2017.09

Hadoop离线大数据分析

MapReduce特性

计数器

- 用户想要充分了解数据
- 计数器用来收集作业统计
- 用来记录某一特定发生的时间
- 获取计数器比获取日志更方便
- 分析计数器比分析日志更方便

内置计数器

Table 9-1. Built-in counter groups

Group	Name/Enum	Reference
MapReduce task counters	`org.apache.hadoop.mapreduce.TaskCounter`	Table 9-2
Filesystem counters	`org.apache.hadoop.mapreduce.FileSystemCounter`	Table 9-3
`FileInputFormat` counters	`org.apache.hadoop.mapreduce.lib.input.FileInputFormatCounter`	Table 9-4
`FileOutputFormat` counters	`org.apache.hadoop.mapreduce.lib.output.FileOutputFormatCounter`	Table 9-5
Job counters	`org.apache.hadoop.mapreduce.JobCounter`	Table 9-6

MR特性

Table 9-2. Built-in MapReduce task counters

Counter	Description
Map input records (MAP_INPUT_RECORDS)	The number of input records consumed by all the maps in the job. Incremented every time a record is read from a RecordReader and passed to the map's map() method by the framework.
Split raw bytes (SPLIT_RAW_BYTES)	The number of bytes of input-split objects read by maps. These objects represent the split metadata (that is, the offset and length within a file) rather than the split data itself, so the total size should be small.
Map output records (MAP_OUTPUT_RECORDS)	The number of map output records produced by all the maps in the job. Incremented every time the collect() method is called on a map's OutputCollector.
Map output bytes (MAP_OUTPUT_BYTES)	The number of bytes of uncompressed output produced by all the maps in the job. Incremented every time the collect() method is called on a map's OutputCollector.
Map output materialized bytes (MAP_OUTPUT_MATERIALIZED_BYTES)	The number of bytes of map output actually written to disk. If map output compression is enabled, this is reflected in the counter value.
Combine input records (COMBINE_INPUT_RECORDS)	The number of input records consumed by all the combiners (if any) in the job. Incremented every time a value is read from the combiner's iterator over values. Note that this count is the number of values consumed by the combiner, not the number of distinct key groups (which would not be a useful metric, since there is not necessarily one group per key for a combiner; see "Combiner Functions" on page 34, and also "Shuffle and Sort" on page 197).
Combine output records (COMBINE_OUTPUT_RECORDS)	The number of output records produced by all the combiners (if any) in the job. Incremented every time the collect() method is called on a combiner's OutputCollector.
Reduce input groups (REDUCE_INPUT_GROUPS)	The number of distinct key groups consumed by all the reducers in the job. Incremented every time the reducer's reduce() method is called by the framework.
Reduce input records (REDUCE_INPUT_RECORDS)	The number of input records consumed by all the reducers in the job. Incremented every time a value is read from the reducer's iterator over values. If reducers consume all of their inputs, this count should be the same as the count for map output records.

任务计数器

Table 9-3. Built-in filesystem task counters

Counter	Description
Filesystem bytes read (BYTES_READ)	The number of bytes read by the filesystem by map and reduce tasks. There is a counter for each filesystem, and *Filesystem* may be Local, HDFS, S3, etc.
Filesystem bytes written (BYTES_WRITTEN)	The number of bytes written by the filesystem by map and reduce tasks.
Filesystem read ops (READ_OPS)	The number of read operations (e.g., open, file status) by the filesystem by map and reduce tasks.
Filesystem large read ops (LARGE_READ_OPS)	The number of large read operations (e.g., list directory for a large directory) by the filesystem by map and reduce tasks.
Filesystem write ops (WRITE_OPS)	The number of write operations (e.g., create, append) by the filesystem by map and reduce tasks.

Table 9-4. Built-in FileInputFormat task counters

Counter	Description
Bytes read (BYTES_READ)	The number of bytes read by map tasks via the `FileInputFormat`.

Table 9-5. Built-in FileOutputFormat task counters

Counter	Description
Bytes written (BYTES_WRITTEN)	The number of bytes written by map tasks (for map-only jobs) or reduce tasks via the `FileOutputFormat`.

