PROCEEDINGS OF SCIENCE

XCFS - An Analysis Disk Pool & Filesystem Based On FUSE And Xroot Protocol

Andreas.Joachim.Peters*
CERN IT-DM-DA
E-mail: Andreas.Joachim.Peters@cern.ch

One of the biggest challenges in LHC experiments at CERN [1] is data-management for data analysis. Event tags and iterative looping over datasets for physics analysis require many file opens per second and (mainly forward) seeking access. Analyses will typically access large datasets reading terabytes in a single iteration. A large user community requires policies for space management and a highly performant, scalable, fault-tolerant and higly available system to store user data. While batch job access for analysis can be done using remote protocols experiment users expressed a need for a direct filesystem integration of their analysis (output) data to support file handling via standard unix tools, browsers,scripts etc. XCFS - the *xroot* [2] based catalog file system is an attempt to implement the above ideas based on *xroot* protocol. The implementation is done via a filesystem plug-in for FUSE [3] using the *xroot* posix client library which has been tested on LINUX and MAC OSX platform. Filesystem meta data is stored on the head node in a XFS filesystem [4] using sparse files and extended attributes. XCFS provides synchronous replica creation during write operations, a distributed unix quota system, krb5/gsi and voms authentication with support for secondary groups (via *xroot* remote protocol and through the mounted filesystem). High availability of the headnode is achieved using a heartbeat [5] setup and filesystem mirroring using DRBD [6]. The first 80 TB test setup allowing to store a maximum number of 200 million files has shown promising results with thousands of file open and meta data operations per second and saturation of gigabit ethernet executing single 'cp' commands on the mounted file system. The average latency for meta data commands is in the order of 1ms, for file open operations it is <4ms. The talk will discuss results of typical LHC analysis applications using remote or mounted filesystem access. A comparison will be made between XCFS and other filesystem implementations like AFS or Lustre. Strength and weaknesses of the approach and its possible usage in CASTOR [7] - the CERN mass storage system - will be discussed.

XII Advanced Computing and Analysis Techniques in Physics Research
November 3-7 2008
Erice, Italy

*Speaker.

PoS(ACAT08)041

1. Introduction

Today two file access methods are in use for data access for LHC analysis: remote access protocols (RAP) and mounted filesystems (MFS). The common way in many GRID (e.g. LCG [8]) or Mass Storage Systems is access via a remote access protocol e.g. *ftp, gridFTP, rfio, xroot, sftp* etc. These require to link the user application with protocol specific libraries. Implementation of strong authentication methods like krb5 [9], GSI [10] or VOMS [11] are existing as regular user-space libraries. RAPs allow additionally application specific optimizations e.g.

- asynchronous read-ahead based on read hints given by the application (asyncR)
- compound reads based on read-clustering hints given by the application (readioV)

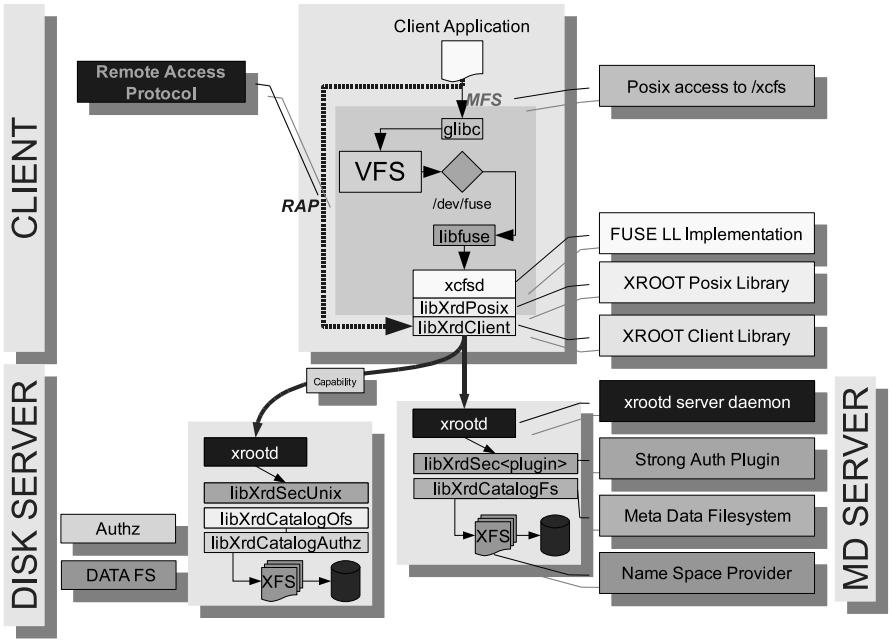
These optimizations target to reduce network latencies and individual client performance (see [12]). However they imply mostly sparse file access on server side which leads in case of spinning disks to degraded random access.

