GLUSTER(FS): A HUNT FOR THE PERFECT FILE SYSTEM AT PCDS/LCLS

Igor Gaponenko and Ling Cherd Ho

April ,20 2012

(*Presentation for SLAC/PPA/SCA+SCS*)

What's in this talk

- PCDS/LCLS requirements for a file storage
- An overview of GlusterFS
- First-hands experience
- Conclusions, plans, etc.

Motivations, requirements, etc.

What we and our users expect from a file storage

What our users need

- **Experiments & users:**
  - 50+ experiments/year (~150 so far) at 6 instruments (detectors)
  - Independent (sometimes competing) experimental groups
  - A typical experiment is taking data for 5 days
  - >100 of users a year: coming and going
- **Data rates & volumes:**
  - Data production rate at DAQ: up to **1 GB/s** or higher
  - No data reduction in ONLINE (and OFFLINE)
  - Large data volumes: over **>1 PB/year** (**2.2 PB** so far)
  - >10,000 of large files: **1..100 GB/file** (over **100,000 files** so far)
  - >1,000,000 of small files: **~10 MB/file** (over **10,000,000 files** so far)
  - Aggregate data processing/analysis rates: a **few GB/s** or higher
  - >100 of processing jobs run in parallel
  - Key data formats: **XTC** and **HDF5**

What our users need (contd.)

- **Key data management policies:**
  - "Unlimited" storage for each experiment
  - Data should be "safe"
    - archived to HPSS (**2 copies** on different sets of tapes)
  - Data retention: **1 year on disk, 10 years on tape**
  - "Zero" DAQ –to-OFFLINE migration latency
    - Fast feedback needed to tune up on-going experiments
  - Little or no data sharing between experimental groups
    - Data privacy is important due to competition
  - "High" data availability on disk: **24/7**

Data processing & analysis

- **Centralized XTC to HDF5 translation for 10% of raw data**
- **Our users have a "zoo" or tools and frameworks:**
  - MATLAB, C, C++, Python, MPI, OpenMP, etc.
- **We have very little control over:**
  - Their data processing frameworks (we provide them with 3)
  - How and when they process data
  - Derived data formats and output rates & volumes
  - Which data are exported
- **And our chances to change that *status-quo* are low:**
  - Too many groups
  - Too little time for a group to prepare for their experiment
  - Too few of us to "baby-seat" each user/group

