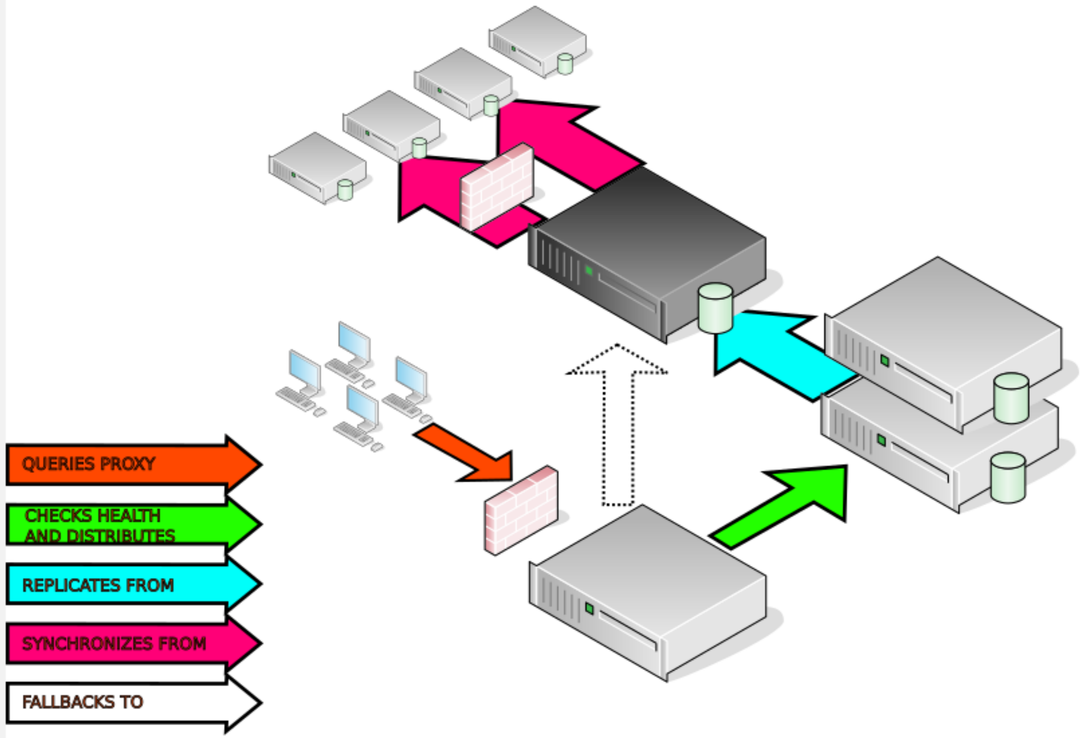
EGI's top BDII



2022-12-01

B. Jacobs, sysadmin for

The EGI Foundation

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Running CentOS and outdated software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 unwarrented)

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 BDIIs
- 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