The MFS approach guarantees high performance due to low-level kernel implementations (e.g. AFS [13], NFS4 [14], Lustre [15]) and easy application integration due to the POSIX [16] standard. Most production versions of distributed filesystems support none or only kerberos authentication. GSI authentication is implemeneted for NFS4 as an up-call to a user-space daemon. VOMS authentication is not implemented for any filesystem and a crucial point is a consistent mapping of virtual roles and groups to physical UID/GID pairs as used by the operating system. IO Optimizations for mounted filesystems are individual read-ahead algorithms. There is no POSIX function to issue compound reads with different offsets neither can applications issue asynchronous reads receiving delayed responses. Data Analysis as needed for the LHC experiments at CERN can range from linear file access with large files and large IO to extremely sparse forward seeking access with small IO volume. The first benefits from optimization for maximum IO output of the storage system while the latter from name-space and file access latency optimizations and server side read-ahead.From a user's perspective the client interface differs mainly during interactive work. The user community prefers data handling via a mounted filesystem. For applications running as batch or GRID jobs the user interface plays a minor role, strong authentication and efficient linear and sparse file access are most important. XCFS was implemented to merge both approaches into one, providing an interactive user interface via FUSE as a mounted filesystem and allow non-POSIX optimizations for batch processing using *xroot* protocol as remote access protocol. A first prototype was based on Lustre as back-end filesystem and an integration of *xroot* protocol on top of the Lustre disk servers.

2. Architecture & Implementation

XCFS characteristics are: Simplicity, High Performance, Scalability, Fault tolerance, Resource & Access Policies (Quota, ACLs), Dual-Access via MFS & RAP, Strong Authentication (Krb5, GSI, VOMS). Key elements for the implementation are as already mentioned the *xroot* protocol and the FUSE low-level API. *xroot* Protocol allows low-latency and high-bandwidth data access. The *xrootd* file server daemon offers a complete plug-in infrastructure in C++. Every component can be extended or replaced by customized implementations (Authentication, Authorization,

Figure 1: Architecture of XCFS using xrootd client & FUSE.

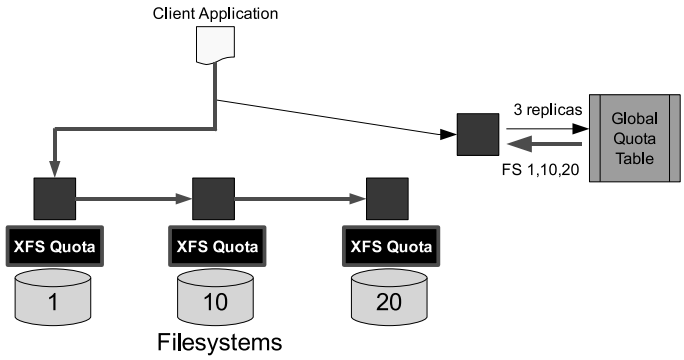


Filesystem Access, Filename Translation). The *xroot* client is already packaged in ROOT [17] which is a standard application used in all software frameworks of LHC experiments and simplifies the deployment. The FUSE low-level API allows multi-user filesystem implementations in user-space using file attribute, kernel caching and the usage of the kernel read-ahead algorithm. An overview of the architecture and plug-in implementation for access via RAP & MFS is shown in figure 1. The architecture is analog to NFS4 & Lustre. The namespace is provided by a meta data server, files are stored on several disk servers.

