory size, and hard drive capacity.

Dell	Precision M6700	i7	3.2GHz	32GB	1TB
Dell	Precision M6600	i7	2.8GHz	24GB	1TB
Dell	Precision M6500	i7	2.4GHz	16GB	500GB
Acer	A300	i5	2.8GHz	12GB	500GB
Acer	A100	i5	2.4GHz	8GB	500GB
Lenovo	LR770	i7	3.0GHz	16GB	750GB
Lenovo	LR550	i5	2.4GHz	8GB	500GB
Acme	AC55	i5	2.4GHz	8GB	500GB
Acme	AC22	i3	2.2GHz	4GB	250GB
Asus	T600	17	3.2GHz	16GB	1TB
Asus	T400	i5	2.4GHz	8GB	500GB
Asus	T200	i3	2.2GHz	4GB	250GB

Solution

- The solution uses a series of MapReduce jobs to:
  - Extract the features of each model of laptop and classify each feature as high, medium, or low
  - Score each ranked feature numerically
  - Cluster the laptops based on the feature scores

Solution
Classification

- Setup
  - Train a Naïve Bayes Classifier with information about processor speed, memory size, and hard drive size
- Map Phase
  - Read file one line at a time. Tokenize line and extract the three features. Use Naïve Bayes Classifier to assign classification to each feature.
- Reduce Phase
  - Gather the three classifications for each laptop and output to a tab-separated values file

Mapper Class

```java
// The setup function is called once - here it is used to
// train the Bayes Classifier.
protected void setup(Context context)
  throws IOException, InterruptedException
{
  // Train the Bayes classifier for the laptop input data.
  bayesClassifier.learn(MEM_LOW, Arrays.asList(lowMem));
  bayesClassifier.learn(MEM_MED, Arrays.asList(mediumMem));
  bayesClassifier.learn(MEM_HIGH, Arrays.asList(highMem));
  bayesClassifier.learn(PROC_LOW, Arrays.asList(slowProc));
  bayesClassifier.learn(PROC_MED, Arrays.asList(mediumProc));
  bayesClassifier.learn(PROC_HIGH, Arrays.asList(fastProc));
  bayesClassifier.learn(HD_LOW, Arrays.asList(smallHD));
  bayesClassifier.learn(HD_MED, Arrays.asList(mediumHD));
  bayesClassifier.learn(HD_HIGH, Arrays.asList(bigHD));
}
```

```java
/**
  * The mapper class receives the data from the Map/Reduce
  * framework one line at a time and passes
  * the data through the Bayes classifier.  It emits a key/value pair for
  * each feature extracted.
  */
public static class FeatureExtractionMapper extends
    Mapper<LongWritable, Text, Text, Text>
{
  protected static Text modelOut = new Text();
  protected static Text classOut = new Text();

 public void map(LongWritable key, Text value, Context context)
  throws IOException, InterruptedException
  {
    String[] tokens = value.toString().split("\t");
    if (tokens.length > 2)
    {
      modelOut.set(tokens[0] + " " + tokens[1]);
      Collection<Classification<String, String>> classifications =
        bayesClassifier.classifyDetailed(Arrays.asList(tokens));

      classOut.set(getCategory(MEM_PREFIX, classifications));
      context.write(modelOut, classOut);
      classOut.set(getCategory(PROC_PREFIX, classifications));
      context.write(modelOut, classOut);
      classOut.set(getCategory(HD_PREFIX, classifications));
      context.write(modelOut, classOut);
    }
  }
```

Reducer Class

```
/**
  * The reducer class gathers the output of the mappers and creates a single line of output for each input
  * key with the features in alphabetical order.
  */
 public static class FeatureExtractionReducer extends Reducer<Text, Text, Text, Text>
 {
   // Output variables are static to avoid object creation on each call.
   protected static Text out = new Text();
   protected static SortedSet<String> outSet = new TreeSet<String>();
   protected static StringBuilder sb = new StringBuilder();

   // The reduce method is called once for each key emitted by the mapper, with an iterable collection of
   // each value emitted for that key.
   public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException
   {
     outSet.clear();
     sb.setLength(0);
     for (Text t : values)
     {
       outSet.add(t.toString());
     }
     for (String s : outSet)
     {
       sb.append(s).append("\t");
     }
     out.set(sb.toString().trim());
     context.write(key, out);
   }
 }
```

