Tron Documentation

Release 0.8.0.5

Yelp, Inc.

Apr 24, 2018

Contents

Tron is a centralized system for managing periodic batch processes across a cluster. If this is your first time using Tron, read *Tutorial* and *Overview* to get a better idea of what it is, how it works, and how to use it.

Note: Please report bugs in the documentation at our Github issue tracker.

CHAPTER 1

Table of Contents

1.1 What's New

1.1.1 0.6.1

- tronweb was replaced with a clientside version
- more ssh options are now configurable
- adding an experimental feature to support a max_runtime on jobs
- adding tronctl kill to SIGKILL a service
- add a *–no-header* option to tronfig

1.1.2 0.6.0

- *action.requires* must be a list (string has been deprecated since 0.3.3)
- *tronctl zap* has been removed (it shouldn't be necessary anymore)
- service monitoring code has been re-written (services should not longer get stuck in a stopping state)
- hosts can not be validated by specifying a *known_hosts* file
- additional validation for ssh options and context variables has been moved into configuration validation
- tronview now displays additional details about jobs and services
- the API has changed slightly (href is now url, service status is now state)

1.1.3 0.5.2

- Tron now supports the ability to use different users per node connection.
- Fragmented configuration is now possible by using namespaced config files.

- Additional cleanup and stability patches have been applied.
- State persistence configuration can now be changed without restarting *trond*
- State saving now includes a namespace, you will need to run *tools/migration/migrate_state.py* to migrate old state.
- *trond* now expects a configuration directory. Use *tools/migration/migrate_config_0.5.1_to_0.5.2.py* to convert your existing config to the new format.
- Patched an issue with SSH connections that caused an exception on channel close

1.1.4 0.5.1

- Jobs which are disabled will no longer be re-enabled when part of their configuration changes.
- Individual actions for a Job can no longer be started independently before a job is started. This was never intentionally supported.
- Adding a new configuration option *allow_overlap* for Jobs, which allows job runs to overlap each other.
- Jobs can now be configured using crontab syntax. see *Scheduling*

1.1.5 0.5.0

- Names for nodes, jobs, actions and service can now contain underscore characters but are restricted to 255 characters.
- trond now supports a graceful shutdown. Send trond SIGINT to have it wait for all currently running jobs to complete before shutting down. SIGTERM also performs some cleanup before terminating.
- State serialization has changed. See *State Persistence* for configuration options. *tools/migration/migrate_state.py* is included to migrate your existing Tron state to a new store. YAML store is now deprecated.
- All relative path options to *trond* and relative paths in the configuration will now be relative to the `--working-dir` directory instead of the current working directory.
- Old style config, which was deprecated in 0.3 will no longer work.

1.1.6 0.4.1

- *tronview* will once again attempt to find the tty width even when stdout is not a tty.
- Fixed last_success for job context.
- Job runs which are manually cancelled will now continue to schedule new runs.

1.1.7 0.4.0

- Jobs now continue to run all possible actions after one of its actions fail
- Enabling a disabled job now schedules the next run using current time instead of the last successful run (which could cause many runs to be scheduled in the past if the job had been disabled for a while)
- Command context is now better defined. see *Built-In Command Context Variables*. Also adds support for a last_success keyboard which supports date arithmetic.

- Resolved many inconsistencies and bugs around Job scheduling.

1.1.8 0.3.3

- Logging is now configured from logging.conf, see *Logging*
- Old style configuration files can be converted using *tools/migration/migrate_config_0.2_to_0.3.py*
- working_dir in the configuration has been replaced by output_stream_dir

1.1.9 0.3.0

- **!** (tags), ***** (references), and **&** (anchors) are now deprecated in the *trond* configuration file. Support will be removed for them in 0.5.
- Adding an enabled option for jobs, so they can be configured as disabled by default
- tron commands (*tronview*, *tronfig*, *tronctl*) now support a global config (defaults to /etc/tron/tron.yaml)
- tronview will now pipe its output through `less` if appropriate

1.1.10 0.2.10

- ssh_options is actually optional
- Cleanup actions no longer cause jobs using an interval scheduler to stop being scheduled if an action fails
- Failed actions can be skipped, causing dependent actions to run

1.1.11 0.2.9

- *tronweb* works and is documented.
- Daylight Savings Time behavior is more well-defined. See *Notes on Daylight Saving Time* for more information.
- Jobs that fail after running over their next scheduled time are no longer forgotten.
- Reconfiguring syslog no longer requires restarting *trond* to take effect.
- Syslog formatter is more meaningful (0.2.8.1).
- Prebuilt man pages are included so you don't need Sphinx to have them (0.2.8.1).