Namespace

The XCFS namespace is stored in an XFS filesystem using sparse files and extended attributes. A sparse file represents the full POSIX attributes (creation time, access time, permissions, size) and the extended attributes contain the location of a file on disk servers and a unique file id. Every disk server filesystem in XCFS uses a unique bit (out of 8192 by default) in the location attribute. For consistency checks a reverse lookup filesystem exists to back-reference a logical file in the namespace using the three keys <*disk server*> +<*filesystem ID*> + <*unique file id*> via symbolic links. One stored file in the namespace occupies in default configuration 4 kb in the meta data filesystem and 2 inodes. High availability of the namespace provider is currently done using DRBD and heartbeat with an active-passive configuration. An active node configures a virtual IP and runs the meta data *xrootd* server. All writes on the namespace filesystem are mirrored to the passive node which can fail-over the namespace filesystem, virtual IP and *xrootd* server. To avoid split-brain situations the previous master node is powered off using IPMI [18] in case of a fail-over. It is however desirable, more performant and more flexible to aim for a high-level change-log protocol

Figure 2: Synchronous replica creation in XCFS. The example shows the creation of three replicas. A client connects to the endpoint for the first replica. The second and third replica is created and written synchronously in a chain as shown.



inside XCFS with master-slave replication as used e.g. in MySQL[19].

Space and Quota Model

An important feature to accommodate for diverse hardware and resources is the grouping of disk server filesystems into spaces. Each space is made by a set of filesystem bits which reference the corresponding target filesystems. Default policies for the selection of a space can be defined based on user identity or namespace location - otherwise can be selected individually by each user for individual file.
Quota is implemented using standard Linux (XFS) quota on the target filesystems on disk servers. Quota can be assigned per space to users or groups and is distributed equally over the assigned filesystems. Quota contains inode/block soft and hard limits.
The meta data node redirecting clients to target filesystems has a real-time view of the free space and assigned soft/hard limits on every target filesystem. File creation requests are distributed round-robin over filesystems providing sufficient quota for file creation (default: at least 1 GB free space in soft quota).

Replication Model - High-Availability Of Data

XCFS supports synchronous replication of files e.g. every write operation is done synchronously on all replicas as seen in figure 2. The policy interface allows to define default policies for replica creation based on file pattern/suffix matching, namespace location or per space. A command line interface (CLI) allows to view, add, delete or reduce the number of replicas for a specific file. The meta data server detects if a target filesystem is offline or faulty and tries to select (if possible) always an on-line replica. If a file with an offline replica is opened for rewrite XCFS creates currently an IO error on client side. To further improve the HA an option would be to trigger dropping the offline replica and creation of a new on-line replica.This would allow the client operation to proceed without IO error.

Command Line Interface

The XCFS CLI covers standard shell commands like mkdir, rmdir, rm, ls, cp and XCFS specific user commands for replica handling and stage-in and -out commands for Mass Storage interaction. Finally it provides a set of administrator commands to deal with quota, draining of filesystems, moving of filesystems and a global file system check.

Read- & Write-Cache

The *xroot* client provides a read cache for prefetching and read-ahead using an asynchronous thread with user defined read-ahead and cache size. Additionally FUSE allows to enable the kernel read-ahead algorithm with a maximum chunk-size of 128 kb. Many tools like 'cp' create sequential 4 kb page writes which lead to a low total bandwidth for latency reasons. To improve the bandwidth for writing an additional write-cache with a default size of 4M per file has been introduced. This cache aggregates consecutive writes and flushes when the cache size is reached. Because of the synchronous implementation only half of the theoretical network bandwidth can be achieved.

3. Performance Evaluation

3.1 XCFS Testbed

A generic testbed has been setup with ten standard CERN Linux SLC-4 disk servers (8 Gb memory) with 20 active disks with four hardware RAID-5 arrays combined via a software RAID-0 device (8TB). Each of the ten RAID-0 filesystems got a unique filesystem bit assigned. Up to 100 SLC-4 CPU server (16 Gb memory) have been used as clients. The meta data namespace was created on two standard CERN Linux SLC-4 small disk servers using four disks in RAID-10 configuration, configured with DRBD and heartbeat as fail-over pair. All machines were equipped with Gbit Ethernet and eight cores.

