Chapter 8

MapReduce Features

Learning Objectives

- Counters
- Sorting
- Joins
- Side Data Distribution

Counters

- Counters are a useful channel for gathering statistics about the job:
  - for quality control;
  - for application-level statistics;
  - for problem diagnosis (# of invalid records);
  - Easier to use and to retrieve than log output.

Built-In Counters

- Built-in counters for every job
  - Report various metrics for Map-Reduce jobs.
  - Counters for the number of bytes and records processed.

Table 8-1. Built-in counter groups

Group	Name/Enum	Reference
MapReduce task counters	`org.apache.hadoop.mapred.Task$Counter` (1.x) `org.apache.hadoop.mapreduce.TaskCounter` (post-1.x)	Table 8-2
Filesystem counters	`FileSystemCounters` (1.x) `org.apache.hadoop.mapreduce.FileSystemCounter` (post 1.x)	Table 8-3
FileInputFormat counters	`org.apache.hadoop.mapred.FileInputFormat$Counter` (1.x) `org.apache.hadoop.mapreduce.lib.input.FileInputFormatCounter` (post-1.x)	Table 8-4
FileOutputFormat counters	`org.apache.hadoop.mapred.FileOutputFormat$Counter` (1.x) `org.apache.hadoop.mapreduce.lib.output.FileOutputFormatCounter` (post-1.x)	Table 8-5
Job counters	`org.apache.hadoop.mapred.JobInProgress$Counter` (1.x) `org.apache.hadoop.mapreduce.JobCounter` (post-1.x)	Table 8-6

Task and Job Counters

- **Task counters** gather information about tasks over the course of their execution.
  - MAP_INPUT_RECORDS counter counts the input records read by each map task;
  - Then aggregates over all map tasks in a job.
- **Job counters** are maintained by the jobtracker (or application master in YARN).
  - Measure job-level statistics,
  - TOTAL_LAUNCHED_MAPS counts the number of map tasks that were launched (including tasks that failed).

User-Defined Java Counters – 1

- MapReduce allows user code to define a set of counters, which are then incremented as desired in the mapper or reducer.
- Counters are defined by a Java enum, which serves to group related counters.
  - The name of the enum is the group name.
  - The enum's fields are the counter names.

User-Defined Java Counters – 2

- P. 264: Example 8-1

Retrieving Counters – P. 267

Example 8-2. Application to calculate the proportion of records with missing temperature fields

```
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class MissingTemperatureFields extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {
    if (args.length != 1) {
      JobBuilder.printUsage(this, "<job ID>");
      return -1;
    }
    String jobID = args[0];
    JobClient jobClient = new JobClient(new JobConf(getConf()));
    RunningJob job = jobClient.getJob(JobID.forName(jobID));
    if (job == null) {
      System.err.printf("No job with ID %s found.\n", jobID);
      return -1;
    }
    if (!job.isComplete()) {
      System.err.printf("Job %s is not complete.\n", jobID);
      return -1;
    }

    Counters counters = job.getCounters();
    long missing = counters.getCounter(
        MaxTemperatureWithCounters.Temperature.MISSING);

    long total = counters.getCounter(Task.Counter.MAP_INPUT_RECORDS);

    System.out.printf("Records with missing temperature fields: %.2f%%\n",
        100.0 * missing / total);
    return 0;
  }
  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new MissingTemperatureFields(), args);
    System.exit(exitCode);
  }
}
```

Sorting

- The ability to sort data is at the heart of MapReduce.
- Even if your application isn't concerned with sorting per se, it may be able to use the sorting stage that MapReduce provides to organize its data.
- P. 269: Example 8-3.

Sorting a Sequence File with IntWritable – P. 270

Example 8-4. A MapReduce program for sorting a SequenceFile with IntWritable keys using the default HashPartitioner

```
public class SortByTemperatureUsingHashPartitioner extends Configured
  implements Tool {

  @Override
  public int run(String[] args) throws Exception {
    Job job = JobBuilder.parseInputAndOutput(this, getConf(), args);
    if (job == null) {
      return -1;
    }

    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);
    SequenceFileOutputFormat.setCompressOutput(job, true);
    SequenceFileOutputFormat.setOutputCompressorClass(job, GzipCodec.class);
    SequenceFileOutputFormat.setOutputCompressionType(job,
        CompressionType.BLOCK);

    return job.waitForCompletion(true) ? 0 : 1;
  }

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

Produce MapFiles as Output – 271

Example 8-5. A MapReduce program for sorting a SequenceFile and producing MapFiles as output

