Architect of an Open World™

Centralized Lustre monitoring on Bull platforms

September 16, 2013

Florent THERY

Parallel File Systems
Extreme Computing R&D

Agenda

- Bull Extreme Computing
- Goals of our approach
- Architecture and chosen metrics
- Expected benefits

Bull Extreme Computing

Actor of the HPC market

- 1st european manufacturer
- Several “high-scale” supercomputers delivered in 2013
  - MeteoFrance
    - Direct Liquid Cooling (DLC) technology
    - first “full CPU” system at green500
  - Sara, Dresden, IT4I

Contributor to Lustre development

- EOFS founding member
- Integrator of lustre 2.1, 2.4, incoming 2.5
- Many bugs reported and patches submitted
- Working on Lustre static code analysis project

Goals of our approach

Lustre monitoring complexity

- The devil is in the detail
  - /proc holds all the needed information, but ..
  - Metrics are spread over a multitude of nodes and devices
  - Aggregating and visualizing this information is not an easy work
- How to detect an abnormal situation ?
  - Normality depends on the observer's point of view
  - Hard to find a generic algorithm for all situations
  - Need to define configurable and flexible mechanisms to adapt to several contexts

Bull Lustre monitoring goals

General goals and constraints

- Collect data and display graphs for lustre targets, nodes and filesystems
- Raise alerts based on administrator's configured thresholds
- Easy integration in current Bull HPC software base

Bull Lustre monitoring goals

Functional objectives

- Three main usability objectives:
  - live visualization (based on graphs and alerts system)

    → "*OSTxxxx is close to space saturation, 6 hours left if the current workload goes on like this, I need to do something now*"

  - backward analysis (based on graphs)

    → "*Two days ago, the request mean wait time on node ossX has exploded while the number of requests per second remained stable, what happened ?*"

  - easy to configure alerts system

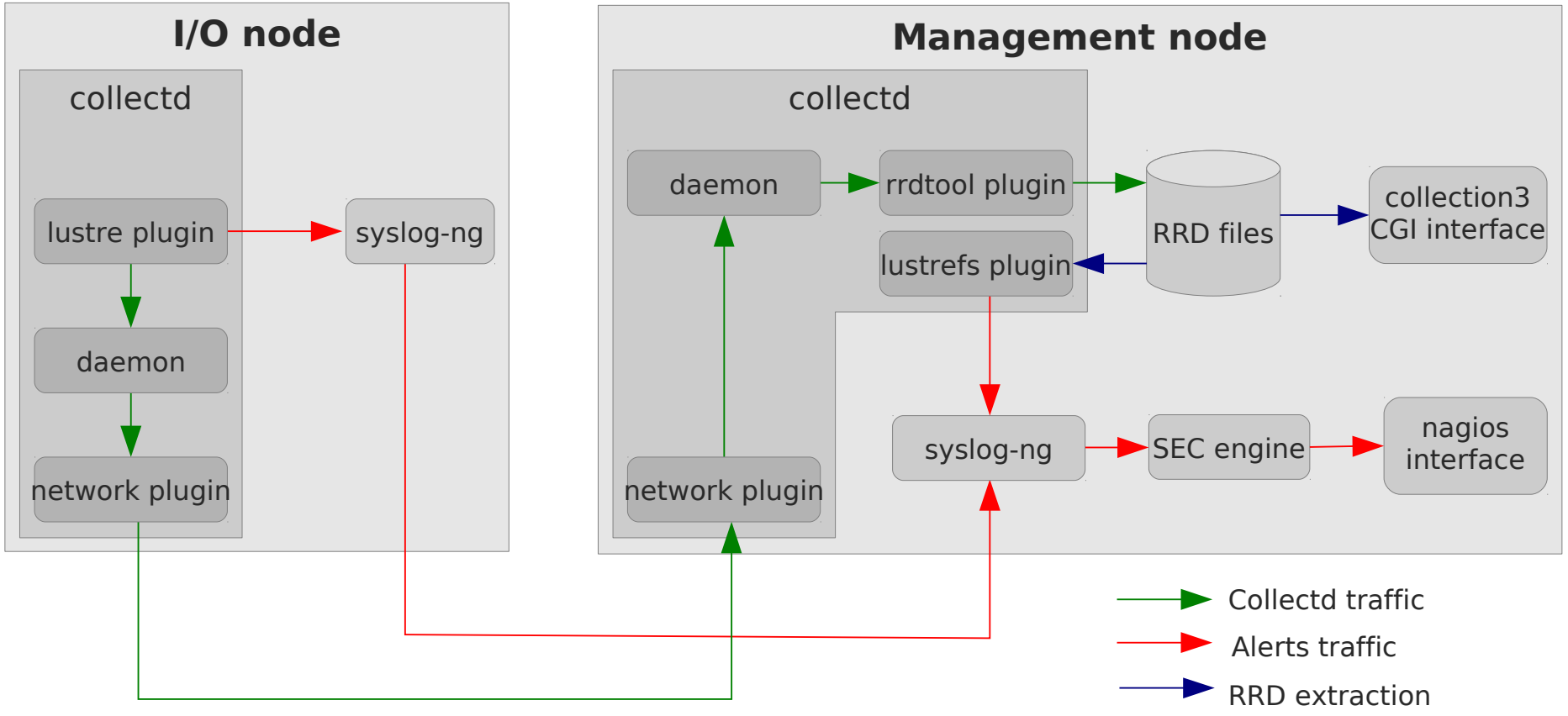
    → "*When node zzz has crashed, the number of RPC/s was so high, I need to setup a threshold to be notified if it reproduces*"

