Observations made while running a multi-petabyte storage system

Miguel Coelho dos Santos
IT Department
CERN
Geneva, Switzerland
miguel.coelho.santos@cern.ch

Dennis Waldron
IT Department
CERN
Geneva, Switzerland
dennis.waldron@cern.ch

***Abstract*— We take an overview of the CERN Advanced Storage (CASTOR) version 2 system and its usage at CERN while serving the High Energy Physics community. We further explore some of the observations made between 2005 and 2010 while managing this multi-petabyte distributed storage system.**

I. Introduction

CASTOR, CERN Advanced STORage manager, is a file based hierarchical storage management (HSM) system developed at CERN[1] that is used to store physics production data and user data. Files can be stored, listed, retrieved and accessed using command line tools or applications built on top of the different data transfer protocols like RFIO (Remote File IO), ROOT libraries, GridFTP and XROOT.

In the last 10 years over 453 million files have been written to the system, of which slightly over 143 million files (25PB) are part of the current data store (figure1).

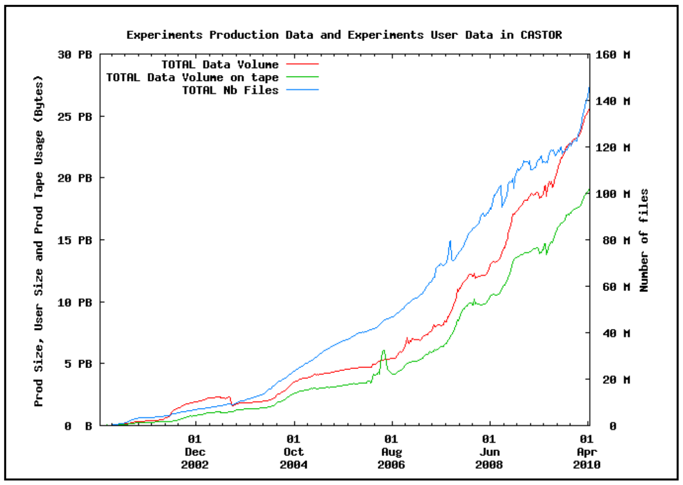

Figure 1. Evolution of files on CASTOR

The system's function is to enable High Energy Physics (HEP) computing by storing and serving files containing collisions data from different accelerator and experiments, supporting task such as:

- Data processing, in general terms measuring quality of recorded data, filtering and aggregating interesting events into larger event collections.
- Archiving a copy of the generated event collections.
- Distribution of the collected data over other HEP sites.
- Analysis of the collected data.

Data processing, archiving and distribution are processes which happen immediately after data is collected from the physics detectors.

A. *Files Stored on CASTOR*

Physics data is stored in the form of files. The distribution of the size of files on the system (figure 2) shows that of the 127 million files in store the majority of the files are not very large, 52% are under 10MB and 75% are under 100MB.

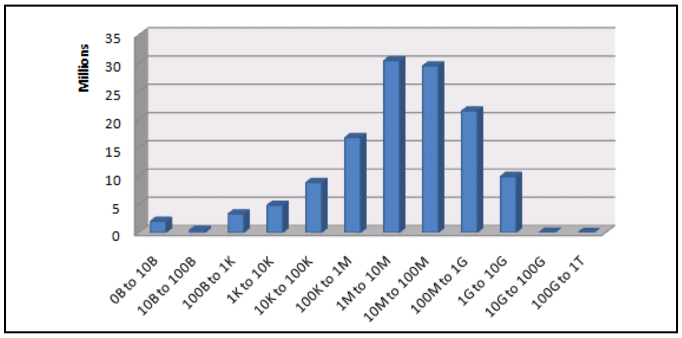

Figure 2. Frequency of files by size

A closer look at the current population of online disk resident files (figure 3) confirms the prevalence of files between 1MB and 10MB but shows that very small files between 100KB and 1MB are almost as present as files between 10MB and 100MB (figure 3).