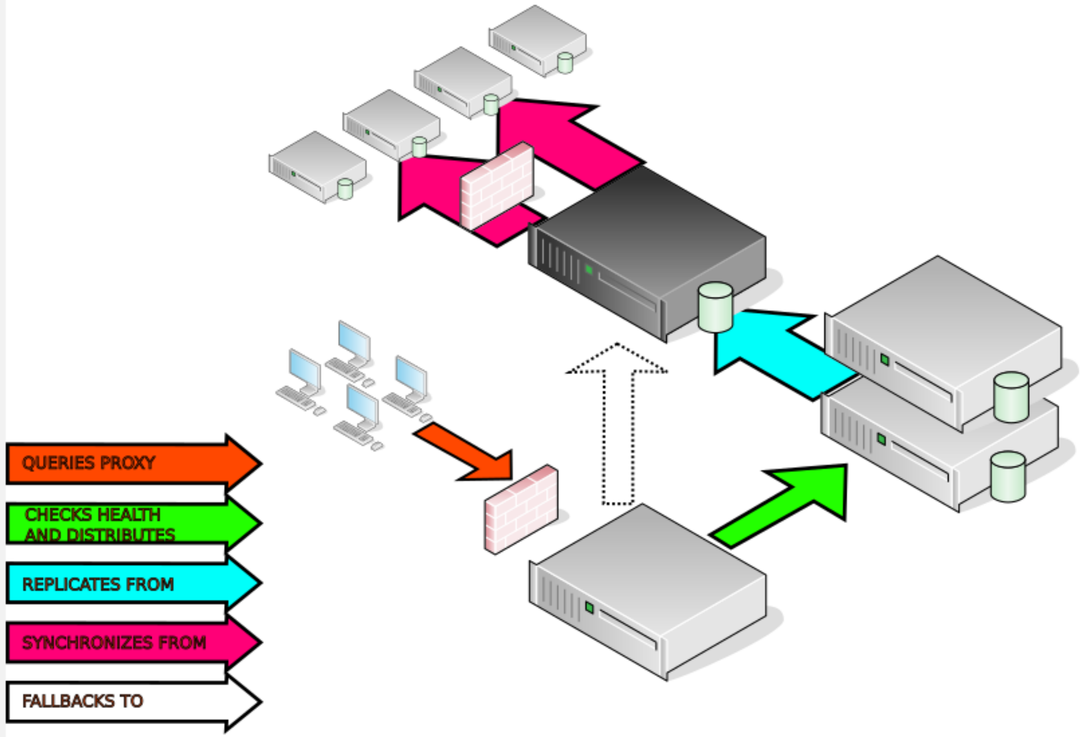
Comparison

- (talk here)