    → "*I'm expecting a huge metadata load in the next few days, I must increase metrics xxx and yyy threshold values*"

Architecture and chosen metrics

Bull Lustre monitoring architecture

General overview



- Collected data flow through collectd mechanism
- Alerts flow through syslog-ng, SEC and nagios
- Data are stored and displayed on the management node

Bull Lustre monitoring architecture

Metrics collection and display

- Metrics are collected on each I/O node for each OST/MDT target
  → 1 RRD file generated per “target” metric

- Two sources for data aggregation
  - lustre plugin on I/O nodes
    → aggregate and store “node” data
    → 1 RRD file generated per “node” metric

  - web CGI script on management node
    → display “target” and “node” data
    → aggregate and display “filesystem” data from “target” RRD files

Bull Lustre monitoring architecture

Collected metrics

Metric	Lustre Node Type	One graph displayed per: (Target, Node, Filesystem)
Read/Write bandwidth	OSS	Target, Node, Filesystem
Read/Write IOPS	OSS	Target, Node, Filesystem
I/O size	OSS	Target
Request mean wait time	OSS	Node
RPC/s	OSS (I/O requests) MDS (metadata requests)	Node
Disk usage	OSS, MDS	Target, Filesystem
Inode usage	MDS	Target, Filesystem
Metadata operations/s	MDS	Target
Clients connected	MDS	Target

Bull Lustre monitoring architecture

Thresholds

- Thresholds setup is performed through configuration files on each I/O node
  - by default, no thresholds, must be activated and configured by administrator
- Thresholds are checked in two places
  - lustre plugin on I/O nodes
    → check “target” and “node” thresholds
  - lustrefs plugin on management node
    → aggregate and check “filesystem” thresholds from “target” RRD files
- Overtaken threshold means syslog warning message
    → SEC engine matches it and pushes an alert to nagios.
- So far, only maximum thresholds supported

Bull Lustre monitoring architecture

Currently defined thresholds

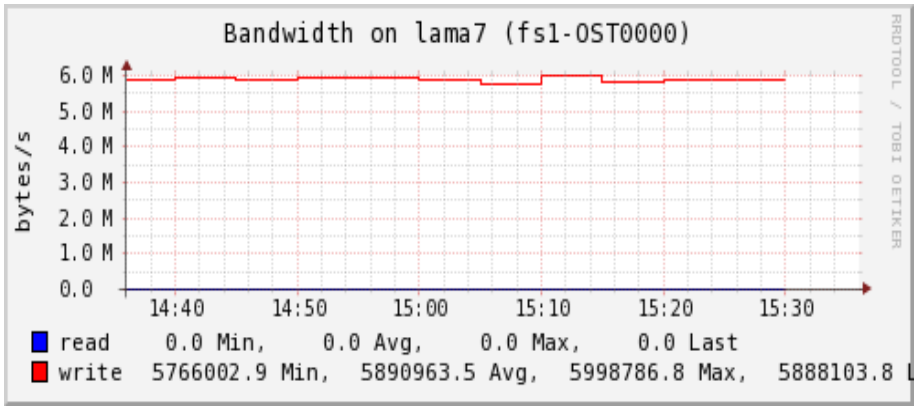
Threshold name	Scope	Unit
oss_requests_per_second	OSS	Absolute value
oss_requests_waittime	OSS	Micro-seconds
iops_per_ost	OST	Absolute value
iops_per_oss	OSS	Absolute value
ost_disk_usage	OST	Percentage
mds_requests_per_second	MDS	Absolute value
mdt_disk_usage	MDT	Percentage
mdt_inode_usage	MDT	Percentage
fs_disk_usage	Filesystem	Percentage
fs_iops	Filesystem	Absolute value