The Driver

```java
/**
 * Implements the run() method of the Tool interface.
 *
 * @param args - Command line arguments.
 *             The first argument should contain the location of the input
 *             data file, e.g. /examples/data/LaptopSpecs.tsv
 *             The second argument should specify a directory in which
 *             to place the results, e.g. /examples/results/features
 *
 * @return 0 if the job is successful, 1 otherwise
 *
 */
@Override
public int run(String[] args) throws Exception
{
  // Get the default configuration object
  Configuration conf = new Configuration();

  // Create a new job
  Job job = new Job(conf, this.getClass().getName());

  // Set the output types
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(Text.class);

  // Set the mapper and reducer classes
  job.setMapperClass(FeatureExtractionMapper.class);
  job.setReducerClass(FeatureExtractionReducer.class);

  // Set the input and output formats
  job.setInputFormatClass(TextInputFormat.class);
  job.setOutputFormatClass(TextOutputFormat.class);

  // Set the input and output paths
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));

  // set the jar file to run
  job.setJarByClass(this.getClass());

  // set the number of reducers to one
  job.setNumReduceTasks(1);

  // submit the job
  int exitCode = job.waitForCompletion(true) ? 0 : 1;

  // Return the exit code
  return exitCode;
}
```

Solution Classification

- Output

Acer A100	hd_medium	memory_low	proc_medium
Acer A300	hd_medium	memory_medium	proc_medium
Acme AC22	hd_low	memory_low	proc_low
Acme AC55	hd_medium	memory_low	proc_medium
Asus T200	hd_low	memory_low	proc_low
Asus T400	hd_medium	memory_low	proc_medium
Asus T600	hd_high	memory_medium	proc_high
Dell Precision M6500	hd_medium	memory_medium	proc_medium
Dell Precision M6600	hd_high	memory_high	proc_medium
Dell Precision M6700	hd_high	memory_high	proc_high
Lenovo LR550	hd_medium	memory_low	proc_medium
Lenovo LR770	hd_medium	memory_medium	proc_high

Solution Scoring

- Map Phase
  - Reads the output file from the feature extractor one line at a time. Assigns a numeric score to each feature.
- Reduce Phase
  - Aggregates the scores for each model of laptop. Writes the output to a sequence file that will be the input to the clustering job.

Mapper Class

```java
/**
   * The mapper class receives the data from the
   * Map/Reduce framework one line
   * at a time.
   */
  public static class ScoringMapper extends
    Mapper<LongWritable, Text, LongWritable, DataPoint>
  {
    protected static DataPoint out    = new DataPoint();

    // Map is called once per line in the file.
    public void map(LongWritable key, Text value,
                    Context context)
      throws IOException, InterruptedException
    {
      String[] fields = value.toString().split("\t");
      if (fields.length >= 4)
      {
        // Set up output data point and score the
        // features.
        out.setName(fields[0]);
        out.setX(score(fields[1]));
        out.setY(score(fields[2]));
        out.setZ(score(fields[3]));
        context.write(key, out);
      }
    }
```

```java
 protected double score(String feature)
   {
     double retval = 0.0;
     // Hard coded feature scoring. This is not terribly
     // efficient.
     // Possible improvements might included using enums
     // instead of strings.

     if (HD_LOW.equals(feature))
     {
       retval = 10.0;
     }
     else if (HD_MED.equals(feature))
     {
       retval = 30.0;
     }
     else if (HD_HIGH.equals(feature))
     {
       retval = 50.0;
     }
/**
 ** Trimmed for brevity…
 */
     return retval;
   }
  }
```

Reducer Class

```
/**
   * The reducer class gathers the output of the mappers.
   */
  public static class ScoringReducer extends Reducer<LongWritable, DataPoint, DataPoint, DataPoint>
  {
    public void reduce(LongWritable key, Iterable<DataPoint> values, Context context)
      throws IOException, InterruptedException
    {
      for (DataPoint v : values)
      {
        context.write(DataPoint.ORIGIN, v);
      }
    }
  }
```

The Driver

```java
  /**
   * Implements the run() method of the Tool interface.
   *
   * @param args
   *          - Command line arguments.
   *          The input directory where the output
   *          of the feature extraction was stored.
   *          The output directory of the scoring job
   *
   * @return 0 if the job is successful, 1 otherwise
   *
   */
 @Override
 public int run(String[] args) throws Exception
 {
   // Get the default configuration object
   Configuration conf = new Configuration();

   // Create a new job
   Job job = new Job(conf, this.getClass().getName());
   job.setJobName("Scoring");

   // Set the mapper output types
   job.setMapOutputKeyClass(LongWritable.class);
   job.setMapOutputValueClass(DataPoint.class);

   // Set the final output types
   job.setOutputKeyClass(DataPoint.class);
   job.setOutputValueClass(DataPoint.class);

   // Set the mapper and reducer classes
   job.setMapperClass(ScoringMapper.class);
   job.setReducerClass(ScoringReducer.class);

   // Set the input and output formats
   job.setInputFormatClass(TextInputFormat.class);
   job.setOutputFormatClass(SequenceFileOutputFormat.class);

   // Set the input and output paths
   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));

   // set the jar file to run
   job.setJarByClass(this.getClass());

   // set the number of reducers to one
   job.setNumReduceTasks(1);

   // submit the job
   int exitCode = job.waitForCompletion(true) ? 0 : 1;

   // Return the exit code
   return exitCode;
 }
```