1.1.12 0.2.8

Features

- New HTML documentation. Hello!
- Cleanup actions let you run a command after the success or failure of a job. You can use them to clean up temp files, shut down Elastic MapReduce job flows, and more. See *Cleanup Actions*.
- Log to syslog by setting **syslog_address** in your config. See *Logging*.
- "zap" command for services lets you force Tron to see a service or service instance as **DOWN**. See *tronctl*.
- `simplejson` is no longer a dependency for Python 2.6 and up

Bug Fixes

- Fixed weekday-specified jobs (mon, tues, . . .) running a day late
- Fixed services being allowed in jobs list and causing weird crashes
- Fixed missing import in www.py

1.2 Tutorial

To install Tron you will need:

- A copy of the most recent Tron release from either github or pypi (see *Installing Tron*).
- A server on which to run **trond**.
- One or more batch boxes which will run the Jobs.
- An SSH key and a user that will allow the tron daemon to login to all of the batch machines without a password prompt.

1.2.1 Installing Tron

The easiest way to install Tron is from PyPI:

```
$ sudo pip install tron
```

You can also get a copy of the current developmnent release from github. See *setup.py* in the source package for a full list of required packages.

If you are interested in working on Tron development see *Contributing to Tron* for additional requirements and setting up a dev environment.

1.2.2 Running Tron

Tron runs as a single daemon, **trond**.

On your management node, run:

```
$ sudo -u <tron user> trond
```

The chosen user will need SSH access to all your worker nodes, as well as permission to write to the working directory, log file, and pid file (see `trond --help` for defaults). You can change these directories using command line options. Also see *Logging* on how to change the default logging settings.

Once **trond** is running, you can view its status using **tronview** (by default tronview will connect to localhost, use `--server=<host>:<port> -s` to specify a different server, and have that setting saved in `~/.tron`):

```
$ tronview

Jobs:
No jobs
```

1.2.3 Configuring Tron

There are a few options on how to configure tron, but the most straightforward is through tronfig:

```
$ tronfig
```

This will open your configured `$EDITOR` with the current configuration file. Edit your file to be something like this:

```
ssh_options:
  agent: true

notification_options:
  smtp_host: localhost
  notification_addr: <your email address>

nodes:
  - name: local
    hostname: 'localhost'

jobs:
  - name: "getting_node_info"
    node: local
    schedule: "interval 10 mins"
    actions:
      - name: "uname"
        command: "uname -a"
      -
        name: "cpu_info"
        command: "cat /proc/cpuinfo"
        requires: [uname]
```

After you exit your editor, the configuration will be validated and uploaded to *trond*.

Now if you run **tronview** again, you'll see `getting_node_info` as a configured job. Note that it is configured to run 10 minutes from now. This should give you time to examine the job to ensure you really want to run it.

```
Jobs:
Name              State      Scheduler             Last Success
getting_node_info ENABLED    INTERVAL:0:10:00      None
```

You can quickly disable a job by using **tronctl**:

```
$ tronctl disable getting_node_info
Job getting_node_info is disabled
```

This will stop scheduled jobs and prevent anymore from being scheduled. You are now in manual control. To manually execute a job immediately, do this:

```
$ tronctl start getting_node_info
New job getting_node_info.1 created
```

You can monitor this job run by using **tronview**:

```
$ tronview getting_node_info.1
Job Run: getting_node_info.1
State: SUCC
Node: localhost
```

```
Action ID & Command  State   Start Time           End Time             Duration
.uname               SUCC    2011-02-28 16:57:48  2011-02-28 16:57:48  0:00:00
.cpu_info            SUCC    2011-02-28 16:57:48  2011-02-28 16:57:48  0:00:00

$ tronview getting_node_info.1.uname
Action Run: getting_node_info.1.uname
State: SUCC
Node: localhost

uname -a

Requirements:

Stdout:
Linux dev05 2.6.24-24-server #1 SMP Wed Apr 15 15:41:09 UTC 2009 x86_64 GNU/Linux
Stderr:
```

Tron also provides a simple, optional web UI that can be used to get tronview data in a browser. See *tronweb* for setup instructions.

That's it for the basics. You might want to look at *Overview* for a more comprehensive description of how Tron works.

1.3 Overview

Batch process scheduling on a single UNIX machines has historically been managed by **cron** and its derivatives. But if you have many batches, complex dependencies between batches, or many machines, maintaining config files across them may be difficult. Tron solves this problem by centralizing the configuration and scheduling of jobs to a single daemon.

The Tron system is split into four commands:

trond Daemon responsible for scheduling, running, and saving state. Provides an HTTP interface to tools.

tronview View job state and output.

tronctl Start, stop, enable, disable, and otherwise control jobs.

tronfig Change Tron's configuration while the daemon is still running.