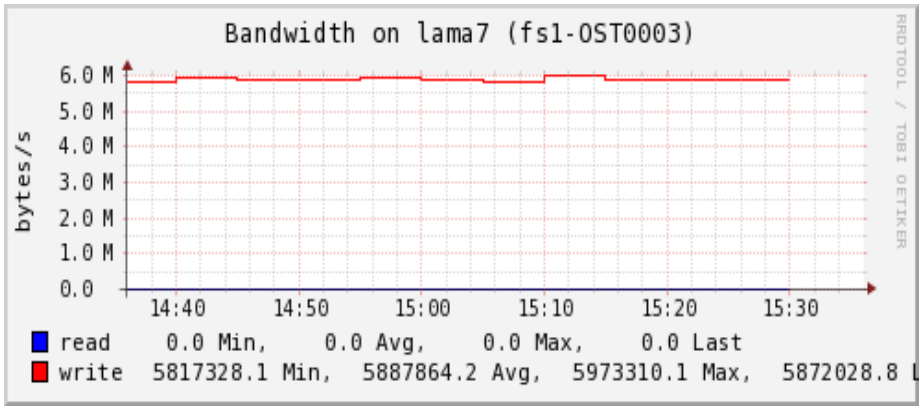
Expected benefits

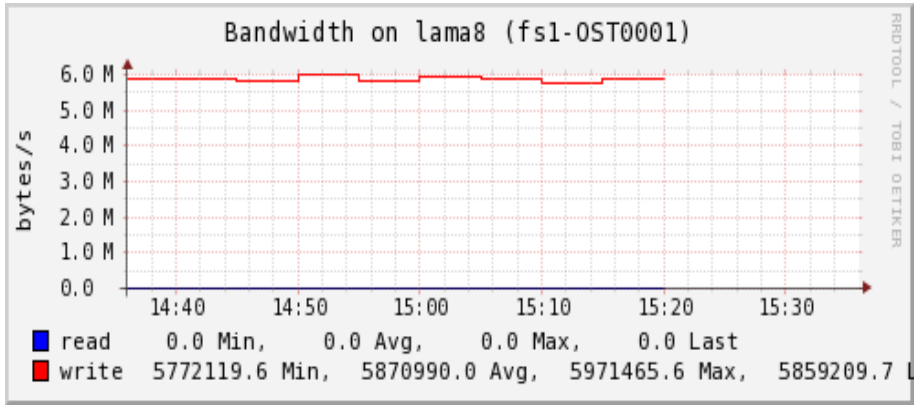
Use case

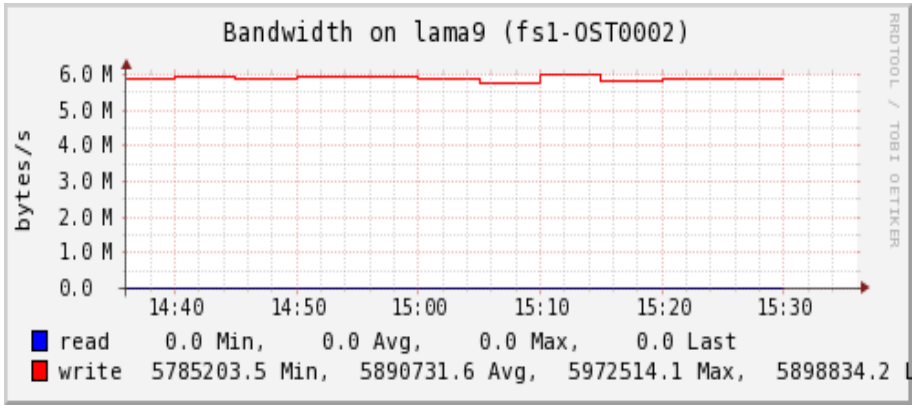
- Job xxx takes too long to finish compared to yesterday, what's happening ?

