Refactoring the SWH scheduler and listers

for better handling of recurrent origin visit tasks

Nicolas Dandrimont
olasd@softwareheritage.org
@olasd

Fondation Inria

16 December 2020
tech talk - ~~Inria Paris~~the Internet

Some history

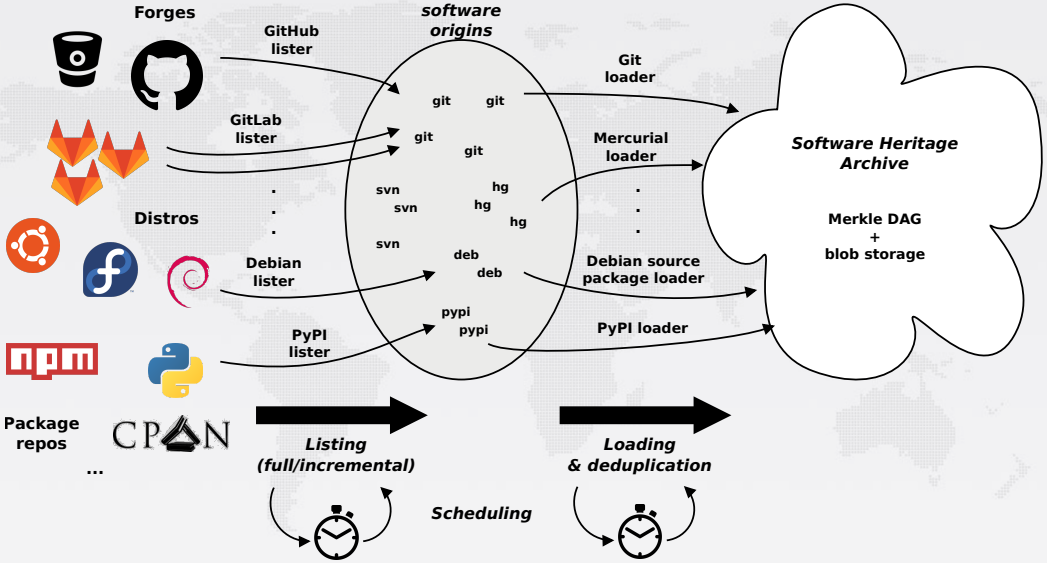
In the beginning, there was…

- First listing of GitHub: ad-hoc scripts by zack
- First git clones: basic Celery worker setup (swh-cloner-git repository)
- First imports in the archive: basic Celery worker setup setup too (see early swh-loader-git history)

How to future-proof this infra?

- What to do for recurrent imports?
- Lots of Celery limitations quickly apparent:
  - no real support of (adaptive) recurrent tasks in celery
  - rabbitmq is generally FIFO: no task priorities
  - recurrent data-loss on single-node rabbitmq setup with lots of messages in flight

Data flow


Forges
GitHub lister
GitLab lister
Distros
Debian lister
PyPI lister
Package repos
...
software origins
git
git
git
git
svn
svn
svn
hg
hg
hg
deb
deb
pypi
pypi
Git loader
Mercurial loader
Debian source package loader
PyPI loader
Software Heritage Archive
Merkle DAG
+
blob storage
Listing (full/incremental)
Loading & deduplication
Scheduling

swh-scheduler original goals / design

Goals

- "persistence layer" for recurrent task definitions around Celery
- adaptive recurrence interval according to task results
- single source of truth for scheduling
- secondary: implement priorities and automatic retry of tasks

Design

- core: storage of task definitions (type, args, queue position, recurrence interval)
- scheduler runner: sends next tasks to run to the celery queues
- scheduler listener: updates tasks in db from celery events
- celery/rabbitmq:
  - only a buffer for tasks between database queries
  - used for its worker management framework

a peek at the database

```
> select * from task where id=1;
-[ RECORD 1 ]----+-------------------------------------
id               | 1
type             | load-git
arguments        | {"args": [], "kwargs": {"url": "https:
   //github.com/hylang/hy"}}
next_run         | 2020-03-06 17:11:05.482501+00
current_interval | 12:00:00
status           | next_run_not_scheduled
policy           | recurring
retries_left     | 0
priority         |

Temps : 103,580 ms
```