The config file uses YAML syntax, and is further described in *Configuration*.

1.3.1 Nodes, Jobs and Actions

Tron's orders consist of *jobs*. *Jobs* contain *actions* which may depend on other actions in the same job and run on a schedule.

trond is given access (via public key SSH) to one or more *nodes* on which to run jobs. For example, this configuration has two nodes, each of which is responsible for a single job:

```
nodes:
    hostname: 'localhost'
  - name: node1
    hostname: 'batch1'
  - name: node2
    hostname: 'batch2'

jobs:
  - name: "job0"
    node: node1
    schedule: "interval 20s"
    actions:
      - name: "batch1action"
        command: "sleep 3; echo asdfasdf"
  - name: "job1"
    node: node2
    schedule: "interval 20s"
    actions:
      - name: "batch2action"
        command: "cat big.txt; sleep 10"
```

How the nodes are set up and assigned to jobs is entirely up to you. They may have different operating systems, access to different databases, different privileges for the Tron user, etc.

See also:

- *Jobs*
- *Configuration*

1.3.2 Node Pools

Nodes can be grouped into *pools*. To continue the previous example:

```
node_pools:
    - name:pool
      nodes: [node1, node2]

jobs:
    # ...
    - name: "job2"
      node: pool
      schedule: "interval 5s"
      actions:
        - name: "pool_action"
          command: "ls /; sleep 1"
      cleanup_action:
        command: "echo 'all done'"
```

`job2`'s action will be run on a random node from `pool` every 5 seconds. When `pool_action` is complete, `cleanup_action` will run on the same node.

For more information, see *Jobs*.

1.3.3 Caveats

While Tron solves many scheduling-related problems, there are a few things to watch out for.

Tron keeps an SSH connection open for the entire lifespan of a process. This means that to upgrade **`trond`**, you have to either wait until no jobs are running, or accept an inconsistent state. This limitation is being worked on, and should be improved in later releases.

Tron is under active development. This means that some things will change. Whenever possible these changes will be backwards compatible, but in some cases there may be non-backwards compatible changes.

Tron does not support unicode. Tron is built using twisted which does not support unicode.

1.4 Configuration

1.4.1 Syntax

The Tron configuration file uses YAML syntax. The recommended configuration style requires only strings, decimal values, lists, and dictionaries: the subset of YAML that can be losslessly transformed into JSON. (In fact, your configuration can be entirely JSON, since YAML is mostly a strict superset of JSON.)

Past versions of Tron used additional YAML-specific features such as tags, anchors, and aliases. These features still work in version 0.3, but are now deprecated.

1.4.2 Basic Example

```
ssh_options:
  agent: true

notification_options:
  smtp_host: localhost
  notification_addr: <your email address>

nodes:
  - name: local
    hostname: 'localhost'

jobs:
  - name: "getting_node_info"
    node: local
    schedule: "interval 10 mins"
    actions:
      - name: "uname"
        command: "uname -a"
      - name: "cpu_info"
        command: "cat /proc/cpuinfo"
        requires: [uname]
```

1.4.3 Command Context Variables

command attribute values may contain **command context variables** that are inserted at runtime. The **command context** is populated both by Tron (see *Built-In Command Context Variables*) and by the config file (see *Command Context*). For example:

```
jobs:
    - name: "command_context_demo"
      node: local
      schedule: "1st monday in june"
      actions:
        - name: "print_run_id"
          # prints 'command_context_demo.1' on the first run,
          # 'command_context_demo.2' on the second, etc.
          command: "echo %(runid)"
```

1.4.4 SSH

ssh_options (optional) Options for SSH connections to Tron nodes. When tron runs a job on a node, it will add some jitter (random delay) to the run, which can be configured with the options below.

agent (optional, default `False`) Set to `True` if **`trond`** should use an SSH agent. This requires that `$SSH_AUTH_SOCK` exists in the environment and points to the correct socket.

identities (optional, default `[]`) List of paths to SSH identity files

known_hosts_file (optional, default `None`) The path to an ssh known hosts file

connect_timeout (optional, default `30`) Timeout in seconds when establishing an ssh connection

idle_connection_timeout (optional, default `3600`) Timeout in seconds that an ssh connection can remain idle after which it is closed

jitter_min_load (optional, default `4`) Minimum *load* on a node before any jitter is introduced. See *jitter_load_factor* for a description of how load is calculated

jitter_max_delay (optional, default `20`) Maximum number of seconds to add to a run

jitter_load_factor (optional, default `1`) Factor used to increment the count of running actions for determining the upper bound of jitter to add (ex. A factor of 2 would increase the upper bound by 2 seconds per running action)

