Zebra and MapReduce

Table of contents

1 Overview....2
2 Hadoop MapReduce APIs....2
3 Zebra MapReduce APIs....2
4 Zebra MapReduce Examples....5

1. Overview

MapReduce allows you to take full advantage of Zebra's capabilities.

2. Hadoop MapReduce APIs

Zebra requires Hadoop 20. However, this release of Zebra only supports the "old" jobconf-style MapReduce APIs.

- "old" mapreduce API - org.apache.hadoop.mapred.* - supported
- "new" mapreduce API - org.apache.hadoop.mapreduce.* - not supported

3. Zebra MapReduce APIs

Zebra includes several classes for use in MapReduce programs. The main entry point into Zebra are the two classes for reading and writing tables, namely TableInputFormat and BasicTableOutputFormat.

3.1. BasicTableOutputFormat

Static	Method	Description
yes	void setOutputPath(JobConf, Path)	Set the output path of the BasicTable in JobConf
yes	Path[] getOutputPaths(JobConf)	Get the output paths of the BasicTable from JobConf
yes	void setStorageInfo(JobConf, ZebraSchema, ZebraStorageHint, ZebraSortInfo)	Set the table storage information (schema, storagehint, sortinfo) in JobConf
yes	Schema getSchema(JobConf)	Get the table schema in JobConf
yes	BytesWritable generateSortKey(JobConf, Tuple)	Generates a BytesWritable key for the input key
yes	String getStorageHint(JobConf)	Get the table storage hint in JobConf
yes	SortInfo getSortInfo(JobConf)	Get the SortInfo object

yes	void close(JobConf)	Close the output BasicTable, No more rows can be added into the table
yes	void setMultipleOutputs(JobConf, String commaSeparatedLocs, Class < extends ZebraOutputPartition> theClass)	Enables data to be written to multiple zebra tables based on the ZebraOutputPartition class. See Multiple Table Outputs.

3.2. TableInputFormat

Static	Method	Description
yes	void setInputPaths(JobConf, Path... paths)	Set the paths to the input table
yes	Path[] getInputPaths(JobConf)	Get the comma-separated paths to the input table or table union
yes	Schema getSchema(JobConf)	Get the schema of a table expr
yes	void setProjection(JobConf, ZebraProjection)	Set the input projection in the JobConf object
yes	String getProjection(JobConf)	Get the projection from the JobConf
yes	SortInfo getSortInfo(JobConf)	Get the SortInfo object regarding a Zebra table
yes	void requireSortedTable(JobConf, String sortcolumns, BytesComparator comparator)	Requires sorted table or table union
yes	TableRecordReader getTableRecordReader(JobConf ZebraProjection)	Get a TableRecordReader on a single split
yes	void setMinSplitSize(JobConf, long minSize)	Set the minimum split size, default of 1M bytes

3.3. TableRecordReader

Static	Method	Description

no	boolean seekTo(BytesWritable key)	Seek to the position at the first row which has the key (returning true) or just after the key(returning false); only applicable for sorted Zebra table.

3.4. ZebraOutputPartition

Static	Method	Description
no	public abstract int getOutputPartition(BytesWritable key, Tuple value)	Abstract method from ZebraOutputPartition abstract class. App implements this to stream data to different table
no	void setConf(Configuration jobConf)	Initialization routine giving JobConf to application. Zebra implements it
no	Configuration getConf()	returns JobConf. Zebra implements it
yes	Class< extends ZebraOutputPartition> getZebraOutputPartitionClass(Jo conf)	return user implemented ZebraOutputPartition class

3.5. ZebraProjection

Static	Method	Description
yes	ZebraProjection createZebraProjection(String)	Create a ZebraProjection object from a string representing projection information.

3.6. ZebraSchema

Static	Method	Description
yes	ZebraSchema createZebraSchema(String)	Create a ZebraStorageHint object from a string representing storage hint information.

3.7. ZebraStorageHint

Static	Method	Description
yes	ZebraStorageHint createZebraStorageHint(String)	Create a ZebraStorageHint object from a string representing storage hint information.

