Agenda

- Available options to run a top BDII
- Implementation design
- Current status and challenges
- Plan
Infra at EGI

- Managing collaborations tools
- < 30 VMs, mostly Debian
- One sysadmin
How to run BDII? (pick one)

Running CentOS and outdated software

or

Running BDII on a modern debian

or

Trying something else?
Problem: running on CentOS

- Does not fit in our infra (Debian)

- Hence, everything must be special-cased: monitoring, logging, automation, security, update cycles, ...

- Backporting is a pain

- Personal distaste & experiences

- Too much effort for me

- What were the reasons for which EGI has been asked to run this service?
Problem: using BDII on debian

- BDII is made of a many (integrated) pieces
- Each of those will have to be reviewed and ported accordingly
- In the long run better than having CentOS, in my opinion
- But still too much effort for me
- What were the reasons for which EGI has been asked to run this service?
More problems

- BDII Code = Favorite Italian dish?
- Most of it is not needed for a top-BDII.
- Py3 migration + text file processing = bad feelings (*)
- Not operating a site: de facto not an user, never touched a real BDII.
- Dilemma: fixing bugs in unused functionality?

(*: possibly unwarranted)
New design: goals

- Implementation simplicity
- Cost/resource effectiveness
- High integration, low maintenance
- Long term operations
- Having fun and learning something
Non-goals

- Maintaining and evolving the BDII infrastructure
Different choices

- Does only one thing: aggregation of remote BDII
- Per site(s) synchronizations (small step)
  - rather than a global aggregation (big step)
- Trade peak memory usage for more CPU (in theory)
- Native libldap2 calls
  - avoiding LDIF processing and encoding issues(*)
- Using newer OpenLDAP mmap db engine and on-line configuration (OLC)
  - An in-memory database rather than an in-memory filesystem

(*: in reality, things are more complex)
Design pros

- Concurrency flattens resource usage (vs spiky batch behavior)
- Finer grained monitoring and metrics is dead simple to add on
- Luajit, libldap2 and systemd are here to stay...
  - ... for a long time
  - ... without breaking changes*

(*: excepting systemd, maybe... )
Design cons

- Need to keep track of entries per site (for deletions)
- Complicates site removal (to avoid orphaned entries)
- LMDB engine is not a perfect fit for this workload*

(*: mitigation planned, cf. issues slides and thereafter)
Other bonus

- Read-scalability
Comparison

- (talk here)
Status

- POC implementation (2nd half August)
- Public instance w/ basic optimizations (1st week of September)
- Replicated (September)
  - to provide availability and read scalability
- Just above 1 kLOC of Lua, deployed with Ansible
Possible redundancy
Lightweight

- Few dependencies: OpenLDAP, Luajit, systemd, xmlstarlet, sh

- HAProxy

- Master node has 2 CPU, 4GB RAM 20GB storage.
Issues

- Surprising mmap behaviour under Linux (for a non-MySQL guy)
- An interesting catastrophic failure phenomenon (see next slide)
- New software, unknown bugs, still a bit crude
- Rare LDAP errors (yet to be investigated)
- Improve and polish
- ENOTIME
csmubdd01.ops.eqi.eu-c556ea0791154b69ab9e0f15b06deed6c: CPU usage

<table>
<thead>
<tr>
<th>Metric</th>
<th>Last</th>
<th>Min</th>
<th>Avg</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU guest nice time</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
</tr>
<tr>
<td>CPU guest time</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
</tr>
<tr>
<td>CPU system time</td>
<td>1.4151 %</td>
<td>0.2012 %</td>
<td>0.7932 %</td>
<td>1.8888 %</td>
</tr>
<tr>
<td>CPU user time</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
</tr>
<tr>
<td>CPU system time</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
</tr>
<tr>
<td>CPU idle time</td>
<td>56.4956 %</td>
<td>0.621 %</td>
<td>23.2387 %</td>
<td>69.8942 %</td>
</tr>
<tr>
<td>CPU nice time</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
</tr>
<tr>
<td>CPU system time</td>
<td>0.9361 %</td>
<td>0.6454 %</td>
<td>5.6402 %</td>
<td>11.0823 %</td>
</tr>
</tbody>
</table>

CPU usage and Memory Utilization over time.
- DB growth caused by lmdb's MVCC
- Affects both changelogs and databases
- Begins to swap when core size > some % of RAM (80-90?)
  
  ... Then the fun starts :D
- Possible mitigation?
  - smaller transactions
  - less concurrency
  - adaptive scheduling: f(load), f(nconn), other?
  - compaction cycle
  - more RAM/$$ (*)

(*): affording to be bothered less often, not a fix per se)
- Even greater DB growth on replica
- However no swapping observed: working set always in memory.
- Hypothesis (speculation):
  - write txn are much bigger due to the way the replication stream is committed
  - pages are less fragmented (# of writer = 1)
- Unknown: how will it behave under load/clients?
- Mitigations:
  - compaction cycle
  - timelimit (?)
  - -> real observation under load is needed!
Graph: proxy outbound

last 3 months
Graph: disk usage

master, last 3 months
Plan (ideally)

- Implementing concurrency mitigations
- Better systemd integration
- Metric extractions, grafana dashboard, ...
Plan (reality)
Last slide