HSM Coordinator Bypass

Matt Rásó-Barnett, University of Cambridge
LAD'18

Why are Cambridge interested in this?

Lustre HSM is a core part of our cold-storage/archival tier.

We went this route because we felt that the Lustre+Robinhood combination for HSM provided more flexibility to the Tape-HSM solution we use as our backend

Lustre

IEEL 3.1
Lustre 2.7

~200TiB
Filesystem

Cache

Robinhood v3.0

Archive *all* files

Release to keep
OSTs ~80% full

Lemur Copy Tool - v0.5.1

QStar Archive Manager

Posix Filesystem interface to Tape library with cache

Spectra Logic T950

2x Libraries with ~10PB of Tape per DC
Data copied to two tapes - one in each DC

HSM Backend

Lustre HSM Experiences

- ➔ Cold Storage Area: small 200TiB filesystem **dedicated** to inactive data (instead of HSM policy run off our larger scratch area). We've since copied the model with 3 of these archive-filesystems now for different user-groups.

- ➔ ARCHIVE policy archives **all files**
  RELEASE policy aims to keep OSTs between 70-85% full, preferring to release largest files first. Reserve as much of the disk for smallest-files as long as possible, to have fewer requests from Tape System

- ➔ Lustre acts primarily as the dropoff area and 'cache' for archival data
  Users purchase Archive space on this 'filesystem'. Monitor usage via Robinhood reports.

- ➔ Encourage users to tar up files to make best use of this filesystem. We use inode quotas on Lustre to limit number of files user's can purchase per-TiB as additional nudge to force them to bundle data up after copying over.

- ➔ Predominantly **ARCHIVE-heavy** workload that can generate large-backlogs of hundreds-of-thousands to millions of jobs in the Coordinator, but with occasional bursts of RESTOREs when a user wants to pull data back

Why HSM upcall?

Discussion at LUG’18 with John Hammond about our experiences when the Coordinator becomes full up with ARCHIVE jobs.

➔ As the number of jobs in the HSM coordinator queue grows large (~few hundred thousand jobs and up) both the rate at which jobs are fulfilled by the copytools, and the rate at which jobs are dispatched by Robinhood policy run, slows down dramatically. Issue has been discussed in LU-7988 and LU-8626

➔ No QoS on requests, just first-in-first-out. ARCHIVES lower priority for us. Other approach to this discussed in LU-8324

➔ Robinhood 3.1 helps with this as we can use the rate_limit feature to slow down the submission of archival commands to the coordinator - allow RESTORE jobs more chance to be interleaved and prevent large backlogs of archival jobs building up

```
rate_limit {
          period_ms = 100;
          max_count = 100; # 100/100ms = 1k/sec
          max_size = 1GB;  # 1GB/100ms = 10GB/sec
}
```

Why HSM upcall?

Idea of moving the coordinator queue into user-space has appeal to us:

- Site-Admin has the flexibility to set their own priorities
  Prioritise RESTORE over ARCHIVE for example
  Can also treat requests differently as suits the site - could use Archive ID as a QoS system, prioritising certain IDs over others, or prioritise one Filesystem's jobs over another in the queue

- Greater ability to monitor, log and understand what's happening in the queue.
  Admin can take action when there is errors, easier to PAUSE, MODIFY or REQUEUE jobs

