

CERN-IT-2013-005

<https://cds.cern.ch/record/1630387>

September 2013

Technologies for Large Data Management in Scientific Computing

Alberto Pace

Head, Data and Storage Services group, CERN IT department, CH-1211 Geneva, Switzerland.

Technologies for Large Data Management in Scientific Computing

Alberto Pace

Head, Data and Storage Services group, CERN IT department, CH-1211 Geneva, Switzerland.

1. Introduction

Scientific computing could be defined as one of the techniques used in many sciences to make progress in research using computer science. The use of computers in research has become an essential tool in nearly all sciences for several unrelated activities, including modeling, data acquisition, simulation, numerical analysis.

Historically, scientific computing was focused on processing information but has evolved in the recent years in the areas of storage, processing, distribution and long-term preservation. This evolution explains why scientific computing is today built from three essential components (figure 1):

- 1) A large number of processing cores which are necessary to process, analyze, transform the data associated to the research project. This is the traditional and well known component of scientific computing that maps into services like 'batch job submission', 'virtualization on demand', 'interactive computing services'.
- 2) A network infrastructure with large switching and routing capabilities within all nodes of the computing infrastructure, but also a large bandwidth to the internet, so as to ensure proper access to the computing infrastructure by the various universities and research laboratories participating in research projects.
- 3) A data and storage infrastructure that is able to store the information, dynamically place it where the processing resources are available, distribute it to a large number of stakeholders (universities and research laboratories participating in the project), and preserve the information over time.

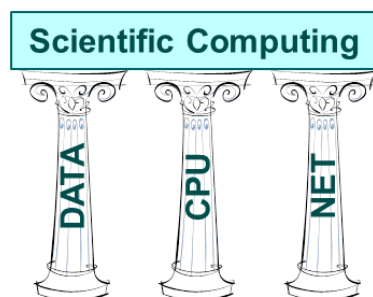


Figure 1: The three pillars of scientific computing

While all three pillars are equally important and necessary to the success of a scientific computing infrastructure, this paper will discuss only the data and storage infrastructure and the associated technologies.

2. The growth of scientific computing

The technological improvements in computing of the last few decades have gone beyond the most optimistic predictions. When, in the late '80s, the new 32-bit architecture received a large market acceptance, all IT professionals thought that this would put an end to the problem of address space shortage. More than 4 gigabytes (GB) of address space was considered 'infinite', being 3 orders of magnitude higher than the average RAM size of the time: just a few megabytes (MB). Nobody could then imagine that only 20 years after, a new 64-bit architecture would have become necessary to again overcome the address space limitation of the legacy 32-bit processor. Similarly today, it is common belief that the switch to 64 bit computing has solved forever the problem of lack of address space as no one can now imagine installing more than 16 exabytes (EB) of addressable memory in a single computer. Only time will tell if the coming technological improvements have been again under-estimated, or if we will live with 64-bit computing for many years to come.

Similarly, the average desktop hard disk in 1990 had a size of 10 – 20 MB while at present days we have 1 – 2 TB. An increase in storage capabilities of more than a million times in less than 25 years.

This impressive evolution in computing has been a major enabler of scientific research, opening unprecedented opportunities for calculation, simulation, data analysis and modeling. The use of computing has also been one of the key strategies in multiplying the productivity in several research fields, by enabling the possibility of a systematic, brute force simulation of all possible configurations and/or solutions, rather than being limited to few experimental cases.

The strategic necessity for high performance computing in scientific research has been clearly recognized in science: the fraction of investment dedicated to computing has even been growing over the years, with the consequence that the usage of computing in research has increased beyond the already impressive technological evolution mentioned above. De facto, the access to large computing facilities has become essential for any exact science.

And within this general evolution, the fraction of computing resources targeted at data storage and data management have been of the most strategic importance. For example, as explained later in section 2.3 the CERN computer center hosts 100% of the physics data from the Large Hadron Collider (LHC) experiments, while it has invested in only 20% of the CPU power necessary to process it.

2.1 The data management component of scientific computing

The data component of scientific computing has been of growing importance over the recent years, with several keywords being used to describe this activity, such as 'big data' or 'cloud storage'. The amount of data involved in large scientific projects has grown exponentially, inducing scalability issues, which have limited traditional data-processing activities.

The most important limiting factor in a project requiring the processing of large amounts of data is that CPU resources are not close enough to the data storage location. This leads to the transfer of hundreds of gigabytes or terabytes of data, often across the internet, and requires a lot of time, during which the CPU resources are idle. This demonstrates the importance of the strategy of 'data placement' and the impact a suboptimal strategy could have on the overall efficiency of the computing resources usage.

In addition to ‘data placement’, data caching strategies (from offline to online storage, from remote site to local site, from data server to local disk, from local disc to local RAM memory) and the number of available replicas can also have a high impact on the overall efficiency. Similarly, the strategy used for error correction, data replication, has considerable consequences for both the availability and reliability of data, as well as ensuring the data’s long-term preservation.

For these reasons, the role of ‘data management’, seen as the architecture, policies and procedures that manage all data activities and lifecycle, has become of the utmost importance in scientific computing.

2.2 Components of data management: storage services and data pools

In a large research project, the workflow of data can be extremely complex: data can be duplicated across multiple sites to reduce the probability of loss in case of severe operational incidents, or to increase the number of CPU cores able to process the data in parallel, or to increase the available throughput for on-demand transfers (as multiple copies can serve the data in parallel).

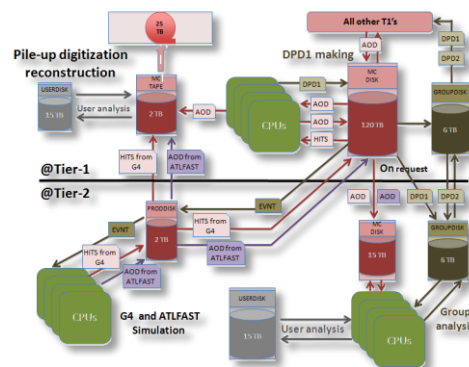


Figure 2: Example of ‘big-data’ workflows in the ATLAS experiment on the LHC at CERN

A significant example is the data management activities of the four LHC¹ experiments at CERN². Data is moved from one storage pool to another to minimize costs or because the quality of service differs among the different pools: for example, a storage pool based on solid state drives with multiple data replicas and fast networking can be used to store the recent high-value raw data that needs to be analyzed (analysis pool). Once the data processing has been completed, the same data can be moved to a high latency, low cost repository, which mainly focuses on high reliability (archive). On the other hand, some intermediate analysis results, which can be recalculated, will likely be moved to a low latency, high throughput, low cost, unreliable temporary storage (scratch space).

The data model of a LHC experiment (see the figure 2 as an example) shows the complexity of the data workflow and its main components: clearly there is a need of data pools with different qualities of service (in terms of reliability, performance, cost), and then plenty of tools are required to transfer and process this data across pools.

¹ LHC is the 27 km long Large Hadron collider accelerator located at CERN, Geneva, see <http://cern.ch/lhc>

² CERN is the European Organization for Nuclear Research, <http://cern.ch>

This requirement of having multiple pools with a specific quality of service is probably the major difference between the 'cloud storage' and 'big data' approaches:

- In the 'cloud storage' model, all data are stored into a unique, huge, flat, uniform storage pool containing all data (typically with three replicas on two sites), there is no need to move the data and the role of 'data management' is entirely delegated to the pool service management team. This single-pool approach implies a uniform quality of service with the consequence that this becomes suboptimal (and therefore anti-economical) when the amount of data grows beyond the point where storage contributes to a large fraction of the cost of the whole scientific project.
- Conversely, the 'big data' approach goes beyond the single unique storage pool. It introduces the concepts of data workflows, data life time, data movement, data placement and a storage infrastructure based on multiple pools with different quality of service where the cost of storage can be significantly reduced and, hopefully, optimized.

These concepts of storage with 'on demand' variable quality of service and the tools required for an effective data management are the main topics of this paper and will be discussed in the next chapters.

2.3 Scientific computing in high energy physics

A clear reference for scientific computing in high energy physics is the CERN Data Centre which stores, preserves and distributes the data from the four LHC experiments namely ALICE, ATLAS, CMS and LHCb³.

At CERN, physics data is received in the computer center at rates between 4 and 6 gigabytes/s and stored in an infrastructure that has approximately 1,500 storage servers with 40,000 disks representing a raw storage space of 75,000 TB. This infrastructure is combined with more than 150 tape servers handling several robotized libraries with more than 50,000 tape cartridges. There are 45 PB of data stored online in hard disks and 100 PB of data stored in magnetic tapes (the quoted numbers are from Oct 2013).

CERN has more than 90,000 cores in 10,000 servers dedicated to process this data, but this represents only 20% of the total computing power necessary to analyze the LHC data. For this reason the data is replicated to more than 150 computer centers worldwide that allow to have between 250,000 processing jobs running in average (figure 3).

³ More information on the four LHC experiments at CERN is available respectively from <http://cern.ch/alice>, <http://cern.ch/atlas>, <http://cern.ch/cms> and <http://cern.ch/lhcb>.



Figure 3: a map to the 150 Tier 1 and 2 data centers participating in the LHC computing grid⁴

This explains the need for the three pillars (Data, Computing and Networking). And if we focus on the ‘Data’ component object of this document, we will now discuss in the following key ingredients:

- An infrastructure to manage metadata information (chapter 3)
- Quality of service that is necessary to define and understand the key indicator of a storage service (chapter 4)
- Tools for data transfer across multiple data pools, especially across wide area networks (chapter 5).
- Data reliability and preservation (chapter 6).
- Service scalability, consistency and performance (chapters 7 and 10)
- Data security which includes Authentication, Authorization and Accounting (chapter 8)
- Service monitoring, alarms and incident responses (chapter 9).

While all these chapters are all essential for ensuring well managed storage service, they can all be designed and implemented independently from each other. This is also a strategy to reduce the complexity seen from the service managers and to allow a large number of persons to work together in a large computing project which typically has a complexity that goes beyond the understanding of a single person.

3. Metadata information on the data

A multi-petabyte data repository is not very useful to a scientists unless he has access to detailed description of the data itself. This information about the data is generally called ‘metadata’ and it is the information that allows to locate, understand and manipulate the data to achieve a particular goal.

Immediately we see that different types of metadata may be needed for different usage of the storage repository: for example a physicist participating in analyzing LHC data at CERN may be interested to search data based on:

⁴ You can read from <http://cern.ch/wlwg> additional information and architecture details on the Worldwide LHC computing Grid (WLCG). There you can find the list of the 12 Tier 1 and the more than 140 Tier 2 data centers participating in the WLCG project.

- Descriptive information and documentation on how the data is structured so that it can be interpreted and analyzed.
- The type of beams that the LHC accelerator was colliding when the data was taken.
- All variable parameters from the accelerator (for example the energy and the intensity of the beams) and from the experiment (trigger rates, thresholds, luminosity) at the moment the data was recorded. This information, called the 'Condition Data', is essential knowledge which is necessary for the physicist to be able to process and transform the data. The 'condition data' represents thousands of recorded parameters.
- Once the data of interest has been logically found, information on where the data is physically located and how it can be retrieved.

On the other hand, the computer scientists in charge of the operation of the storage service is interested in a different type of metadata:

- Information on growth rates so that new storage can be provisioned where necessary.
- Information on access pattern so that data can be pre-placed to locations where access times are reduced increasing the overall efficiency (in case of hierarchical storage systems).
- Information on storage failures and incidents so that appropriate responses can be promptly be organized.
- All the low level attributes of the data: size, owner, creation and last modification dates, history of past requests, list of authorized persons to read, modify and delete the data, encoding, compression level and checksum information that are necessary to implement measurable quality of service in storage
- Location and number of alternate copies of the same data.

In practice the term 'metadata' can have different means between the physicist and the computer scientist because of a clear split of responsibility: the physicists access the content of the data only for the knowledge it represents while the computer scientists to store, preserve and distribute binary blobs of data without requiring any knowledge on its content.