Use case



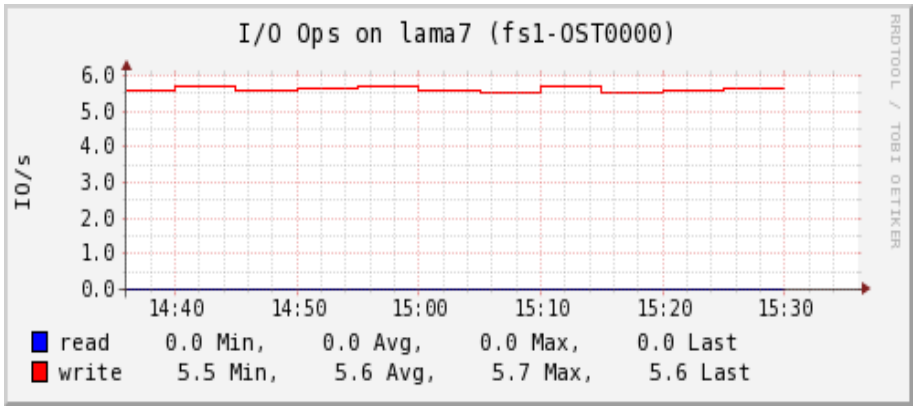


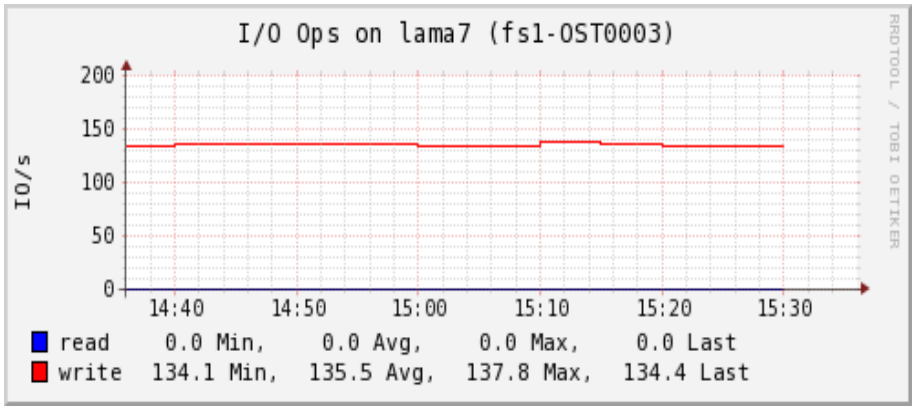


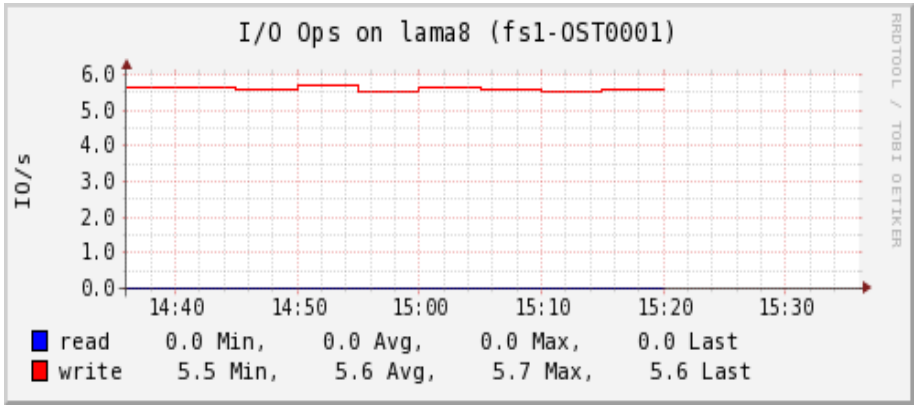


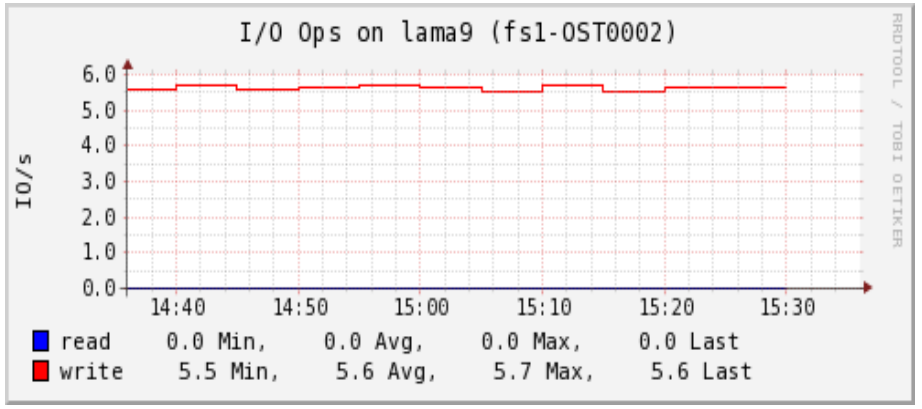
- *“Mhm, bandwidth is low, but about the same on all OSTs. What about IOPS ?”*