Example:

```
ssh_options:
    agent:                      false
    known_hosts_file:           /etc/ssh/known_hosts
    identities:
        - /home/batch/.ssh/id_dsa-nopasswd

    connect_timeout:            30
    idle_connection_timeout:    3600

    jitter_min_load:            4
    jitter_max_delay:           20
    jitter_load_factor:         1
```

1.4.5 Notification Options

notification_options Email settings for sending failure notices.

smtp_host (required) SMTP host of the email server

notification_addr (required) Email address to send mail to

Example:

```
notification_options:
    smtp_host: localhost
    notification_addr: batch+errors@example.com
```

1.4.6 Time Zone

time_zone (optional) Local time as observed by the system clock. If your system is obeying a time zone with daylight savings time, then some of your jobs may run early or late on the days bordering each mode. See *Notes on*

Daylight Saving Time for more information.

Example:

```
time_zone: US/Pacific
```

1.4.7 Command Context

command_context Dictionary of custom *command context variables*. It is an arbitrary set of key-value pairs.

Example:

```
command_context:
    PYTHON: /usr/bin/python
    TMPDIR: /tmp
```

See a list of *Built-In Command Context Variables*.

1.4.8 Output Stream Directory

output_stream_dir A path to the directory used to store the stdout/stderr logs from jobs. It defaults to the `--working_dir` option passed to *trond*.

Example:

```
output_stream_dir: "/home/tronuser/output/"
```

1.4.9 State Persistence

state_persistence Configure how trond should persist its state to disk. By default a *shelve* store is used and saved to *./tron_state* in the working directory.

store_type

Valid options are: **shelve** - uses the *shelve* module and saves to a local file

sql - uses sqlalchemy to save to a database (tested with version 0.7).

yaml - uses *yaml* and saves to a local file (this is not recommend and is provided to be backwards compatible with previous versions of Tron).

You will need the appropriate python module for the option you choose.

name The name of this store. This will be the filename for a **shelve** or **yaml** store. It is just a label when used with an **sql** store.

connection_details Ignored by **shelve** and **yaml** stores.

A connection string (see sqlalchemy engine configuration) when using an **sql** store.

Valid keys are: hostname, port, username, password. Example: `"hostname=localhost&port=5555"`

buffer_size The number of save calls to buffer before writing the state. Defaults to 1, which is no buffering.

Example:

```
state_persistence:
    store_type: sql
    name: local_sqlite
    connection_details: "sqlite:///dest_state.db"
    buffer_size: 1 # No buffer
```

1.4.10 Action Runners

Note: this is an experimental feature

action_runner Action runner configuration allows you to run Job actions through a script which records it's pid. This provides support for a max_runtime option on jobs, and allows you to stop or kill the action from **`tronctl`**.

runner_type

Valid options are:

none Run actions without a wrapper. This is the default

subprocess Run actions with a script which records the pid and runs the action command in a subprocess (on the remote node). This requires that **`bin/action_runner.py`** and **`bin/action_status.py`** are avialable on the remote host.

remote_status_path Path used to store status files. Defaults to */tmp*.

remote_exec_path Directory path which contains **`action_runner.py`** and **`action_status.py`** scripts.

Example:

```
action_runner:
    runner_type:        "subprocess"
    remote_status_path: "/tmp/tron"
    remote_exec_path:   "/usr/local/bin"
```

1.4.11 Nodes

nodes List of nodes. Each node has the following options:

hostname (required) The hostname or IP address of the node

name (optional, defaults to `hostname`) A name to refer to this node

username (optional, defaults to current user) The name of the user to connect with

port (optional, defaults to 22) The port number of the node

Example:

```
nodes:
    - name: node1
      hostname: 'batch1'
    - hostname: 'batch2'     # name is 'batch2'
```

1.4.12 Node Pools

node_pools List of node pools, each with a `name` and `nodes` list. `name` defaults to the names of each node joined by underscores.

Example:

```
node_pools:
    - name: pool
      nodes: [node1, batch1]
    - nodes: [batch1, node1]     # name is 'batch1_node1'
```

1.4.13 Jobs and Actions

jobs List of jobs for Tron to manage. See *Jobs* for the options available to jobs and their actions.

1.4.14 Logging

As of v0.3.3 Logging is no longer configured in the tron configuration file.

Tron uses Python's standard logging and by default uses a rotating log file handler that rotates files each day. The default log directory is `/var/log/tron/tron.log`.

To configure logging pass -l <logging.conf> to trond. You can modify the default logging.conf by copying it from tron/logging.conf. See http://docs.python.org/howto/logging.html#configuring-logging

Interesting logs

Most tron logs are named by using pythons *__file__* which uses the modules name. There are a couple special cases:

twisted Twisted sends its logs to the *twisted* log

tron.api.www.access API access logs are sent to this log at the INFO log level. They follow a standard apache combined log format.