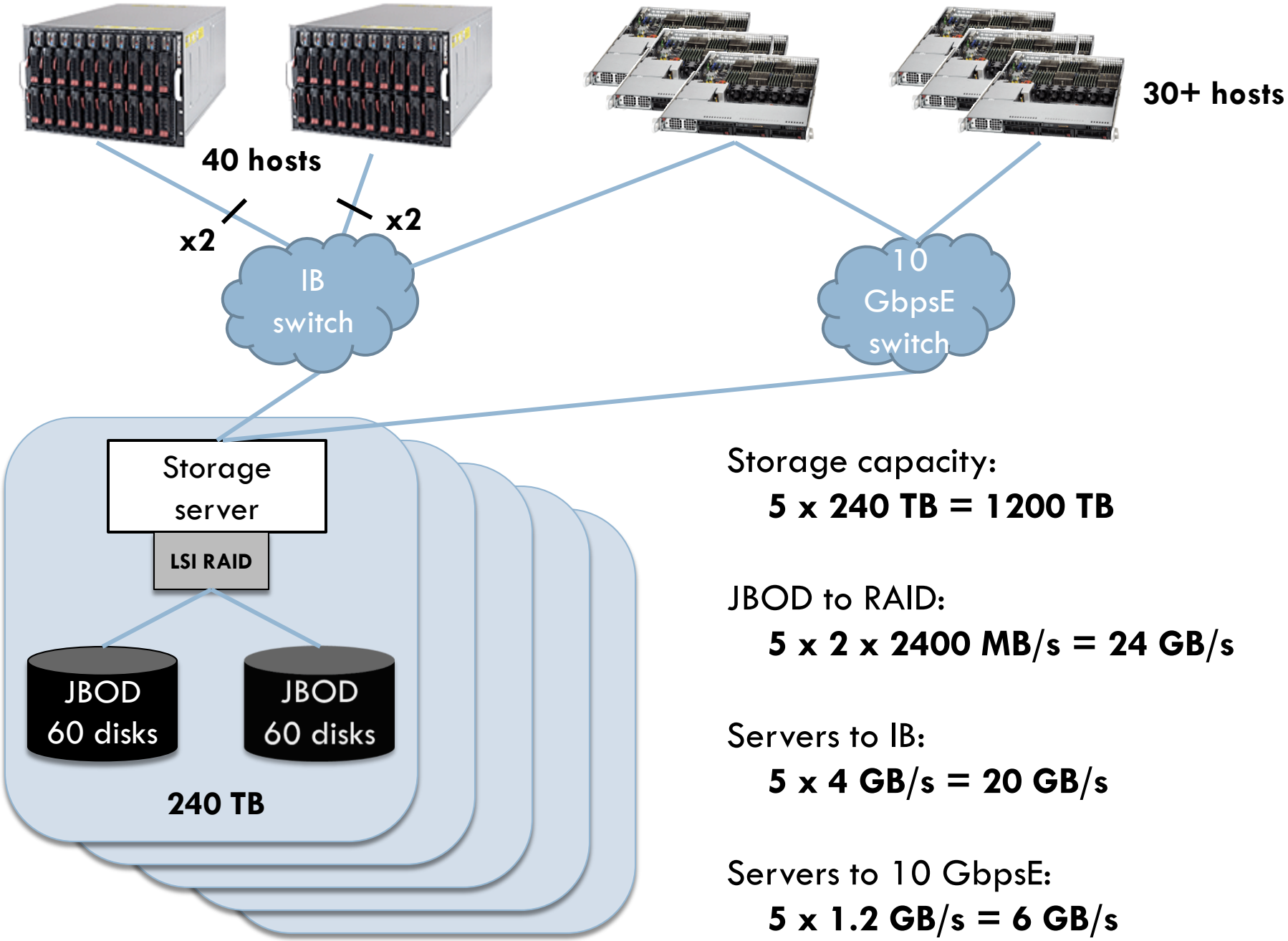
The only solution which would work : POSIX file system

A file system we (PCDS) want

- **Can run on top (and efficiently utilize) cheap hardware**
- **Is inexpensive to own:**
  - license, support, etc.
- **Is easy to install, maintain and troubleshoot:**
  - By PCDS and SLAC Comp.Dept.
  - Preferably w/o off-SLAC paid expertise
- **Must have a broad user base:**
  - Benefiting from someone else finding and fixing bugs

Our “cheap” hardware

- **Servers and storage:**
  - 5 file servers: dual CPU x5520, 32 GB, RHEL6
  - One LSI MegaRaid 9285e (1 GB memory cache) per server
  - 2 JBODs per server connect to one RAID card
  - JBOD: x60 of 2 TB SAS disks => 120 TB
  - 4 LUNs per JBOD: RAID6 13+2, formatted as XFS, 8 MB read-ahead per LUN
  - Total raw disk capacity of 10 JBODS: 1200 TB
- **Network: QDR IB and 10 GbpsE**
- **Clients:**
  - IB: 40 SuperMicro blades
  - 10 GbpsE : 18 DAQ and 10+ interactive analysis machines
  - 3 dual-network (IB + E) data migration machines



Storage capacity:

5 x 240 TB = 1200 TB

JBOD to RAID:

5 x 2 x 2400 MB/s = 24 GB/s

Servers to IB:

5 x 4 GB/s = 20 GB/s

Servers to 10 GbpsE:

5 x 1.2 GB/s = 6 GB/s

Gluster overview

Most interesting features. Main focus on the same functionality which we're already relying upon in Lustre…

GlusterFS

- **Cluster type file system similar to Lustre, GPFS and others:**
  - Data are spread between “peers” (computers)
  - Each pear has 1 or many storage “bricks” (local file systems)
    - Bricks are just file system paths. Two bricks can share the same OST (LUN)
  - Transport: TCP/IP, IB RDMA, TCP/IP over IB
  - Single namespace
  - POSIX ACLs, storage quotas
  - Supports “striping” and “mirroring”of files
  - Expandable, scalable
- **Open source**
  - www.gluster.org
  - Recently acquired by RedHat (committed to keep it as open-source)
- **Community edition is free (no official support)**
- **RedHat support for Gluster “appliance”:**
  - Boxed installation on certified hardware