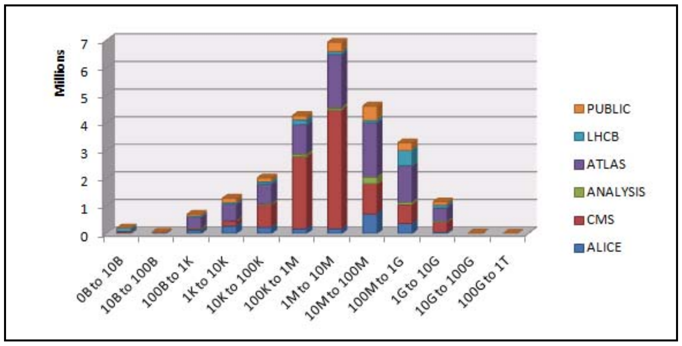

Figure 3. Frequency of online files by size

B. *Brief System Description*

The current system is a collection of approximately 1300 disk servers, typically with around 22 SATA disks grouped into disk arrays of varying sizes (between 2 and 5 disks per array).

Disk servers are grouped into six separate instances managed by an independent set of nodes. Files on disk are said to be online. Each catalogue of the files on disk at each instance (stager) is kept in a separate database.

All files are also registered in a central catalogue containing the name space (file paths), ACLs, checksum information, and tape locality.

Each file can have one or more copies on tape, stored in any of the five different tape libraries.

File movement is centrally managed when done to/from tape and when done inside one of the six instances.

I. OBSERVATIONS AND EXPERIENCES WITH CASTOR

During the last five years of running CASTOR we have made many observations and encountered some challenges which we believe not only apply to CASTOR but also to other distributed storage systems. Some of these are more focused on distributed storage systems while others are more related to distributed systems in general. We will explore these briefly based on this perspective, without any other link between the different topics.

A. *Component Failure, a fact of life*

On such distributed storage systems mainly composed by commodity hardware the components have relatively high fault rates[2][3][4]. In-house measurements show average server uptime of 97%. Faults and other events and processes contribute to server uptime:

- Hardware failures: fan, motherboard, cabling, raid controller, memory, CPU, double power supply failure, etc
- Operating System and Kernel upgrades
- Firmware patching (disks, controllers, etc)
- File system reconfiguration, e.g. reconfiguring RAID arrays on a file server from RAID5 to RAID1
- Power-cuts

Drive failures are also common (2.9E10^{-6} disk drive failures/hour). Currently their impact is usually a RAID rebuild. Transient or permanent data unavailability of a disk array because of double disk drive failure or disk controller failure is a rarer event (1.8E10^{-7} failures/hour).

A large distributed storage system like CASTOR needs to be recovering from faults continuously, either by doing the mentioned RAID rebuilds or by moving data between servers and updating metadata servers to reflect the data movements on the catalogues.

Other events reduce the useful usage time of a server, some particular examples:

- Disk server draining (moving all data in a disk server to other nodes sibling servers)
- Moving disk server between pools, i.e. redistribution of servers from one logical group to another
- Retiring hardware going out of production

B. *Data unavailability*

We generally consider two types of data unavailability:

- Transient, when data is temporarily unavailable
- Permanent, when data is lost

Four petabytes of disk space - 50% of total disk space available - is dedicated to user managed pools, i.e. no automatic file deletion in place. Files written to these pools are disk resident (online) only, there is no copy on tape - Tape0Disk1[6]. There is also the possibility to recall files from tape so they become online for accessing.

Under this Tape0Disk1 storage class, transient data unavailability of unique data, maps directly to server unavailability, i.e. 3% of downtime. Permanent data unavailability, or data loss, maps directly to the average disk array failure rate of 1.8E10^{-7} failures/hour.

To reduce the impact of transient and permanent data loss some of the other disk pools are configured as Tape1Disk1, meaning the pool is user managed and a copy of the data is kept on tape. Under this storage class transient data unavailability after a disk server failure depends on tape and tape-library unavailability, and time to read file from tape to disk. Permanent unavailability depends on occurrence of both disk array loss and tape loss. Measured tape loss points to an average failure rate of 5.4E10^{-8} failures/hour.

Other pools are configured as Tape1Disk0 which means files should have a copy on tape and the pool in centrally managed, i.e. files that have a copy on tape can be deleted from disk according to different cache management algorithms: least recently used, first in first out, etc. On Tape1Disk0 pools permanent data unavailability depends mostly on tape loss and transient data unavailability depends mostly on tape recall time.

Tape2DiskX is a solution with few clients, mainly used for low quantities of precious data. This solution provides higher availability but not higher performance. TapeXDisk2 is a solution that has been proposed but not widely adopted because having two copies reduces the available disk space by 50%. The trade-off between disk space and uptime has yet to trigger a general move towards higher file availability.

Another way to improve data availability is to move data as efficiently as possible before predictable downtimes; this option is especially relevant for Tape0Disk1 storage class.

