Rumen

Table of contents

1. Overview

Rumen is a data extraction and analysis tool built for *Apache Hadoop*. *Rumen* mines *JobHistory* logs to extract meaningful data and stores it in an easily-parsed, condensed format or *digest*. The raw trace data from MapReduce logs are often insufficient for simulation, emulation, and benchmarking, as these tools often attempt to measure conditions that did not occur in the source data. For example, if a task ran locally in the raw trace data but a simulation of the scheduler elects to run that task on a remote rack, the simulator requires a runtime its input cannot provide. To fill in these gaps, Rumen performs a statistical analysis of the digest to estimate the variables the trace doesn't supply. Rumen traces drive both Gridmix (a benchmark of Hadoop MapReduce clusters) and Mumak (a simulator for the JobTracker).

1.1. Motivation

- Extracting meaningful data from *JobHistory* logs is a common task for any tool built to work on *MapReduce*. It is tedious to write a custom tool which is so tightly coupled with the *MapReduce* framework. Hence there is a need for a built-in tool for performing framework level task of log parsing and analysis. Such a tool would insulate external systems depending on job history against the changes made to the job history format.
- Performing statistical analysis of various attributes of a *MapReduce Job* such as *task runtimes, task failures etc* is another common task that the benchmarking and simulation tools might need. *Rumen* generates *Cumulative Distribution Functions (CDF)* for the Map/Reduce task runtimes. Runtime CDF can be used for extrapolating the task runtime of incomplete, missing and synthetic tasks. Similarly CDF is also computed for the total number of successful tasks for every attempt.

1.2. Components

Rumen consists of 2 components

- *Trace Builder* : Converts *JobHistory* logs into an easily-parsed format. Currently `TraceBuilder` outputs the trace in *JSON* format.
- *Folder* : A utility to scale the input trace. A trace obtained from *TraceBuilder* simply summarizes the jobs in the input folders and files. The time-span within which all the jobs in a given trace finish can be considered as the trace runtime. *Folder* can be used to scale the runtime of a trace. Decreasing the trace runtime might involve dropping some jobs from the input trace and scaling down the runtime of remaining jobs. Increasing the trace runtime might involve adding some dummy jobs to the resulting trace and scaling up the runtime of individual jobs.

2. How to use Rumen?

Converting *JobHistory* logs into a desired job-trace consists of 2 steps

1. Extracting information into an intermediate format
2. Adjusting the job-trace obtained from the intermediate trace to have the desired properties.

> **Note:**
> Extracting information from *JobHistory* logs is a one time operation. This so called *Gold Trace* can be reused to generate traces with desired values of properties such as `output-duration`, `concentration` etc.

Rumen provides 2 basic commands

- `TraceBuilder`
- `Folder`

Firstly, we need to generate the *Gold Trace*. Hence the first step is to run `TraceBuilder` on a job-history folder. The output of the `TraceBuilder` is a job-trace file (and an optional cluster-topology file). In case we want to scale the output, we can use the `Folder` utility to fold the current trace to the desired length. The remaining part of this section explains these utilities in detail.

> **Note:**
> Examples in this section assumes that certain libraries are present in the java CLASSPATH. See *Section-3.2* for more details.

2.1. Trace Builder

`Command:`

```
java org.apache.hadoop.tools.rumen.TraceBuilder [options] <jobtrace-output>
<topology-output> <inputs>
```

This command invokes the `TraceBuilder` utility of *Rumen*. It converts the JobHistory files into a series of JSON objects and writes them into the `<jobtrace-output>` file. It also extracts the cluster layout (topology) and writes it in the`<topology-output>` file. `<inputs>` represents a space-separated list of JobHistory files and folders.

> **Note:**
> 1) Input and output to `TraceBuilder` is expected to be a fully qualified FileSystem path. So use '*file://*' to specify files on the `local` FileSystem and '*hdfs://*' to specify files on HDFS. Since input files or folder are FileSystem paths, it means that they can be globbed. This can be useful while specifying multiple file paths using regular expressions.

Note:

2) By default, TraceBuilder does not recursively scan the input folder for job history files. Only the files that are directly placed under the input folder will be considered for generating the trace. To add all the files under the input directory by recursively scanning the input directory, use '-recursive' option.

Cluster topology is used as follows :

- To reconstruct the splits and make sure that the distances/latencies seen in the actual run are modeled correctly.
- To extrapolate splits information for tasks with missing splits details or synthetically generated tasks.

`Options :`

Parameter	Description	Notes
`-demuxer`	Used to read the jobhistory files. The default is `DefaultInputDemuxer`.	Demuxer decides how the input file maps to jobhistory file(s). Job history logs and job configuration files are typically small files, and can be more effectively stored when embedded in some container file format like SequenceFile or TFile. To support such usage cases, one can specify a customized Demuxer class that can extract individual job history logs and job configuration files from the source files.
`-recursive`	Recursively traverse input paths for job history logs.	This option should be used to inform the TraceBuilder to recursively scan the input paths and process all the files under it. Note that, by default, only the history logs that are directly under the input folder are considered for generating the trace.

2.1.1. Example

```
java org.apache.hadoop.tools.rumen.TraceBuilder
file:///home/user/job-trace.json file:///home/user/topology.output
file:///home/user/logs/history/done
```

This will analyze all the jobs in `/home/user/logs/history/done` stored on the `local` FileSystem and output the jobtraces in `/home/user/job-trace.json` along with topology information in `/home/user/topology.output`.

2.2. Folder

`Command`:

```
java org.apache.hadoop.tools.rumen.Folder [options] [input] [output]
```

Note:

Input and output to `Folder` is expected to be a fully qualified FileSystem path. So use '*file://*' to specify files on the `local` FileSystem and '*hdfs://*' to specify files on HDFS.