No dedicated catalog server in Gluster

- **Interesting architecture: distributed namespace**
  - Bricks have the same directory structure as file system namespace seen by clients
  - Files are distributed to bricks using a static formula ('elastic hash algorithm')
  - Client-side catalog operations ('*ls*') would merge file lists from all bricks
  - Bricks may also have special files not seen by users. These are used to keep a persistent state of the storage.
  - NO namespace caching at GlusterFS client side => simple implementation
- **Benefits:**
  - **Transparency**: Bricks are mounted locally on servers, so that their contents can be introspected, 'fsck'-ed, backed up, etc. Any file system can be used for bricks.
  - **Higher tolerance to hardware problems**:
    - bricks or even peers can go down w/o affecting other components
    - Remaining files (not just names in the catalog!) will be still available
    - Automatic failover (Gluster clients "know" about all peers)
    - Failed bricks can be brought offline and investigated separatedly

Client side

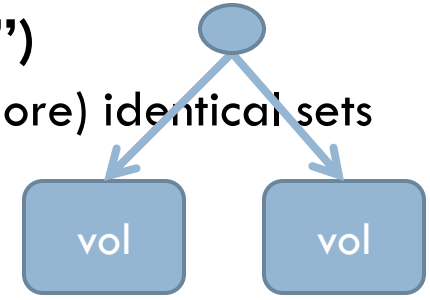
- **User-level file system (FUSE)**
  - No need to build a special kernel
  - Much less sensitive to kernel versions
  - Has certain performance implications (for small records)
- **Built-in NFS and CIFS (SAMBA) servers:**
  - Both see the same full namespace as the native client
  - Both support ACL and quota
  - => Opens an interesting possibility for exporting the file system to non-UNIX environment, or to “legacy” hosts where NFS is the only option

Multi-volume support

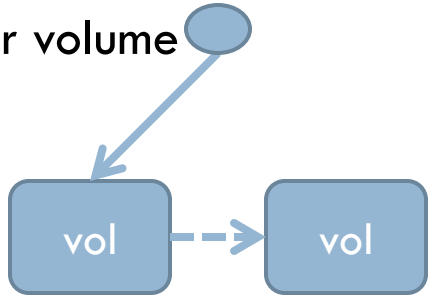
- **The same hardware (of servers & bricks) can be partitioned into one or many exportable volumes (file systems)**
  - Opens a possibility for more efficient (hardware) resources utilization
  - While keeping an option of tuning different volumes differently to meet specific needs:
    - e.g. one large volume for bulk data and sequential read, and the other one for small files cached for frequent/subsequent read

Automatic data replication options

- **Synchronous ("replication volumes" = "mirroring")**
  - Each file is being written simultaneously onto two (or more) identical sets of bricks
  - Possible side effect: performance slow down?
    - hasn't been studied yet



- **Asynchronous ("geo volumes" similar MySQL replication)**
  - A file written to a (master volume would eventually be copied to a chained (slave) replication volume
  - This can be further expanded to a replication tree
    - an interesting option for increasing aggregate read performance by xN
  - Designed to work for clusters, LAN and WAN (high latency network)
  - No noticeable side effect on performance of the master volume
    - We're still studying it



Setting it up: 1 PB

Hands-on experience in installing, configuring and managing. Trying to break it up, bring its components up and down…

A lot of credits goes to Ling who made all of this possible!!!

Installing and making available

- **Required**: RHEL 5.1+ or 6.2
- **Installed** 3 RPMs at both client and server sides
- **Installed** and **configured** IB (if needed); noting special for TCP/IP
- **Start** from one host and register others to the federation (1 simple command per each host to add)
- **Create** a volume from distributed bricks (1 simple command per each host)
- **Tune up**: a small number of (easy to understand) parameters; no need to reboot/restart anything; default parameters still usable
- **Mount** the volume (as a file system at clients)
- **=> no expert level knowledge is required beyond basic sys-admin skills; spent a few hours reading the Installation and Administration guides (a few dozen Web pages)**

