Swift log processing tools

Source control $LastChangedRevision$

Table of Contents

1. Overview

There is a package of Swift log processing utilties.

2. Prerequisites

gnuplot 4.0, gnu m4, gnu textutils, perl

3. Web page about a run

```
swift-plot-log /path/to/readData-20080304-0903-xgqf5nhe.log
```

This will create a web page, report-readData-20080304-0903-xgqf5nhe If the above command is used before a run is completed, the web page will report information about the workflow progress so far.

4. CEDPS logs

The log processing tools can output transition streams in CEDPS logging format:

```
swift-plot-log /path/to/readData-20080304-0903-xgqf5nhe.log execute.cedps
```

5. Event/transition channels

Various event channels are extracted from the log files and made available as `.event` and `.transition` files. These roughly correspond to processes within the Swift runtime environment.

These streams are then used to provide the data for the various output formats, such as graphs, web pages and CEDPS log format.

The available streams are:

Table 1.

Stream name	Description
execute	Swift procedure invocations
execute2	individual execution attempts
kickstart	kickstart records (not available as transitions)
karatasks	karajan level tasks, available as transitions (there are also four substreams karatasks.FILE_OPERATION, karatasks.FILE_TRANSFER and karatasks.JOB_SUBMISSION available as events but not transitions)
workflow	a single event representing the entire workflow
dostagein	stage-in operations for execute2s
dostageout	stage-out operations for execute2s

Streams are generated from their source log files either as .transitions or .event files, for example by `swift-plot-log whatever.log foo.event`.

Various plots are available based on different streams:

Table 2.

Makefile target	Description
foo.png	Plots the foo event stream
foo-total.png	Plots how many foo events are in progress at any time
foo.sorted-start.png	Plot like foo.png but ordered by start time

Text-based statistics are also available with `make foo.stats`.

Event streams are nested something like this:

```
workflow
  execute
    execute2
      dostagein
        karatasks (fileops and filetrans)
      clustering (optional)
        karatasks (execution)
          cluster-log (optional)
            wrapper log (optional)
              kickstart log
      dostageout
        karatasks (fileops and filetrans)
```

6. Internal file formats

The log processing code generates a number of internal files that follow a standard format. These are used for com-

munication between the modules that parse various log files to extract relevant information; and the modules that generate plots and other summary information.

```
need an event file format of one event per line, with that line
containing start time and duration and other useful data.

col1 = start, col2 = duration, col3 onwards = event specific data - for
some utilities for now should be column based, but later will maybe
move to attribute based.

between col 1 and col 2 exactly one space
between col 2 and col 3 exactly one space

start time is in seconds since unix epoch. start time should *not* be
normalised to start of workflow

event files should not (for now) be assumed to be in order

different event streams can be stored in different files. each event
stream should use the extension  .event



.coloured-event files
=====================
third column is a colour index
first two columns as per .event (thus a coloured-event is a specific
form of .event)
```

7. hacky scripts

There are a couple of hacky scripts that aren't made into proper commandline tools. These are in the libexec/log-processing/ directory:

```
  ./execute2-status-from-log [logfile]
     lists every (execute2) job and its final status

  ./execute2-summary-from-log [logfile]
     lists the counts of each final job status in log
```