Table of Contents

Recurrent task and services Review

This document is the output of the Visualisation working group: Alex and Marian.

As requested in the WLCG monitoring consolidation meeting 18 July 2013, we attempt to answer the following questions regarding Recurrent task and services:

- Are we doing things consistently for all of our applications?
- If we are doing things in multiple ways, is there a good reason for it?
- Are there any new technologies that would help us here?
- How long would it take to change?
- How would that impact the other layers?

Are we doing things consistently for all of our applications?

Three kind of Scheduling are used by our applications:

- UNIX Cronjob : SAM Synchronizer
- UNIX Daemon/process : Dashb-agent
- DBMS Scheduler:
    - Oracle DBMS Scheduler (PL/QSL Procedures)
    - MyQSL Events (MySQL Procedures)

Scope of the previous scheduling system:

- DBMS scheduler
    - All DB internal processing should use it (unless multiple implementation, oracle + mysql).
- Unix Daemon (Performance):
    - Heavy initialization procedure
    - Can achieve higher frequencies
    - Stateful (can improve performance)
    - Can be forked (easy to spawn new instances)
- Unix Cronjob (Robustness):
    - Automatic restart on failures (simplevisor can achieve it for daemon)
    - Low frequency (daily operation)
    - Does not use any ressources when inactive
    - Less sensitive to memory leaks
    - Implementation could be simplified

These 3 technologies are (almost always) properly used and their usage is consitent between application. Few counter-example due to historical reason:

- DDM Stored Procedure which are called by dashb-agent
    - No multiple implementation at the DB level
- Interactive view view used dashb-agent for daily job
    - Frequency not adapted
- Topology retrieval using dashb-agent in WLCG Transfers and XRootD Dashboard
    - Code could be simplified a lot (Use of more than 100 lines of code for a simple wget)

If we are doing things in multiple ways, is there a good reason for it?

Each technologies have their own properties and should be used for what they have been designed.

Are there any new technologies that would help us here?

- **Scheduling is not a new problem in IT**
- **We do not have specific requirements**
- **Standard solutions exist**

For these three reasons, we should try to re-invent the wheel. Standard solution should always be privileged.

However, there is space for improving our strategy.

- UNIX Services:
  - Use of the python-daemon (PEP3143)
  - Possible code improvement (clarity)
  - Systemd is coming (rhel7?)
- Cron and DB internal scheduler
  - Code convention?
- (Re-)defined some best-practice
  - When to use which toolHow long would it take to change?

How long would it take to change?

Step to get there:

- Discussion required
  - Writer of Dashb-agent, SAM sync (and external person?)
  - Define a common strategy
- Small development effort
  - Mostly integration

The time to change will mainly depend on the direction we would like to take: try to unify everything, define best-practice. But it shouldn't took much (few weeks maximum).

How would that impact the other layers?

Should be independant and without any impact on others layer in the actual state.

However, depending on the choice of the "data-aggregation" group if computation should go inside or outside the storage system, scheduling of that jobs will be impacted.

Conclusion

- Things are not 100% consistant today and could be done in many ways. But does it make sense to use a single technology?
- Having a single way of scheduling our tasks would probably lead to over-engineering and complicate the overall system.

- A clear policy should be defined to help "service" writer to choose between each technology and get the best from each one.

This topic: LCG > RecurrentTaskReview
Topic revision: r1 - 2013-08-02 - AlexandreBeche