3.8. ZebraSortInfo

Static	Method	Description
yes	ZebraSortInfo createZebraSortInfo(String sortColumns, Class< extends RawComparator < Object >> comparatorClass)	Create a ZebraSortInfo object from a sort columns string and a comparator class.

4. Zebra MapReduce Examples

4.1. Table Output Format

This MapReduce example demonstrates the Zebra table output format. The Zebra table in this example has two unsorted columns groups, each of which has one column. The output format is specified as follows:

```
BasicTableOutputFormat.setStorageInfo(jobConf,
    ZebraSchema.createZebraSchema("word:string, count:int"),
    ZebraStorageHint.createZebraStorageHint("[word];[count]"),
    null);
```

The input file for this example should contain rows of word and count, separated by a space. For example:

```
this 2
is 1
a 5
test 2
hello 1
world 3
```

The example works like this. The first job is in Zebra format. The second job reads output from the first job, where Count is specified as a projection column. The table input format projects an input row which has both Word and Count into a row containing only the Count column and hands it to map. The reducer sums the counts and produces a sum of counts

which should match total number of words in original text.

```
package org.apache.hadoop.zebra.mapred;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.pig.data.Tuple;
import org.apache.hadoop.zebra.parser.ParseException;
import org.apache.hadoop.zebra.schema.Schema;
import org.apache.hadoop.zebra.types.TypesUtils;

import java.io.IOException;
import java.util.Iterator;

public class TableMapReduceExample extends Configured implements Tool {

  static class Map extends MapReduceBase implements Mapper<LongWritable,
Text, BytesWritable, Tuple> {
    private BytesWritable bytesKey;
    private Tuple tupleRow;

    /**
     * Map method for reading input.
     */
    @Override
    public void map(LongWritable key, Text value,
                    OutputCollector<BytesWritable, Tuple> output, Reporter
reporter)
        throws IOException {

      // value should contain "word count"
      String[] wordCount = value.toString().split(" ");
      if (wordCount.length != 2) {
        // LOG the error
        throw new IOException("Value does not contain two fields:" +
value);
      }

      byte[] word = wordCount[0].getBytes();
      bytesKey.set(word, 0, word.length);
      tupleRow.set(0, new String(word));
      tupleRow.set(1, Integer.parseInt(wordCount[1]));

      output.collect(bytesKey, tupleRow);
    }
```

```
    /**
     * Configuration of the job. Here we create an empty Tuple Row.
     */
    @Override
    public void configure(JobConf job) {
      bytesKey = new BytesWritable();
      try {
        Schema outSchema = BasicTableOutputFormat.getSchema(job);
        tupleRow = TypesUtils.createTuple(outSchema);
      } catch (IOException e) {
        throw new RuntimeException(e);
      } catch (ParseException e) {
        throw new RuntimeException(e);
      }
    }

  }

  static class ProjectionMap extends MapReduceBase implements
Mapper<BytesWritable, Tuple, Text, IntWritable> {
    private final static Text all = new Text("All");

    /**
     * Map method which gets count column after projection.
     *
     * @throws IOException
     */
    @Override
    public void map(BytesWritable key, Tuple value,
                    OutputCollector<Text, IntWritable> output, Reporter
reporter)
        throws IOException {
      output.collect(all, new IntWritable((Integer) value.get(0)));
    }
  }

  public static class ProjectionReduce extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable> {
    /**
     * Reduce method which implements summation. Acts as both reducer and
combiner.
     *
     * @throws IOException
     */
    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));
    }
```

```
  }

  /**
   * Where jobs and their settings and sequence is set.
   *
   * @param args arguments with exception of Tools understandable ones.
   */
  public int run(String[] args) throws Exception {
    if (args == null || args.length != 3) {
      System.out.println("usage: TableMapReduceExample
input_path_for_text_file output_path_for_table output_path_for_text_file");
      System.exit(-1);
    }

    /*
       First MR Job creating a Table with two columns
     */
    JobConf jobConf = new JobConf();
    jobConf.setJobName("TableMapReduceExample");
    jobConf.set("table.output.tfile.compression", "none");

    // Input settings
    jobConf.setInputFormat(TextInputFormat.class);
    jobConf.setMapperClass(Map.class);
    FileInputFormat.setInputPaths(jobConf, new Path(args[0]));

    // Output settings
    jobConf.setOutputFormat(BasicTableOutputFormat.class);
    BasicTableOutputFormat.setOutputPath(jobConf, new Path(args[1]));

    // set the storage info of logical schema with 2 columns;
    // and create 2 physical column groups;
    // unsorted table

    BasicTableOutputFormat.setStorageInfo(jobConf,
        ZebraSchema.createZebraSchema("word:string, count:int"),
        ZebraStorageHint.createZebraStorageHint("[word];[count]"), null);

    // set map-only job.
    jobConf.setNumReduceTasks(0);

    // Run Job
    JobClient.runJob(jobConf);

    // Need to close Zebra output streams
    BasicTableOutputFormat.close(jobConf);

    /*
      Second MR Job for Table Projection of count column
    */
    JobConf projectionJobConf = new JobConf();
    projectionJobConf.setJobName("TableProjectionMapReduceExample");

    // Input settings
```

```
    projectionJobConf.setMapperClass(ProjectionMap.class);
    projectionJobConf.setInputFormat(TableInputFormat.class);
    TableInputFormat.setProjection(projectionJobConf, "count");
    TableInputFormat.setInputPaths(projectionJobConf, new Path(args[1]));
    projectionJobConf.setMapOutputKeyClass(Text.class);
    projectionJobConf.setMapOutputValueClass(IntWritable.class);

    // Output settings
    projectionJobConf.setOutputFormat(TextOutputFormat.class);
    FileOutputFormat.setOutputPath(projectionJobConf, new Path(args[2]));
    projectionJobConf.setReducerClass(ProjectionReduce.class);
    projectionJobConf.setCombinerClass(ProjectionReduce.class);

    // Run Job
    JobClient.runJob(projectionJobConf);

    return 0;
  }

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

4.2. Table Input/Output Formats

This MapReduce examples demonstrates how to perform a simple union. To run this program, we need two basic tables that contain the data as in the example above (word, count). In this example they are: /user/mapredu/t1 and /user/mapredu/t2. The resulting table is /user/mapredu2/t.