five year retrospective of swh-scheduler

Some good

- OK permanence layer for celery, which has some useful features for worker mgmt
- when used sparingly, task priorities work fine (e.g. save code now)

lots of drawbacks, however

- poor introspectability:
  - difficult to index on free-form task arguments
  - inscrutable queue positions and adaptive recurrence:
    - currently processing `load-git` tasks scheduled to run 11 months ago
    - impossible to know at what point recurrent `load-git` tasks inserted now will run
- no input for external information about task scheduling
  - tons of useless task runs for repos not updated in years
- the celery events feedback loop is a hack
  - the events queue isn't persistent by default, and we struggle to work around this

Current swh-lister design (1/2)

Basic operation

- iterate all pages of the upstream API
- insert records for origins found, in an ad-hoc database/table for each lister
- generate recurrent tasks for origin visits in swh.scheduler

Two main modes of operation

- incremental: if possible, only get "new" pages of results from the upstream API
- full: list the upstream API completely again, updating the stored information

Current swh-lister design (2/2)

Design iterations:

- Originally based on the one-off GitHub listing scripts by zack
- Generalized from GitHub + BitBucket by fiendish in 2016
  - extracted common patterns useful to write listers (http, rate limiting, etc.)
  - extracted a common, overridable database schema from the GitHub and BitBucket commonalities
- Grew lots of tentacles to implement a bunch of listers (13 different kinds of upstreams supported)

swh-lister design shortcomings (1/2)

(strong) opinions ahead

deep and *wide* inheritance hierarchy

- lots of subtly different mixins to implement common functionality
  - that end up being overridden to handle peculiarities of every upstream
- lots of copy/paste to get a working lister
- debugging is quite painful

Way too much magic in tests

- based on UnitTest with a fairly opaque base class
- Provide two (good/bad) api responses and you're done...
- ...but it's not clear what's covered or not when reading the tests for a given lister

swh-lister design shortcomings (2/2)

Unhelpful generic database schema

- generic but needs very specific overrides
- lots of GH-specific/useless fields
- hard to do cross-cutting analysis of listed origins

Supposed to be an "easy" entry point for new contributors

- all in all, pretty hard to actually implement anything

Scheduler for recurrent origin visits

Tracking task

T2345

Scope

Only handle recurrent origin visit tasks. "One-shot" tasks are out of scope.

Design elements

- A single, unified storage for lister state and listed origins, within the scheduler database
  - Implemented, to be used by refactored listers
- TODO: A cache for quick, bulk access to information about the status of a given origin in the archive
- TODO: A scheduling policy component merging information from the two previous tables to send tasks for processing in workers

Scheduler for recurrent origin visits data model

Lister

id	uuid
name	str (f.e. "github", "phabricator")
instance_name	str (f.e. "softwareheritage")
current_state	dict
created	timestamp of creation
updated	timestamp of last update

ListedOrigin

lister_id	uuid
url	str
visit_type	str
extra_loader_arguments	Dict[str, str]
last_update	timestamp of last update (if provided upstream)
enabled	bool
first_seen	timestamp of earliest listing
last_seen	timestamp of latest listing

Lister refactoring

Forge references

https://forge.softwareheritage.org/T2453 and open diffs D3425, D3526, D3527, D4700, D4705, D4706

Scope

- Replace direct recurrent task scheduling with the new swh-scheduler based lister storage
- Drop the ad-hoc database schemas for listers
- Improve/clarify test coverage

Current status

- Base patterns implemented (with state storage: D3425, stateless: D4705)
- GitHub lister reimplemented, with full test coverage (D3527)
- Phabricator lister reimplemented, with no test coverage (D4706)

A call for help!

All alone in the rabbit hole

- Progress on this has been (very, very) slow
- Even if the updated listers land, they'll write their data to a dead-end table

Multiple tasks can be distributed

- Review of the current code
- Implementation of the rest of the scheduler components
- And of course the implementation of more listers, as well as hopefully some refactoring of common behaviors…

Maybe a sprint topic to get us started on 2021?