MAP REDUCE ALGORITHM FOR SORTINGpackage org.apache.hadoop.examples.terasort;

import java.io.IOException;

import java.io.PrintStream;

import java.net.URI;

import java.util.ArrayList;

import java.util.List;

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.filecache.DistributedCache;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.SequenceFile;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.mapred.Partitioner;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

/\*\*

\* Generates the sampled split points, launches the job, and waits for it

to

\* finish.

\* <p>

\* To run the program:

\* <b>bin/hadoop jar hadoop-\*-examples.jar terasort in-dir out-dir</b>

\*/

public class TeraSort extends Configured implements Tool {

private static final Log LOG = LogFactory.getLog(TeraSort.class);

/\*\*

\* A partitioner that splits text keys into roughly equal partitions

\* in a global sorted order.

\*/

static class TotalOrderPartitioner implements Partitioner<Text,Text>{

private TrieNode trie;

private Text[] splitPoints;

/\*\*

\* A generic trie node

\*/

static abstract class TrieNode {

private int level;

TrieNode(int level) {

this.level = level;

}

abstract int findPartition(Text key);

abstract void print(PrintStream strm) throws IOException;

int getLevel() {

return level;

}

}

/\*\*

\* An inner trie node that contains 256 children based on the next

\* character.

\*/

static class InnerTrieNode extends TrieNode {

private TrieNode[] child = new TrieNode[256];

InnerTrieNode(int level) {

super(level);

}

int findPartition(Text key) {

int level = getLevel();

if (key.getLength() <= level) {

return child[0].findPartition(key);

}

return child[key.getBytes()[level]].findPartition(key);

}

void setChild(int idx, TrieNode child) {

this.child[idx] = child;

}

void print(PrintStream strm) throws IOException {

for(int ch=0; ch < 255; ++ch) {

for(int i = 0; i < 2\*getLevel(); ++i) {

strm.print(' ');

}

strm.print(ch);

strm.println(" ->");

if (child[ch] != null) {

child[ch].print(strm);

}

}

}

}

/\*\*

\* A leaf trie node that does string compares to figure out where the

given

\* key belongs between lower..upper.

\*/

static class LeafTrieNode extends TrieNode {

int lower;

int upper;

Text[] splitPoints;

LeafTrieNode(int level, Text[] splitPoints, int lower, int upper) {

super(level);

this.splitPoints = splitPoints;

this.lower = lower;

this.upper = upper;

}

int findPartition(Text key) {

for(int i=lower; i<upper; ++i) {

if (splitPoints[i].compareTo(key) >= 0) {

return i;

}

}

return upper;

}

void print(PrintStream strm) throws IOException {

for(int i = 0; i < 2\*getLevel(); ++i) {

strm.print(' ');

}

strm.print(lower);

strm.print(", ");

strm.println(upper);

}

}

/\*\*

\* Read the cut points from the given sequence file.

\* @param fs the file system

\* @param p the path to read

\* @param job the job config

\* @return the strings to split the partitions on

\* @throws IOException

\*/

private static Text[] readPartitions(FileSystem fs, Path p,

JobConf job) throws IOException {

SequenceFile.Reader reader = new SequenceFile.Reader(fs, p, job);

List<Text> parts = new ArrayList<Text>();

Text key = new Text();

NullWritable value = NullWritable.get();

while (reader.next(key, value)) {

parts.add(key);

key = new Text();

}

reader.close();

return parts.toArray(new Text[parts.size()]);

}

/\*\*

\* Given a sorted set of cut points, build a trie that will find the

correct

\* partition quickly.

\* @param splits the list of cut points

\* @param lower the lower bound of partitions 0..numPartitions-1

\* @param upper the upper bound of partitions 0..numPartitions-1

\* @param prefix the prefix that we have already checked against

\* @param maxDepth the maximum depth we will build a trie for

\* @return the trie node that will divide the splits correctly

\*/

private static TrieNode buildTrie(Text[] splits, int lower, int upper,

Text prefix, int maxDepth) {

int depth = prefix.getLength();

if (depth >= maxDepth || lower == upper) {

return new LeafTrieNode(depth, splits, lower, upper);

}

InnerTrieNode result = new InnerTrieNode(depth);

Text trial = new Text(prefix);

// append an extra byte on to the prefix

trial.append(new byte[1], 0, 1);

int currentBound = lower;

for(int ch = 0; ch < 255; ++ch) {

trial.getBytes()[depth] = (byte) (ch + 1);

lower = currentBound;

while (currentBound < upper) {

if (splits[currentBound].compareTo(trial) >= 0) {

break;

}

currentBound += 1;

}

trial.getBytes()[depth] = (byte) ch;

result.child[ch] = buildTrie(splits, lower, currentBound, trial,

maxDepth);

}

// pick up the rest

trial.getBytes()[depth] = 127;

result.child[255] = buildTrie(splits, currentBound, upper, trial,

maxDepth);

return result;

}

public void configure(JobConf job) {

try {

FileSystem fs = FileSystem.getLocal(job);

Path partFile = new Path(TeraInputFormat.PARTITION\_FILENAME);

splitPoints = readPartitions(fs, partFile, job);

trie = buildTrie(splitPoints, 0, splitPoints.length, new Text(), 2);

} catch (IOException ie) {

throw new IllegalArgumentException("can't read paritions file", ie);

}

}

public TotalOrderPartitioner() {

}

public int getPartition(Text key, Text value, int numPartitions) {

return trie.findPartition(key);

}

}

public int run(String[] args) throws Exception {

LOG.info("starting");

JobConf job = (JobConf) getConf();

Path inputDir = new Path(args[0]);

inputDir = inputDir.makeQualified(inputDir.getFileSystem(job));

PathpartitionFile=newPath(inputDir,TeraInputFormat.PARTITION\_FILEN

AME);

URI partitionUri = new URI(partitionFile.toString() +

"#" + TeraInputFormat.PARTITION\_FILENAME);

TeraInputFormat.setInputPaths(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.setJobName("TeraSort");

job.setJarByClass(TeraSort.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setInputFormat(TeraInputFormat.class);

job.setOutputFormat(TeraOutputFormat.class);

job.setPartitionerClass(TotalOrderPartitioner.class);

TeraInputFormat.writePartitionFile(job, partitionFile);

DistributedCache.addCacheFile(partitionUri, job);

DistributedCache.createSymlink(job);

job.setInt("dfs.replication", 1);

TeraOutputFormat.setFinalSync(job, true);

JobClient.runJob(job);

LOG.info("done");

return 0;

}

/\*\*

\* @param args

\*/

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

int res = ToolRunner.run(new JobConf(), new TeraSort(), args);

System.exit(res);

}

}