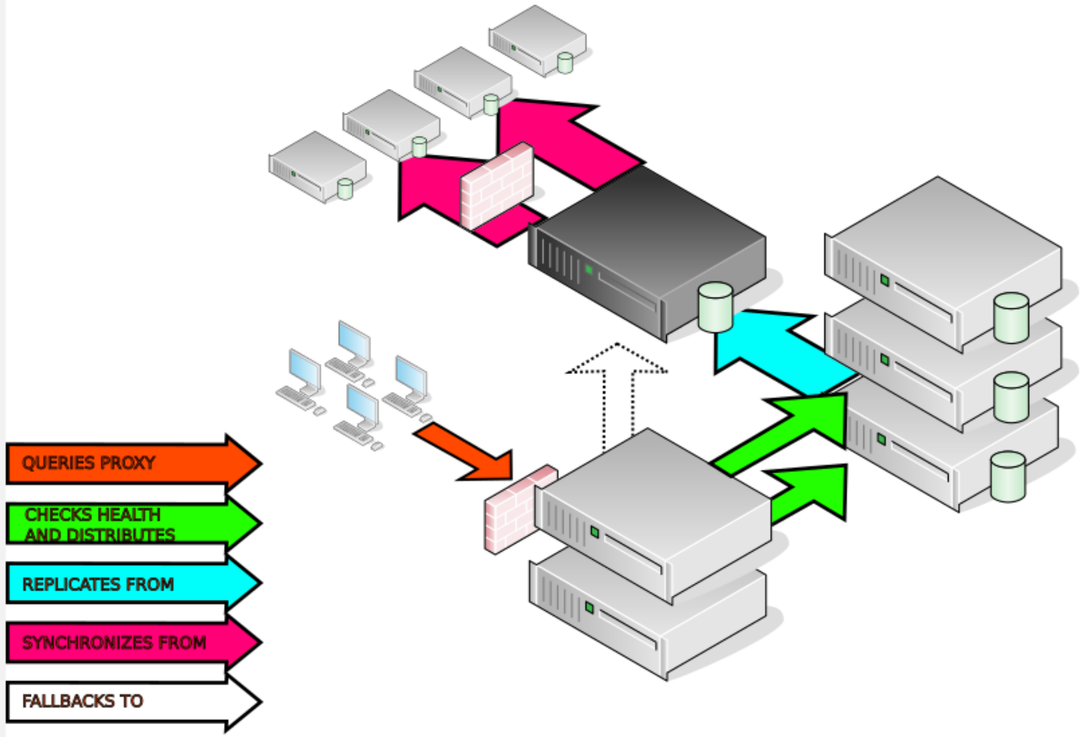
Status

- 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

Architecture diagram



Possible redundance



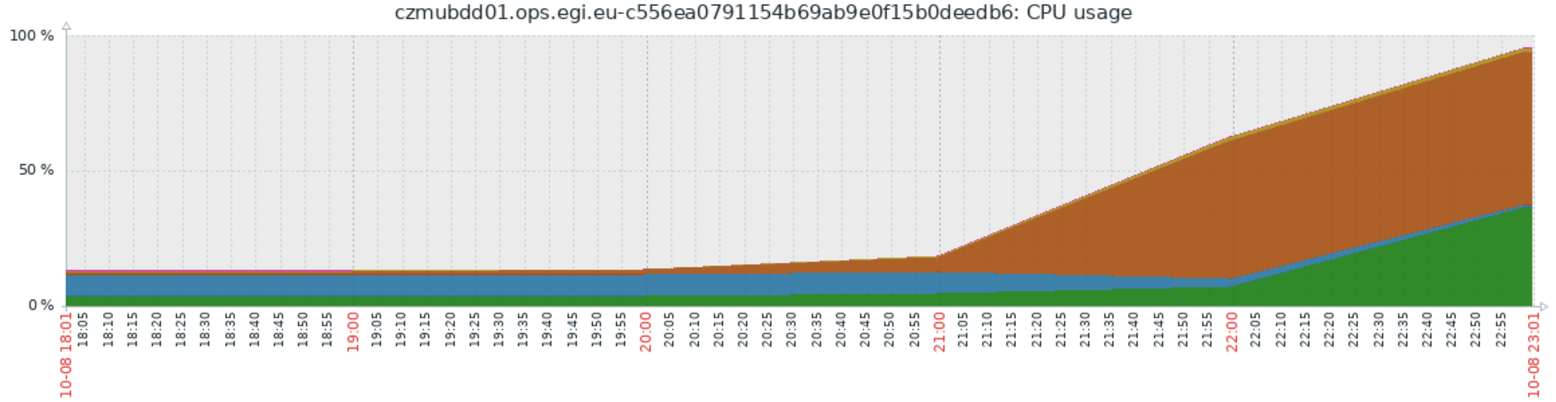
Lightweight

- Few dependencies: OpenLDAP, Luajit, systemd, xmlstarlet, sh
- HAProxy
- Master node has 2 CPU, 4GB RAM 20GB storage.

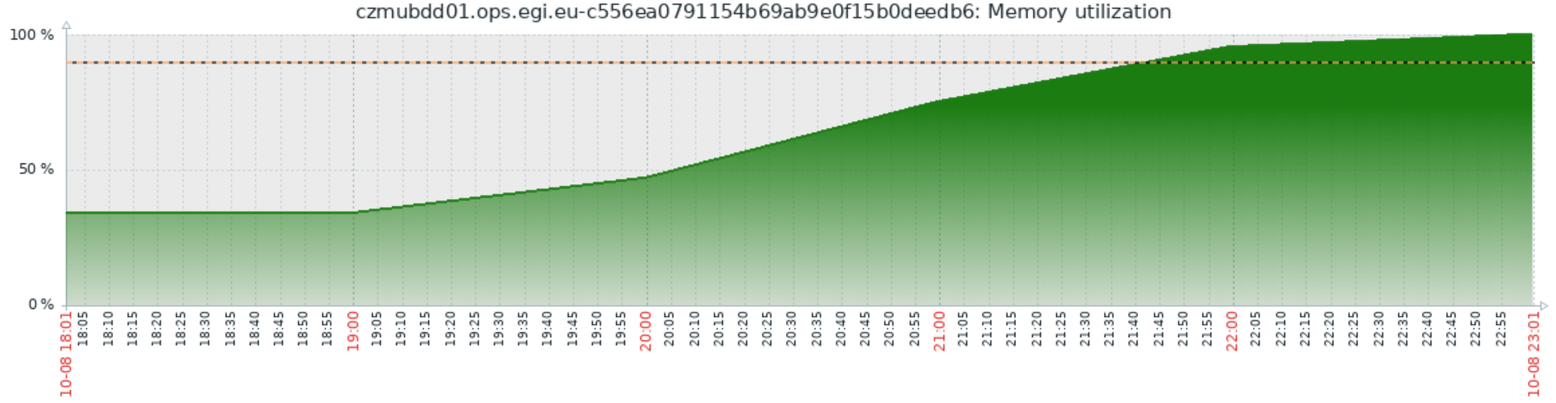
Issues

- 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

Issues (cont.)