3.1 Separation of metadata management between knowledge of content and binary preservation

The two examples above shows the different needs for metadata handling and the split of responsibilities between the researchers in the experiment (who have the knowledge) and the computing infrastructure, custodian of the data.

The metadata representing the knowledge and the structure of the data is of the utmost strategic importance to the experiment and it is a core component that is specific to each experiment as its design must carefully match its research goals, as briefly described in the next section (3.2).

On the other hand, the low-level metadata that allows an effective management of the data within the storage infrastructure can have a common structure across multiple experiments: once the data has been written to a file, the attributes like size, checksum, owner, permissions are pretty standard metadata information. This allows the delegation of the data storage, preservation and distribution activities to a set of computer scientists.

For example, the 4 LHC experiments at CERN (ALICE, ATLAS, CMS, LHCb) have all their own specific ways of representing their raw data (as the research objectives are different) but they all share a common storage infrastructure provided by the CERN IT department that will be described in details in the rest of this paper.

3.2 Metadata on knowledge and content of the repository

At CERN, the ALICE experiment⁵ uses has a data bandwidth of 4 GB/s which adds to the conditions data derived from the raw data, i.e. calibration and alignment information, which have to be available from the beginning of the reconstruction and therefore are not be included in the raw data and becomes part of the ‘knowledge metadata’. Similarly the ATLAS and CMS experiments⁶ have data bandwidth up to 2 GB/s with different condition data structures and different computing models. The fourth experiment, LHCb⁷, also has his own computing model and his own framework with raw data bandwidth that are around 500 MB/s.

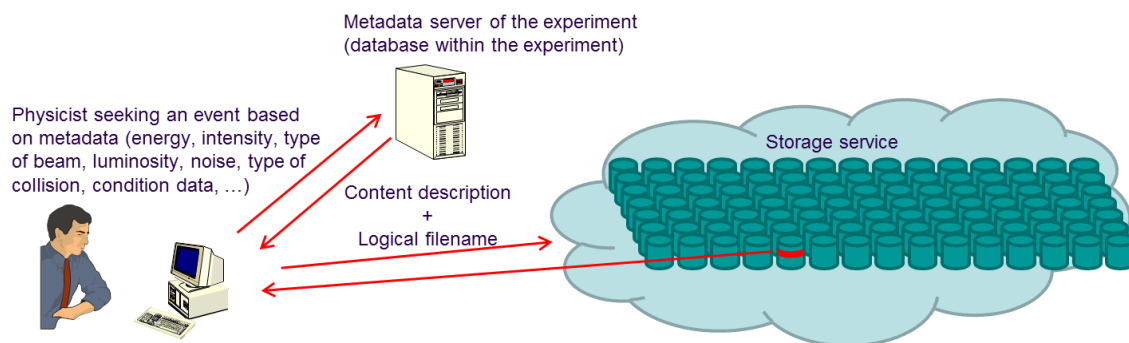


Figure 4: Data discovery by the physicist requires access to databases specific to the experiment

Clearly this shows how the knowledge management and the computing interfaces offered to a researcher to search and understand the data stored are specific to each experiment and requires specific training on the data structures used (figure 4).

This specificity of the experiment’s computing frameworks is essential to guarantee the independence of the scientific discoveries. However, in the long term, all experiments have concerns on how to preserve the knowledge in addition to the binaries raw data already preserved by the storage infrastructure.

Preserving the knowledge in the long term on the experiment is a difficult task and is object of challenging investigations. In High Energy Physics, this is a recent activity that has been started under the project ‘Data Preservation and Long Term Analysis in High Energy Physics

⁵ Information on the ALICE experiment is available at <http://cern.ch/alice>. Additional information on its data acquisition and storage infrastructure is available at http://aliceinfo.cern.ch/Public/en/Chapter2/Chap2_DAO.html

⁶ The ATLAS experiment information is available at <http://cern.ch/atlas>. The CMS experiment is at <http://cern.ch/cms>. A detailed page on the Atlas computing infrastructure is available at <https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/AtlasComputing>

⁷ The official LHCb web site is at <http://cern.ch/lhcb>. The computing model and the LHCb framework (DIRAC, GANGA) are described in the LHCb computing pages available at <https://twiki.cern.ch/twiki/bin/view/LHCB/LHCBComputing>

(<http://www.dphep.org/>)' which represents an attempt to define future standards and best practices on knowledge preservation to be used across multiple experiments. The project is currently specific to High Energy Physics but it is not excluded to extend it to other sciences and, as the reader can imagine, discussing this point in-depth goes beyond the aim of this paper.

3.3 Metadata for the computer scientists

On the other hand, the low-level metadata is used by the storage service to ensure data placement, preservation, and integrity is more consolidated. The way metadata on binary information is structured is extremely important and major consolidation efforts have been made in recent years do converge to a well identified techniques and common practice that are common to all experiments at CERN and worldwide across the Worldwide LHC Computing Grid (WLCG).

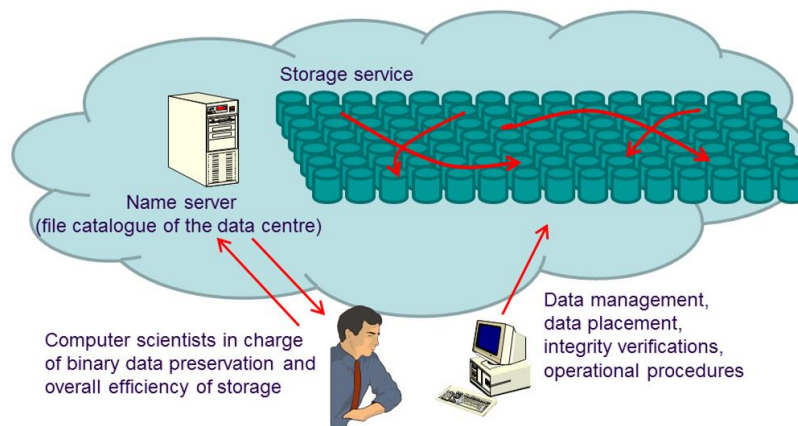


Figure 5: Data management by the storage service is common to all experiments

The main repository of this metadata is the 'name server' (described in detail in section 7.2) which is the database used to store and access this metadata (figure 5). As the reader can imagine, the structure and type of metadata attached to the binary raw data have a huge impact on the performance, reliability and the quality of the storage service. For this reason, the possible choices available will be discussed in depth throughout the rest of this paper.

4. Quality of services for storage pools

The ideal storage pool should be simple to use, scalable, consistent, reliable, available, manageable, flexible, performing, cheap, and secure. Easy said, but when facing practical implementation, compromises become necessary and the choices made impact how each requirement is met. For this reason, we need to define and measure quality in storage services.

In a simplified approach, the definition of quality could be reduced to three main parameters: performance, reliability and cost. In a graphical representation of these parameters, at the edge of a triangle it is easy to understand the compromises that must be done: you can choose your point on the triangle, but you always end up with a pool that is either unreliable or expensive or with poor performance. The simplified commonplace of this approach is that storage hardware can be used to implement all three corners of the triangle (figure 6):

- ‘unmirrored, striped⁸ disks pools (RAID-0)’ are fast, cheap but unreliable
- ‘mirrored⁹ (RAID-1) disks pools’ are fast, reliable but expensive
- ‘pools based on tape technology’ are reliable, cheap but slow¹⁰

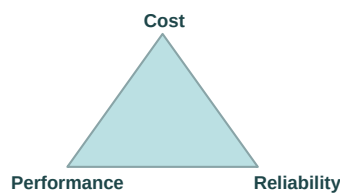


Figure 6: Quality of a storage service can be simplified to three parameters

The real world is unfortunately far more complex. In order to measure cost or performance or reliability quantitatively, there are several sub parameters among which additional compromises are possible or necessary.

The cost of a storage pool can be split among the hardware investment, the power consumption and the manpower required to operate the service. Plenty of choices are possible to shift costs from one component to the other: typically higher investments allows reduced power consumption or reduced human effort. Again we get a triangle representing the various options to split costs.

Reliability also has multiple sub components. For example, we can separate the probability of losing data (reliability) from the probability that data is temporarily not accessible (availability). As discussed later in the reliability chapters, there are data-encoding techniques that are resilient to hardware failures only in terms of data loss but not in terms of availability. As an example, RAID5 and RAID6 disk array¹¹ offer a rather good protection against data loss but after the failure of one disk it takes a long time to recover when the content of the failed disk needs to be rewritten. And, during this time, performance significantly drops, especially in a multi-user environment. At the opposite extreme, a storage pool based on 3 or more disk replicas can expect a better resilience of operation during the recovery and recreation of a missing replica that follows a disk failure. But the two solutions do not have the same cost, hence a choice between different levels of quality of service.

When discussing performance, things becomes more complicated, because the number of sub parameters that defines performance is higher and each parameter is often difficult to measure. It is also important to highlight that a storage solution must manage both the ‘data’ and the ‘metadata’, where the metadata is the information about ‘the objects containing the data’ (file names, file sizes, last modified/accessed/created date, file attributes, access control lists/permissions, storage locations, chunks, checksums/hashes, quotas, locations, replicas, etc.). If we attempt to list the various parameters that define performances we have:

⁸ See http://en.wikipedia.org/wiki/Data_striping

⁹ See http://en.wikipedia.org/wiki/Disk_mirroring

¹⁰ ‘tapes are slow’ is a good example of commonplace, as tapes typically outperforms disks in throughput. Their perceived slowness is related to their high latency which can have a significant impact on performances depending on the data access pattern.

¹¹ See http://en.wikipedia.org/wiki/Standard_RAID_levels for a definition of RAID 6

- Throughput: this is a measure of how fast the data can be read or written to a storage pool (bytes/seconds). Throughput can be measured against both data and metadata, leading to multiple throughput performance indicators.
- Latency: a measure of the average time delay introduced by every complete I/O operation (milliseconds). Applies to both data and metadata
- Both throughput and latency parameters can have significant different values between read and write operations depending on the architectural choice. This doubles the number of performance parameters defined so far.
- Scalability: this is a curve that plots performance parameters against the size of the storage pool and the number of operations per second to identify possible performance degradation when the storage pool exceeds a certain size (scalability of data) or exceeds a certain number of stored objects (scalability of metadata) or exceeds a certain number of operations per second (and there are several operations possible). This creates several additional dimensions to each performance parameter defined so far.
- Consistency: this is the maximum time that is required for a change of data to propagate to all client and server nodes of the storage elements. This ensures that a client cannot access obsolete data that has been either deleted or modified by a different client.

To make things more complicated, scalability measurements are extremely complex because they depend on the access pattern: for example, a storage solution may be highly scalable when accessing data that is reasonably contiguous, while it may suffer from performance degradation in random access patterns. This adds another dimension to all performance parameters.

The result is that the simple triangle 'performance, cost, reliability' becomes a multi-dimensional radar chart. If we add scalability, this would become a 3D surface showing the evolution of the radar plot with the chosen scalability parameter plotted on the Z axis.

However, comparing performances becomes yet more difficult. In the example shown in the figure 7, how would you rank performance of pool 1 and pool 2? Unless all indicators favor one of the two solutions, this is impossible. And comparing the various parameters under a unique figure of merit is also impossible because the weighting depends on the application for which the storage will be used.

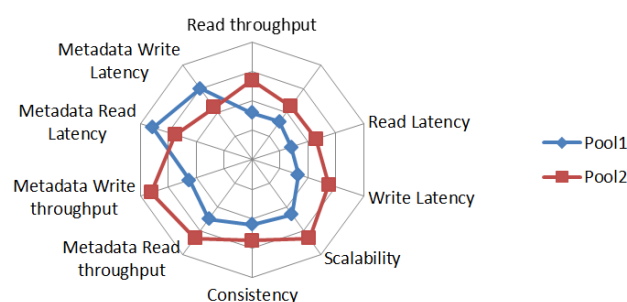


Figure 7: A radar chart defining quality of service

Given that there is not a unique figure of merit available, the definition of 'quality of service' is represented by a set of parameters which have an effect on the efficiency of the application accessing the data. And given that any research application requiring large data processing has a complex workflow, it would be naïve to imagine that a unique quality of service would be optimal for

all steps of the data processing foreseen in the workflow. Therefore it is legitimate to imagine that any scientific application handling a large amount of data ('big data') will require multiple storage pools with different levels of quality of service. Otherwise, some part of the architecture is likely to turn out to be severely inefficient, given that some pools will inevitably be over-dimensioned (and therefore too expensive) in some unnecessary areas to match the common denominator of all requirements.