C. *Bulk data movement*

A particularly important area growing in relevance as data volume expands is data movement. Disk and tapes are constantly increasing in capacity but this has not been followed by a proportional increase in bandwidth, neither a proportional decrease in access times.

An approach put in place to counteract these phenomena was a reduction in RAID array size. First reducing the number of disks in a RAID5 array from 5 to 3 disks and, shortly after, adopting RAID1 as the default configuration. At his point in time this path is exhausting itself and if another move is necessary than multiple copies need to be considered.

The current disk servers have 24TB of usable disk space, which corresponds to a 140% increase in comparison to the previous generation. There is no expectation that this rate will slow down in the next few years. For significant changes in bandwidth and access times, media other than SATA disks needs to be considered.

On the other hand of this capacity increase is the fact that file size of the file population is not changing significantly. As can be seen from figure 2 and 3 the size of the online file population is even lower than that of the overall file population, 62% of the online files are 10MB or smaller, 52% of the total file population is 10MB or smaller.

These conditions - increasingly large disks, reduced bandwidth per terabyte ratio, constant seek times and the continued prevalence of small files - together with the need of keeping metadata catalogues up-to-date result in making data movements an increasingly difficult challenge and at the same time a more common activity.

All user activities (data processing, archiving, distribution and analysis) require large data movements of groups of files. From the system side, removing out of warranty hardware and pool reconfigurations are critical activities that also require movement of large groups of files. As mentioned before some types of interventions could also benefit from more efficient ways of displacing large quantities of data.

Currently these large data movements are done on a file by file basis. An approach which suffers from overhead problems (preparation of source and destination, metadata updates, mover scheduling, etc) and from bandwidth under-utilization. The later comes from the fact that files are small and that scheduling is somewhat chaotic as there is a detachment between (user) name space and disk locality. Requests to move data are done based on metadata information such as name space but there is no system optimization to enhance the reading of data from disk in a streaming and performing manner. Writing is not a large problem because the system is load balanced and therefore optimized to receive data.

D. *Monitoring*

Within any multi-petabyte storage system the need to be aware of how the system is performing and whether SLAs (Service Level Agreement), MoUs (memorandum of understanding), OLAs (Operation level Agreement) are being achieved is paramount. Monitoring of such systems and the environment in which they operate is a challenging task covering many different areas from accountability, user traceability, resource management, performance, user perceptions, feedback and planning of future system enhancements.

Often monitoring is required not only of the physical storage but also its dependant infrastructure, for example, networking and databases, without which the service cannot function correctly. In periods of network instability the service for the end user can be considered degraded yet the monitoring of the storage system itself can be reported as ok.

It is therefore important to not only monitor the application itself but also what the user potentially sees. One way of doing this is through the use of service level probes, whereby the system is able to perform self diagnostics or tests remotely outside of the system to test response times, check for errors and identify possible bottlenecks or areas of contention. Often this will indicate what problem exists. Delivering a quality service requires service level monitoring not just application level monitoring.

When developing mass storage systems it is important to consider monitoring and its constituent parts, data collection, interpretation and visualization as being essential to the correct running and operation of the service. Unfortunately, these key areas are often overlooked and not considered essential in the early development phases. The problem with ignoring or overlooking these key critical areas is that retrofitting monitoring into a pre existing system is costly, inefficient and requires many software releases to perfect, especially in a large distributed system. Furthermore, it is essential to define key performance indicators (KPI) early on in the development phase so they can be measured as development progresses and as the system evolves and becomes more robust.

A key problem with monitoring is knowing exactly what to measure, it is not always obvious. Take for example the different perspectives of a software developer versus that of an operations person. Software developers tend to think in terms of raw values; the system opens x number of files per second the processing time for this subcomponent is X milliseconds.

This raw quantitative information is fine but from the operations perspective it doesn't necessarily tell you how well your system is performing. Questions such as, what happens if x reaches this value? What are the expected values? Is it good or bad? Should I be concerned? Are all reasonable questions for someone who may not have a low level knowledge and understanding of how the system works. Developers and operations personnel need to work together collaboratively to establish useful metrics for measuring the system.

Monitoring needs to be sensitive to the target audience, it should provide additional help and assistance - not confusion - and should be visualized in such a way that a quick glance at the data tells you if there is problem.