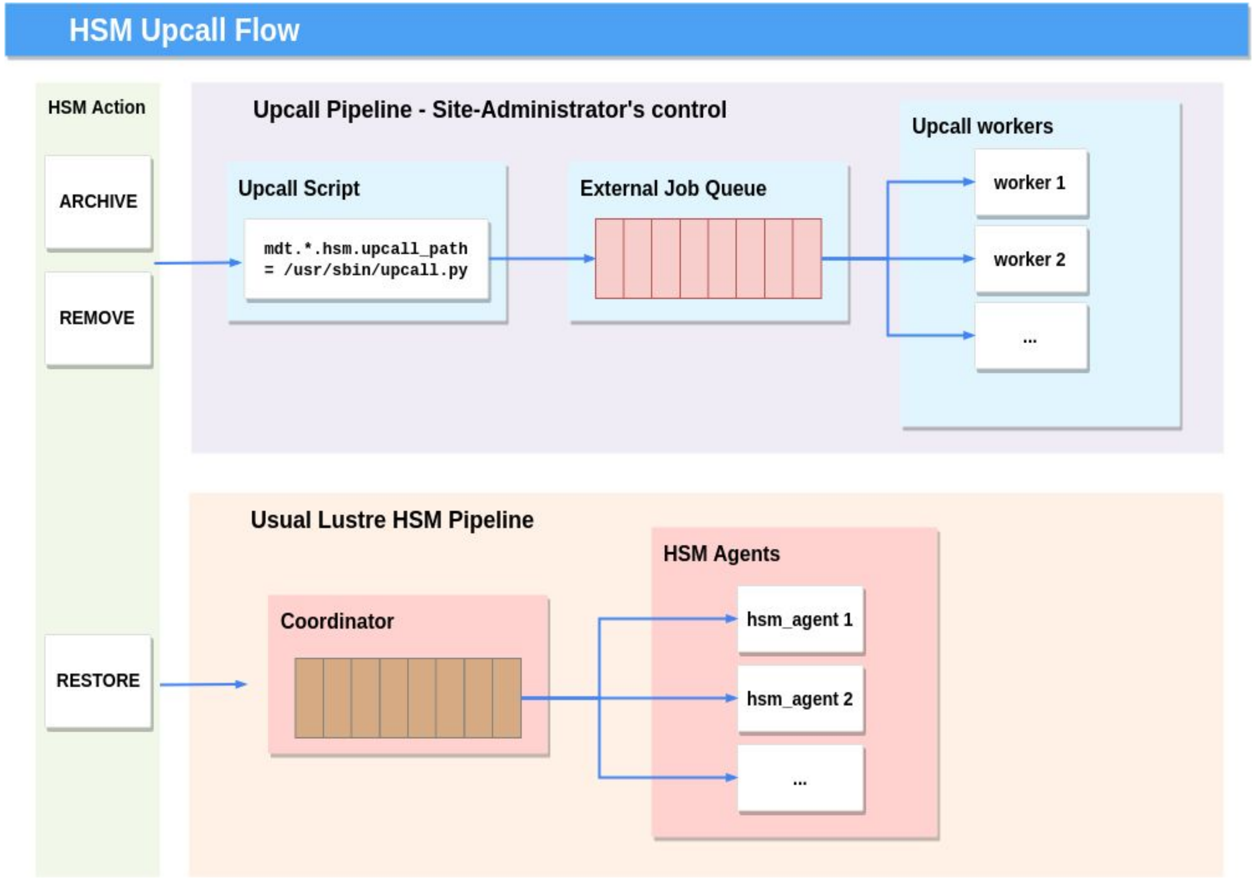
- Enables more-complex behaviours if desired:
  For example spreading low-priority ARCHIVE jobs as backfill tasks in existing SLURM infrastructure.
  Less static infrastructure, could auto-scale-up HSM-agents on-demand in response to growing backlog, and scale-down afterwards

- Simpler copy-tool design for site-admins (at least for upcall-enabled actions) akin to Robinhood team's: `lhsmtool_cmd`

Upcall Testing

From the Upcall design, we need to provide:

- MDT upcall script
  - Executed on every action
  - Should be as simple as possible, ideally without complex environment needs or failure conditions
- Some form of job/message queue to pass HSM action to the HSM agents
- Upcall Worker Script
  - Receives HSM job from the queue and invokes `lfs hsm_upcall …`



