Map-Reduce

Craig Ward, cward@isi.edu

An Introduction to Map-Reduce with Examples using Hadoop.

Agenda

- Basics of Map-Reduce
- Overview of Hadoop
- Data mining using K-Means in Hadoop
- Bibliography

Basics of Map-Reduce

- Inspired by higher-order functions.
  - Functions that take functions as arguments.
- Map and Reduce take a function and apply it to a list of elements, returning the result.
- A hallmark of functional programming.

Map-Reduce

- Map()
  - Processes key/value pairs to generate intermediate key/value pairs.
- Reduce()
  - Merges all intermediate values associated with the same key.

Word Count

- Counting words in a document or documents.
- How input is divided doesn't change results.
- Mapping in parallel across large numbers of nodes.
- Reducing also in parallel.

Word Count

- Simple pseudo code

```
map(string key, string value)
//key: document name
//value: document contents
    for each word w in value
        EmitIntermediate(w, "1");

reduce(string key, iterator values)
//key: word
//values: list of counts
    int results = 0;
    for each v in values
        result += ParseInt(v);
    Emit(AsString(result));
```

Hadoop Overview

- Hadoop is an open source implementation of a Map-Reduce framework from the Apache Software Foundation.
  - http://hadoop.apache.org/
- In production use by several organizations, most notable Yahoo!
- It is a rapidly evolving system. The examples here use 0.17.x; latest is 0.20.x.
  - Parts of Hadoop keep splitting into separate projects.

Hadoop Overview

- Distributed, fault tolerant system
  - Yahoo runs cluster with over 10,000 cores.
- Hadoop Distributed File System (HDFS)
  - Data files replicated in multiple blocks across multiple nodes.
  - If one node fails, another node has copy of each block.

Hadoop Overview

- Node Types
  - Namenode
    - Central control node
  - Task processing nodes
  - Storage nodes

Hadoop Overview

- The Hadoop Map-Reduce framework provides three operations:
  - Map – Map data to a key.
  - Combine – Add data for a key on a node.
  - Reduce – Add data for all keys.
- In Hadoop, both Combine and Reduce implement the same interface.
  - Combine data on one node before reducing across all nodes.

Data Mining with Map-Reduce

- K-Means example
  - K-Means is an algorithm for analyzing clustered data.
  - Find K center points in clustered data.
- K-Means used as a test case for experiments to test networks using Hadoop in various distributed configurations.

Data Mining with Map-Reduce

- Experiment generated large sets of data points to use as input for a Map-Reduce run in Hadoop.
- The experiment wasn't about K-Means, but about Hadoop and network performance.
- Nonetheless, the K-Means job is a useful illustration.

Map

```
 static class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, Text>
{
    private Text word = new Text();
    private int verboseLevel = 0;

    public void map(LongWritable key, Text value,
                    OutputCollector<Text, Text> output, Reporter reporter)
    throws IOException {

        String line = value.toString();
        NumberedPoint point = new NumberedPoint(line);
        int newKey = 0;
        double minDistance = 0.0;

        for (int i = 0; i < clusters.length; i++) {
            NumberedCenterPoint cluster = clusters[i];
            double distance = cluster.distance(point);
            if (minDistance == 0 || distance < minDistance) {
                minDistance = distance;
                newKey = i;
            }
        }

        clusters[newKey].incrementPointCount();
        word.set(point.toString());
        output.collect(new Text(clusters[newKey].toString()), word);
    }
```

Combine

```
static class Combine extends MapReduceBase
    implements Reducer<Text, Text, Text, Text>
 {
    private JobConf myJob;
    private int verboseLevel;

    public void reduce(Text key, Iterator<Text> values,
                    OutputCollector<Text, Text> output, Reporter reporter)
    throws IOException {
       int iteration = myJob.getInt("RunCount", 0);
       double points = 0.0;
       double cx = 0.0;
       double cy = 0.0;
       NumberedCenterPoint cp = new NumberedCenterPoint(key.toString());

       while (values.hasNext()) {
          String line = values.next().toString();
          NumberedPoint p = new NumberedPoint(line);
          cx += p.getX().doubleValue();
          cy += p.getY().doubleValue();
          points++;
       }

       Text value = new Text();
       value.set(result);
       output.collect(key, value);
    }
```