Use case





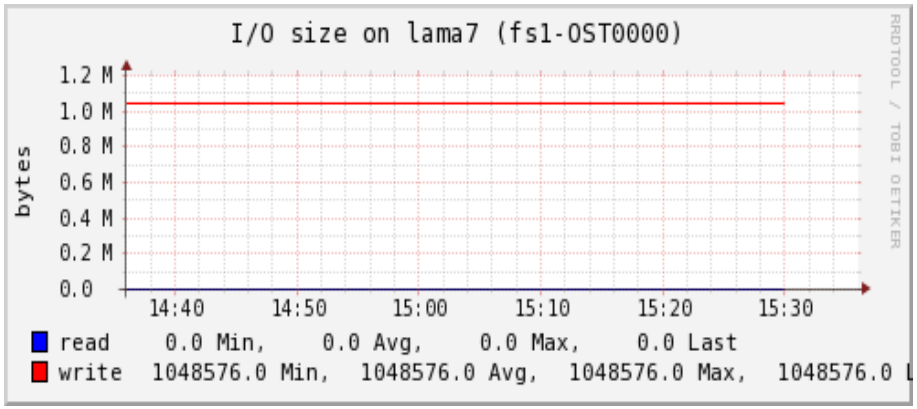


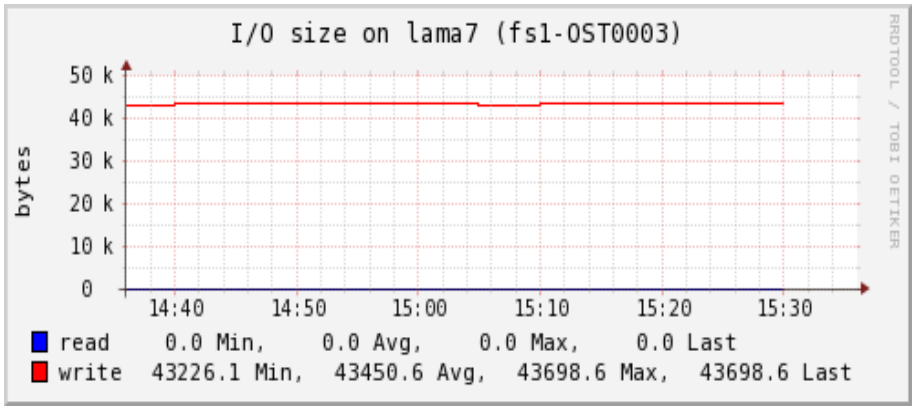


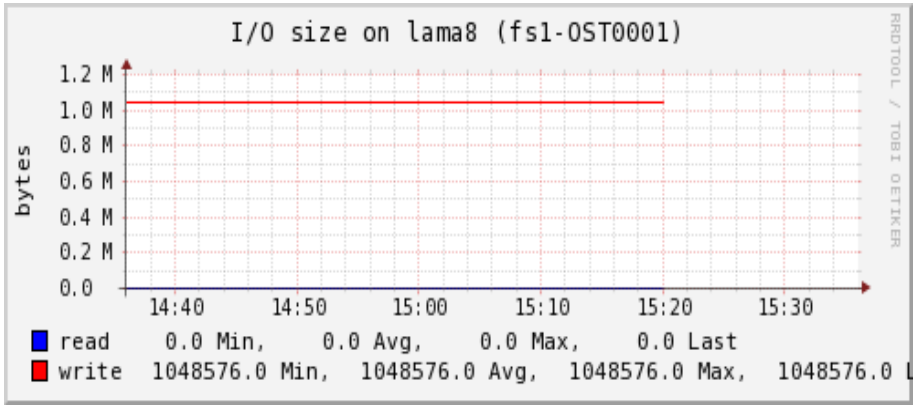
- *“OST0003 handles nearly 150 IOPS while other OSTs are handling between 5 and 6. Weird, isn't it ?*