Upcall Testing: rabbitmq

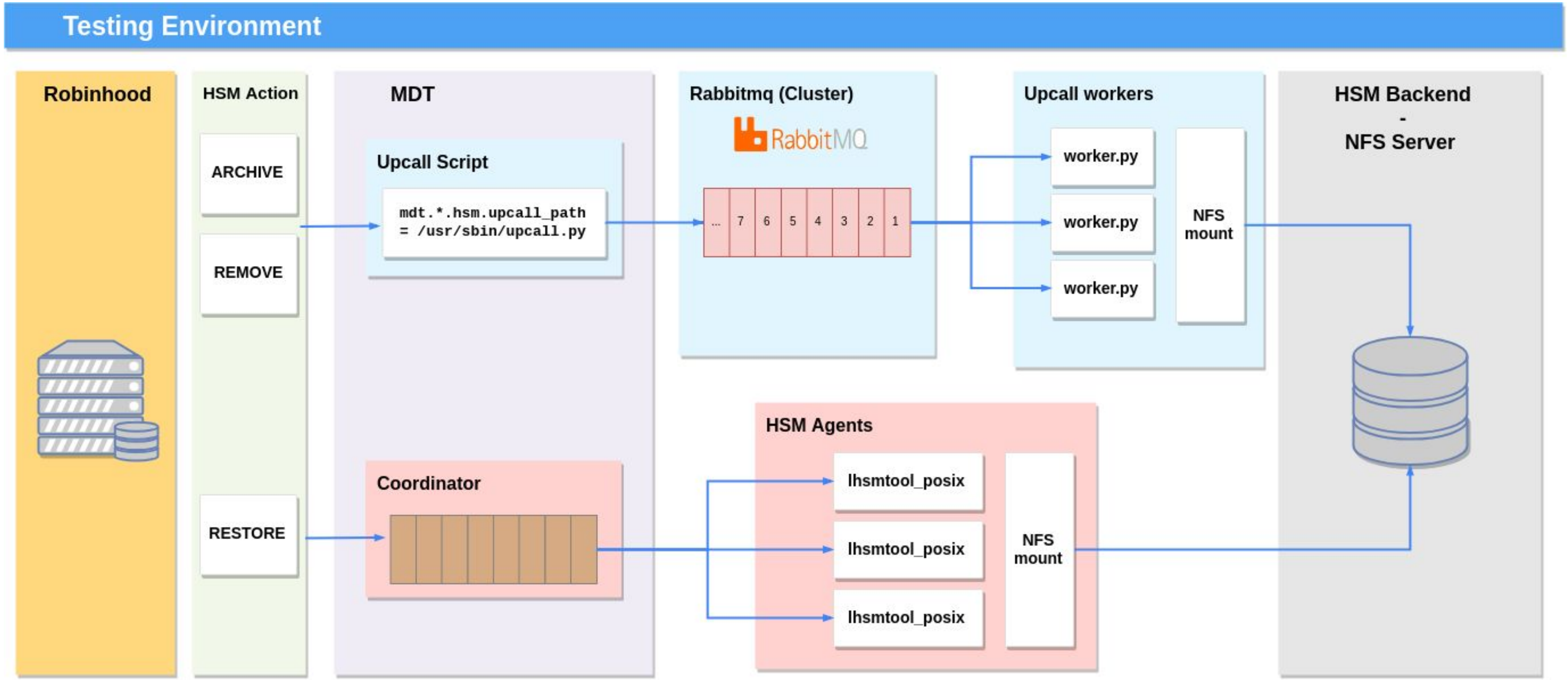
So far I’ve been testing this with a message queue: rabbitmq

Originally I planned to begin with Slurm as the message queue - still planning this (perhaps a blog post or mailing list post for anyone interested?)

Reasons for starting with rabbitmq:

- Simple publisher/clients - Queuing intelligence is in the the Rabbitmq broker. Very simple scripts to publish to and receive messages from the queue
- Client libraries in many languages - we’ve been mostly testing with python and a bit of golang so far
- Has message acknowledgement and persistence
- Queues can be easily and simply declared on the fly by either the publisher or clients
- Robust ecosystem and wide user-base in production contexts in other environments, with native clustering for high-availability (more a future consideration for a production setup)
- We already use this in other contexts in Cambridge so had some familiarity

Upcall Testing Environment



Upcall scripts

Tested with two very simple scripts (<100 LOC) using the python rabbitmq library 'pika':

- upcall.py - MDT upcall script
  - Invoked by MDT in the form:

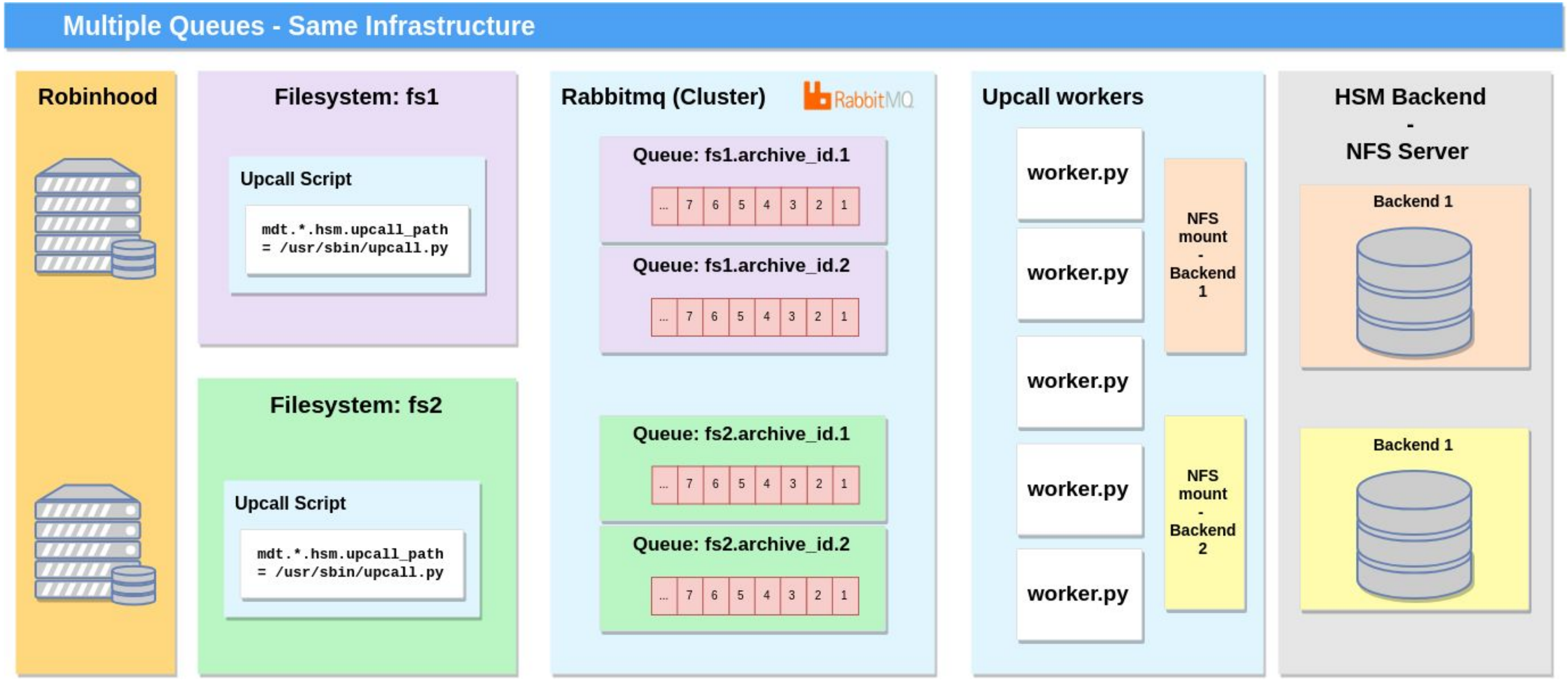
    ```
    upcall.py ACTION FSNAME ARCHIVE_ID FLAGS DATA FID…
    ```