MR特性

Table 9-6. Built-in job counters

Counter	Description
Launched map tasks (TOTAL_LAUNCHED_MAPS)	The number of map tasks that were launched. Includes tasks that were started speculatively (see "Speculative Execution" on page 204).
Launched reduce tasks (TOTAL_LAUNCHED_REDUCES)	The number of reduce tasks that were launched. Includes tasks that were started speculatively.
Launched uber tasks (TOTAL_LAUNCHED_UBERTASKS)	The number of uber tasks (see "Anatomy of a MapReduce Job Run" on page 185) that were launched.
Maps in uber tasks (NUM_UBER_SUBMAPS)	The number of maps in uber tasks.
Reduces in uber tasks (NUM_UBER_SUBREDUCES)	The number of reduces in uber tasks.
Failed map tasks (NUM_FAILED_MAPS)	The number of map tasks that failed. See "Task Failure" on page 193 for potential causes.
Failed reduce tasks (NUM_FAILED_REDUCES)	The number of reduce tasks that failed.
Failed uber tasks (NUM_FAILED_UBERTASKS)	The number of uber tasks that failed.
Killed map tasks (NUM_KILLED_MAPS)	The number of map tasks that were killed. See "Task Failure" on page 193 for potential causes.
Killed reduce tasks (NUM_KILLED_REDUCES)	The number of reduce tasks that were killed.
Data-local map tasks (DATA_LOCAL_MAPS)	The number of map tasks that ran on the same node as their input data.
Rack-local map tasks (RACK_LOCAL_MAPS)	The number of map tasks that ran on a node in the same rack as their input data, but were not data-local.
Other local map tasks (OTHER_LOCAL_MAPS)	The number of map tasks that ran on a node in a different rack to their input data. Inter-rack bandwidth is scarce, and Hadoop tries to place map tasks close to their input data, so this count should be low. See Figure 2-2.
Total time in map tasks (MILLIS_MAPS)	The total time taken running map tasks, in milliseconds. Includes tasks that were started speculatively. See also corresponding counters for measuring core and memory usage (VCORES_MILLIS_MAPS and MB_MILLIS_MAPS).
Total time in reduce tasks (MILLIS_REDUCES)	The total time taken running reduce tasks, in milliseconds. Includes tasks that were started speculatively. See also corresponding counters for measuring core and memory usage (VCORES_MILLIS_REDUCES and MB_MILLIS_REDUCES).

MR特性

```
public class MaxTemperatureWithCounters extends Configured implements Tool {

  enum Temperature {
    MISSING,
    MALFORMED
  }

  static class MaxTemperatureMapperWithCounters
      extends Mapper<LongWritable, Text, Text, IntWritable> {

    private NcdcRecordParser parser = new NcdcRecordParser();

    @Override
    protected void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

      parser.parse(value);
      if (parser.isValidTemperature()) {
        int airTemperature = parser.getAirTemperature();
        context.write(new Text(parser.getYear()),
            new IntWritable(airTemperature));
      } else if (parser.isMalformedTemperature()) {
        System.err.println("Ignoring possibly corrupt input: " + value);
        context.getCounter(Temperature.MALFORMED).increment(1);
      } else if (parser.isMissingTemperature()) {
        context.getCounter(Temperature.MISSING).increment(1);
      }

      // dynamic counter
      context.getCounter("TemperatureQuality", parser.getQuality()).increment(1);
    }
  }
```

自定义Counter

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

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


    job.setMapperClass(MaxTemperatureMapperWithCounters.class);
    job.setCombinerClass(MaxTemperatureReducer.class);
    job.setReducerClass(MaxTemperatureReducer.class);

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

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

排序

- 输出结果部分排序
- key的排序是根据WritableComparable中的compareTo方法
- 全局排序：
- 使用partitioner，将结果分为从大到小的不同区
- 不同区内的数据进行排序

排序

- 输出结果部分排序
- key的排序是根据WritableComparable中的compareTo方法
- 全局排序：
- 使用partitioner，将结果分为从大到小的不同区
- 不同区内的数据进行排序

2017.08

THE END