This command invokes the `Folder` utility of *Rumen*. Folding essentially means that the output duration of the resulting trace is fixed and job timelines are adjusted to respect the final output duration.

`Options :`

Parameter	Description	Notes
`-input-cycle`	Defines the basic unit of time for the folding operation. There is no default value for `input-cycle`. **Input cycle must be provided**.	'`-input-cycle 10m`' implies that the whole trace run will be now sliced at a 10min interval. Basic operations will be done on the 10m chunks. Note that *Rumen* understands various time units like *m(min), h(hour), d(days) etc*.
`-output-duration`	This parameter defines the final runtime of the trace. Default value if **1 hour**.	'`-output-duration 30m`' implies that the resulting trace will have a max runtime of 30mins. All the jobs in the input trace file will be folded and scaled to fit this window.
`-concentration`	Set the concentration of the resulting trace. Default value is **1**.	If the total runtime of the resulting trace is less than the total runtime of the input trace, then the resulting trace would

		contain lesser number of jobs as compared to the input trace. This essentially means that the output is diluted. To increase the density of jobs, set the concentration to a higher value.
`-debug`	Run the Folder in debug mode. By default it is set to **false**.	In debug mode, the Folder will print additional statements for debugging. Also the intermediate files generated in the scratch directory will not be cleaned up.
`-seed`	Initial seed to the Random Number Generator. By default, a Random Number Generator is used to generate a seed and the seed value is reported back to the user for future use.	If an initial seed is passed, then the `Random Number Generator` will generate the random numbers in the same sequence i.e the sequence of random numbers remains same if the same seed is used. Folder uses Random Number Generator to decide whether or not to emit the job.
`-temp-directory`	Temporary directory for the Folder. By default the **output folder's parent directory** is used as the scratch space.	This is the scratch space used by Folder. All the temporary files are cleaned up in the end unless the Folder is run in `debug` mode.
`-skew-buffer-length`	Enables *Folder* to tolerate skewed jobs. The default buffer length is **0**.	'`-skew-buffer-length 100`' indicates that if the jobs appear out of order within a window size of 100, then they will be emitted in-order by the folder. If a job appears out-of-order outside this window, then the Folder will bail out provided `-allow-missorting` is not set. *Folder* reports the maximum skew size seen in the input trace for future use.
`-allow-missorting`	Enables *Folder* to tolerate out-of-order jobs. By default mis-sorting is not allowed.	If mis-sorting is allowed, then the *Folder* will ignore out-of-order jobs that cannot be

		deskewed using a skew buffer of size specified using `-skew-buffer-length`. If mis-sorting is not allowed, then the Folder will bail out if the skew buffer is incapable of tolerating the skew.

2.2.1. Examples

2.2.1.1. Folding an input trace with 10 hours of total runtime to generate an output trace with 1 hour of total runtime

```
java org.apache.hadoop.tools.rumen.Folder -output-duration 1h  -input-cycle
20m  file:///home/user/job-trace.json file:///home/user/job-trace-1hr.json
```

If the folded jobs are out of order then the command will bail out.

2.2.1.2. Folding an input trace with 10 hours of total runtime to generate an output trace with 1 hour of total runtime and tolerate some skewness

```
java org.apache.hadoop.tools.rumen.Folder -output-duration 1h -input-cycle
20m  -allow-missorting -skew-buffer-length 100
file:///home/user/job-trace.json file:///home/user/job-trace-1hr.json
```

If the folded jobs are out of order, then atmost 100 jobs will be de-skewed. If the 101st job is *out-of-order*, then the command will bail out.

2.2.1.3. Folding an input trace with 10 hours of total runtime to generate an output trace with 1 hour of total runtime in debug mode

```
java org.apache.hadoop.tools.rumen.Folder -output-duration 1h -input-cycle
20m  -debug -temp-directory file:///tmp/debug
file:///home/user/job-trace.json file:///home/user/job-trace-1hr.json
```

This will fold the 10hr job-trace file `file:///home/user/job-trace.json` to finish within 1hr and use `file:///tmp/debug` as the temporary directory. The intermediate files in the temporary directory will not be cleaned up.

2.2.1.4. Folding an input trace with 10 hours of total runtime to generate an output trace with 1 hour of total runtime with custom concentration.

```
java org.apache.hadoop.tools.rumen.Folder -output-duration 1h -input-cycle
20m  -concentration 2  file:///home/user/job-trace.json
file:///home/user/job-trace-1hr.json
```

This will fold the 10hr job-trace file `file:///home/user/job-trace.json` to finish within 1hr with concentration of 2. `Example-2.3.2` will retain 10% of the jobs. With *concentration* as 2, 20% of the total input jobs will be retained.

3. Appendix

3.1. Resources

MAPREDUCE-751 is the main JIRA that introduced *Rumen* to *MapReduce*. Look at the MapReduce rumen-component for further details.

3.2. Dependencies

Rumen expects certain library *JARs* to be present in the *CLASSPATH*. The required libraries are

- `Hadoop MapReduce Tools` (`hadoop-mapred-tools-{hadoop-version}.jar`)
- `Hadoop Common` (`hadoop-common-{hadoop-version}.jar`)
- `Apache Commons Logging` (`commons-logging-1.1.1.jar`)
- `Apache Commons CLI` (`commons-cli-1.2.jar`)
- `Jackson Mapper` (`jackson-mapper-asl-1.4.2.jar`)
- `Jackson Core` (`jackson-core-asl-1.4.2.jar`)

Note:

One simple way to run Rumen is to use '$HADOOP_HOME/bin/hadoop jar' option to run it.