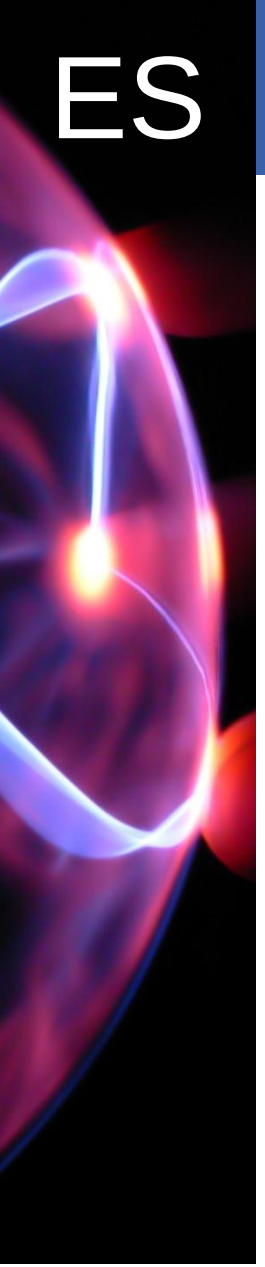

OPS TEG WG4

Maarten Litmaath
on behalf of WG4

v1.1

Introduction

- WG4 and WG5 share a TWiki page
  - https://twiki.cern.ch/twiki/bin/view/LCG/WLCGTegOperationsWG4WG5
- Contents for WG4
  - Input from experiments, sites and infrastructures
    - Top problems
    - Strategic directions
    - EGI middleware requirements process
  - Summary document for Dec F2F meeting
    - Overview
    - Technologies and tools
    - Procedures and policies
    - Areas of improvement ← will be covered here

Areas of improvement

Impact	Area
5	Documentation
7	Logging
7	Error messages
8	Robustness strategies

- On the following pages first a quick overview of those areas
- Then a set of observations and ideas for progress

Documentation

- Good documentation will help both site admins and users understand how a particular service or client is supposed to behave with a given configuration.
  - Allowing for a reasonable setup to be designed from the start and subsequent logging messages and occasional errors to be dealt with more easily.

Logging

- Log files should have sensible rotation policies compliant with the 90-day retention policy required by WLCG.
- Messages logged at a given level need to identify unambiguously which operation was executed at which time and, when applicable, for which client connection and/or by which thread.
- They need to avoid clutter that should only be visible at a higher verbosity or debug level.

Error messages

- On the server side they need to identify unambiguously which operation was attempted for which client connection at which time and why it failed.
- On the client side they need to indicate to a reasonable degree why and when the operation failed and if the problem may be transient.
- They need to avoid clutter that should only be visible at a higher verbosity or debug level.

Robustness strategies (1)

- Complexity should be reduced where possible and carefully divided between sites, experiments and infrastructure providers.
  - If sites are asked to do more, they will be more likely to fail in some respects.
- Middleware should conform to standards and popular technologies as much as possible.
  - This will allow site admins to spend less time in setting up services and understanding their behavior, and allow common tools to be used.

Robustness strategies (2)

- Services should be designed with long-term scalability in mind.
- Services and their clients should as much as possible be designed to allow for load-balanced deployment of redundant service instances.
- Services should as much as possible be designed to allow for High Availability solutions in their deployment.

Robustness strategies (3)

- Services should be more prepared to deal with minor problems preferably by design, but in any case as much as possible out of the box.
  - For example cron jobs for graceful restarts.
- Clients should allow for configurable retry and fail-over strategies where they make sense and have reasonable defaults for the maximum number of attempts and the maximum time spent.
  - A client should in particular back off from a service that is overloaded.

Observations and ideas (1)

- Documentation
  - There is a lot in good shape already
    - But not always easy to find…
  - Input/feedback from sites would help a lot
    - How is service X actually used?
    - What do admins care about?
    - Can developers stop caring about irrelevant scenarios? Remove unused features?
- Logging and error messages
  - A longstanding problem !!!
  - Bugs and RFEs in this area should have high priority

Observations and ideas (2)

- Complexity reduction
  - Reassess experiment models and actual practice
  - Examples
    - Space token consolidations by ATLAS and LHCb → simpler configuration of SEs and VO frameworks
    - LFC consolidation by ATLAS → fewer non-trivial services to be run at sites
    - CVMFS → simpler SW distribution to sites
- Standards
  - Moving to standards is happening in EMI/EGI/OSG
  - Legacy components still need support for years to come

Observations and ideas (3)

- Scalability, load balancing and High Availability
  - Assess existing and proposed services
    - How can service X scale?
      - More boxes?
      - Where are the bottlenecks?
    - Where is the stateful data?
      - Can it be shared between multiple front ends?
      - Can it be put into a H/A setup?
- Client and service robustness
  - Teething problems should be dealt with in early production tests → pilot instances
    - Collaboration between developers, VOs, very few sites
      - Example: FTS 2.2.8
    - Followed by staged rollout as usual

Requirements gathering

- Sustainability can be improved by:
  - Joint EGI/OSG/… and WLCG task forces in requirements gathering
  - Exploiting more the existing EGI/OSG/… resources for these matters
- Open bugs and RFEs should be reassessed periodically
  - Is an item still relevant?
  - Does it still have the same priority?