The monitoring information provided by CASTOR is generated by the analysis of log messages recorded by various components in the system. On average there are 26 messages recorded per transfer. Due to architectural problems within CASTOR it cannot perform many opens/second but imagine a system that could do 1000 opens/second, this would result in a minimum of 26,000 messages/second to be centrally collected, recorded and analyzed, potentially hundreds of gigabytes per day. At these rates the overhead of logging itself becomes a performance issue as the cost of logging increases with the number of operations. Why was this approach taken? Because monitoring was an afterthought, not needed at first but essential for operating the system later, the easy but non scalable option was to parse log messages.

Software developers working on distributed systems should take serious consideration towards the topic of monitoring by recording concise, meaningful information and avoid generating massive quantities of data.

E. Databases & Metadata

The architectural model of CASTOR has at its core two databases, one which stores metadata about all files, the central catalogue and another which holds information on the files which are currently available, online. Both databases share common information and as a result some information is duplicated across the two. As a consequence of this duplication maintaining consistency is always a problem. As the system grows in size the level of cross consistency checks that are required to keep the system usable becomes more and more problematic. Not only is the operation fairly expensive but the time taken to perform a check increases in proportion to the number of files. This is an inherent problem in having two distinct databases where the online database references the namespace data in a non transactional way.

The problem becomes even more difficult because of the "Double commit" problem whereby information is updated in one system and the attempt to update the second fails. At this point the system is in an inconsistent state, it could rollback or revert the modifications to the first system but there is no guarantee that this will work. Furthermore, some other operations may have taken place in the system outside of the databases control which cannot be reverted. In a system such as CASTOR these problems are not handled gracefully and no logic exists to rectify the state automatically, the cost of which is time spent on manually fixing the problem by direct database manipulation.

When designing distributed mass storage systems the use of databases needs to be carefully analyzed, although convenient and flexible it may not be the best choice in terms of operational cost, the architectural restrictions the technology imposes and system performance. The key is to have an appropriate model for the metadata, one which scales and can deal with inconsistencies gracefully. Inconsistencies are unavoidable; it's how the application handles or deals with them which determines the impact and cost.

Considerations such as where to store metadata are key to system robustness. Within CASTOR for example if the namespace database is lost all data is lost, it cannot be reconstructed from the content on disk or tape, the operational cost and impact to the end user under such a situation is major. One way to solve this issue is to have files which are self describing so that a full reconstruction could be performed. This solution however is not free from its own problems, issues such as having multiple replicas of a file results in multiple locations of metadata and this brings potential consistency and synchronicity related problems. A trade off needs to be made between centralized and distributed metadata storage.

II. CONCLUSIONS

Multi-petabyte storage systems are intrinsically difficult to operate and manage. Their inherent size, complexity and number of system components make failures a persistent problem that system needs to handle automatically in order to be robust and usable. As a consequence dealing with data unavailability and moving large quantities of data are regular operations which need to be scalable and efficient.

Monitoring and key performance indicators are important aspects that should be defined early on in the development lifecycle in order to trace the evolution of the system with regards to its performance and to make sure that the system fulfils its user requirements under different load conditions.

The scalability of metadata storage and the capability to recover from disaster scenarios should be considered as a key principle when making design choices. Having weak elements of the architecture or single points of failure could potentially lead to data loss which should be avoided at all costs.

[1] Presti, G. L., Barring, O., Earl, A., Rioja, R. M., Ponce, S., Taurelli, G., Waldron, D., and Santos, M. C. 2007. CASTOR: A Distributed Storage Resource Facility for High Performance Data Processing at CERN. In Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies (September 24 - 27, 2007). MSST. IEEE Computer Society

[2] E. Pinheiro, W.-D. Weber, and L. A. Barroso, “Failure trends in a large disk drive popula- tion,” in Proceedings of 5th USENIX Conference on File and Storage Technologies (FAST 2007), San Jose, CA, February 2007.

[3] B. Schroeder and G. A. Gibson, “Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?,” in Proceedings of the 5th USENIX Conference on File and Storage Technologies, February 2007.

[4] A. Wolman et al., “On the Scale and Performance of Cooperative Web Proxy Caching,” Proc. 17th Symp. Operating Systems Principles 1999, ACM Press, New York, 1999, pp. 16-31

[5] Hoelzle, U. and Barroso, L. A. “The Datacenter as a Computer: an Introduction to the Design of Warehouse-Scale Machines.” 2009, 1st. Morgan and Claypool Publishers.

[6] F.Donno and M. Litmaath, Data management in WLCG and EGEE, CERN-IT-Note-2008-002.