```
 package org.apache.hadoop.zebra.mapred;

import java.io.IOException;
import java.util.List;
import java.util.ArrayList;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.zebra.mapred.BasicTableOutputFormat;
import org.apache.hadoop.zebra.mapred.TableInputFormat;
import org.apache.hadoop.zebra.parser.ParseException;
import org.apache.hadoop.zebra.schema.Schema;
import org.apache.hadoop.zebra.types.TypesUtils;
import org.apache.pig.data.Tuple;
```

```
public class TableMRSample2 {
  static class MapClass implements
      Mapper<BytesWritable, Tuple, BytesWritable, Tuple> {
    private BytesWritable bytesKey;
    private Tuple tupleRow;

    @Override
    public void map(BytesWritable key, Tuple value,
        OutputCollector<BytesWritable, Tuple> output, Reporter reporter)
        throws IOException

    {
      System.out.println(key.toString() + value.toString());
      output.collect(key, value);
    }

    @Override
    public void configure(JobConf job) {
      bytesKey = new BytesWritable();
      try {
        Schema outSchema = BasicTableOutputFormat.getSchema(job);
        tupleRow = TypesUtils.createTuple(outSchema);
      } catch (IOException e) {
        throw new RuntimeException(e);
      } catch (ParseException e) {
        throw new RuntimeException(e);
      }
    }

    @Override
    public void close() throws IOException {
      // no-op
    }

    public static void main(String[] args) throws ParseException,
IOException {
      JobConf jobConf = new JobConf();
      jobConf.setJobName("tableMRSample");
      jobConf.set("table.output.tfile.compression", "gz");