Solution Scoring

- Output

```
ORIGIN (0.0, 0.0, 0.0)[0]     Acer A100 (30.0, 10.0, 70.0)[0]
ORIGIN (0.0, 0.0, 0.0)[0]     Acer A300 (30.0, 50.0, 70.0)[0]
ORIGIN (0.0, 0.0, 0.0)[0]     Acme AC22 (10.0, 10.0, 20.0)[0]
ORIGIN (0.0, 0.0, 0.0)[0]     Acme AC55 (30.0, 10.0, 70.0)[0]
ORIGIN (0.0, 0.0, 0.0)[0]     Asus T200 (10.0, 10.0, 20.0)[0]
ORIGIN (0.0, 0.0, 0.0)[0]     Asus T400 (30.0, 10.0, 70.0)[0]
ORIGIN (0.0, 0.0, 0.0)[0]     Asus T600 (50.0, 50.0, 120.0)[0]
ORIGIN (0.0, 0.0, 0.0)[0]     Dell Precision M6500 (30.0, 50.0, 70.0)[0]
ORIGIN (0.0, 0.0, 0.0)[0]     Dell Precision M6600 (50.0, 100.0, 70.0)[0]
ORIGIN (0.0, 0.0, 0.0)[0]     Dell Precision M6700 (50.0, 100.0, 120.0)[0]
ORIGIN (0.0, 0.0, 0.0)[0]     Lenovo LR550 (30.0, 10.0, 70.0)[0]
ORIGIN (0.0, 0.0, 0.0)[0]     Lenovo LR770 (30.0, 50.0, 120.0)[0]
```

Solution
Clustering

- Use k-means clustering to determine which category the laptops belong in. Number of iterations is configurable since convergence is difficult to detect in the MapReduce paradigm.
- Setup
  - Creates an initial centroids file with arbitrary values on the first iteration. Reads the centroids file from the previous iteration from then on.
- Map Phase
  - Reads the scored laptop records one line at a time. Finds the closest centroid for the record using Euclidean distance. Outputs the closest centroid as key, record as value.
- Reduce Phase
  - Gets a cluster centroid as key, list of records assigned to that cluster as value. Calculates a new centroid for the cluster based on the assigned records.
- Cleanup
  - Writes a new centroids file to be used for the next iteration
- Final report written after iterations are complete

Mapper Class

```java
/**
 * The setup function is run once at the beginning of the
 * map task, and reads the updated centroids produced by
 * the last iteration (initial values arbitrarily
 * assigned)
 */
@Override
protected void setup(Context context)
  throws IOException, InterruptedException
{
  super.setup(context);
  Configuration conf = context.getConfiguration();
  Path centroidsPath = new
    Path(conf.get("centroids.path"));
  FileSystem fs = FileSystem.get(conf);

  try (SequenceFile.Reader reader = new
    SequenceFile.Reader(fs, centroidsPath, conf))
  {
     DataPoint key = new DataPoint();
     IntWritable value = new IntWritable();
     while (reader.next(key, value))
     {
       DataPoint clusterCenter = new DataPoint(key);
       centroids.add(clusterCenter);
     }
   }
}
```

```java
// Map is called once per line in the file.  The distance
// of each value from all of the centroids is calculated,
// and a new closest centroid possibly found.
// Output is the closest centroid as key, and the record
// as value.
public void map(DataPoint key, DataPoint value,
  Context context)
  throws IOException, InterruptedException
 {
   DataPoint closest = null;
   double shortest = Double.MAX_VALUE;
   for (DataPoint c : centroids)
   {
     double distance = c.distance(value);
     if (distance < shortest || null == closest)
     {
       shortest = distance;
       closest = c;
     }
   }
   value.setCluster(closest.getCluster());
   context.write(closest, value);
 }
}
```