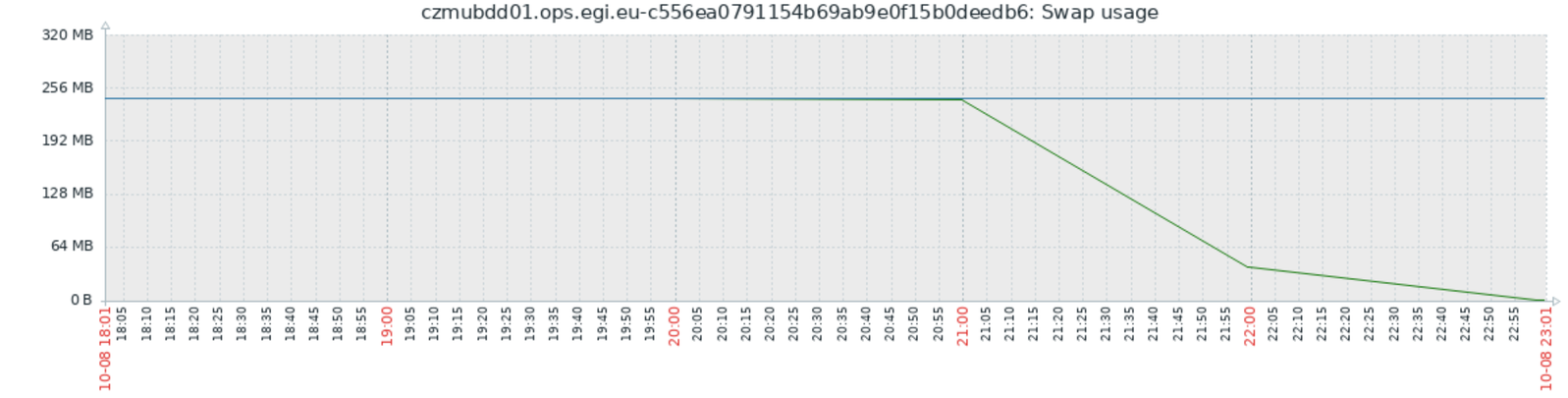


		last	min	avg	max
CPU guest nice time	[avg]	0 %	0 %	0 %	0 %
CPU guest time	[avg]	0 %	0 %	0 %	0 %
CPU softirq time	[avg]	1.4551 %	0.2012 %	0.7932 %	1.8888 %
CPU interrupt time	[avg]	0 %	0 %	0 %	0 %
CPU steal time	[avg]	0 %	0 %	0 %	0 %
CPU iowait time	[avg]	56.4956 %	0.621 %	23.2387 %	69.8942 %
CPU nice time	[avg]	0 %	0 %	0 %	0 %
CPU user time	[avg]	0.9161 %	0.6454 %	5.6402 %	11.8022 %
CPU system time	[avg]	36.2308 %	2.4081 %	10.6591 %	54.1647 %