Expanding

- **Adding a peer or a brick translates into a couple of simple commands, and:**
  - No need to remount clients or stop servers
    - Though, we had some troubles with carrying over ACLs. This didn't work the way we hoped for
  - Changes happen on the fly (some delay at a client side was observed before the expanded file)
  - An optional rebalance of data files can be requested; will run in background by "lazy" moving files between bricks
    - Still may be a lengthy process for a large file system
- **Peers and bricks can be removed, replaced in the same way**
  - We actually have tried all of these

Using it

Testing performance and features. Exploring the "guts"…

Gluster server performance

- **Tested maximum output of a server (one peer) from "native" (Gluster/FUSE) clients**
- **Reading the same large file simultaneously from 20 client hosts over IB:**

  dd bs=1M if=… of=/dev/null
- **Aggregate performance as measured by clients:**
  - **3700 MB/s**
  - Maxed out IB bandwidth of a QDR connection?
- **Observed very high (close to 100%) CPU utilization at a server side**

Gluster-NFS server performance

- **NFS mount points at 10 GbpsE machines:**

  % mount –t nfs psanaoss212:/ana12 /tmp/ana12

  % df –h /tmp/ana12

  psanaoss212:/ana12 946T 22T 925T 3% /tmp/ana12

- **1 client host performance (MB/s):**
  - 1 mount point:
    - 1 file: **read:** **419**, **write:** **253**
    - 2 streams from different files: **read:** 139 + 218 = **357**, **write:** 117 + 117 = **234**
  - 2 mount points (different NFS servers)
    - 2 streams from different files: **read:** 291 + 284 = **575**
- **2 client hosts performance (MB/s):**
  - 1 mount point per host (same NFS server)
    - 2 streams from same file: **read:** 297 + 276 = **573**
  - 1 mount point per host(different NFS servers)
    - 2 streams from different files: **read:** 461 + 476 = **936**
- **Notes:**
  - Multi-file read is limited by an NFS server performance (similar to the traditional NFS)
  - Some degree of scalability can be achieved by using all 5 Gluster servers as NFS servers:
    - **read:** 5 * 400 = **2000**, **write:** 5 * 250 = **1250**

Single stream performance: IP and IB

- **10 GbpsE**
  - Read: **555 MB/s**
  - Write: **569 MB/s**
- **QDR IB RDMA**
  - Read: **726 MB/s**
  - Write: **927 MB/s** (server side write-behind window:16 MB)
- **Compared with Lustre on the same hardware:**
  - Read performance is the same, write performance is 30% better

Parallel write performance [MB/s]

- **Setup:** 20 client hosts, IB RDMA
- **Monitoring method:** '*iostat –m 10*' at each server
- **Default configuration** (no server tuning):
  - 40 files (2 files per host): **8130** (**23**/40 bricks)
  - 80 files (4 files per host): **5661** (**35**/40 bricks)
- **Tuned up server** (enable HT: 16 cores, increased buffer sizes, number of threads per server: 64):
  - 80 files (4 files per host): **7439**
- **Notes:**
  - '*Elastic hashing*' algorithm of Gluster isn't so perfect in distributing load between servers. It's based on a namespace rather than load.
  - Aggregate write performance is *on-par* with Lustre set up on similar hardware: **8000+ MB/s**

Parallel read performance [MB/s]

- **Setup**: 20 client hosts, IB RDMA
- **Monitoring method**: '*iostat –m 10*' at each server
- **Default configuration** (no server tuning):
  - 40 files (2 files per host): **7160**
  - 80 files (4 files per host): **9057**
- **Tuned up server** (enable HT: 16 cores, increased buffer sizes, number of threads per server: 64):
  - 80 files (4 files per host): **9313**
- **Notes**:
  - Gluster read performs is *on-par* Lustre set up on similar hardware: **10000+ MB/s**
  - Catalog operations were noticeably affected (slowed down) when Gluster was under extreme load. Though, we didn't test catalog performance with Lustre.