As an example of multiple storage pools, the storage infrastructure of a typical LHC experiment at CERN is composed of 1- fast and reliable pools for analysis containing 'hot' data that are replicated across multiple data centers, 2- storage pools for home directories and personal data of the scientists, 3- temporary pools (scratch) used to store intermediate results, 4- cheap and reliable pools to archive scientific results and raw data that require long term preservation.

The requirement of having a multi-pool architecture to efficiently handle the complex data-workflows in scientific research becomes evident by remembering that even the simplest computer has multiple types of storage: L1 and L2 caches, RAM memory, hard disk. It is obvious that running a computer from a single type of storage, would be a simplistic approach leading to unavoidable inefficiencies.

4.1 Initial quality of service for storage services

In the chapters above, we defined the quality of service of storage, with the assumption that the architecture of the software in the storage solution could improve some aspects at the expenses of others. But the baseline quality of any storage service starts from the raw quality of the storage media used. The higher the starting quality of the media used, the easier the task of the software will be.

Similarly, the lower the media cost, the greater the margin of the software to use additional media to increase either performance or reliability.

4.2 Baseline performances and reliability of Disks, Tapes, and Solid State Drives

Every person using a computer has a clear picture of what can be expected from hard disks. Common sense says that they are pretty fast and cheap. A majority of laptop or desktop users would also say that they are pretty reliable because they never had a data loss due to disk failures, although they are aware that this can happen. But, you can also find a small minority of users that would claim that hard drives are not reliable (as they have experienced one or more severe data losses in the past).

The difficulty in estimating the reliability of hard drives is related to the fact that end-users are questioned in estimating occurrences of rare events that have a large impact, and this is an area where the individual experience is typically not enough to have an objective opinion.

When we come to tapes, end users often propound outdated and erroneous notions, such as 'tapes are slow'. Similarly the user experience in solid-state drives (SSD) is also surrounded by several misconceptions.

Luckily, in the CERN data centre we have the opportunity to measure the quality of the various media systematically over a very large volume of data and we have reached some surprising conclusions. The first is that disks are by far the most cost-effective (0.04 USD/GB) solution for online

storage (without counting electrical consumption cost of a 24x7 operation), their performance is acceptable (100 MB/s in reading and writing a single stream with a few milliseconds of seek time) but their reliability is too low to ensure any acceptable service without data loss. The CERN data centre has 80,000 disks and we experience a typical failure of 5 disks every day. This gives a daily data loss of 10 TB which is the equivalent of 5,000 average user mailboxes per day. This is unacceptable and is why storing only one copy of data on a single disk is not a viable strategy for a storage service.

Concerning solid-state drives we measure reliability figures that are also below what the manufacturers advertise, although we do not have sufficient volumes to publish reliable statistics. Generic disk controller designed for hard disks have proven not always optimal to be used with SSDs and several operational incidents with data corruption have occurred when SSDs have been used as fast persistent cache in front of hard disks. Also, with the present level of prices, the usage of SSD as a caching technology to improve performance cannot (yet) compete with the possibility of adding additional RAM memory that we now have in a scalable 64 bit architecture. Similarly, the general usage of SSD as primary storage instead of hard disk remains expensive and limited to very specific cases. Therefore, at the current level of prices, the SSDs usage in a data center remains limited ... although we firmly believe that we are getting very close to a crossover point: a small reduction of SSD prices expected in the coming years will make SSD very interesting technology for both persistent data caching and for fast primary storage.

The situation we measure for tapes is rather different. Tapes comes in cartridges of 4-5 TB and have a cost (0.03 USD/GB) comparable to the disks and when it comes to performance, tapes have a high access time (latency) as it can take a couple of minutes to rewind a tape, unmount it, mount another cartridge and fast forward to the location where the data needs to be read. However, once the tape is mounted and positioned, data can be written or read at speeds (> 200 MB/s) that are typically twice those experienced with hard disks. In addition, tape drives have separate heads for reading and writing, allowing the data to be read 'on the fly' just after having been written. In a scenario where a read-after-writing verification is necessary, tapes give another factor of two in data streaming improvement giving the equivalent figure of $200 \text{ MB/s write} + 200 \text{ MB/s read} = 400 \text{ MB/s}$ of throughput.

Another difference between disks and tapes comes when comparing reliability. When a tape fails, the amount of data loss is limited to a localized area of the tape, and the remaining part of the tape remains readable. This 'tape failure incident' generates a data loss that ranges from few hundred of megabytes up to few gigabytes, thus representing 3 orders of magnitude less than the data loss in a disk failure. This figure is confirmed by the CERN data centre, which, with more than 50,000 tapes, experiences a loss of tape data of just a few hundreds of gigabytes. This compares well against the few hundred terabytes of data lost due to disk failure every year.

Failures with tapes are also much less correlated between each other: the probability of a read operation in tape A failing is quite independent from the probability of a failure on tape B. With hard disks, there are several correlations when disks are in the same servers or under the same controller, as the probability of a second failure after a first one is much higher than the probability of the initial failure. We will see (in chapter 6) that independence of failures is a key strategy to increase the

service reliability and this independence is often missing when servers with large number of disks are used in the infrastructure.

5. Tools for data transfers and replications

Once accepted that the underlying storage infrastructure for scientific computing is composed of multiple storage pools, we need to understand the techniques to move data across these pools.

We may start with the operation 'moving' data from one pool to another, where the data is first copied and then deleted in an atomic (and consistent) operation. However, as there is often the need to separate the 'copy' from the 'delete' process, we have three primitive operations (copy, delete, move) that are well known to all storage systems.

The next step, is to be able to extend the primitive operation to aggregated sets of data, which we call 'freight train' approach. For example, file system users can operate on folders and directories, and in many cases also recursively through a hierarchy of folders. Similarly, in a 'big data' infrastructure, we need to be able to aggregate data beyond the concept of file, folder or directory using metadata information. In many cases, to overcome the limited scalability of hierarchical storage based on directories, data is stored in a flat container using a 'dataset' property in the metadata that can be used for aggregated operations. This is a good example where a compromise is necessary between the complexity of the metadata (which limits the scalability) and the efficiency of the data management tools.

The availability of basic primitives, allowing aggregated operations based on metadata properties, makes possible the deployment of several automated data movements that are generically called data synchronization (or data syncing) activities that can be planned under different scenarios.

5.1 Master/slave or multi-master scenarios

The first scenario which is also the easiest to implement is the one-way master/slave replication, where data are copied from one (source, master) location to another (target, slave) location, with no data ever copied back. The slave copy is read/only for end-users with only the replication process allowed to modify its content.

The alternative scenario, more complicated, is the two-way multiple-masters implementation, where all data pools are read/write. Each write operation is detected (synchronously or asynchronously – see later) and replicated to the other master(s). Within this approach, conflicts and temporary inconsistencies are possible and must be handled, either manually, by raising an alarm and stopping the syncing process or automatically, based on predefined policies, as described later.

A synchronization process scans and replicates differences which have been found. The scan can be at different levels of detail which affect the performance and the reliability of the result. The process can take decisions on data movements only by comparing metadata information and/or file descriptors in the storage elements. A more detailed scan may also check the existence of every file and object. But how can we ensure that the data stored in two files of the same name, size and modified date is the same? One step further is to also calculate and compare checksums and take data movement decisions by relying on the checksum information. And if we want to really ensure

that two storage repositories are really in sync and thus be immune to checksum collisions, we need to execute a binary level compare on all data!

The paragraph above should give an idea of the problem of 'consistency' of a storage repository which comes in as an additional compromise with performance: to have a fast synchronization you may temporarily lose consistency or vice versa to ensure high consistency, synchronization must be slow, done in real time, with performance affected.

In theory, in a perfect world, where software always behaves as designed, replication based on metadata information should be enough to ensure a sufficient level of consistency. In practice, there are several cases where this is not enough, in particular due to replication errors that happen when a source or target storage element is temporarily unavailable or corrupted. In this case, the replication software must be able to handle the integrity and consistency of the data by dynamically switching consistency check level from metadata to checksum to binary data whenever this is necessary.

5.2 Synchronous, asynchronous, scheduled, and manual synchronizations

To ensure the maximum consistency each create/modify/delete operation on a storage element must be intercepted and replicated in real-time to the other slave or master pools. This approach (called 'synchronous' synchronization) has a real-time synchronization process working only on metadata to guarantee maximum levels of performance but it is associated with a secondary, asynchronous, scheduled scan into checksums or binary data that ensure consistency beyond errors and temporary unavailability of pools.

Consistency of data across multiple pools is an important requirement and many applications want to control the consistency manually as they often deliberately need to allow part of the storage to temporarily loose consistency. For this reason the 'synchronous' sync can be replaced by a 'scheduled' synchronization enabling timelines of data changes: for example, in a daily synchronization on 3 pools, one pool can contain the active data, another the same data as they were yesterday and a third pool could contain the data from two days before. This allows the implementation of backups, history and data roll-back procedures.

Similarly, there is always the need to trigger data synchronizations manually without schedule in order to create checkpoints, baselines, or data archives. An example from the CERN LHC experiments is the large data recalls from offline tape to online disks that are necessary for some types of data re-processing.

As there are so many scenarios to handle the consistency of data, the catch-all solution is the 'custom' synchronization. In this case the owner of the data writes automation scripts to implement a specific transfer policy within his application. This is no longer manual nor scheduled at precise intervals but is triggered by external conditions under the control of the data owner. This is the ultimate feature that empowers 'consistency on demand'.

5.3 Synchronization algorithms

Imagine two data pools containing files that are being synchronized. There is a file that exists in the first pool but is missing in the second pool. It is essential to distinguish between the case where the file is missing from the second pool because it was deleted from it (and therefore the sync program

should also delete the additional copy left in the first pool) or the file is missing because it is a new file that was created since the last sync (and therefore the sync program should copy it to the pool where it is missing).

This problem could be solved by remembering the last synchronization time (when consistency was last enforced) and comparing it with the 'creation time' of the file's metadata. But this simplistic approach is prone to errors (and data loss) because the metadata cannot always be trusted (because it is under control of the client application). A more reliable syncing programme must create and maintain a database of the metadata of all past synchronized files to be able to differentiate the two cases. This is the approach taken by many implementations. It is safe but relies on uniquely identifying files (and their content) using their metadata. A more robust approach would be to remember, within the database of the metadata of all past synchronized files, all checksums of the files synchronised in the past. This guarantees the correct action but if the file system does not offer primitives to know the checksums of existing files stored, the syncing programme must read the entire file content to calculate its checksum and compare it with the database of previously synchronized files. This is impractical and hostile to high performance.

This last example should explain why modern storage elements add checksum information to the metadata: it is an essential extension, not part of the POSIX standard, that allows reliable synchronizations and data integrity checks, as described later in the 'integrity' paragraphs (chapters 5.9 and 8.5).

5.4 Handling errors

So we think we have solved the problem of correctly discriminating between a file deletion in pool A from a new file creation in pool B. Now imagine the case where the absence of our file from pool A is related to an unforeseen failure of pool A. If we apply our syncing algorithm, knowing (from the database of past synchronization) that the file was synced before and now it is no longer on pool A we would delete it (incorrectly!) from pool B.

This example shows how delicate the syncing algorithm is and the importance of the software having a complete awareness of possible failures of all components delivering the storage service. The only possible workaround for this problem is to have a reliable monitoring infrastructure that keeps an up-to-date status of the availability of all components involved in delivering the storage service. And this monitoring infrastructure is perfectly integrated within the syncing programme that uses this additional information to correctly discriminate and decide which actions to take.

However, this approach is not entirely robust as there is always a delay between a pool failure and its detection by the monitoring system. During this delay, as short as it can be, there is always a small probability for the syncing process to take an incorrect decision and corrupt the data.