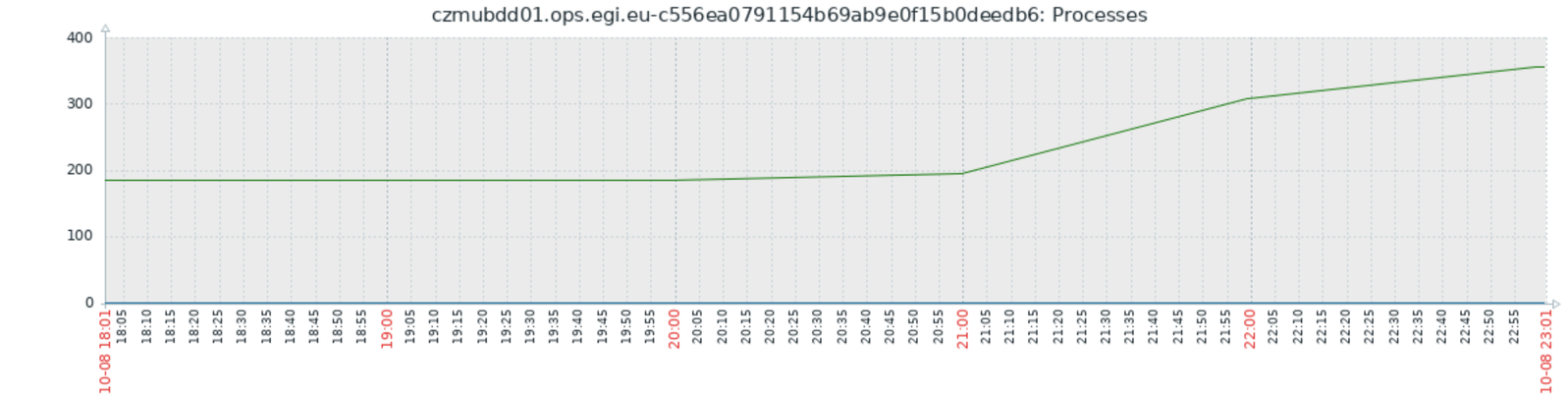


		last	min	avg	max
Memory utilization	[avg]	100 %	28.6337 %	70.0961 %	100 %
Trigger: High memory utilization (>90% for 5m)	[> 90]				

Issues (cont.)



		last	min	avg	max
Free swap space	[avg]	0 B	0 B	154.06 MB	244 MB
Total swap space	[avg]	244 MB	244 MB	244 MB	244 MB



		last	min	avg	max
Number of processes	[avg]	356	182	246	360
Number of running processes	[avg]	0	0	0	4
Maximum number of processes	[no data]				

Trigger: Configured max number of processes is too low (< 1024) [< 1024]

Issues (cont.)

- 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)

Issues (cont.)

- Even greater DB growth on replica

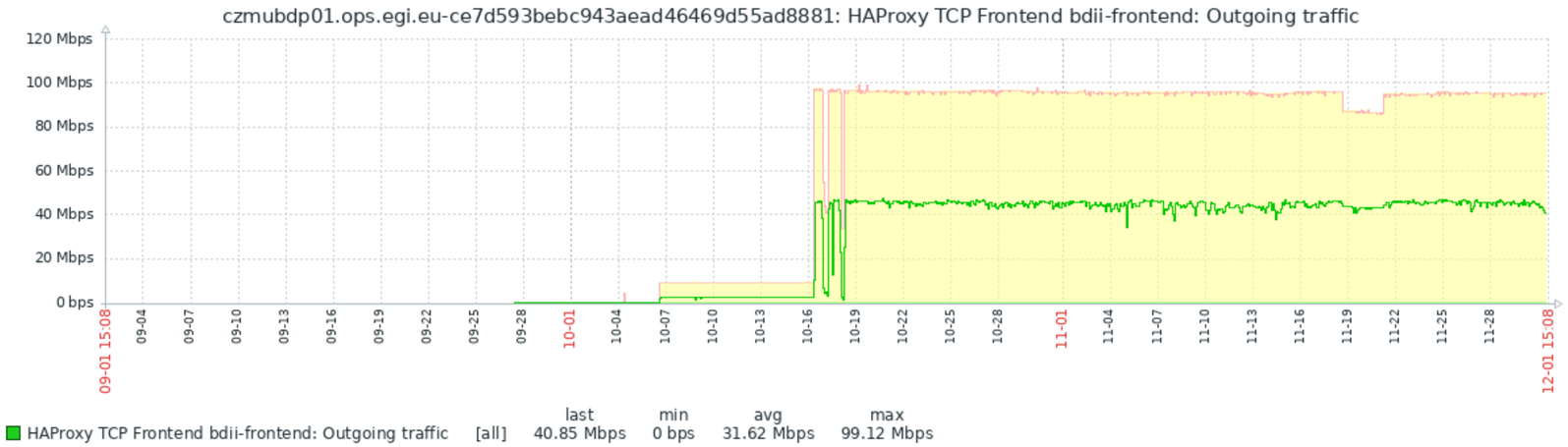
- However no swapping observed: working set always in memory.

- Hypothesis (speculation):
  - write txn are much bigger due to the way the replication stream is committed
  - pages are less fragmented (# of writer = 1)

- Unkown: how will it behave under load/clients?