1.5 Jobs

A job consists of a name, a node/node pool, a list of actions, a schedule, and an optional cleanup action. They are periodic events that do not interact with other jobs while running.

If all actions exit with status 0, the job has succeeded. If any action exists with a nonzero status, the job has failed.

1.5.1 Required Fields

name Name of the job. Used in **`tronview`** and **`tronctl`**.

node Reference to the node or pool to run the job in. If a pool, the job is run in a random node in the pool.

schedule When to run this job. Schedule fields can take multiple forms. See *Scheduling*.

actions List of *actions*.

1.5.2 Optional Fields

monitoring (default {}) (Beta Feature) Dictionary of key: value pairs to inform the monitoring framework on how to alert teams for job failures.

queueing (default True) If a job run is still running when the next job run is to be scheduled, add the next run to a queue if this is **True**. Otherwise, cancel the job run. Note that if the scheduler used for this job is not defined to queue overlapping then this setting is ignored. The ConstantScheduler will not queue overlapping.

allow_overlap (default False) If **True** new job runs will start even if the previous run is still running. By default new job runs are either cancelled or queued (see **queuing**).

run_limit (default 50) Number of runs which will be stored. Once a Job has more then run_limit runs, the output and state for the oldest run are removed. Failed runs will not be removed.

all_nodes (default False) If **True** run this job on each node in the node pool list. If a node appears more than once in the list, the job will be run on that node once for each appearance.

If **False** run this job on a random node from the node pool list. If a node appears more than once in the list, the job will be more likely to run on that node, proportionate to the number of appearances.

If **node** is not a node pool, this option has no effect.

cleanup_action Action to run when either all actions have succeeded or the job has failed. See *Cleanup Actions*.

enabled (default True) If **False** the job will not be scheduled to run. This configuration option is only relevant when a Job is first added to the configuration, after which this value will be ignored.

max_runtime (default None) A time interval (ex: "2 hours") that limits the duration of each job run. If the job run is still running after this duration, all of it's actions are sent SIGTERM.

Note: This requires an *Action Runners* to be configured. If *action_runner* is none max_runtime does nothing.

time_zone (default None) Time zone used for calculating when a job should run. Defaults to None, which means it will use the default time_zone set in the master config.

1.5.3 Actions

Actions consist primarily of a **name** and **command**. An action's command is executed as soon as its dependencies (specified by **requires**) are satisfied. So if your job has 10 actions, 1 of which depends on the other 9, then Tron will launch the first 9 actions in parallel and run the last one when all have completed successfully.

If any action exits with nonzero status, the job will continue to run any actions which do not depend on the failed action.

Required Fields

name Name of the action. Used in `tronview` and `tronctl`.

command Command to run on the specified node. A common mistake here is to use shell expansions or expressions in your command. Commands are run using `exec` so bash (or other shell) expressions will not work, and could cause the job to fail.

Optional Fields

requires List of action names that must complete successfully before this action is run. Actions can only require actions in the same job.

node Node or node pool to run the action on if different from the rest of the job.

Example Actions

```
jobs:
    - name: convert_logs
      node: node1
      schedule:
        start_time: 04:00:00
      actions:
        - name: verify_logs_present
          command: "ls /var/log/app/log_%(shortdate-1).txt"
        - name: convert_logs
          command: "convert_logs /var/log/app/log_%(shortdate-1).txt /var/log/app_
↪converted/log_%(shortdate-1).txt"
          requires: [verify_logs_present]
```

1.5.4 Scheduling

Tron supports four methods for configuring the schedule of a job. Schedulers support a jitter parameter that allows them to vary their runtime by a random time delta.

Interval

Run the job every X seconds, minutes, hours, or days. The time expression is `<interval> days|hours|minutes|seconds`, where the units can be abbreviated.

Short form:

```
schedule: "interval 20s"
```

Long form:

```
schedule:
    type:   "interval"
    value:  "5 mins"
    jitter: "10 seconds"        # Optional
```

With alias:

```
schedule:
    type:   "interval"
    value:  "hourly"
```

Daily

Run the job on specific days at a specific time. The time expression is `HH:MM:SS[ MTWRFSU]`.

Short form:

```
schedule: "daily 04:00:00"
```

Short form with days:

```
schedule: "daily 04:00:00 MWF"
```

Long form:

```
schedule:
    type:   "daily"
    value:  "07:00:00 MWF"
    jitter: "10 min"            # Optional
```

Cron

Schedule a job using cron syntax. Tron supports predefined schedules, ranges, and lists for each field. It supports the *L* in day of month field only (which schedules the job on the last day of the month). Only one of the day fields (day of month and day of week) can have a value.