```
public class SortByTemperatureToMapFile extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {
    Job job = JobBuilder.parseInputAndOutput(this, getConf(), args);
    if (job == null) {
      return -1;
    }

    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputFormatClass(MapFileOutputFormat.class);
    SequenceFileOutputFormat.setCompressOutput(job, true);
    SequenceFileOutputFormat.setOutputCompressorClass(job, GzipCodec.class);
    SequenceFileOutputFormat.setOutputCompressionType(job,
        CompressionType.BLOCK);

    return job.waitForCompletion(true) ? 0 : 1;
  }

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

Retrieve the First Entry

Example 8-6. Retrieve the first entry with a given key from a collection of MapFiles

```
public class LookupRecordByTemperature extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {
    if (args.length != 2) {
      JobBuilder.printUsage(this, "<path> <key>");
      return -1;
    }
    Path path = new Path(args[0]);
    IntWritable key = new IntWritable(Integer.parseInt(args[1]));

    Reader[] readers = MapFileOutputFormat.getReaders(path, getConf());
    Partitioner<IntWritable, Text> partitioner =
      new HashPartitioner<IntWritable, Text>();
    Text val = new Text();
    Writable entry =
      MapFileOutputFormat.getEntry(readers, partitioner, key, val);
    if (entry == null) {
      System.err.println("Key not found: " + key);
      return -1;
    }
    NcdcRecordParser parser = new NcdcRecordParser();
    parser.parse(val.toString());
    System.out.printf("%s\t%s\n", parser.getStationId(), parser.getYear());
    return 0;
  }

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

Total Sort

- Produce a set of sorted files that, if concatenated, would form a globally sorted file.
  - Use a partitioner that respects the total order of the output.
- Although this approach works, you have to choose your partition sizes carefully to ensure that they are fairly even so that job times aren't dominated by a single reducer.

Temperature range	< –10°C	[–10°C, 0°C)	[0°C, 10°C)	>= 10°C
Proportion of records	11%	13%	17%	59%

Sampling

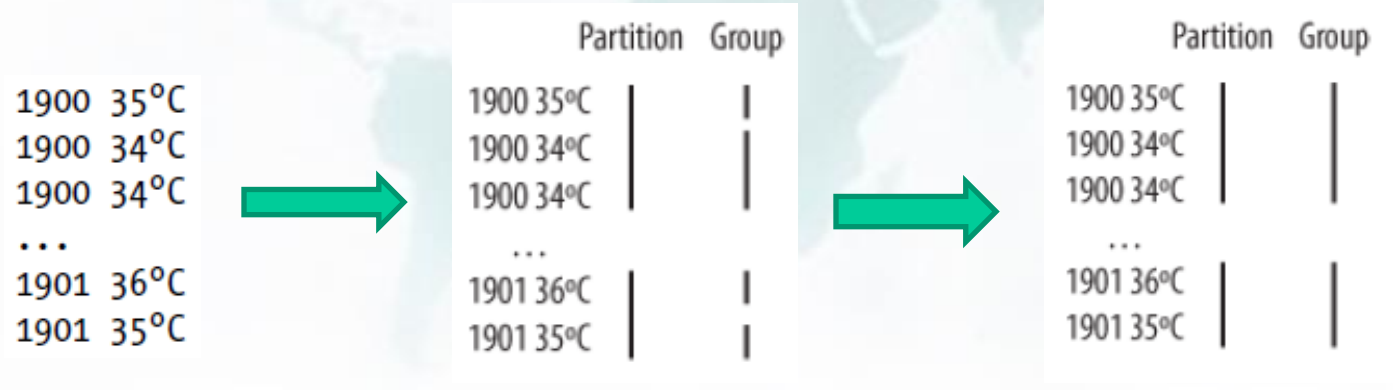
- To construct more even partitions, we need to have a better understanding of the distribution for the whole dataset.
- It's possible to get a fairly even set of partitions, by *sampling* the key space.
- The idea behind sampling is that you look at a small subset of the keys to approximate key distribution, which is then used to construct partitions.

Global Sort

- P. 276: Example 8-8

Secondary Sort – 1

- Example: calculating the maximum temperature for each year.
- It is possible to impose an order on the values by sorting and grouping the keys in a particular way.
  - Make the key a composite of the natural key and the natural value.
  - The key comparator should order by the composite key, that is, the natural key and natural value.
  - The partitioner and grouping comparator for the composite key should consider only the natural key for partitioning and grouping.



Secondary Sort – 2

- P. 279: Example 8-9

Map-Side Join – 1

- A map-side join works by performing the join before the data reaches the map function. Requirements:
  - Each input dataset must be divided into the same number of partitions.
  - It must be sorted by the same key (the join key) in each source.
- Above requirements actually fit the description of the output of a MapReduce job:
  - A map-side join can be used to join the outputs of several jobs that had the same number of reducers, the same keys, and output files that are not splittable.

Map-Side Join – 2

Stations

Station ID	Station Name
011990-99999	SIHCCAJAVRI
012650-99999	TYNSET-HANSMOEN

Records

Station ID	Timestamp	Temperature
012650-99999	194903241200	111
012650-99999	194903241800	78
011990-99999	195005150700	0
011990-99999	195005151200	22
011990-99999	195005151800	-11

Join

Station ID	Station Name	Timestamp	Temperature
011990-99999	SIHCCAJAVRI	195005150700	0
011990-99999	SIHCCAJAVRI	195005151200	22
011990-99999	SIHCCAJAVRI	195005151800	-11
012650-99999	TYNSET-HANSMOEN	194903241200	111
012650-99999	TYNSET-HANSMOEN	194903241800	78

Reduce-Side Join – 1

- Reduce-side join is more general than a map-side join:
  - Input datasets don't have to be structured in any particular way.
  - Less efficient as both datasets have to go through the MapReduce shuffle.
  - The mapper tags each record with its source.
  - Uses the join key as the map output key so that the records with the same key are brought together in the reducer.

Reduce-Side Join – 2

Example 8-12. Mapper for tagging station records for a reduce-side join