Reduce

```
static class Reduce extends MapReduceBase
    implements Reducer<Text, Text, Text, Text>
 {
    private JobConf myJob = null;
    private int verboseLevel = 0;

    public void reduce(Text key, Iterator<Text> values,
                OutputCollector<Text, Text> output, Reporter reporter)
    throws IOException {
       int iteration = myJob.getInt("RunCount", 0);
       double points = 0.0;
       double cx = 0.0;
       double cy = 0.0;
       NumberedCenterPoint oldCenter =
          new NumberedCenterPoint(key.toString());

       while (values.hasNext()) {
          String value = values.next().toString();
          String[] line = value.split(", *");
          String msg = value + " => " + line[0] + " " + line[1] + " "
                    + line[2];
          out.println(msg);
          reporter.setStatus(msg);
          cx += Double.valueOf(line[0]);
          cy += Double.valueOf(line[1]);
          points += Double.valueOf(line[2]);
       }
       cx /= points;
       cy /= points;
       NumberedCenterPoint newCenter = new NumberedCenterPoint(cx, cy);
       Text nKey = null;
       Text nValue = null;
       StringBuffer sb = new StringBuffer("Reduce iteration "
             + String.valueOf(runCount));
       if (oldCenter.equals(newCenter)) {
          if (verboseLevel > VERBOSE_OFF) {
             String msg = sb.toString() + ": cluster center unchanged";
             out.println(msg);
             reporter.setStatus(msg);
          }
          nKey = key;
       } else {
          if (verboseLevel > VERBOSE_OFF) {
             String msg = sb.toString() + ": cluster now "
             + newCenter.toString();
             out.println(msg);
             reporter.setStatus(msg);
          }
          nKey = new Text(newCenter.toString());
       }
       output.collect(nKey, nValue);
    }
}
```

Running K-Means

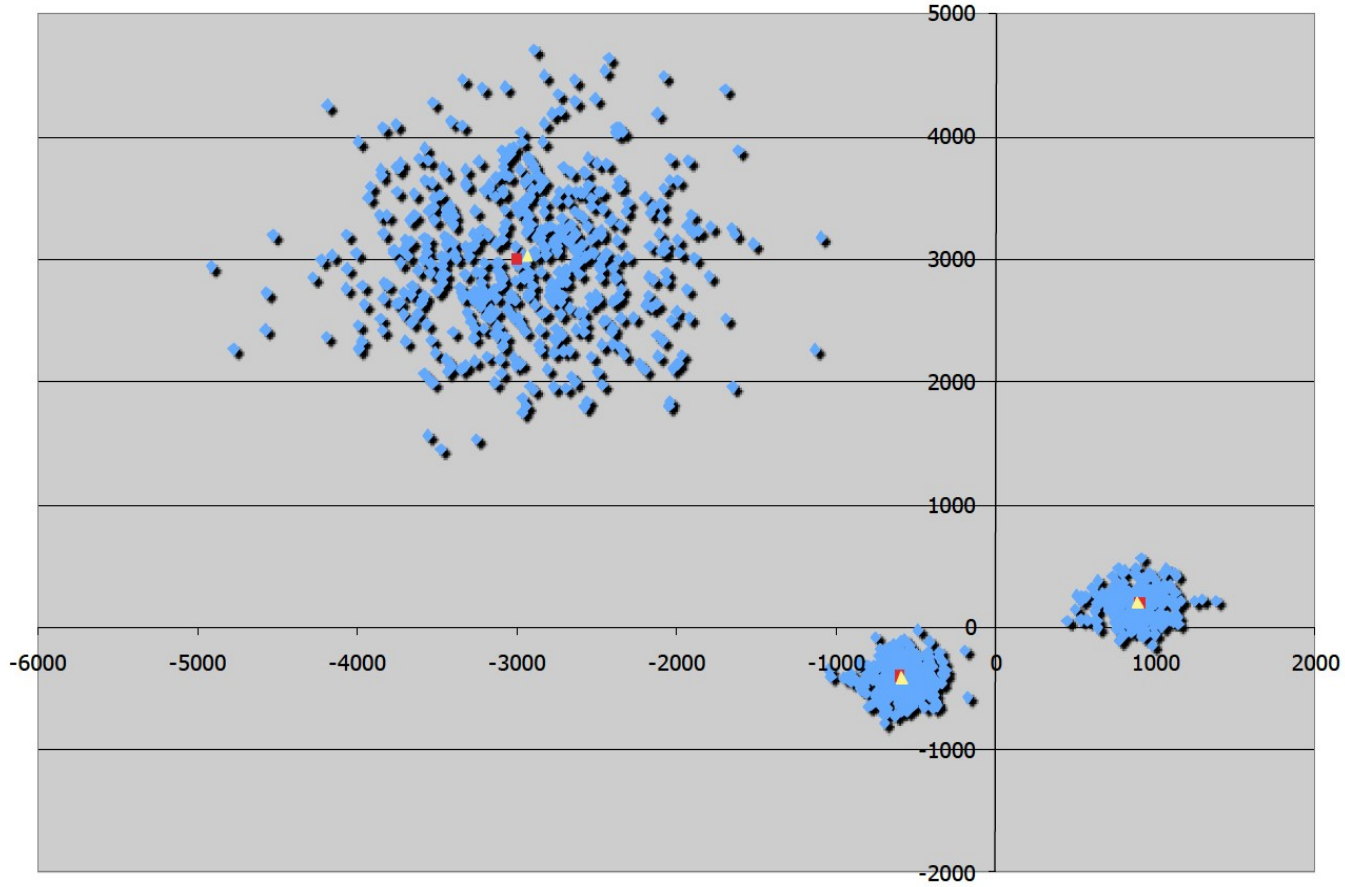
- Files containing points created around known centers were loaded into HDFS.
- The K-Means job was submitted to search for varying numbers of cluster points.
- The job terminated either when the points stopped changing or the maximum allowed iterations had been exceeded.

