Data Intensive Scalable Computing on TeraGrid: A Comparison of MapReduce and Swift

Quan T. Pham
Computer Science Department
The University of Chicago
quanpt@cs.uchicago.edu

Atilla S. Balkir
Computer Science Department
The University of Chicago
soner@cs.uchicago.edu

Jing Tie
Computer Science Department
The University of Chicago
jtie@cs.uchicago.edu

Ian Foster
Math and Compute Science Div.
Argonne National Lab
Computer Science Department
The University of Chicago
foster@mcs.anl.gov

Michael J. Wilde
Math and Compute Science Div.
Argonne National Lab
wilde@mcs.anl.gov

Ioan Raicu
Computer Science Department
The University of Chicago
iraicu@cs.uchicago.edu

Abstract

Many programming models and frameworks have been introduced to abstract away the management details of running applications in distributed environments.

MapReduce is regarded as a power-leveler that solves computation problems using brutal-force resources. It provides a simple programming model and powerful runtime system for processing large datasets. The model is based on two key functions: "map" and "reduce", and the runtime system automatically partitions input data and schedules the execution of programs in a large cluster of commodity machines. MapReduce has been applied to document processing problems, such as distributed indexing, sorting, and clustering.

Swift is a parallel programming tool for rapid and reliable specification, execution, and management of large-scale science and engineering workflows. Swift consists of a concise scripting language called SwiftScript for specifications of complex parallel computations based on dataset typing and iterations, and dynamic dataset mappings. The runtime system relies on CoG Karajan workflow engine for scheduling and load balancing, and integrates the Falkon light-weight task execution service for optimized task throughput, resource provisioning, and to leverage data-locality found in application access patterns.

Applications that can be implemented in MapReduce are a subset of those that can be implemented in Swift due to the more generic programming model found in Swift. Contrasting Swift and Hadoop are interesting as it could potentially attract new users and applications to systems which traditionally were not considered.

Methods

We compared two benchmarks, Sort and WordCount, and tested them at different scales and with different datasets. The testbed consisted of a 270 processor cluster TeraPort at UChicago. Hadoop the MapReduce implementation from Yahoo! was configured to use Hadoop Distributed File System HDFS, while Swift used Global Parallel File System GPFS.

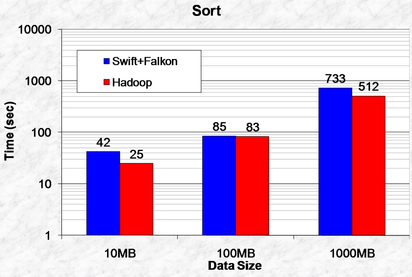
Input data size	
WordCount	large number of 10kB text files of total 70MB, 700MB
Sort	one file of size 10M, 100M, 1GB

Results

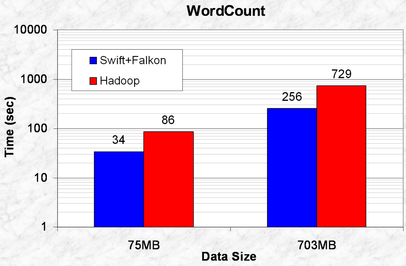
We found Swift offered comparable performance with Hadoop, a surprising finding due to the choice of benchmarks which favored the MapReduce model. In Sorting over a range of small to large files, Swift execution times were on average 38% higher when compared to Hadoop. However, for WordCount, Swift execution times were on average 60% lower. Our experience with Swift and Hadoop indicate that the file systems GPFS and Hadoop are the main bottlenecks as applications scale; HDFS is more scalable than GPFS, but it still has problems with small files, and it requires applications be modified. There are current efforts in Falkon to enable Swift to operate over local disks rather than shared file systems and to cache data across jobs, which would in turn offers comparable scalability and performance to HDFS without the added requirements of modifying applications. We plan to do additional experiments on the TeraGrid, with larger testbeds and data sets, as well as additional benchmarks.

Performance

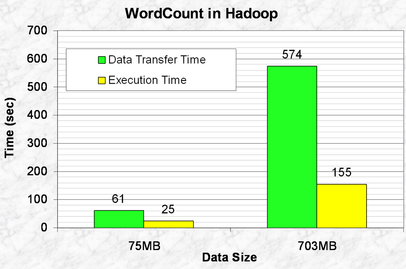
Hadoop better than Swift + Falkon



Swift + Falkon better than Hadoop



Why Swift + Falkon is faster?