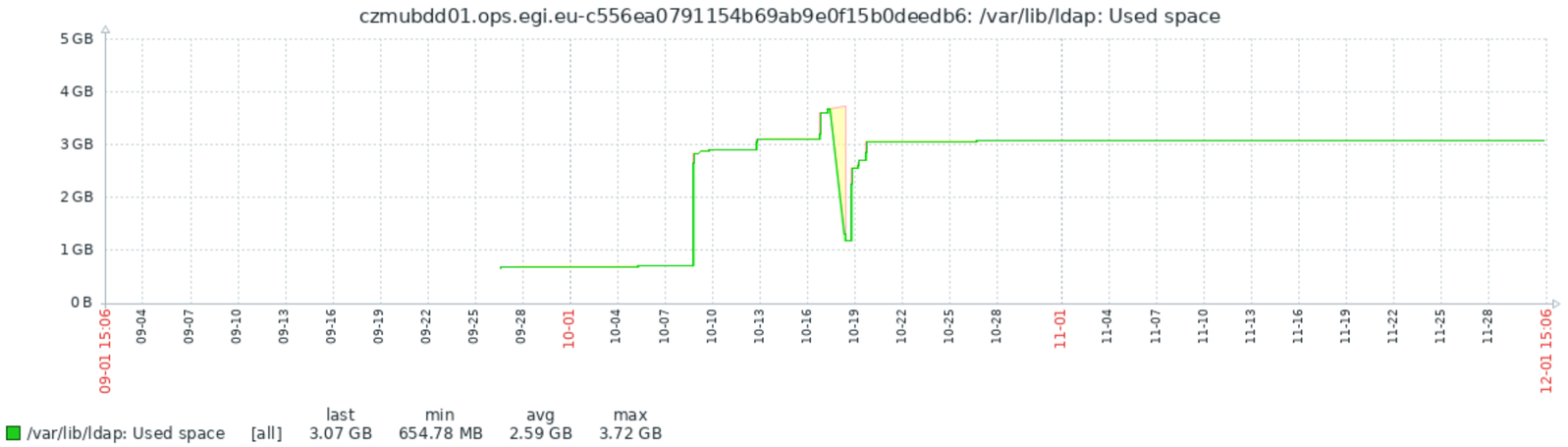
- Mitigations:
  - compaction cycle
  - timelimit (?)
  - -> real observation under load is needed!

Graph: proxy outbound



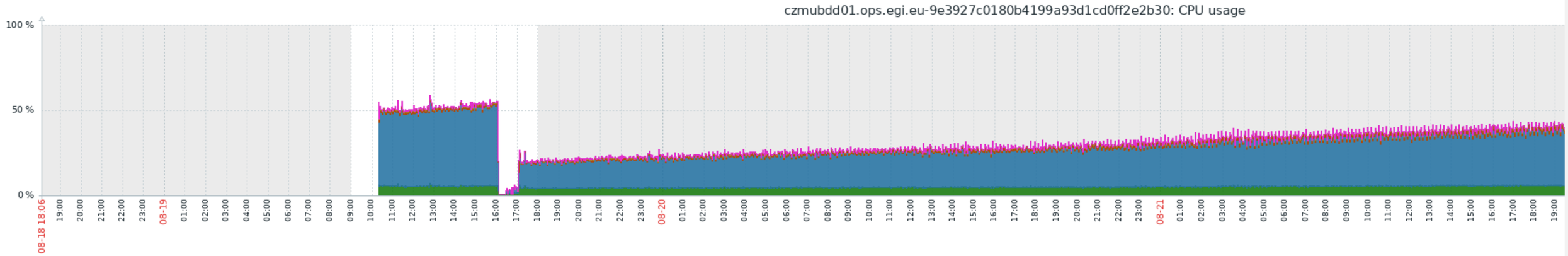
last 3 months

Graph: disk usage



master, last 3 months

Graph: no-optimization



		last	min	avg	max
CPU guest nice time	[avg]	0 %	0 %	0 %	0 %
CPU guest time	[avg]	0 %	0 %	0 %	0 %
CPU softirq time	[avg]	0.5595 %	0.008355 %	0.3732 %	1.0155 %
CPU interrupt time	[avg]	0 %	0 %	0 %	0 %
CPU steal time	[avg]	0 %	0 %	0 %	0 %
CPU iowait time	[avg]	1.9344 %	0 %	1.3877 %	14.0344 %
CPU nice time	[avg]	0 %	0 %	0.000009961 %	0.03343 %
CPU user time	[avg]	50.1389 %	0.09184 %	29.5584 %	62.5816 %
CPU system time	[avg]	4.9609 %	0.06676 %	4.8344 %	7.7393 %

Plan (ideally)

- Implementing concurrency mitigations

- Better systemd integration

- Metric extractions, grafana dashboard, ...

Plan (reality)

Last slide