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
technical 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
- Debian lister
- PyPI lister

Distros

Software origins
- git
- git
- hg
- hg
- git
- git
- git
- git
- svn
- svn
- svn
- deb
- deb
- deb
- pypi
- pypi
- pypi

Package repos

Listing (full/incremental)

Loading & deduplication

Scheduling

Software Heritage Archive

Merkle DAG + blob storage

Nicolas Dandrimont

Refactoring the SWH scheduler and listers 16 Dec 2020
### 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

```sql
> select * from task where id=1;

<table>
<thead>
<tr>
<th>id</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>load-git</td>
</tr>
<tr>
<td>arguments</td>
<td>{&quot;args&quot;: [], &quot;kwargs&quot;: {&quot;url&quot;: &quot;<a href="https://github.com/hylang/hy%22%7D%7D">https://github.com/hylang/hy&quot;}}</a></td>
</tr>
<tr>
<td>next_run</td>
<td>2020-03-06 17:11:05.482501+00</td>
</tr>
<tr>
<td>current_interval</td>
<td>12:00:00</td>
</tr>
<tr>
<td>status</td>
<td>next_run_not_scheduled</td>
</tr>
<tr>
<td>policy</td>
<td>recurring</td>
</tr>
<tr>
<td>retries_left</td>
<td>0</td>
</tr>
<tr>
<td>priority</td>
<td></td>
</tr>
</tbody>
</table>

Temps : 103,580 ms
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
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)
(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
## Lister

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>uuid</td>
<td>Unique identifier for the lister.</td>
</tr>
<tr>
<td>name</td>
<td>str (f.e. &quot;github&quot;, &quot;phabricator&quot;)</td>
<td>Name of the lister (e.g., GitHub or Phabricator).</td>
</tr>
<tr>
<td>instance_name</td>
<td>str (f.e. &quot;softwareheritage&quot;)</td>
<td>Instance name of the origin.</td>
</tr>
<tr>
<td>current_state</td>
<td>dict</td>
<td>Current state of the lister.</td>
</tr>
<tr>
<td>created</td>
<td>timestamp</td>
<td>Timestamp of creation.</td>
</tr>
<tr>
<td>updated</td>
<td>timestamp</td>
<td>Timestamp of last update.</td>
</tr>
</tbody>
</table>

## ListedOrigin

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lister_id</td>
<td>uuid</td>
<td>Unique identifier for the lister's origins.</td>
</tr>
<tr>
<td>url</td>
<td>str</td>
<td>URL of the origin.</td>
</tr>
<tr>
<td>visit_type</td>
<td>str</td>
<td>Visit type of the origin.</td>
</tr>
<tr>
<td>last_update</td>
<td>timestamp (if provided)</td>
<td>Timestamp of last update (if provided upstream).</td>
</tr>
<tr>
<td>enabled</td>
<td>bool</td>
<td>Enable status of the origin.</td>
</tr>
<tr>
<td>first_seen</td>
<td>timestamp</td>
<td>Timestamp of earliest listing.</td>
</tr>
<tr>
<td>last_seen</td>
<td>timestamp</td>
<td>Timestamp of latest listing.</td>
</tr>
</tbody>
</table>

Nicolas Dandrimont
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?