      // input settings
      jobConf.setInputFormat(TableInputFormat.class);
      jobConf.setOutputFormat(BasicTableOutputFormat.class);
      jobConf.setMapperClass(TableMRSample2.MapClass.class);

      List
      <Path> paths = new ArrayList<Path>(2);
      Path p = new Path("/user/mapredu/t1");
      System.out.println("path = " + p);
      paths.add(p);
      p = new Path("/user/mapredu/t2");
      paths.add(p);
```

```
      TableInputFormat.setInputPaths(jobConf, paths.toArray(new Path[2]));
      ZebraProjection zebraProjection =
ZebraProjection.createZebraProjection("word");
      TableInputFormat.setProjection(jobConf, zebraProjection);
      BasicTableOutputFormat.setOutputPath(jobConf, new
Path("/user/mapredu2/t1"));

      ZebraSchema zebraSchema =
ZebraSchema.createZebraSchema("word:string");
      ZebraStorageHint zebraStorageHint =
ZebraStorageHint.createZebraStorageHint("[word]");
      BasicTableOutputFormat.setStorageInfo(jobConf, zebraSchema,
zebraStorageHint, null);

      // set map-only job.
      jobConf.setNumReduceTasks(0);
      jobConf.setNumMapTasks(2);
      JobClient.runJob(jobConf);
    }
  }
}
```

4.3. Sort Columns

This MapReduce code snippet demonstrates how to sort Zebra columns.

```
/* user provides a Comparator Class for creating ZebraSortInfo */

public static final class MemcmpRawComparator implements
    RawComparator<Object>, Serializable {
      @Override
        public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2,
int l2) {
          return WritableComparator.compareBytes(b1, s1, l1, b2, s2, l2);
      }

      @Override
      public int compare(Object o1, Object o2) {
        throw new RuntimeException("Object comparison not supported");
      }
}

…
…

ZebraSchema zSchema = ZebraSchema.createZebraSchema(schemaString);

ZebraStorageHint zStorageHint =
ZebraStorageHint.createZebraStorageHint(storageHintString);

/* Here we can use above Comparator Class to create a ZebraSortInfo object
*/
```

```
ZebraSortInfo zSortInfo =
ZebraSortInfo.createZebraSortInfo(sortColumnsString,
MemcmpRawComparator.class);

BasicTableOutputFormat.setStorageInfo(jobConf, zSchema, zStorageHint,
zSortInfo);
```

4.4. Drop Column Groups

This example illustrates how to drop column groups (CG) in Zebra tables. This is not a MapReduce program since the API for deleting columg groups is a simple Java API.

How to compile:

```
# this command requires pig.jar and latest zebra jar.
$ javac -cp pig.jar:zebra-0.1.0.jar DropColumnGroupExample.java

# create a jar file
$ jar cvf dropcg.jar DropColumnGroupExample.class
```

How to run:

```
# run the example.
$ java -cp pig.jar:zebra-0.1.0.jar:dropcg.jar DropColumnGroupExample

# This creates a table under the directory "dropCGExample".
# If run the same command again, it fails since the destination
# directory still exists. Please remove the directory before running.

#This program takes one argument : directory for the example table
$ java -cp pig.jar:zebra-0.1.0.jar:dropcg.jar DropColumnGroupExample
tableDir
```

Source code:

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.zebra.io.BasicTable;
import org.apache.hadoop.zebra.io.TableInserter;
import org.apache.hadoop.zebra.io.TableScanner;
import org.apache.hadoop.zebra.types.TypesUtils;
import org.apache.pig.data.Tuple;

public class DropColumnGroupExample {

  private static String schema = "url, outcount:int, content:string";
  private static String storageHint = "[url, outcount]; [content] as data";
```

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

    // Create configuration and process generaic Hadoop options like -D.
    Configuration conf = new Configuration();
    args = new GenericOptionsParser(conf, args).getRemainingArgs();