Complexity

Sort in Swift: 50 lines of code

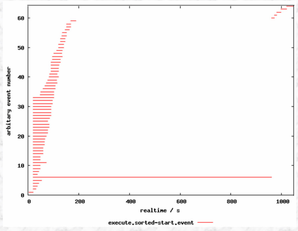
```
type inputfile {}
type sortedfile {}
type textfile {}

(sortedfile t) sort (string f, textfile fileList) {
  app {
    sort "-u" f stdout= @filename(t);
  }
}

(sortedfile sfiles[]) sportAll(textfile fileList, string ifiles[]) {
  foreach f,i in ifiles {
    sfiles[i] = sort(f, fileList);
  }
}

(sortedfile sf) merge(sortedfile f1, sortedfile f2) {
  app {
    merge -g @filename(f1) @filename(f2) stdout= @filename(sf);
  }
}

(sortedfile nfiles[]) mergeHalf(sortedfile sfiles[]) {
  foreach f,i in sfiles {
    if(i %% 2 == 1) {
      mfiles[%(i/2)=mergeo(sfiles[i-1], sfiles[i]);
    }
  }
}

(sortedfile rfiles[]) mergeHalfRec(sortedfile sfiles[], int k) {
  if(k >1) {
    sortedfile tmpfiles[];
    tmpfiles = mergeHalf(sfiles);
    mfiles = mergeHalfRec(tmpfiles, k -1);
  } else {
    mfiles = mergeHalf(sfiles);
  }
}

main (textfile fileList, string strFiles[]) {
  sortedfile sfiles[] <simple_mapper;location="./data",prefix="s",suffix=".sorted.g">;
  sortedfile rfiles[] <simple_mapper;location="./data",prefix="final",suffix=".data">;
  sfiles = sportAll(fileList, strFiles);
  rfiles= mergeHalfRec(sfiles, 1);
}

(textfile post_condition) prepareData(string localpath) {
  app {
    rplit "data.txt" 32 localpath stdout= @post_condition;
  }
}

string localpath = "/home/quanpt/swift/sort/data";
string strFiles[] = readdata("data/datalist.txt");
textfile fileList;
fileList = prepareData(localpath);
main(fileList, strFiles);
```

Sort in Hadoop: 96 lines of code

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.lib.IdentityReducer;
import org.apache.hadoop.util.ToolRunner;

public class WordCount extends Configured implements Tool {

  public static class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value,
                    OutputCollector<Text, IntWritable> output,
                    Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        output.collect(word, one);
      }
    }
  }

  public static class Reduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
                       OutputCollector<Text, IntWritable> output,
                       Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) {
        sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
    }
  }

  static int printUsage() {
    System.out.println("wordcount [-m <maps>] [-r <reduces>] -input> -output>");
    ToolRunner.printGenericCommandUsage(System.out);
    return -1;
  }

  public int run(String[] args)throws Exception {
    JobConf conf = new JobConf(getConf(), WordCount.class);
    conf.setJobName("wordcount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(MapClass.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    List<String> other_args = new ArrayList<String>();
    for(int i=0; i < args.length; ++i) {
      try {
        if("-m".equals(args[i])) {
          conf.setNumMapTasks(Integer.parseInt(args[++i]));
        } else if ("-r".equals(args[i])) {
          conf.setNumReduceTasks(Integer.parseInt(args[++i]));
        } else {
          other_args.add(args[i]);
        }
      } catch (NumberFormatException except) {
        System.out.println("ERROR: Integer expected instead of " + args[i]);
        return printUsage();
      } catch (ArrayIndexOutOfBoundsException except) {
        System.out.println("ERROR: Required parameter missing from " +
                           args[-1]);
        return printUsage();
      }
    }
    if (other_args.size() != 2) {
      System.out.println("ERROR: Wrong number of parameters: " +
                         other_args.size() + " instead of 2.");
      return printUsage();
    }
    conf.setInputPath(new Path(other_args.get(0)));
    conf.setOutputPath(new Path(other_args.get(1)));

    JobClient.runJob(conf);
    return 0;
  }

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

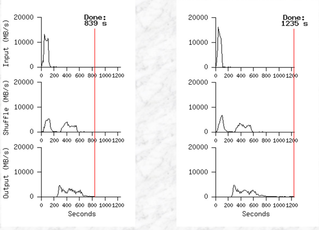
Error handling

Swift with an abnormal slow computing node



Notes: recent version of Swift has a new feature to resubmit slow tasks to other available resources .

Hadoop with an abnormal slow computing node



Use backup job submission

Not use backup job submission

http://labs.google.com/papers/mapreduce-osdi04-slides/

Conclusions

We conclude with three questions.

1) Can MapReduce applications run on workflow systems? We believe yes, and with even better performance in some cases.

2) Is the MapReduce model an option for scientific applications?

3) What parallel programming model will be best suited for scientific applications in the coming decade? We hope future work will help answer question 2 and 3.

Related material & Acknowledgment

[1] Zhao Y., Hategan, M., Clifford, B., Foster, I., vonLaszewski, G., Raicu, I., Stef-Praun, T. and Wilde, M "Swift: Fast, Reliable, Loosely Coupled Parallel Computation", IEEE International Workshop on Scientific Workflows 2007.

[2] J. Dean, S. Ghemawat "Mapreduce: simplified data processing on large clusters", Commun. ACM, vol. 51, no. 1, pp. 107-113, January 2008. Presentation slide available online at http://labs.google.com/papers/mapreduce-osdi04-slides/

[3] Hadoop website, available online at http://hadoop.apache.org/core/

[4] Ioan Raicu, Yong Zhao, Catalin Dumitrescu, Ian Foster, Mike Wilde "Falkon: a Fast and Light-weight tasK executiON framework", IEEE/ACM SuperComputing 2007.