Short form:

```
schedule: "cron */5 * * 7,8 *"  # Every 5 minutes in July and August
```

```
schedule: "cron 0 3-6 * * *"    # Every hour between 3am and 6am
```

Long form:

```
schedule:                       # long form
    type: "cron"
    value: "30 4 L * *"         # The last day of the month at 4:30am
```

Complex

More powerful version of the daily scheduler based on the one used by Google App Engine's cron library. To use this scheduler, use a string in this format as the schedule:

```
("every"|ordinal) (days) ["of|in" (monthspec)] (["at"] HH:MM)
```

ordinal Comma-separated list of `1st` and so forth. Use `every` if you don't want to limit by day of the month.

days Comma-separated list of days of the week (for example, `mon`, `tuesday`, with both short and long forms being accepted); `every day` is equivalent to `every mon,tue,wed,thu,fri,sat,sun`

monthspec Comma-separated list of month names (for example, `jan`, `march`, `sep`). If omitted, implies every month. You can also say `month` to mean every month, as in `1,8th,15,22nd of month 09:00`.

HH:MM Time of day in 24 hour time.

Some examples:

```
2nd,third mon,wed,thu of march 17:00
every monday at 09:00
1st monday of sep,oct,nov at 17:00
every day of oct at 00:00
```

In the config:

```
schedule: "every monday at 09:00"
```

```
schedule:
    type: "groc daily"
    value: "every day 11:22"
    jitter: "5 min"
```

Notes on Daylight Saving Time

Some system clocks are configured to track local time and may observe daylight savings time. For example, on November 6, 2011, 1 AM occurred twice. Prior to version 0.2.9, this would cause Tron to schedule a daily midnight job to be run an hour early on November 7, at 11 PM. For some jobs this doesn't matter, but for jobs that depend on the availability of data for a day, it can cause a failure.

Similarly, some jobs on March 14, 2011 were scheduled an hour late.

To avoid this problem, set the *Time Zone* config variable. For example:

```
time_zone: US/Pacific
```

If a job is scheduled at a time that occurs twice, such as 1 AM on "fall back", it will be run on the *first* occurrence of that time.

If a job is scheduled at a time that does not exists, such as 2 AM on "spring forward", it will be run an hour later in the "new" time, in this case 3 AM. In the "old" time this is 2 AM, so from the perspective of previous jobs, it runs at the correct time.

In general, Tron tries to schedule a job as soon as is correct, and no sooner. A job that is schedule for 2:30 AM will not run at 3 AM on "spring forward" because that would be half an hour too soon from a pre-switch perspective (2 AM).

Note: If you experience unexpected scheduler behavior, file an issue on Tron's Github page.

1.5.5 Cleanup Actions

Cleanup actions run after the job succeeds or fails. They are specified just like regular actions except that there is only one per job and it has no name or requirements list.

If your job creates shared resources that should be destroyed after a run regardless of success or failure, such as intermediate files or Amazon Elastic MapReduce job flows, you can use cleanup actions to tear them down.

The command context variable `cleanup_job_status` is provided to cleanup actions and has a value of `SUCCESS` or `FAILURE` depending on the job's final state. For example:

```
-
    # ...
    cleanup_action:
      command: "python -m mrjob.tools.emr.job_flow_pool --terminate MY_POOL"
```

1.5.6 States

The following are the possible states for a Job and Job Run.

Job States

ENABLED A run is scheduled and new runs will continue to be scheduled.

DISABLED No new runs will be scheduled, and scheduled runs will be cancelled.

RUNNING Job run currently in progress.

Job Run States

SCHE The run is scheduled for a specific time

RUNN The run is currently running

SUCC The run completed successfully

FAIL The run failed

QUE The run is queued behind another run(s) and will start when said runs finish

CANC The run was scheduled, but later cancelled.

UNKWN The run is in and unknown state. This state occurs when tron restores a job that was running at the time of shutdown.

Action States

Job states are derived from the aggregate state of their actions. The following is a state diagram for an action.