The only way to solve this last problem is to ensure that all actions taken by the syncing process are journaled and reversible, like transactions that can be rolled back. We will discuss later how the journal can be implemented, but for the moment it is important that, to ensure consistency, all actions must be journaled for a duration at least equal to the maximum time required by the monitoring system or the syncing process to detect the error, so that that in case of late detection of a failure, any transaction which could have corrupted the data or created inconsistencies can be rolled back.

This maximum time within which a transaction can be rolled back also has an impact on the consistency versus performance compromise: for a maximum consistency, every write operation requested by the user should be acknowledged only after this delay has expired, this creating significant performance degradation. For a maximum performance, every write operation can be immediately acknowledged, but in case of rollback, temporary or permanent inconsistencies may be introduced in the client application (which is unaware that an acknowledged request has been rolled back).

5.5 Handling conflicts

Conflicts are events that the syncing process cannot resolve unambiguously. They are similar but different from errors because they are generated by an incorrect usage of the storage service by the end-user or by an external application. A conflict is a normal event in a storage service and not a symptom of a problem (although it is an indication of a problem in the client application).

The typical example of a conflict is a file that has been simultaneously modified in two different masters: the synchronization program discovers two files have both different metadata from the one seen during the last synchronization. In this case, the conflict is immediately detected but cannot be resolved without the knowledge of the origin of the conflict.

When a conflict is detected, the easiest option is to stop the synchronization and notify the owner with the details of the inconsistency and expect them to manually resolve the issue. However, this mechanism is impractical or even impossible, when managing large amounts of data. An alternative is to define a policy that must be agreed and accepted by the data owner on how conflicts should be resolved automatically.

In the example mentioned above, where multiple master copies of the same file have been independently modified, a possible policy could be:

- Define which master is the 'reference' (super-master) to be used in case of conflict.
- Use the most recent modification as the reference copy. Note that in order to implement this policy it is essential that timestamps of actions taken in different servers are consistently ordered (and this is an interesting challenge by itself).
- Duplicate the data object of the conflict.

In multi-master storage services, conflicts are unavoidable and therefore handling conflicts must be foreseen. It is very important to stress that the existence of a policy to resolve the conflict does not guarantee that there will be no data loss. The policy is only a contract between the storage service and the data owner on how the conflict will be resolved such that the data owner is aware of the clear conditions that may generate data loss in case of inconsistent usage of the storage service.

5.6 Syncing over Wide Area Networks (WAN)

We have just stressed the importance of having a reliable and consistent monitoring program that is necessary in a multi-master synchronized environment. Typically this can be achieved by having the pools being synchronized, supervised by a unique monitoring programme. This implies that the pools are located in the same datacentre and under a common monitoring and management infrastructure.

On the other hand, when syncing over wide area network, typically across a generic Internet connection, this is not possible. In this scenario, the syncing programme, located at one end or at a third location, cannot have access to reliable monitoring information at both ends and this requires to have a journal that can ensure rollbacks over a period of time that is much longer compared to the local sync scenario.

In addition, as data transfer over wide area tends to be more limited in bandwidth and significantly more expensive, several additional software developments are necessary to ensure a reliable syncing:

- first of all, the possibility of having complete awareness of the data already transferred, so as to avoid retransmitting the same data again after a transfer failure. Basically it is essential to be able to resume transfers from the exact point they failed.
- Then comes the requirement of splitting a large transfer over multiple parallel data streams. This also minimizes the impact of transfer failures but also allows the usage of full Internet bandwidth between data centres even in the case where the aggregated bandwidth is composed of multiple parallel routes with different latencies.

These requirements both push in the direction of chunked data transfers, similar to the one used in the bittorrent protocol¹². This will be discussed again later in the chapter on implementing reliability.

5.7 Wide Area Networks (WAN) syncing and data reliability

Finally, while syncing over a wide area network may be a little bit harder in terms of the software quality required, it also has the advantage that failures of storage services located at different locations, under different management, may be considered independent.

The independence of failures is a key ingredient for increasing reliability: for example, in many scenarios, reliability is increased by storing the same information multiple times in different locations (multiple replicas). If the multiple locations are all in the same data centre there are incidents that may impact all copies of the data, voiding entirely the motivation for the multiple replicas. For example, incidents like fire, floods, or over temperature due to loss of cooling capacity may have catastrophic consequences on all data stored at a unique location. Similarly, whenever there is a common management team on all data, consequences of human errors (or even deliberate sabotage) in applying operational procedures may also be catastrophic.

The usage of multiple locations to increase reliability will be discussed further in the data reliability chapter when discussing the need of having multiple, independent events necessary to trigger any data loss.

5.8 Syncing across reliable or unreliable storage pools

In previous chapters, we mentioned that synchronous, scheduled, and manual types of syncing was at the origin of the several motivations for syncing, including minimizing wide area traffic and increasing reliability.

¹² See [http://en.wikipedia.org/wiki/BitTorrent_\(protocol\)](http://en.wikipedia.org/wiki/BitTorrent_(protocol))

However, it is worth mentioning that syncing two unreliable pools can be a way often used to have only one but reliable storage pool, with two replicas of the data. In this case, if reliability is the ultimate and only goal, there may be more effective alternative ways to achieve identical high reliability figures, that are described in the reliability chapter.

But is it really necessary to replicate the entire storage pools content in a scheduled or manual sync process when the reliability is already ensured at the storage pool level? Clearly not, especially if high performance is not at stake. In this case, the synced snapshot does not require having the entire data, but only the differences from the master one. In this case, we must remember that if the storage pool ensures reliability and journaling, scheduled and manual synchronization process to create archives or for the implementation of history and data roll-back procedures is probably not necessary, because all these functionalities can be derived from the journal of the reliable pool.

5.9 Ensuring data integrity: data verification

Whenever the data is replicated across multiple pools, what are the procedures to ensure that there is no difference of content between the multiple replicas? Data integrity is, again, a basic function of a storage pool that will be discussed in details in the reliability chapter but in the case of multiple pools, it is important to mention that in addition to the procedures to ensure the integrity of every individual pool, the architecture must foresee additional procedures to ensure integrity of data across the multiple pools.

If every pool already ensures a sufficient level of integrity for its own data, then this can be implemented by a process that compares metadata and hash information from the different pools.

However, in some cases the pool does not have hash or checksum information for its data or the checksum information cannot be trusted to represent the content and this corresponds to the scenario when the individual pools do not offer a sufficient level of integrity for its own data. In this case, to properly ensure data integrity across multiple pools it is necessary to have an infrastructure based on multiple process: a process that constantly scans the entire data content and calculates hash and checksum information associated with a process running at higher frequency that compares the calculated hash information across the different pools.

6. Reliability

Reliability of a storage service is related to the probability of losing data and could be defined in relation to failures expected per I/O operation or failure expected per time unit. The definition that we retain is the latter, i.e. we define reliability as ‘the probability that a storage service will perform an arbitrarily large number of I/O operations without data loss during a specified period of time’.

The overall reliability of a storage service depends from many environmental factors that can affect the service: for example, energy failures with sudden stops can trigger unexpected rate of hardware failures. Similarly cooling malfunctioning in the machine room can generate high temperature that can increase the rate of media failures. Last but not least, the people managing the service, their competence and their procedures have a significant impact on reliability. To keep the reliability discussion focussed on computing, these aspects which are more related to management and work organization will only be mentioned briefly in this paper, despite being important.

As for quality of service, discussed in chapter 4.1, to estimate the reliability of a 'storage service' we start from the reliability of the underlying hardware. For example, if a storage service is composed of disk servers with directly connected disks, we can imagine that the reliability of service is equal to the reliability of the hard disks used. But the whole point of using data management and cloud storage solutions is to increase the reliability of the hardware at the expenses of performance and/or additional hardware. This is achieved using software and well defined procedures and workflows.

6.1 Extending performances and reliabilities using software

Traditionally, to increase the reliability of a storage service 'Redundant Array of Inexpensive Disks (RAID)'¹³ have been used. RAID solutions typically include:

- 'Disk Mirroring' (RAID1) where the data is always written to two disks in parallel so that the storage service can survive without data loss to the physical failure of any of the two disks.
- RAID5, where one additional disk is added to a pool of disks so that parity information distributed across all disks can be used to reconstruct the content of any failed disk in the set. In this scenario, the storage service can survive without data loss to the physical failure of any of the many disks that compose the pool.
- RAID6, where two additional disks are added to a pool of disks so that additional error correction information distributed across all disks can be used to reconstruct the content of any two failed disks. In this scenario, the storage service can survive without data loss to the physical failure of any two of the many disks that compose the pool.

When trying to calculate reliability of RAID systems we could naively use the probability to fail of a single disk and calculate the probability of having two simultaneous disk failures (that would generate data loss in a RAID1, RAID10, RAID5 scenario) or the probability of having a triple simultaneous disk failure in the case of RAID6 by squaring or cubing the individual probabilities of a single disk failure. This approach is not only naïve but fundamentally incorrect because the probability for two events to happen can be calculated as the product of the individual events only if the events are independent:

$$P(A \text{ and } B) = \begin{cases} P(A) \times P(B), & \text{if } A \text{ and } B \text{ are independent} \\ P(A) \times (B/A), & \text{otherwise} \end{cases}$$

The failure of two disks within the same server, connected to the same disk controller, in the same rack, using the same power supply, in the same building, under the same management scheme cannot be considered independent events. On the contrary, there are so many dependencies and coupling factors that after the initial failure $P(A)$, the conditional probability of having a second failure $P(B/A)$ is at least one order of magnitude higher compared to the initial failure $P(A)$. This because the initial failure could be related to any of the coupling factors: faulty disk controller, high temperature, faulty fan, power loss, faulty power supply, etc.

¹³ See <http://en.wikipedia.org/wiki/RAID> for more information about RAID and RAID levels

If we repeat this reasoning to three disks, we immediately see that the conditional probability of having a third disk failure after two have already failed in the same disk array is several orders of magnitudes higher than the initial probability $P(A)$, because the simultaneous loss of two disks is probably due to one of the many coupling factors that are likely to severely increase the probability of a third disk failure.

6.2 Reducing the dependencies of failures

From the previous chapter, it becomes clear that to increase reliability of a storage service, all development efforts must be targeted at reducing all possible sources of dependencies and common points of failures in the infrastructure. If we achieve this, then $P(B/A) = P(B) = P(A)$. And when we reach the independence of failures, we can then estimate the probability of a data loss by simply multiplying the individual probabilities.

Removing common points of failures in a data centre is a difficult task that can be tackled in multiple steps. The first is to remove all common source of failures related to common servers, common power supplies, common racks, common networking equipment. This can be achieved by spreading the N disks composing an array of aggregated data on N different servers and implementing, in software, the RAID functionality across the network. This is typically what the data management software at CERN has implemented under the name of RAIN (Redundant Array of Inexpensive Nodes, code name EOS) which allows the implementation of RAID 1 (with arbitrary number of replicas), 10, 6 (using diagonal parity algorithms) across independent servers on the network (80'000 disks involved).

This is a major step forward towards creating independency of failures. However, several common point of failures remains if the various servers are located in the same data centre:

- Common energy supply
- Common cooling system
- Common software
- Common management personnel and operational procedures

For this reason, the data management strategy of the CERN LHC experiments are based on the Worldwide LHC Computing Grid (WLCG) where the data is guaranteed to be replicated across multiple data centres. With this last architectural choice we reach quite a good independence of failures because we can expect a media failure in Geneva, Switzerland to be pretty independent from a simultaneous media failure on the same data in one of the 11 Tier 1 data centres located in different continents used within the WLCG infrastructure.

Why did I write 'quite a good independence of failures' instead of 'independence of failures'? Are there still any coupling dependencies? The answer is, unfortunately, yes: the software.

There is a common framework in every data management architecture which is distributed to all sites and which may be at the origin of a common and simultaneous failure at multiple sites. Also the usage of 'common operating systems' and software solutions (often with common vulnerabilities) has been at the origin of common failures. And these failures can be accidental or the results of organized sabotage.