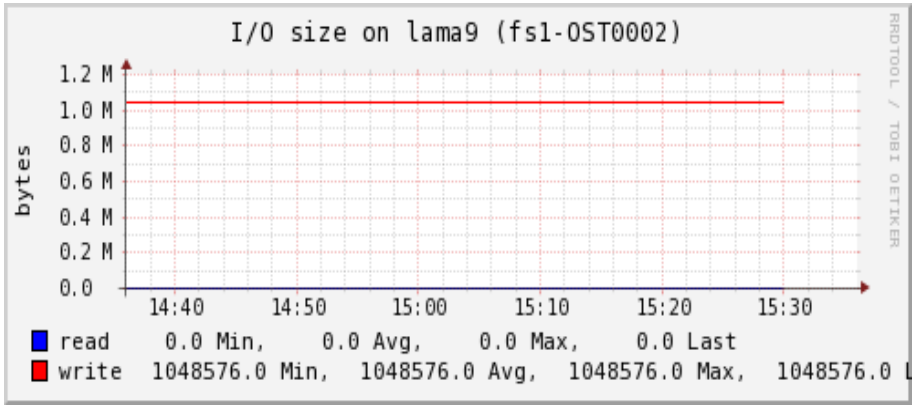
  Let's have a look at the I/O size ..”

Use case









- *“Gosh ! Why is OST0003 request size only 40k ???*
  Hum, that explains the number of IOPS treated on this OST.
  # clush -w lama7 'give me -a reason'”

Future work

Testing at scale

- Figure out the impact on I/O nodes CPU/memory consumption
- Evaluate current metrics relevance
- Evaluate RRD databases relevance for data storage

Enhancement directions

- Add support for job_stats lustre feature
- Study a more user friendly graphs display mechanism
- Extend thresholds possibilities
  - min values
  - more automatic cross-referencing schemes

Conclusion

- Don't expect exact figures !
  - not a performance measurement tool
  - helps analysis and prevention of odd behaviours
- Facilitate information cross-reference
  - e.g. easy to compare OSTs bandwidth values
- Uses recognized tools in HPC world
- Easy configuration and extensibility

Architect of an Open World™

Appendix 1

Metrics retrieval

Metric	Where do we get information
Read/Write bandwidth Read/Write IOPS I/O size	/proc/fs/lustre/obdfilter/<ost label>/stats Entries “read_bytes” and “write_bytes”
Request mean wait time	/proc/fs/lustre/obdfilter/ost_io/stats, entry “req_waittime”
RPC/s	OSS: /proc/fs/lustre/OSS/ost_io/stats, entry “req_waittime” MDS: /proc/fs/lustre/mds/MDS/mdt/stats, entry “req_waittime”
Disk usage	MDS: /proc/fs/lustre/osd-ldiskfs/<mdt label>/{kbytestotal & kbytesfree} OSS: /proc/fs/lustre/obdfilter/<ost label>/{kbytestotal & kbytesfree}
Inode usage	/proc/fs/lustre/osd-ldiskfs/<mdt label>/{filestotal & filesfree}
Metadata operations/s	/proc/fs/lustre/mdt/<mdt label>/md_stats Entries: open, close, mknod, unlink, mkdir, rmdir, rename, getattr, setattr, getxattr, link, statfs
Clients connected	A = /proc/fs/lustre/mdt/<mdt label>/num_exports B = Number of non-clients connected to the MDT (count uuids in /proc/fs/lustre/mdt/<mdt label>/exports/* matching the regular expression: /^([\w-]+)-(OST|MDT).*/ Nbclients = A – B – 1

Appendix 2

RRD files

- RRD files loose precision over time, but ..
- RRD files keep a fixed size over time
- One or more archives are created inside the file with different resolutions
  - Default collectd “rrdtool” plugin archives:
    - Last week archived with a resolution of 5 minutes
    - Last month archived with a resolution of 35 minutes
    - Last year archived with a resolution of 435 minutes
- RRD file size with resolution and history above
  - with 1 variable saved: 172KB
  - with 2 variables saved: 336KB
  - with 12 variables saved: 1968KB