`images/action.png`

1.6 Built-In Command Context Variables

Tron includes some built in command context variables that can be used in command configuration.

shortdate Run date in `YYYY-MM-DD` format. Supports simple arithmetic of the form `%(shortdate+6)s` which returns a date 6 days in the future, `%(shortdate-2)s` which returns a date 2 days before the run date.

year Current year in `YYYY` format. Supports the same arithmetic operations as *shortdate*. For example, `%(year-1)s` would return the year previous to the run date.

month Current month in *MM* format. Supports the same arithmetic operations as *shortdate*. For example, `%(month+2)s` would return 2 months in the future.

day Current day in *DD* format. Supports the same arithmetic operations as *shortdate*. For example, `%(day+1)s` would return the day after the run date.

hour Current hour in *HH* (0-23) format. Supports the same arithmetic operations as *shortdate*. For example, `%(hour+1)s` would return the hour after the run hour (mod 24).

unixtime Current timestamp. Supports addition and subtraction of seconds. For example `%(unixtime+20)s` would return the timestamp 20 seconds after the jobs runtime.

daynumber Current day number as an ordinal (datetime.toordinal()). Supports addition and subtraction of days. For example `%(daynumber-3)s` would be 3 days before the run date.

name Name of the job

node Hostname of the node the action is being run on

1.6.1 Context variables only available to Jobs

runid Run ID of the job run (e.g. `sample_job.23`)

actionnname The name of the action

cleanup_job_status `SUCCESS` if all actions have succeeded when the cleanup action runs, `FAILURE` otherwise. `UNKNOWN` if used in an action other than the cleanup action.

last_success The last successful run date (defaults to current date if there was no previous successful run). Supports date arithmetic using the form `%(last_success:shortdate-1)s`.

1.7 tronweb

tronweb is the web-based UI for tron.

See http://localhost:8089/web/

1.8 Man Pages

1.8.1 tronctl

Synopsys

```
tronctl [--server <host:port>] [--verbose] <command> <job_name | job_run_id |
action_run_id>
```

Description

tronctl is the control interface for Tron. **`tronctl`** allows you to enable, disable, start, stop and cancel Tron Jobs and Services.

Options

`--server=<config-file>` Config file containing the address of the server the tron instance is running on

`--verbose` Displays status messages along the way

`--run-date=<YYYY-MM-DD>` For starting a new job, specifies the run date that should be set. Defaults to today.

Job Commands

disableall Disables all jobs

enableall Enables all jobs

disable <job_name> Disables the job. Cancels all scheduled and queued runs. Doesn't schedule any more.

enable <job_name> Enables the job and schedules a new run.

start <job_name> Creates a new run of the specified job and runs it immediately.

start <job_run_id> Attempt to start the given job run. A Job run only starts if no other instance is running. If the job has already started, it will attempt to start any actions in the SCH or QUE state.

start <action_run_id> Attempt to start the action run.

restart <job_run_id> Creates a new job run with the same run time as this job.

rerun <job_run_id> Creates a new job run with the same run time as this job (same as restart).

cancel <job_run_id | action_run_id> Cancels the specified job run or action run.

success <job_run_id | action_run_id> Marks the specified job run or action run as succeeded. This behaves the same as the run actually completing. Dependant actions are run and queued runs start.

skip <action_run_id> Marks the specified action run as skipped. This allows dependant actions to run.

fail <job_run_id | action_run_id> Marks the specified job run or action run as failed. This behaves the same as the job actually failing.

stop <action_run_id> Stop an action run

kill <action_run_id> Force stop (SIGKILL) an action run

Examples

```
$ tronctl start job0
New Job Run job0.2 created

$ tronctl start job0.3
Job Run job0.3 now in state RUNN

$ tronctl cancel job0.4
Job Run job0.4 now in state CANC

$ tronctl fail job0.4
Job Run job0.4 now in state FAIL

$ tronctl restart job0.4
Job Run job0.4 now in state RUNN
```

```
$ tronctl success job0.5
Job Run job0.5 now in state SUCC
```

Bugs

Post bugs to http://www.github.com/yelp/tron/issues.

See Also

trond (8), **tronfig** (1), **tronview** (1),

1.8.2 trond

Synopsys

```
trond [--working-dir=<working dir>] [--verbose] [--debug]
```

Description

trond is the tron daemon that manages all jobs.

Options

`--version` show program's version number and exit

`-h, --help` show this help message and exit

`--working-dir=WORKING_DIR` Directory where tron's state and output is stored (default /var/lib/tron/)

`-l LOG_CONF, --log-conf=LOG_CONF` Logging configuration file to setup python logger

`-c CONFIG_FILE, --config-file=CONFIG_FILE` Configuration file to load (default in working dir)

`-v, --verbose` Verbose logging

`--debug` Debug mode, extra error reporting, no daemonizing

`--nodaemon` Indicates we should not fork and daemonize the process (default False)

`--pid-file=PIDFILE` Where to store pid of the executing process (default /var/run/tron.pid)

`-P LISTEN_PORT, --port=LISTEN_PORT` What port to listen on, defaults 8089

`-H LISTEN_HOST, --host=LISTEN_HOST` What host to listen on defaults to localhost

Files

Working directory The directory where state and saved output of processes are stored.

Pid file Contains the pid of the daemonized process.