3.2 Streaming Tests

There are two possibilities to reach high read streaming performance. First possibility is to use the *xroot* read-ahead cache. Setting a read-ahead of 1M allows in a 1 Gbit environment to get close to the theoretical network bandwidth with a single client (115 Mb/s). Second possibility is to disable *xroot* read-ahead and use the kernel read-ahead via FUSE. This algorithm uses max. 128 Kb chunks and allows to reach 70 Mbytes/s with default TCP parameters. Write streams perform reasonable well if the write chunk size is bigger and a multiple of the default page size of 4 Kb. Around 60 Mb/s are reached in this case.
The streaming test was additionally performed using a single disk server with a 10 GBit interface. A single 1 GBit-client can stream 118 Mb/s unbuffered from the 10 GBit disk server using *xroot* read-ahead of 1M. 20 clients are able to read 1.050 Mb/s from the buffer cache of the 10 GBit disk server.

3.3 Sparse Access Tests

Physicists doing LHC analysis will use event lists to describe a set of interesting data for analysis. This results in sparse forward-seeking access reading only a small fraction in large sets of data files. The following tests tried to evaluate these access patterns.

Single Client - Single Server

In these tests a ROOT h1 analysis macro has been used which reads sparsely 4% of a 267 Mb datafile using 4 Kb reads. The macro can be downloaded from here [17]. Table 1 shows the measurement results for XCFS with remote access and mounted filesystem access compared to AFS, Lustre(1.6) and NFS4. The 'Config' row shows the used read optimization (readiov, asyncio or read-ahead-/buffer-size). While the fastest real-time results are reached with systems using large linear read-ahead (Lustre + XCFS via MFS), the data volume passing the network differs by a factor 25 with respect to non-read-ahead systems. In all tests it was taken care that no data is cached on client side. For consistency checks the same measurement was repeated and filesystems implementing client side caching don't need to transfer the data again and reach an identical (optimal) minimal real-time for the application.

Table 1: ROOT h1 Analysis Measurements with single client

	RAP	RAP	MFS	MFS	AFS	Lustre	Lustre	NFS4
Config	readiov	asyncio	0k	96/512k		1M/40M	0M/0M	
1. Execution								
Real/Cpu[s/s]	3.4/3.0	3.3/3.5	5.7/3.4	6.0/3.42	11.2/3.1	5.6/3.4	28.4/3.1	5.8/3.4
On Wire[Mb]	12.5	12.5	25	267	275	277	25	25
2. Execution								
Real/Cpu[s/s]	3.4/2.9	3.3/3.5	2.9/2.9	2.9/2.9	2.9/2.9	3.0/3.0	2.9/2.9	3.2/3.2
On Wire[Mb]	12.5	12.5	0	0	0	0	0	0

Multiple Clients - Single Server

The same test was repeated using 12 Gb of data (9 files) with five concurrent applications on a single client machine. The results are visible in table 2 .There is no big difference in different access methods (besides Lustre without read-ahead). Again main difference is the used network bandwidth. Remarkable is that for XCFS access via RAP the disk utilization is at the same level providing 11 MB/s (random access) as in the case of 105 Mb/s (sequential access) for Lustre with 1M read-ahead. Increasing the number of clients on a single machine to eight favors the remote access protocols. The read-ahead used by Lustre satures already the network with five clients as seen before. Obviously none of the two access methods is perfectly suited for all use cases. There are many factors like sparseness, network bandwidth, disk latency, disk ratio between random and sequential access and ratio between CPU and disk servers to be considered to choose the best access way.

Table 2: ROOT h1 Analysis Measurements with five clients.

	RAP	RAP	MFS	MFS	MFS	Lustre	Lustre
Config	readiov	asyncio	0k	96/512k	512k/4M	1M/40M	0M/0M
Real/Cpu[s/s]	131/37	138/37	150/28	139/28	134/28	134/31	324/29
Raid Util.[%/Mb/s]	60/11	60/11	55/8	60/105	60/110	60/110	-/-

3.4 Meta Data Performance Tests

The meta data performance has been measured as shown in table 3. The meta data server is able to keep the information of four million files in the buffer cache - hence most read operations don't involve any disk access. The performance for read operations are therefore mainly limited by CPU usage and thread locking dead time. The latency for a single file open for read is 2.8 ms, for file creation 5.7 ms, for a stat command it is 1 ms. The overall write performance is limited by the random write performance of the device storing the namespace. Moreover DRBD involves additional overhead. A big improvement can be expected using SSD disks of the latest generation with low random write latencies.

Table 3: XCFS Meta Data Performance Measurements.

