Python Scheduler Outline

Zak Karimjee

December 21, 2019

This briefly summarises the scheduler script, its structure and why it has been written in this manner.

1 Structure

The script reads in from a crontabs-style file which specifies values or intervals of minutes, hours, days and months at which to run the given shell command. It reads this file in, parses it and validates the inputs. Each 'job' is then stored as a dictionary in a list of jobs.

The program then runs on a loop that executes every 60 seconds. This loop checks each of the 'job' timing specifications against the current datetime, and runs it if it fits. It runs each script with the `subprocess` library, as this is the new recommended way to execute shell commands in Python 3.

An alternative was to use Timer threads from the `threading` library to run the jobs with a specified delay until the required execution time. This would be straightforward if the jobs were specified with a fixed interval between executions, but more convoluted to have specified times mixed in with 'wildcard' times. The program would have had to calculate when the next execution would occur, and then create a Timer thread to execute it at that time. This has two problems - firstly, finding the next execution time is actually a complicated process when wildcards and multiples are included; secondly, if the threads were suspended by the system the execution time would be off.

Re-checking at regular intervals means that the script will execute based on the system time instead of a timer which could be paused by process suspension. However, it does load the system slightly more especially when the interval between executions is large, as the loop runs every minute. The time-checking routine is very simple and should not noticeably load the system.

The script logs all its actions to a file. This is important as it is likely to be running in the background rather than an open shell, so outputs will not be visible.

2 Assumptions

- Users will not need to run jobs at a timing precision smaller than minutes
- Jobs will be scheduled primarily based on time of day rather than regular intervals of time.
- Any redirecting or piping of shell command outputs can be done in a wrapper script, as they cannot be done within the job specification file.

The limitations of the script are laid out in the file `README.md`.

3 Potential Improvements

- Adding day-of-the-week support - this is a feature of cron that is not currently present here. This would be useful in many workplace situations for reporting on given weekdays etc.
- Reducing unnecessary time-checks - the period of the program loop could be varied depending on the smallest execution interval - for example, if all jobs only execute in 30 minute intervals or more, the program loop can check every 30 minutes instead of every 1.
- Adding options for command outputs to be fed into the output log or a separate output file. This would be especially important for handling errors that come from the shell commands.