To reduce even further coupling related to common software, the usage of magnetic tapes as storage media turns out to have a very strong advantage over hard disk media: while it would take few minutes for a privileged operation (either as a result of bugs in the software, or human error or deliberate sabotage) to wipe out a 100 petabyte repository stored on hard disks, with tapes this is impossible, as erasing 100 PB of tapes, even with a large drive infrastructure exceeding 100 drives like the one available at CERN would take several years.

This last point shows again why tapes have a clear role in a storage infrastructure aiming to deliver a reliable service.

6.3 Arbitrary reliability, arbitrary performances

We discussed earlier the importance of delivering pools with different quality of services in terms of performance, reliability, costs and derived sub-parameters.

These requirements impose to go beyond few (3 or 4) fixed configurations to cover all the complex, multi-dimensional space of quality of services. The data management software must allow to reconfigure storage pools and increase or decrease any parameter at will.

This new requirement increases the complexity of our storage model because it adds a new set of parameters which are the partial derivatives between any two existing quality parameters. For example it may be essential to know

- How much I can increase reliability by reducing storage space
- How much I can increase reliability by reducing throughput (performance)
- How much I can increase reliability by increasing latency
- How much I can increase reliability by paying more (cost)
- How much I can increase any performance indicator by paying more (cost)
- How much I can increase any performance indicator by reducing reliability (or availability)
- ... and many more possible combination of two parameters.

While it is easy to understand the link between reliability and cost (using more hardware we can make storage service more reliable) as well as the link between performance and cost (using more hardware we can make storage services faster – with some limits) the link between reliability and performance is more subtle but it is as important.

The importance of being able to trade between reliability and performance in a storage service is because scientific research projects often do not have the flexibility to change significantly the overall computing budget, and the cost is typically the less flexible parameter to change from the ‘customer’ point of view (despite it being the easiest parameter to change from the ‘service provider point of view by asking to pay more when the customer needs more).

In this scenario where the computing budget is fixed and the storage service must provide the most ‘bangs for the bucks’, the necessity of tuning performance versus reliability becomes essential. This can be done in software by supporting multiple encoding algorithms to provide error correction. Not all error-correction algorithms are equal: some are very efficient in term of storage, other are very efficient in terms of CPU overhead and there is no ‘best of all’ solution.

6.4 Splitting storage data in chunks

Before entering in the review of few data encoding and error correction algorithms, there is an intermediate optimization that becomes necessary: so far we have discussed RAID and RAIN (Redundant Array of Inexpensive Nodes) using the 'disk' as the base storage unit.

Using the 'disk' as the base storage unit is unpractical and lacks of flexibility: disks may not have identical sizes and they typically hold a large amount of data that makes the encoding algorithm time consuming.

To ensure equal size and reasonable sizes, the data is split in 'chunks' (or 'blocks') of equal size (figure 8). Then the process used by nearly all error correction algorithms is:

- Group all chunks in sets of m chunks.
- For each group of m chunks, generate n additional chunks so that for any set of m chunks chosen among the $m+n$, the missing n chunks can be reconstructed.
- Scatter the $m+n$ chunks on independent storage.

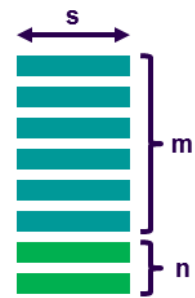


Figure 8: splitting in chunks

Note that there are few generalities that can be said on this approach: The first remark is that the reliability is independent from the chunk size, s , which is arbitrary. However, the size of s matters for performance. As both large and small s impact performance, the value of s must be optimized to match the average size of the data request that we want to optimize (which can be significantly different from the average file size and very specific to the application owning the data).

As the storage system is immune to the loss of n simultaneous chunks from pools of $m+n$ storage chunks and both m and n are arbitrary, we can achieve arbitrary reliability, with an increase of raw storage usage equal to $n / (n + m)$.

This overhead for error correction code (n blocks) can also be reduced arbitrarily by increasing m . Clearly this increases the amount of data lost which happens when more than n chunks becomes unavailable.

6.5 Adding checksum and integrity information to each chunk

The various error correcting algorithms allow the recovery of any n chunks provided that you know how to differentiate the valid chunks from the corrupted ones. If you do not know which chunks are good and which chunks are bad than you can only reconstruct $n-1$ chunks.

To restore the error correcting capability to its full potential of n chunks, it is therefore important to add a checksum verification field to every chunk so that its integrity can be verified independently from the error correction (figure 9).

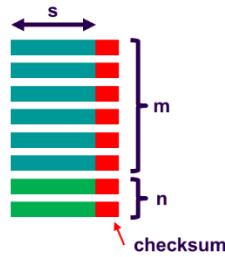


Figure 9: Adding checksum to the chunks to guarantee data integrity

With this trick we do not waste any of the additional n chunks to verify the integrity and we keep them all available for error correction.

The storage overhead of the checksum is typically of few hundred bytes and can be easily neglected compared to the chunk size that is of few megabytes. However, to guarantee a high efficiency in transferring the chunks it is essential that the sum of the chunk size with its checksum is an exact multiple or divisor of the physical block size of the storage being used.

An error to avoid is to choose a chunk size equal to the physical disk block size leaving no space to save the checksum in the same physical block. This will force the software to waste one additional entire block to save the checksum literally doubling the storage capacity used just to add less than 1% of additional integrity information (figure 10).



Figure 10: ensuring that the checksum information fits in the block size

6.6 Data encoding and error correction algorithms

The first evident case is with $m = 1$ and the n extra chunks are just additional copies. With $m = 1$ and $n = 1$ we have the equivalent of RAID1 (mirroring, one replica), and as the chunks are saved to an arbitrarily high number of hard disks, we can imagine that striping across multiple hard disks comes for free making this a very effective high performance RAID10 (mirroring + striping) implementation. But in this scenario reliability may not be high enough and therefore $m = 1$, $n = 2$ (two additional copies for a total of three copies of the data) are also frequently used. In this scenario with $n=2$ the read performance is even higher as the 3 copies can be used in parallel to serve the data (figure 11).

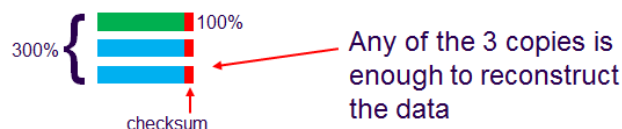


Figure 11: Example of reliability by data replication: $m = 1$ (data) , $n = 2$ (copies)

While replicating data is inefficient in terms of storage lost (200% overhead for the example with $m=1$, $n=2$), it is very effective for boosting read performance and throughput especially in a multi user environment. In the LHC experiments, there are data pools that contain configuration data that must be read literally by several thousands of batch jobs worldwide which are stored in pools with $m=1$ $n=9$ (replica factor of 10) supporting thousands of simultaneous users.

As briefly mentioned the option $m > 1$ with $n = 1$ is equivalent to the RAID 5 solution that is considered too weak in terms of reliability. If we move to $n = 2$, then the solution $m = 4$ becomes interesting because it can be implemented using the row diagonal parity (also called double parity) algorithm which is particularly efficient because it uses only simple fast XOR operations.

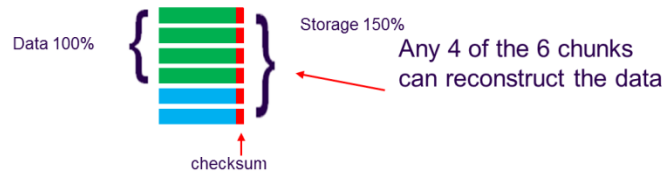


Figure 12: A 4+2 reconstruction schema

The $m = 4$, $n = 2$ solution has only 50% storage overhead and is immune to double failures. The read performance is also interesting because all 6 chunks can be read in parallel to serve the data and only 4 are needed. However, while the read performance in single user mode may be outstanding, in a multi-user environment it could turn out to be reduced as every single read request requires reading data from 4 independent disks. As such, there is a factor of 4 in the number of I/O requests compared to the $m = 1$ solutions. Similarly, in writing, 6 operations are necessary of which only 2 can be deferred, while in the $m = 1$ solution only one operation is necessary and all replication can be done asynchronously.

Typically, the $m = 4$, $n = 2$ solution is a golden standard to achieve high reliability at low cost with outstanding single user performance (figure 12). It can be used for cold storage and archiving where data is transferred using the ‘freight train’ approach and can be used as a valid alternative to tape storage with the advantage that random access mode (low latency) is preserved.

In general, the $m + n$ approach can be generalized and there are several implementation algorithms. The most known is the Reed-Solomon and for $n = 3$ the RAID-Z¹⁴ triple parity used in the Sun-Oracle ZFS¹⁵ file system and its open source version OpenZFS¹⁶. At CERN (in the EOS software used by the LHC experiments) we have evaluated several erasure coding libraries¹⁷ which implements ‘Forward Error Correction’ (FEC) with arbitrary m and n . In general, as n increases the computational effort to decode or reconstruct missing data can become significantly important and reduces the area of application. However, depending on the data access pattern, they can be used to provide extremely reliable storage at a minimal cost. Two erasure coding libraries were integrated in the service and one was finally selected for production¹⁸.

¹⁴ See http://en.wikipedia.org/wiki/Non-standard_RAID_levels

¹⁵ See the page <http://en.wikipedia.org/wiki/ZFS> which points to several web sites involved in ZFS development

¹⁶ See <http://open-zfs.org/>

¹⁷ See J.S. Plank, J. Luo, C. D. Schuman, L. Xu, Z. Wilcox-O’Hearn ‘A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries For Storage’ at http://www.usenix.org/legacy/event/fast09/tech/full_papers/plank/plank.html

¹⁸ The CERN Storage service supports the usage of the zfec library available at <http://pypi.python.org/pypi/zfec> and the Jerasure library available at <https://github.com/tsuraan/Jerasure>. The Jerasure library has been retained for the production implementation.

6.7 Increasing both performance and reliability

In the previous chapter, when discussing on generating the additional n chunks from the groups of m chunks, we made the hypothesis that the n additional chunks would be generated so that from any set of m chunks chosen among the $m+n$ we could reconstruct the missing n chunks.

Let's now weaken this requirement and require that the n additional chunks should allow reconstructing the original data from any z chunks, where $m \leq z < m+n$.

When $m=z$ we fall back in the previous error correction scenario (data can be recovered from any m chunks). When $m < z$ we need more chunks (z) to recover the data, and therefore we are less immune to chunk loss or we require more storage to store for more chunks to obtain comparable reliability.

Why is this approach interesting? Because it overcomes the computational effort to decode the data that we have with Reed-Solomon (or equivalent) 'optimal' algorithms and by adding some additional redundancy we can reconstruct the data from simple XOR operations.

As examples of these encoding techniques, we have fountain and raptor codes. One solution that has been studied at CERN includes 'Low-Density Parity-Check' (LDPC) codes which can be seen in the following diagram (figure 13) used for an implementation with $m=8$, $n=6$ and $z=11$.

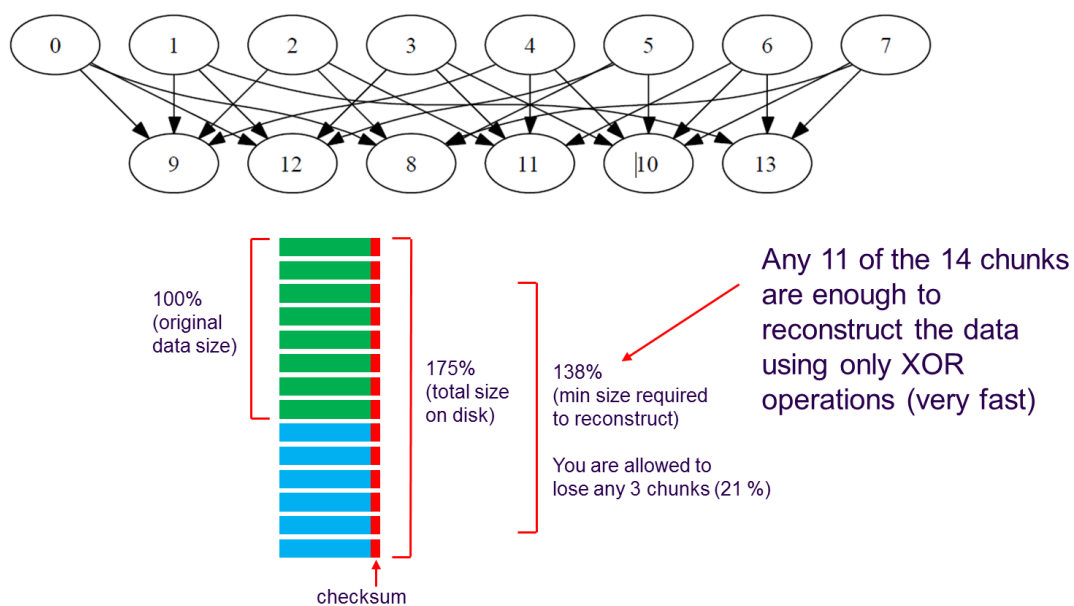


Figure 13: An error correction algorithm based on LDPC using 8+6 chunks

In the example above, chunks 0-7 represents the original data ($m=8$), while the 6 additional chunks 9-13 represent the derived error correction information ($n=6$). The derived error correction information is calculated with simple XOR operations that follows the graph, and to recover the original information, follow the graph in the reverse direction.