	RAP	MFS	Latencies
Seq. ROOT raw file creations/s (single client)	175 files/s	85 files/s	5.7/11.7 ms
Seq. ROOT raw file read opens/s (single client)	385 files/s	90 files/s	2.8/11.0 ms
Par. ROOT raw file creations/s (80 clients)	250 files/s	180 files/s	DRBD-C
Par. ROOT raw file read opens/s (80 clients)	2500 files/s	2100 files/s	52.0/62.0 ms
stats/s (80 clients)		30.000 Hz	(500% CPU)
stats/s (single client)		1.000 Hz	1.0 ms
readdirs+Stats/s (80 clients)		55.000 Hz	(600% CPU)

3.5 Mounted Filesystem Standard Tests

The mounted filesystem has been tested with standard procedures like execution of 'configure', 'tar', 'make', 'make -j 8', random shell functions, software startup. All these tests have been successful. A big limitation however is the file creation rate which is not sufficient if many users execute operations like 'tar -x' at the same time. This bottleneck can only be opened using more performant hardware as described before. Most of this 'standard' operations are a factor 2-3 slower for a single client than AFS or Lustre which is mainly due to advanced caching strategies and stat-ahead.

4. Mass Storage Integration

XCFS supports third party stage-in and stage-out on filesystem nodes using *xroot* and *rfio* protocol. Other protocols could be easily added. The meta data server supports spool file creation for updated/created files to be used with an archiving daemon. Policies for this spool entries can be defined depending on namespace location, space, file size, pattern matching. An external Mass Storage namespace could be made visible within the XCFS namespace as an independent branch (offline namespace). Staging/Migration could be triggered by copying from offline- to on-line namespace and vice-versa.

5. Summary & Outlook

XCFS has been developed as a prototype study in the CERN data-management group according to the requirements of data analysis for LHC experiments to allow batch access to storage via re-

mote access protocol and interactive access via a mounted filesystem. The implementation implies some trade-offs concerning the full implementation of POSIX. However, it allows comprehensive access and resource control as needed by the community. Average access latencies of few *ms* allow for interactive usage. The system has not been optimized to provide user home directories for very small files, software development and compilation. For this use case other filesystem implementations show clear advantages with additional client optimizations. The first prototype interface to Lustre is still a valid option in the future to combine a scalable parallel mountable filesystem with remote access. The XCFS architecture provides one important piece to bring forward a more scalable third generation of the CERN mass storage system CASTOR with a hierarchical and modular separation ranging from low-latency storage front-end to high-latency storage back-ends.

References

[1] *LHC - The Large Hadron Collider*, http://lhc.web.cern.ch/lhc/

[2] *The Scalla Software Suite: xrootd/cmsd*, http://xrootd.slac.stanford.edu/

[3] *FUSE Filesystem in User-Space*, http://fuse.sourceforge.net/

[4] *XFS - A high-performance journaling filesystem*, http://oss.sgi.com/projects/xfs/

[5] *HA - The High Availability Linux Project* http://www.linux-ha.org/

[6] *DRBD - Software Development for High Availability Clusters*, http://www.drbd.org/

[7] *CASTOR - CERN Advanced STORage manager* http://castor.web.cern.ch/castor/

[8] *LCG - Worldwide LHC Computing Grid*, http://lcg.web.cern.ch/LCG/

[9] *MIT - Kerberos Consortium* http://www.kerberos.org/

[10] *GT Security (GSI)*, http://globus.org/toolkit/security/

[11] *VOMS - Virtual Organization Membership Service*, http://hep-project-grid-scg.web.cern.ch/hep-project-grid-scg/voms.html

[12] F. Furano, A. Hanushevsky, *Data access performance through parallelization and vectored access. Some results.* Journal of PHysics, Conference Series 119 (2008) 072016

[13] *OpenAFS*, http://www.openafs.org/

[14] *NFS Version 4 Open Source Reference Implementation*, http://www.citi.umich.edu/projects/nfsv4/

[15] *Lustre - a network clustering FS.* http://www.lustre.org/

[16] *POSIX - IEEE 1003 Standard*, http://ieeexplore.ieee.org/

[17] *ROOT - An Object-Oriented Data Analysis Framework* http://root.cern.ch/

[18] *IPMI - Intelligent Platform Management Interface Specifications* http://www.intel.com/design/servers/ipmi/spec.htm

[19] *MySQL Replication* http://dev.mysql.com/doc/refman/5.0/en/replication.html