Reducer Class

```java
/**
  * The reducer class gathers the output of the
  * mappers.
  */
public static class ClusteringReducer extends
 Reducer<DataPoint, DataPoint, DataPoint, DataPoint>
{
  protected static final SortedSet<DataPoint>
    centroids = new TreeSet<DataPoint>();
  protected static final List<DataPoint> points =
    new ArrayList<DataPoint>();

// Input is a centroid as key, and all records
// calculated as closest to that centroid as
// values.
// New center is calculated from the values.
// Output is new cluster center as key, record as
// value.
public void reduce(DataPoint key,
                   Iterable<DataPoint> values,
                   Context context)
  throws IOException, InterruptedException
{
  points.clear();

  int count = 0;
  double runningX = 0.0;
  double runningY = 0.0;
  double runningZ = 0.0;
```

```java
for (DataPoint d : values)
{
  points.add(new DataPoint(d));
  runningX += d.getX();
  runningY += d.getY();
  runningZ += d.getZ();
  count++;
}
DataPoint center = new
DataPoint(key.getName(),runningX/count,
   runningY/count, runningZ/count, key.getCluster());
 centroids.add(center);
 for (DataPoint d : points)
 {
    context.write(center, d);
 }
}
```

Reducer Class

```java
/**
 * Called once at the end of the reduce phase.  Writes the new centroids into the centroid file
 * to be used by the next iteration.
 */
@Override
protected void cleanup(Context context) throws IOException, InterruptedException
{
  super.cleanup(context);
  Configuration conf = context.getConfiguration();
  Path outPath = new Path(conf.get("centroids.path"));
  FileSystem fs = FileSystem.get(conf);
  fs.delete(outPath, true);

  SequenceFile.Writer out = SequenceFile.createWriter(fs, conf, outPath, DataPoint.class,
    IntWritable.class);
  final IntWritable value = new IntWritable(0);
  for (DataPoint center : centroids)
  {
    out.append(center, value);
  }
  out.close();
}
```

The Driver

```
 // Run the job for the specified number of iterations
 for (int iteration = 1; iteration <= numIters && exitCode
== 0; iteration++)
 {
   // Create a new job
   Job job = new Job(conf, this.getClass().getName());
   job.setJobName("Clustering Pass " + iteration);

   // Set the output types
   job.setOutputKeyClass(DataPoint.class);
   job.setOutputValueClass(DataPoint.class);

   // Set the mapper and reducer classes
   job.setMapperClass(ClusteringMapper.class);
   job.setReducerClass(ClusteringReducer.class);

   // Set the input and output formats

job.setInputFormatClass(SequenceFileInputFormat.class);

job.setOutputFormatClass(SequenceFileOutputFormat.class);
```

```
  // Set the input and output paths
  if (1 == iteration)
  {
     // Read from the initial directory
     FileInputFormat.addInputPath(job, new Path(args[1]));
  }
  else
  {
    // Read from the last directory written
    FileInputFormat.addInputPath(job, new
      Path(outputPrefix + "_" + (iteration - 1)));
  }
  FileOutputFormat.setOutputPath(job, new
    Path(outputPrefix + "_" + iteration));
  // set the jar file to run
  job.setJarByClass(this.getClass());

  // set the number of reducers to one
  job.setNumReduceTasks(1);

  // submit the job
  exitCode += job.waitForCompletion(true) ? 0 : 1;
 }
 if (exitCode == 0)
 {
   writeFinalReport(conf, fs, new Path(outputPrefix + "_"
 + numIters));
 }
```

Solution Clustering

- Output

```
Acme AC22: Entry Level
Asus T200: Entry Level
Acer A100: Midrange
Acer A300: Midrange
Acme AC55: Midrange
Asus T400: Midrange
Dell Precision M6500: Midrange
Lenovo LR550: Midrange
Asus T600: High End
Dell Precision M6600: High End
Dell Precision M6700: High End
Lenovo LR770: High End
```

Additional Reading

References

Apache Software Foundation. (2013, 08 04). *HDFS Architecture Guide.* Retrieved from Hadoop: http://hadoop.apache.org/docs/stable1/hdfs_design.html

Dean, J., & Ghemawat, S. (2004). *MapReduce: Simplified Data Processing on Large Clusters.* Retrieved from Google Research: http://static.googleusercontent.com/media/research.google.com/en/us/archive/mapreduce-osdi04.pdf

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). *The Google File System.* Retrieved from Google Research: http://static.googleusercontent.com/media/research.google.com/en/us/archive/gfs-sosp2003.pdf

Rajaraman, A., Leskovec, J., & Ullman, J. D. (2013). *Mining of Massive Datasets.* Palo Alto, CA: Stanford University.