    Path path = new Path((args.length > 0 ? args[0] : "dropCGExample"));

    if (path.getFileSystem(conf).exists(path)) {
      throw new IOException("Destination path '" + path + "' already
exists. " +
                            "Please remove the directory and run again.");
    }
    // create a simple table.
    createTable(conf, path);

    //verify we can read the table.
    String content = getContentForUrl(path, conf, "http://www.yahoo.com/");
    if (content == null || !content.contains("yahoo")) {
      throw new IOException("Table read failed.");
    }

    /* Now drop the colum group named "data".
     *
     * An exception is thrown if the CG could not be removed for some
reason
     * (e.g. the user might not have enough permissions).
     */
    BasicTable.dropColumnGroup(path, conf, "data");

    // deleting an already deleted CG is a no-op.
    BasicTable.dropColumnGroup(path, conf, "data");

    // now try to read content column.
    // Note that NULL is returned for the third column.
    if (getContentForUrl(path, conf, "http://www.yahoo.com/") != null) {
      throw new IOException("Expected NULL for 3rd column");
    }
    // While reading this table, a warning is logged since the user
    // is trying to reading from a deleted CG.

    //clean up the test directory.
    //for now we are not deleting the directory to let users check it.
    //BasicTable.drop(path, conf);
  }

  // Utility functions:

  /**
   * This is a simple table reader that iterates over the table to
   * find a given url and returns its content.
   */
  private static String getContentForUrl(Path path, Configuration conf,
```

```
                                          String url) throws Exception {

    BasicTable.Reader reader = new BasicTable.Reader(path, conf);
    TableScanner scanner = reader.getScanner(null, true);
    Tuple row = TypesUtils.createTuple(3);

    try {
      while (!scanner.atEnd()) {
        scanner.getValue(row);
        if (url.equals(row.get(0))) {
          return (String)row.get(2);
        }
        scanner.advance();
      }
    } finally {
      scanner.close();
    }
    throw new IOException(url + " is not found");
  }

  private static void createTable(Configuration conf, Path path)
                                  throws Exception {
    /* NOTE: This creates a table using BasicTable API. This is not
     * a committed public API yet. Typically tables are created
     * in M/R or through PIG.
     */
    BasicTable.Writer writer = new BasicTable.Writer(path, schema,
                                                     storageHint, conf);
    TableInserter inserter = writer.getInserter("part-0", true);
    Tuple rowTuple = TypesUtils.createTuple(3);
    BytesWritable emptyKey = new BytesWritable();

    // add two rows:
    rowTuple.set(0, "http://www.cnn.com/");
    rowTuple.set(1, 10);
    rowTuple.set(2, "content for cnn.com");

    inserter.insert(emptyKey, rowTuple);

    rowTuple.set(0, "http://www.yahoo.com/");
    rowTuple.set(1, 20);
    rowTuple.set(2, "content for yahoo.com");

    inserter.insert(emptyKey, rowTuple);

    inserter.close();
  }
}
```

4.5. Multiple Table Outputs

This code snippet illustrates how to work with multiple table outputs.

```
In main()

    String multiLocs = "/user/multi/us" + "," + "/user/multi/india" + "," +
"/user/multi/japan";

    jobConf.setOutputFormat(BasicTableOutputFormat.class);
    BasicTableOutputFormat.setMultipleOutputPaths(jobConf, multiLocs);
    BasicTableOutputFormat.setZebraOutputPartitionClass(jobConf,
MultipleOutputsTest.OutputPartitionerClass.class);


Implement a partition class:

  static class OutputPartitionerClass implements ZebraOutputPartition {
       @Override
       public int getOutputPartition(BytesWritable key, Tuple value, String
commaSeparatedLocs) {

          String reg = null;
          try {
            reg = (String)(value.get(0));
          } catch ( ExecException e) {
             // do something about e
          }

          if(reg.equals("us")) return 0;
          if(reg.equals("india")) return 1;
          if(reg.equals("japan")) return 2;

          return 0;
      }
  }
```