As the error correction algorithm is not optimal, it is not always possible to reconstruct the original 8 chunks from any 8 other chunks. However, it is always possible to reconstruct the original 8 chunks if you have at least 11 other chunks ($z = 11$).

This solution clearly has a storage overhead of 75% but it allows to reconstruct the original data after the loss of any 3 chunks (and sometimes even more as it depends on the chunks that have been lost), which makes it highly reliable. As the XOR operation overhead is limited, also excellent performance will be achieved.

7. Scalability and consistency

In the chapter on quality of service, we defined scalability as the set of curves that correlate the performance parameters with the size of the storage pool and the number of operations per second. Similarly consistency was defined as the time required for a change of data to propagate to all client and server nodes of the storage elements. The two definitions side by side should give a hint that there are several ways to increase scalability at the expenses of consistency and vice-versa.

7.1 The network access protocol

There are several access protocols available to access and modify data remotely over the network, offering a wide range of possibilities: the golden standards are [http](#)¹⁹, [ftp](#)²⁰, [nfs](#)²¹, [S3](#)²² as well as other protocols more optimized for particular applications: for example, among the most used in high energy physics we can quote [xroot](#)²³, [dcap](#)²⁴, and [AFS](#)²⁵.

One of the reasons why there are so many access protocols is because each of them has been historically designed to improve a particular aspect that was (and still is) considered important. Some protocols are more suitable to deliver scalable services and other higher consistency. Some protocols delivers higher throughput in reliable local area networks other are better suited for wide area transfers over the internet.

In the trade between scalability and consistency, the caching algorithm used by the networking client plays an important role: an effective usage of the client cache reduces the load on the network servers and therefore increase scalability... at the risk of not being aware that the data had been invalidated (loss of consistency).

At CERN and in the LHC experiments the [http](#) standard protocol is supported across all implementations allowing any data to be accessible from anywhere in the world. There are many physics research investigations that do not have strong requirements in term of data throughput, latency and consistency that are heavily using the [http](#) protocol. This is combined, for the most demanding applications, with the [xroot](#) protocol when massive data transfers and/or processing with high consistency are required or with the [GridFTP](#)²⁶ protocol optimized for parallel wide-area transfers.

¹⁹ See <http://tools.ietf.org/html/rfc2616>

²⁰ See <http://tools.ietf.org/html/rfc959>

²¹ See <http://tools.ietf.org/html/rfc3530>

²² See <http://aws.amazon.com/s3/>

²³ See <http://xrootd.slac.stanford.edu>

²⁴ See <http://www.dcache.org/downloads/dcap>

²⁵ See <http://www.openafs.org>

²⁶ See <http://en.wikipedia.org/wiki/GridFTP>

Why custom protocols instead of standard ones? There is nothing particular innovative in the core of the xroot protocol itself, apart from being 'custom'. This means that, as we understand and have access to the source code, we can easily introduce customizations (in particular tuning the caching, garbage collection, and the read-ahead components) that ensure the full optimization for the specific data transfers or access patterns that are expected.

By combining the development environment used in high energy physics (for example the root²⁷ framework) with a network protocol that can be easily reconfigured, we offer the option to the research physicists developing the application to use a 'generic implementation' that delivers a 'generic performance' or to optimize his application and control the whole of the data workflows so that he is empowered to tune the levels of performances and consistency desired.

7.2 The name server

Another main component of a storage service that affects performance, scalability and consistency is the name server. It represents 'the' database of a managed storage system containing the catalogue of all data (usually all file names). The file metadata are also typically included in the name server database although depending on the level of performance optimization some metadata columns (like access control list and permissions) may be stored within the data itself.

The name server allows the implementation of the 'name resolution' process that allows the translation of a 'logical name' user by end-user applications into a 'physical name or location' which is used by the data management software.

A name server is not strictly necessary as there are many solutions to 'calculate' or derive the physical location of a file from its logical filename representation using well defined conventions: the best example comes with file systems where the physical name and the logical name are hardcoded together. Clearly not using a names server gives a performance advantage, but when the size of the storage service grows, there are several data management scenarios where the existence of a names server is useful or even indispensable.

The most important requirement for a name server is an increased flexibility in data management that allows transparent / automated data migrations. Data can be moved across different pools with different quality of service according to various rules allowing the optimization of the application performance without modifying the application code. For example, in the implementation of a hierarchical file system, 'hot data' is kept on disk and 'cold data' is migrated to tape, transparently without modifications of the client application that accesses the data wherever this is located using the logical filename.

Another need for a name server appears when the storage architecture manages multiple copies of the data under a unique logical name, with automated and variable number replicas depending on the demand. Or when federating multiple, independent storage service under a unique namespace.

The name server service is, unfortunately, a critical component and a weak point under several aspects:

²⁷ See <http://root.cern.ch/>

- Performances can drop in a scenario where every data request requires a name server database lookup as the name resolution is done in real-time.
- Every meta-data operation requires a database transaction.
- The name server becomes a single point of failure of the entire storage service architecture.

Detailing how an efficient name server should be implemented goes beyond the goals of this chapter. However one important point worth mentioning is where the 'name server' approach is better placed. Between two extreme architectures, one without name server and the other which foresees a name server transaction for every request, there are many intermediate solutions where a name server exist but a large fraction of requests are executed without name server lookup. For example the World Wide Web has a name server (DNS) lookup placed at the 'host' and 'domain' levels and all subsequent requests are redirected to the same physical IP address whatever is the remaining part of the URL or URI of the request. The WWW approach has allowed the federation into a unique namespace billions of web documents while preserving the high speed necessary when an applications talks to one well identified server.

In a large storage services handling billions of files, the problem is very similar: the name server lookup time dictates the performance of the whole storage system and the database becomes the bottleneck of the entire storage process: some data architectures designed for a maximum flexibility (from the data management point of view) turned out to suffer poor scalability and performance due to a constant name server overload.

7.3 Name servers in cloud storage

The name server architecture design is at the heart of the scalability issue in a large storage service. There are many technologies available and several optimization possible, in particular the use of local and distributed hash tables, used in cloud computing solutions. But also in cloud computing architecture, despite data location is calculated from the hash of its name or content, once reached a bucket subsequent requests within the buckets proceeds locally without requiring further lookups. In these scenarios, the use of hash table is an excellent solution for name resolution within a predictable time independent from the size of the storage service (thus allowing scalability). This point will be further discussed in chapters 10.5 and 10.6 when discussing performance issues.

7.4 Name servers to ensure consistency

And what is the role of the name server in ensuring consistency? If we take the conservative approach that every request requires a name server lookup then we can quickly agree that the name server is the component that best ensures consistency. But we know that this approach loses scalability, hence the usual trade between scalability and consistency. However, it is possible to obtain scalability improvements by accepting small reductions in temporary consistency.

The way to achieve this is to allow an arbitrary number of simultaneous (write) requests to be executed hoping that consistency is preserved (for example supposing that no two clients have modified simultaneously the same file from two different locations). The architecture must then provide an asynchronous or synchronous mechanism to detect eventual 'temporary' loss of consistency. When these rare event happens, they trigger a repetition of the request in a more conservative (and slow) mode that ensure consistency. Within this scenario, the vast majority of the request can be executed rapidly at the expenses of few requests that fails and must be repeated.

This approach is similar to ‘optimistic locking’ often used in databases to execute a large number of transaction that could not be executed in parallel if the necessary full locks would be applied to each transactions: instead of executing all transactions sequentially, the database attempt to execute them in parallel and then rollbacks and repeats the few that failed.

7.5 Hierarchical storage management versus manual or planned data placement

We mentioned before in this chapter that a name server gives the flexibility for transparent/automated data migrations. Data can be placed in different pools with different quality of service following some well-defined processes. Why different pools? The multiple motivations could be:

- Pools with different quality (performance/cost/reliability).
- Pools based on different media type (SSDs/disks/tapes).
- Pools located in independent data centres.

In a hierarchical storage management, the decision to place the data in a given pool can be complex and the algorithm can rely on several criteria:

- The frequency of past access requests to the data.
- The number of past simultaneous accesses to the data.
- The size of the data, as moving large amount of data has a higher cost but can save large amount of resources.
- Hints that past access patterns gives on probable future data requests.

Within this architecture with automated data movements across pools, is also important to define the efficiency which represent the fractions of the requests received that finds the data correctly pre-placed. This efficiency can easily swing from outstanding to disastrous as it depends on the ability of the automated data placement process to correctly predict future requests by analyzing the history of the past ones.

This value of the efficiency links directly to caching theories and various caching algorithms²⁸ that could be used. However which is the one to be used? We can’t answer this question as the various caching algorithms rely on generic access patterns that in a particular data analysis scenario can be correct or, sometimes, completely wrong.

For example, in the case of an high energy physics experiment that is processing measured data, a caching algorithm based on ‘Least Recently Used’²⁸ would have a disastrous efficiency because it would keep on high cost storage the data that has already been processed and should be discarded. On the other hand, a ‘Read Ahead’²⁸ algorithm would have a maximum efficiency.

This example shows that generic automated data placement solutions do not guarantee high efficiency and often must be tuned depending on the expected access patterns. And if the access pattern is specific to the application (and this is often the case in a scientific research) then a ‘custom development’ to improve the guessing of future data requests will always be less efficient compared

²⁸ For a non-exhaustive list and a description of various algorithms, see http://en.wikipedia.org/wiki/Cache_algorithms

to a 'custom development' done, within the scientific application, to automate the data pre-placement in the storage service across multiple pools with different quality of services.

This means that if the access pattern is known, this knowledge must be used *a priori* to correctly place the data (in advance) rather than relying on the service infrastructure to be smart enough to be able to guess what will be future requests and place the data accordingly. Why would it be interesting to lose computing resources to estimate something that is known? The second option (inferring the access pattern) will always be less efficient than a custom application that has been designed for a known access pattern.

This is why during recent years, there is been a strong move, in high energy physics, away from 'automated hierarchical storage management' solutions towards a 'managed data placement' approach where data flows across pools with different quality of service in an organized and consistent way. As a last example on this topic that affects the efficiency of a batch processing farm, it is always better to have the data pre-placed before batch job starts rather than starting the job and hoping that the data has been magically pre-placed, especially if you know in advance what is the data that the batch job will need to access.

7.6 Cloud Storage

Cloud storage represents the set of technologies that have been used in recent years to improve scalability issues in large storage environments even beyond the limitations of the name server approach discussed in chapter 7.3. As this technology has also a direct impact on performances it is discussed in chapters 10.5 and 10.6.