Catalog performance: directories

- **Creating 10k subdirectories:**
  - Lustre: 2.3s
  - Gluster: 72s (**30 times slower**)
- **Listing 10k sudirectories:**
  - Lustre: 1s
  - Gluster: 43s (**40 times slower**)
- **Deleting 10k subdirectories from a parent:**
  - Lustre: 73s
  - Gluster: 362s (**5 times slower**)
- **Notes:**
  - Performance was tested on directories w/o ACL
  - Directory operations are slower because for each new file Lustre just makes an entry in its MDS database. Meanwhile Gluster should create the directory (real '*mkdir*' operation) at every single 'brick' (OST) simultaneously. Listing operations would need to merge partial catalogs from all servers. I also have a theory that Gluster would attempt to re-synchronize catalogs across all bricks during directory listing operations.
  - This is the price to pay for more robust distributed catalog in Gluster

Catalog performance: files

- **Creating 10k empty files within a directory from a single process:**
  - Lustre: 5.2s
  - Gluster: 54s (**10 times slower**)
- **Listing 10k files ('ls –al'):**
  - Lustre: 0.22s
  - Gluster: 9s (**40 times slower**)
- **Opening10k files w/o reading them:**
  - Lustre: 2.23s
  - Gluster: 2.90s (**roughly the same performance**)
- **Deleting 10k empty files from a parent:**
  - Lustre: 6.73s
  - Gluster: 12.8s (**2 times slower**)
- **Notes:**
  - The numbers are consistent with our understanding of how the distributed metadata catalog works in Gluster

Exploring features

- **ACLs:**
  - Tested and it seems to work the same way as on Lustre
- **Quota:**
  - Quotas in Gluster are defined for a whole volume or a directory. No quotas for users.
  - Exactly what we need for our experiment-oriented data policies!!!
  - Though, we haven't tried it yet ☺
- **Server performance monitoring:**
  - GlusterFS has built-in server profiling and performance monitoring
  - Partially tested
- **Volume replication options:**
  - Still testing

User group limit (the bad thing!)

- **The current implementation imposes 16 groups limit per user**
  - Same as for NFS v3
  - And it's probably related to NFS support in the FS
- **Was not mentioned in documentation; we discovered it experimentally after deploying FS in production**
  - This created quite a bit of problems for us
  - Required a workaround (register individual users in ACLS)
- **Is supposed to be addressed in the next major version of FS**
  - As we've been reassured by FS developers

Exploring Gluster internals

- **Server configuration files are kept at:**

  /etc/glusterd/
- **Client side configuration (optonal):**

  /etc/glusterfs/
- **The later allows adding/configuring things like:**
  - Read-ahead, I/O cache, timeouts
- **Server-side bricks retain:**
  - The full namespace of the distributed file system
  - All original information about files/directories owners, groups, ACLs
  - Bricks may also have special files (protection mask '--------T') to indicated files which need to be rebalanced between bricks
- **We also experimented by placing new files and directories directly into a brick:**
  - And they instantaneously showed up in the global namespace!
  - Rebalancing of files between bricks can be needed later in background mode if needed
  - Rebalancing for new directories happens automatically just by trying to open the new directory at a client side (global namespace)
  - Nice feature allowing server-side archiving/backups/restores and many more

Conclusions, plans, etc.

Pros and cons…

Summary of benefits

- Simple and transparent architecture
- Observable documentation
- Appears as a thin layer on top of the back-end storage
- Unlimited choices for back-end storage file systems: XFS, EXT3/4, etc.
- Distributed metadata – higher robustness and availability
- Easy to install, manage and troubleshoot
- "Forgiving" management – less effect on clients
- No external expertise required to set it up and "own" it
- Lots of interesting possibilities not available in Lustre:
  - NFS and CIFS clients with full support for ACL and quota
  - Asynchronous replication
  - Direct manipulation (archiving/backups/recovery) of data at server side
  - Server-side monitoring and profiling

Summary of shortcomings

- **Low performance for selected file system catalog operations:**
  - In particular: listing the content of a directory with thousands of subdirectories is 40+ times slower than in Lustre or NFS
- **I/O performance degradation for small record operation on the FUSE-based client side**
  - We've seen 3 times drop compared with Lustre
- **16 groups limitation may create problems**
- **Certain risk exists due to RedHat acquisition**

The final notes, plans

- **GlusterFS is XROOTD "on sterodis" ☺**
- **It's more than suitable to hold and serve bulk experimental data from LCLS**
- **However, a lack of dedicated metadata (file catalog) server may:**
  - Impose certain restrictions on using it for derived data produces by users (x100 more small files)
  - And it can (my personal opinion) make it impossible to use GlusterFS as a general purpose file system
- **There are chances that PCDS may migrate remaining three (<1 PB each) installations from Lustre to GlusterFS:**
  - Not sure about DDN though