```
public class JoinStationMapper
    extends Mapper<LongWritable, Text, TextPair, Text> {
  private NcdcStationMetadataParser parser = new NcdcStationMetadataParser();

  @Override
  protected void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {
    if (parser.parse(value)) {
      context.write(new TextPair(parser.getStationId(), "0"),
          new Text(parser.getStationName()));
    }
  }
}
```

Example 8-13. Mapper for tagging weather records for a reduce-side join

```
public class JoinRecordMapper
    extends Mapper<LongWritable, Text, TextPair, Text> {
  private NcdcRecordParser parser = new NcdcRecordParser();

  @Override
  protected void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {
    parser.parse(value);
    context.write(new TextPair(parser.getStationId(), "1"), value);
  }

}
```

Example 8-14. Reducer for joining tagged station records with tagged weather records

```
public class JoinReducer extends Reducer<TextPair, Text, Text, Text> {

  @Override
  protected void reduce(TextPair key, Iterable<Text> values, Context context)
      throws IOException, InterruptedException {
    Iterator<Text> iter = values.iterator();
    Text stationName = new Text(iter.next());
    while (iter.hasNext()) {
      Text record = iter.next();
  Text outValue = new Text(stationName.toString() + "\t" + record.toString());
  context.write(key.getFirst(), outValue);
}
```

Side Data Distribution

- Side data are extra read-only data needed by a job to process the main dataset.
- The challenge is to make side data available to all the map or reduce tasks (which are spread across the cluster).
  - Using the Job Configuration
  - Distributed Cache

Using the Job Configuration

- Set arbitrary key-value pairs in the job configuration using the various setter methods on **Configuration**.
- Useful if one needs to pass a small piece of metadata to tasks.
- Don’t use this mechanism for transferring more than a few kilobytes of data.

Using Distributed Cache – 1

- Distribute datasets using Hadoop's distributed cache mechanism.
- Provides a service for copying files and archives to the task nodes in time for the tasks to use them when they run.
- GenericOptionsParser – specify the files to be distributed as a comma-separated list of URIs as the argument to the **-files** option.

```
% hadoop jar job.jar MaxTemperatureByStationNameUsingDistributedCacheFile \
  -files input/ncdc/metadata/stations-fixed-width.txt input/ncdc/all output
```

Using Distributed Cache – 2

- P. 290: Example 8-16

Using Distributed Cache – 3

- Hadoop copies the file specified by the –file and -archives options to the jobtracker's filesystem (normally HDFS).
- Before a task is run, the tasktracker copies the files from the jobtracker's filesystem to a local disk—the cache—so the task can access the files.
- The tasktracker also maintains a reference count for the number of tasks using each file in the cache.
- After the task has run, the file's reference count is decreased by one, and when it reaches zero it is eligible for deletion.
- Files are deleted to make room for a new file when the cache exceeds a certain size—10 GB by default.