8. Security

While security is a general buzzword in computing, within data technologies it is the set of solutions that have been chosen to address the issues related to multiuser data access. In particular, the role of data management security is to:

- Ensure that every request that the storage service executes can be mapped to a particular identity. And that this identity can be traced to a physical or moral person (authentication).
- Ensure that the information exchanged between a client and the storage service cannot be eavesdropped or secretly taped by third parties (confidentiality).
- Ensure that the communication exchanged between a client and the storage service cannot be manipulated or tampered (integrity).
- After a client has been authenticated, ensure that the client is allowed to execute only a well-defined set of requests when reading, modifying, or deleting some data (authorization, quotas) depending on the identity or the request that has been verified.
- Ensure that in case of misuse, abuse or stolen identity the service has all necessary tools and information (auditing, journal of transactions) to define and execute procedures to handle the incident (investigation, response, communication, recovery).

Under this definition, the technologies used to implement security are at the core of the storage service architecture. And again, if we add security as an additional parameter to the 'quality of service', we immediately see that even in this case, high security may require a compromise with performance, scalability and costs.

8.1 Cryptographic tools and concepts

Security of data services makes heavy use of cryptography. Cryptography theory provides a wide set of tools and building blocks that allow security to be implemented. This article will not discuss how to implement security from scratch: on the opposite, to ensure a good security in data management it is essential to rely on standard frameworks that use the cryptographic algorithms in a way that is formally encapsulated into well-defined procedures like 'Public Key Infrastructure (PKI)'²⁹ or 'Kerberos'³⁰.

The purpose of this chapter is, to analyze the various processes implemented in the standard security frameworks and understand strengths, weaknesses and limitations of the various implementations when designing data services which include multiple security levels and options in their quality of service. This because many security features like group authorization, quota support, journaling, fine grain access control are functionalities that are not always necessary and that affect performances, scalability and cost and therefore it is important to understand what are the constraints that each functionality adds to the service.

Below, the following paragraphs will also not explain cryptographic algorithms in details. We will only list some concepts and make some comments around these concepts so the reader can get acquainted with the vocabulary that will be used in the following chapters.

Important note: Some statements require the reader to understand that many parameter and choices affects the security of the algorithm being used. It is very easy to build an infallible cryptographic framework which is theoretically perfect, but in practice all implementations require pragmatic choices of which the reader must be aware. Whenever these parameters or choices are mentioned, they will be highlighted and marked with an asterisk (*) indicating that there are consequences and choices involved that will not be discussed.

8.1.1 Symmetric encryption

Symmetric encryption (figure 14) is a transformation of a message (called 'cleartext') into another message (called 'ciphertext' using some input information (called a 'key'). If the same transformation is reapplied to the ciphertext using the same key, the cleartext is revealed.

The key has typically a smaller size (*) than both the cleartext and ciphertext and in theory it should not be possible to infer the key from the knowledge of some portions of cleartext and ciphertext³¹. With this premise the key is considered a 'secret' that unlocks a unique encryption-decryption algorithm and this secret must be shared among all parties requiring sharing of private information.

²⁹ http://en.wikipedia.org/wiki/Public-key_infrastructure

³⁰ [http://en.wikipedia.org/wiki/Kerberos_\(protocol\)](http://en.wikipedia.org/wiki/Kerberos_(protocol))

³¹ Symmetric encryption algorithms that do not satisfy this requirement are vulnerable to plaintext attacks (http://en.wikipedia.org/wiki/Known-plaintext_attack). This may or may not be a problem depending on the application the encryption is used for.

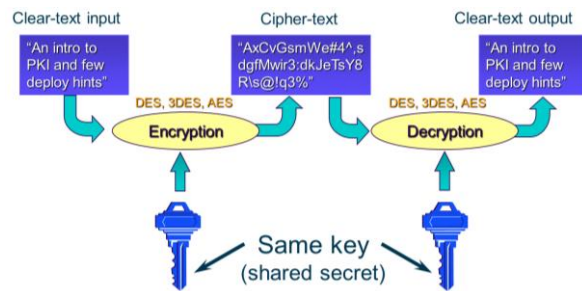


Figure 14: Symmetric encryption

A couple of comments:

- 'cracking' a symmetrically encrypted message means guessing the 'key'. However, cracking a single ciphertext may be easier as it requires a way to verify that a 'supposed' decryption is correct. Typically, this guess relies on supposed redundancy expected in the clear text and it could be useful to compress the cleartext before encryption as compression removes some redundancy.
- symmetric encryption algorithms are very fast. They can be executed in hardware and have, in software, a negligible cpu overhead.

In the various storage implementation used in high energy physics, symmetric encryption is used in nearly all components, as a part of the security frameworks described later not only to ensure confidentiality of data but also user authentication, access control and data integrity.

8.1.2 Asymmetric encryption

Asymmetric encryption (figure 15) looks somewhat similar to symmetric encryption with the difference that there are two different keys involved: one used to encrypt the cleartext the ciphertext and another used for decrypting the ciphertext back into the cleartext. In the various implementations one of the key is made public while the other is kept secret and used by only one identity.

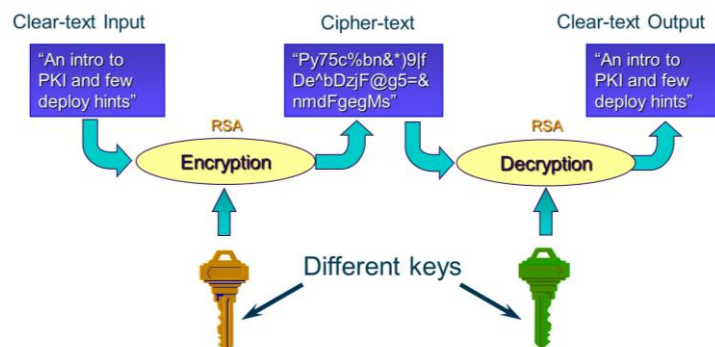


Figure 15: Asymmetric encryption

Despite the drawing seeming superficially similar to the symmetric encryption, the two algorithms are extremely different: Symmetric encryption algorithms are completely 'reversible' because reapplying the same identical algorithm allows you to go back. On the other hand, the entire security of an asymmetric encryption algorithm relies on the fact that the algorithm used is very difficult to reverse despite it is a completely defined mathematical problem given that the key used to encrypt is often known (public key).

In particular, the security of asymmetric encryption relies on the fact that it is:

- 1) Impossible to find the cleartext that corresponds to a cypher text despite having the knowledge of the key used in the encryption process which is often public (note that in symmetric encryption this problem does not exist as the knowledge of the secret encryption key is the base of its security).
- 2) Impossible to obtain knowledge of the decryption key despite having an unlimited amount of cleartext, the corresponding ciphertext, and knowledge of the key used in the encryption process. Again, a perfectly defined, but 'difficult', mathematical problem

One can see that decryption relies on 'reversing' the algorithm used to encrypt. This should be 'computationally infeasible' unless we have a 'hint' that allows us to reverse the algorithm: and this hint is represented by the second key which allows to solve the problem and reverse the algorithm to go back to the cleartext. As an example, the answer to the question 'what are the two numbers that multiplied together give as result 5 549 139 029 240 772 017 554 613 865 259 030 307 060 771 696 148 489?' is a well-defined mathematical problem with only one solution that is difficult to solve without a computer. However, if I give a hint, saying that one of the two numbers is 2 833 419 889 721 787 128 217 599, finding the other number becomes trivial.

Even here, a couple of comments:

- The ideal asymmetric encryption algorithm (which does not exist) should allow the two keys to be interchangeable. However, in reality, reversing the algorithm can be easier with one key and harder with another. This is why when generating a key pair the various implementations name one of the key 'public' and the other 'private' implying that it is safer to expose ciphertext encrypted with the first key rather than with the second (*).
- As mentioned the security relies on the difference between the computational effort between the forward (encryption) and backward (decryption) algorithm when only the knowledge of one key is available. As this is always a 'relative' difference, the more 'computationally intensive' is the forward algorithm, the harder will be the backward decryption (without the second key). This is why, asymmetric encryption is always cpu intensive and encryption is orders of magnitude slower compared to symmetric encryption. Typically this is an area where key length (*) plays a major role in trading between performance and security.
- As cracking asymmetric encryption is a well-defined mathematical problem once you have some ciphertext and the corresponding cleartext, the more your private key is used, the more you expose its value. This explains why the various security frameworks define keys hierarchies and keys expiration rules to ensure that highly exposed keys³² have a short lifetime and *vice versa*³³.

³² As an example, the proxy certificate in a Public Key Infrastructure or the session key in Kerberos environment have lifetime limited to few hours as they are intensively used in all communications during the session.

³³ The opposite example, the root certificate of a certification authority is used only once every few years to sign the signing certificate and can have a lifetime of a century. In Kerberos, the Master key used to obtain a ticket granting ticket is also rarely used and has a lifetime that can be of several years.

The most common example of protocols using asymmetric encryption are the TSL/SSL³⁴ ones. To be precise, the TSL/SSL protocols use the asymmetric encryption to negotiate a shared secret that will then be used symmetrically, and therefore are also examples of an hybrid encryption usage described in the next section.

As for symmetric encryption, high energy physics implementations makes heavy use of asymmetric encryption in authentication, in establishing secure communication channels and in guaranteeing data authenticity and integrity. These usage are described in the following sections of this chapter.

8.1.3 Hybrid encryption

Given the performance overhead, asymmetric encryption may not be suitable for encrypting large amount of data. When this is necessary, hybrid encryption typically gives the advantages of both worlds: the speed of symmetric encryption and the flexibility of the asymmetric one.

With hybrid encryption (figure 16), the data (supposedly large) is encrypted symmetrically using a 'randomly generated key'. The 'randomly generated key' is then encrypted asymmetrically with the public keys of each recipient needing access to the data.

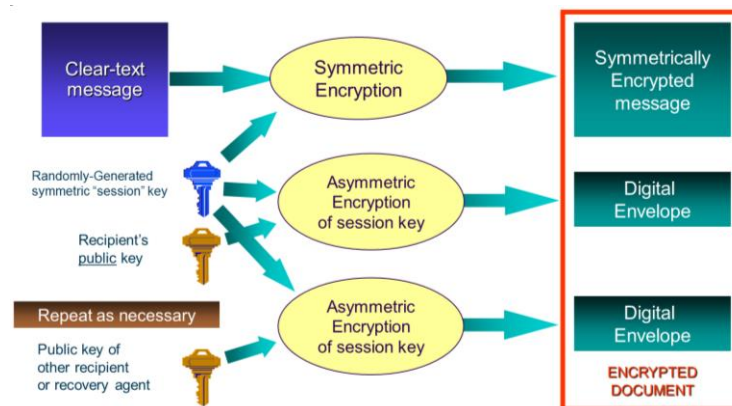


Figure 16: Hybrid encryption

This has the advantage that:

- The encryption process for the large amount of data is faster as it is symmetric.
- Only the 'randomly generated key' is encrypted multiple times, one for each recipient. This saves lot of storage compared to a solution where all the data would have to be encrypted multiple times, one for each potential recipient.

Hybrid encryption is also heavily used in storage element in high energy physics because it is the most used technology to ensure secure communication channels between any two components of a complex storage service (as for the SSL and TLS protocols). In addition, hybrid encryption remains the golden standard for an effective storage encryption, especially when dealing with large amount of data.

³⁴ See specifications and details at http://en.wikipedia.org/wiki/Transport_Layer_Security

8.1.4 Key generation

The process of generating keys in an unpredictable random way (*) is essential in all types of encryption and in contradiction with computers and algorithms that are entirely deterministic. This point will not be discussed further but must be reminded that it is an important point at the core of the security of the storage service and an additional area where performance and security can be traded.

8.1.5 Hash functions

All security frameworks make heavy usage of cryptographic hash functions. A Hash function is a transformation of some arbitrary amount of data that returns a fixed-size short representation (called 'hash', 'digest', 'fingerprint') of the original data that has the following properties:

- It is efficiently computable (hashing speed similar to symmetric encryption)
- Any (small) modification in the data generates a modification in the hash
- It should be computationally impossible to 1) calculate some data that matches a given hash 2) find 'collisions', wherein two different amount of data have the same hash.

One comment:

- Hash collisions exist by construction, so the security relies on the inability to find them. However, collisions happen. When the usage of data hashes is intensive, like in cloud storage which used the key to identify the data, it is essential that collisions are foreseen and correctly handled by the data management software. Overlooking the rare event of collisions in a large data management environment is a guaranteed strategy for data loss and corruption.
- Hashes are an essential tool in designing storage services to ensure data integrity and implement data comparisons, data migration, deduplication, syncing, caching algorithms, associative memories, name servers, cloud storage and many other data management components.