Log File trond error log, configured from logging.conf

Signals

SIGINT Graceful shutdown. Waits for running jobs to complete.

SIGTERM Does some cleanup before shutting down.

SIGHUP Reload the configuration file.

SIGUSR1 If running with `--nodaemon` will drop into an ipdb debugging prompt.

Logging

Tron uses Python's standard logging and by default uses a rotating log file handler that rotates files each day. Logs go to `/var/log/tron/tron.log`.

To configure logging pass -l <logging.conf> to trond. You can modify the default logging.conf by coping it from tron/logging.conf. See http://docs.python.org/howto/logging.html#configuring-logging

Bugs

trond has issues around daylight savings time and may run jobs an hour early at the boundary.

Post further bugs to http://www.github.com/yelp/tron/issues.

See Also

tronctl (1), **tronfig** (1), **tronview** (1),

1.8.3 tronfig

Synopsys

```
tronfig [--server server_name ] [--verbose | -v] [<namespace>] [-p] [-]
```

Description

tronfig allows live editing of the Tron configuration. It retrieves the configuration file for local editing, verifies the configuration, and sends it back to the tron server. The configuration is applied immediately.

Options

`--server <server_name>` The server the tron instance is running on

`--verbose` Displays status messages along the way

`--version` Displays version string

`-p` Print the configuration

`namespace` The configuration namespace to edit. Defaults to MASTER

`-` Read new config from `stdin`.

Configuration

By default tron will run with a blank configuration file. The config file is saved to `<working_dir>/config/` by default. See the full documentation at http://tron.readthedocs.io/en/latest/config.html.

Bugs

Post bugs to http://www.github.com/yelp/tron/issues.

See Also

trond (8), **tronctl** (1), **tronview** (1),

1.8.4 tronview

Synopsys

`tronview [-n <numshown>] [--server <server_name>] [--verbose] [<job_name> | <job_run_id> | <action_run_id>]`

Description

tronview displays the status of tron scheduled jobs.

tronview Show all configured jobs

tronview <job_name> Shows details for a job. Ex:

```
$ tronview my_job
```

tronview <job_run_id> Show details for specific run or instance. Ex:

```
$ tronview my_job.0
```

tronview <action_run_id> Show details for specific action run. Ex:

```
$ tronview my_job.0.my_action
```

Options

`--version` show program’s version number and exit

`-h, --help` show this help message and exit

`-v, --verbose` Verbose logging

`-n NUM_DISPLAYS, --numshown=NUM_DISPLAYS` The maximum number of job runs or lines of output to display(0 for show all). Does not affect the display of all jobs and the display of actions for given job.

`--server=SERVER` Server URL to connect to

`-c, --color` Display in color

`--nocolor` Display without color

-o, --stdout Solely displays stdout

-e, --stderr Solely displays stderr

--events Show events for the specified entity

-s, --save Save server and color options to client config file (~/.tron)

States

For complete list of states with a diagram of valid transitions see http://packages.python.org/tron/jobs.html#states

Bugs

Post bugs to http://www.github.com/yelp/tron/issues.

See Also

trond (8), **tronctl** (1), **tronfig** (1),

1.9 Contributing to Tron

Tron is an open source project and welcomes contributions from the community. The source and issue tracker are hosted on github at http://github.com/yelp/Tron.

1.9.1 Setting Up an Environment

Tron works well with virtualenv, which can be setup using virtualenvwrapper:

```
$ mkvirtualenv tron --distribute --no-site-packages
$ pip install -r dev/req_dev.txt
```

`req_dev.txt` contains a list of packages required for development, including: Testify to run the tests and Sphinx to build the documentation.

1.9.2 Coding Standards

All code should be PEP8 compliant, and should pass pyflakes without warnings. All new code should include full test coverage, and bug fixes should include a test which reproduces the reported issue.

This documentation must also be kept up to date with any changes in functionality.

1.9.3 Running Tron in a Sandbox

The source package includes a development logging.conf and a sample configuration file with a few test cases. To run a development intsance of Tron create a working directory and start **trond** using the following:

```
$ make dev
```

1.9.4 Running the Tests

Tron uses the Testify unit testing framework.

Run the tests using `make tests` or `testify tests`. If you're using a virtualenv you may want to run ``python `which testify` test`` to have it use the correct environment.

This package also includes a `.pyautotest` file which can be used with https://github.com/dnephin/PyAutoTest to auto run tests when you save a file.

1.9.5 Contributing

There should be a github issue created prior to all pull requests. Pull requests should be made to the `Yelp:development` branch, and should include additions to `CHANGES.txt` which describe what has changed.

CHAPTER 2

Indices and tables

- genindex
- search

Index