  - Builds message from these arguments (JSON at the moment, probably unnecessary overhead for this purpose...)
  - Transmits message to particular Rabbitmq queue, based on FSNAME and ARCHIVE_ID
- worker.py
  - Analogous to a traditional HSM agent
  - Runs as a daemon on agent node and subscribes to particular queue or queue(s)
  - For each message, decodes JSON and spawns process to perform copy for each FID in the message:

    ```
    lfs hsm_upcall lhsm_worker_posix ARCHIVE FSNAME ARCHIVE_ID FLAGS DATA FID
    ```

    and acknowledges message with rabbitmq on successful copy

Additional infrastructure overhead not as bad as first seems, as it can be reused across multiple filesystems easily:



Upcall Impressions

- Works well for our primary use-case:
  - All ARCHIVE operations offloaded to dedicated queue
  - RESTORES execute instantly irrespective of ARCHIVE backlog
  - Simple to add logging at all stages of the pipeline, full visibility/monitoring of what's happening

- Particularly enjoy the flexibility this gives us:
  - Dedicated queues per filesystem and per Archive ID
  - Single pool of clients servicing these queues, with flexible Agents servicing any/all queues
  - We can control via client logic which of these queues to prioritise when needed

- More HSM infrastructure to run, monitor, but (for us) this is required with current state of HSM (carefully monitoring coordinator backlog, in-flight requests etc…), so the additional overhead is not so onerous

- Too early for proper impression of performance of upcall at the moment. Currently have quite low message publish rates in the 50-100/s range with my simple python-publisher and lots of logging going on. Some early testing with a golang based publisher shows much faster message rate of few hundred/s.

  Not really put effort into testing speed right now (subject for a more testing)

Next steps for Cambridge

- Continue testing. Scale testbed up to a larger size more comparable to our current production workload.

- Rabbitmq and Slurm comparison (we are a Slurm site on our HPC cluster):
  - Interested in exploring scheduling HSM archives as jobs spread out over larger existing slurm cluster possibly?
  - How Slurm copes with higher-throughput tasks like this

- HSM agent development
  - We are still using Lemur as our main POSIX copy-tool in production: https://github.com/whamcloud/lemur

  - We are still thinking of what our next move is going to be now that it's not being actively developed:
    - Possibly try to take maintenance on and finish pieces that are important to us
    - Or we look at keeping the backend HSM layout of Lemur, but start again with simpler HSM-upcall-based agents for ARCHIVE + `lhsmtool_cmd`-based traditional POSIX agent for RESTOREs?

  - Interested in looking at adding checksumming to the above copy-tools that we could store in an xattr perhaps and verify on RESTORE

Acknowledgements

Special thanks to John Hammond for his work on the patch and his time discussing these issues with us.

Thanks also to my colleague Javier Navarro for his input and help with testing

UNIVERSITY OF CAMBRIDGE