Iterations

```
 private static int runIteration(JobConf parentConf)
throws IOException {
      JobClient client = new JobClient();
      JobConf runConf = new JobConf(parentConf);
      FileSystem fs = FileSystem.get(runConf);
      int iteration = runConf.getInt("RunCount", 0);

      // Setup different output each run
      runConf.setOutputPath(new Path(outputPath + "_" +
                              String.valueOf(iteration)));
      client.setConf(runConf);
      JobClient.runJob(runConf);

      // Now build the name of the reduce result to use for checking
      // results
      Path srcClusterFile = new Path(outputPath + "_" +
                              String.valueOf(iteration) +
                              "/part-00000");
      Path path = runConf.getWorkingDirectory();
      Path dstClusterFile = new Path(path, clusterFileNameTemplate +
                              String.valueOf(iteration+1));
      FileUtil.copy(fs, srcClusterFile, fs, dstClusterFile, false, runConf);

      return countChanges(fs, srcClusterFile, iteration, runConf);
  }
```

```
private static int countChanges(FileSystem fs, Path reduceClusterFile,
                              int iteration, JobConf conf)
    throws IOException {
      Path path = conf.getWorkingDirectory();
      Path inputClusterFile = new Path(path, clusterFileNameTemplate
            + String.valueOf(iteration));
      NumberedCenterPoint[] inputPoints =
         loadClusterFile(fs,inputClusterFile);
      NumberedCenterPoint[] reducePoints =
         loadClusterFile(fs,reduceClusterFile);
      Arrays.sort(inputPoints);
      Arrays.sort(reducePoints);
      int differCount = inputPoints.length;
      for (int i = 0; i < reducePoints.length; i++) {
         if (inputPoints[i].equals(reducePoints[i])) {
            differCount--;
         } else {
            // k-means can get trapped into small oscillations in points. If
            // the differences are small, assume they're close enough.
            double ipx = inputPoints[i].getX().doubleValue();
            double ipy = inputPoints[i].getY().doubleValue();
            double rpx = reducePoints[i].getX().doubleValue();
            double rpy = reducePoints[i].getY().doubleValue();
            double minDiff = 0.0000000001;
            if (Math.abs(ipx - rpx) < minDiff &&
                  Math.abs(ipy - rpy) < minDiff) {
               differCount--;
            }
         }
      }
      return differCount;
   }
```

Plotted Results



Conclusions

- Map-Reduce is a scalable and efficient method of analyzing large volumes of data in parallel.
- Hadoop provides a usable, effective, and reliable framework for Map-Reduce operations.

Bibliography

Dean, J., Ghemawat S. (2004) MapReduce: Simplified Data Processing on Large Clusters. Operating System Design and Implementation. Retrieved 25 June 2009 from http://labs.google.com/papers/mapreduce.html

Hadoop Map-Reduce Tutorial (2007) Retrieved 25 June 2009 from http://hadoop.apache.org/core/docs/r0.17.1/mapred_tutorial.pdf.

The Hadoop Distributed File System: Architecture and Design (2007) Retrieved 25 June 2009 from http://hadoop.apache.org/core/docs/r0.17.1/hdfs_design.pdf.

O'Malley, O. (2008) TeraByte Sort on Apache Hadoop. Retrieved 29 June 2009 at http://www.hpl.hp.com/hosted/sortbenchmark/YahooHadoop.pdf.

Yahoo! Launches World's Largest Hadoop Production Application, 2008. Retrieved 29 June 2009 from http://developer.yahoo.net/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html.

Stonebraker, M, et. al., MapReduce and Parallel DBMSs: Friends or Foes? CACM January 2010, pp. 64-71.

Dean, J. and Ghemawat, S., MapReduce: a Flexible Data Processing Tool. CACM January 2010, pp. 72-77.