Hashing functions are everywhere in storage element implementation in high energy physics: they are used to guarantee integrity of the data, integrity of the software and, most important overall integrity of complex processes requiring transactions and data manipulation. For example, the upcoming 'digital signature' section of this chapter is a significant example of hash function usage that is generally used to guarantee integrity and authenticity.

8.1.6 Digital signatures

An hash represents the data from which it is calculated. As we believe that data cannot be manipulated to meet a given hash, if the data owner encrypts the hash with his own private key, then this information (figure 17), when associated with the data itself, can be used as an evidence that:

- The owner who encrypted the data hash with his own private key has somewhat endorsed (a.k.a. signed) the data content.
- The data has not been modified since this endorsement (otherwise the hash would no longer be matching the hash of the data).

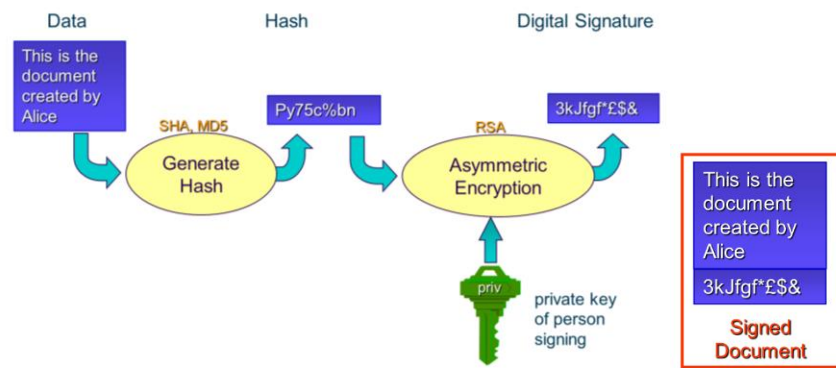


Figure 17: A digitally signed document contains an encrypted had of the content

To verify a digitally signed document, we must extract the digital signature, decrypt it using the public key of the recipient and compare it with the hash of the data that we need to recalculate. If we have a match, then we have evidence (*) that the signature was generated using the private key that corresponds to the public key that we have used in the decryption. And depending on the trust we have on the public key³⁵ belonging to the person pretending having signed the document we can conclude (or deny) that he was at the origin of that signed document.

Digital signatures offer plenty of opportunities for ensuring additional level of security in any computing environment, in particular in storage services. Remember that a digitally signed document is not encrypted and remains public readable. It simply has additional information that guarantees the origin and the integrity of its content.

8.2 Authentication

Authentication is the functionality that allows a storage services (or any computing service in general) to map all requests received to a known identity.

In the trade between security and performance, one can imagine the scenario of a storage service optimized for performance which is accessed only by trusted clients and the storage infrastructure always execute the requests without questioning the origin of the request. This would be similar to a public web server that executes any unauthenticated requests.

However, despite an authentication service is not strictly mandatory, when the service handles multiple users and data can be modified, ensuring that every request received is mapped to a known identity becomes a strong requirement.

In high energy physics the availability of authentication services for data access is a strong requirement. CERN has deployed a complex identity management infrastructure for more than 40,000 scientists worldwide. This service 'physically identifies' every person and map her/him to an electronic credential (account or certificate, as described later) that he can use to access the LHC storage infrastructure. The 'physical identification' is a complex process that relies either in verifying identity documents (typically passports) of end-users or, more often, to establish trust relationship with other identity management services to be able to trust identities that have already been

³⁵ Having an infrastructure that allows us to trust public keys and map them to the corresponding owners is the main goal of a 'public Key Infrastructure' (PKI) described later.

verified by a national identity management infrastructure or by an existing human resource database (*)³⁶.

To give an idea of the volume of ongoing authentication activity, the CERN authentication infrastructure processes an average of 28,000 authentications per day. Nearly all these authentications are motivated to fulfill remote requests that requires some sort of access to data stored in the CERN infrastructure.

8.2.1 Username and password authentication

There are several techniques which allow authentication of a remote client based on a shared secret (the 'password') that has been previously agreed by the two parties. An example is the web basic authentication where both username and password are sent on every request to the server.

An improvement over always sending username and password, are protocols (for example the http 'digest'³⁷ authentication) where the knowledge of the password is never transmitted over the network. This prevents third parties listening to the entire conversation and guess its value. As a simplified explanation, the client willing to authenticate sends a hash of his username, password, and some random data unique to this session called a 'nonce' that he has previously received from the servers, and sometimes also adds a 'cnonce (client nonce)'. The 'nonce' ensures that old communications cannot be reused in replay attacks. The 'cnonce' to allow the server to recognize future requests from the same clients.

The servers is then able to recalculate the hash and verify the correctness of the username and password that the client has supplied.

There are many implementations available, each of them with advantages and disadvantages (typical trade between performances and security) some of which (*) have some security concerns as they are vulnerable to man in the middle attack (like the mentioned example) or require the server to be able to store the password in clear text or reversible encryption.

In high energy physics and at CERN in particular the username/password authentication is used for all web-based accesses. Unfortunately, many implementations still uses the 'basic' method where the password is transmitted from the client to the server and protected (only) by the secure http channel. This technique is heavily used in average for 20,000 authentications/day.

8.2.2 Public key infrastructure and authentication using certificates

When an entity owns a pair of public/private keys, authentication becomes possible by challenging the entity on the knowledge of his private key within the pair. With this approach, the main problem is the practical distribution of public keys which is solved by the use of certificates. The certificate is

³⁶ Identity management in high energy physics is a complex process which federates identities from national infrastructures and international laboratories. This activity takes places under the coordination of the International Grid Trust Federation (ITGF) and the list of all participating countries and institutes can be seen at <http://www.gridpma.org/>. An example of the European activity in this field can be discovered by visiting the www.eugridpma.org site

³⁷ See RFC 2617 on Basic and Digest Access Authentication: <http://tools.ietf.org/html/rfc2617>

a document which associates a public key with an identity which is signed by a third party (a certification authority – CA).

All actors (including the storage service provider) trust the certification authority to have done the correct identity verifications before signing a certificate, and therefore they can use the certificates as a reliable source to obtain any public key corresponding to any entity.

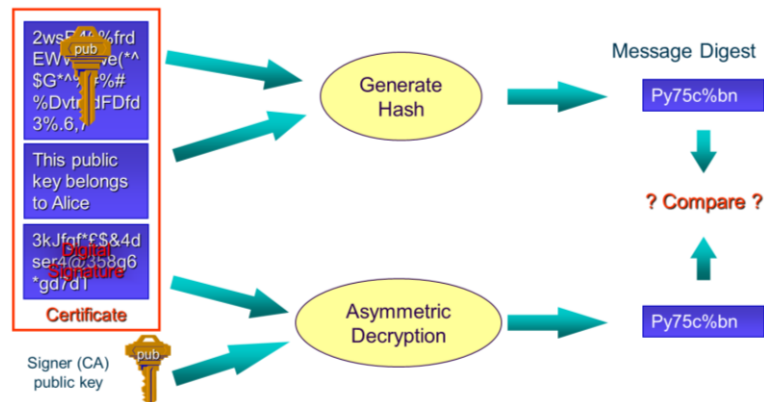


Figure 18: Certificate verification

As the certificate is digitally signed, there is no need to involve the certification authority in the authentication process as the certificate is signed. The certificate is presented by the client willing to authenticate to the servers who can verify the digital signature on the certificate (using the public key of the certification authority) and validate the certificate content which contains the association between the digital identity and the associated public key (figure 18).

After the certificate has been verified, the server can send to the client a random and unique ‘nonce’ and expect the client to return it encrypted using his private key. The unique ‘nonce’ should be interpreted as a challenge to the client on the knowledge of the private key that pairs with the public key the server has found in the certificate. The client response is decrypted by the servers using the client public key (of the certificate) expecting to find back the initial nonce (figure 19).



Figure 19: Authentication with certificate (simplified)

This is the simplified process. In reality there are several additional steps that can be necessary like verifying the certificate of the certification authority (CA) to obtain its public key that is necessary to verify the client’s certificate. Or to verify that the client certificate has not been revoked by the certification authority.

As the reader can imagine, these steps may require additional network roundtrip and some of them are highly non scalable, as downloading the entire certificate revocation lists from the certification authority at every authentication requests. For this reason, there are several options to speed up the authentication (and therefore increase the performance of the storage service) by skipping some part of the authentication protocol (*).

All storage services used in high energy physics supports PKI and certificate-based authentication, mainly to handle wide-area requests coming from outside the local campus intranet. The infrastructure at CERN handles an average of 4,000 certificate-based authentications/day and recognizes certificates issued by a large number of certification authorities³⁸. CERN also has its own certification authority³⁹ and issues its own certificates: for example the number of active certificates issued by CERN is nearly 7,000 for user authentication and more than 47,000 for host (i.e. computer or server) authentication.

8.2.3 Authentication using Kerberos Tickets

An alternative to using certificates with public/private key exchange is the usage of the Kerberos authentication protocol which represents a security framework to share and distribute shared secrets (symmetric encryption). In this infrastructure, the trusted authority is called 'Key Distribution Center (KDC)' with whom every known identity shares a unique secret (Master Key). The master key is different for each identity and allows them to communicate securely with the KDC.

As two users cannot verify each other's identity directly, the role of the KDC is to distribute a unique session key (shared secret) to each pair of users that mutually want to establish a secure communication channel.

When a client wants to authenticate with a server⁴⁰ he asks for a unique session key to be used for the communication between the client and the servers. The KDC returns to the client two copies of the shared session key:

- The first copy is sent back encrypted using the client master key. The client can decrypt it and obtain the session key to be subsequently used in communicating with the server
- The second copy is sent back with in the client name in a document that is encrypted with the master key of the server. This is called the 'Kerberos ticket'. As this ticket is encrypted with the server master key, it cannot be decrypted by the client.

³⁸ The list of accredited certification authorities who issue certificates recognized for authentication at CERN and within the Worldwide Computing Grid (WLCG) project is available at <http://www.eugridpma.org/members/membership> (for Europe), at <http://www.apgridpma.org/members.html> (for the Asia-Pacific region) and at <http://www.tagpma.org/members> (for North and South America).

³⁹ The CERN certification authority is online at <http://cern.ch/ca>. Several statistics on its usage are at <https://ca.cern.ch/ca/statistics/Statistics.aspx>

⁴⁰ The usage of the words 'client' and 'server' is used only to differentiate two identities willing to authenticate each other.

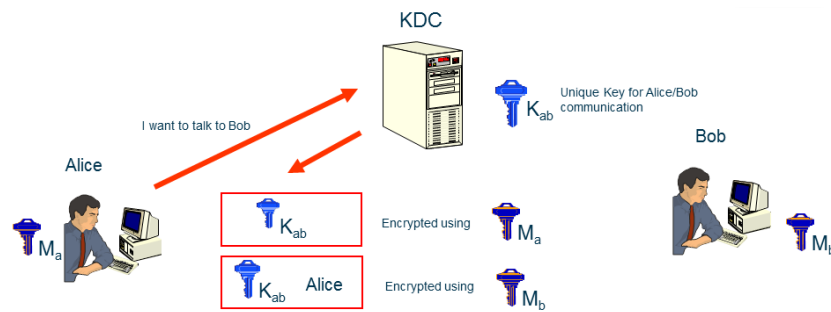


Figure 20: Obtaining the session key and the ticket in a Kerberos environment (simplified)

- After the client has obtained the shared key and the ticket, he attempts the authentication with the server.
- The client sends the Kerberos ticket to the server together with an 'authenticator' which is the client name and the current time encrypted using the session key (figure 20).
- The server, receives the ticket that he decrypts to obtain knowledge of the session key (he is the only entity able to do this as he knows the server master key).
- Using the session key, the server can decrypt the authenticator. He verifies that the time used in the authenticator has never been used in the past⁴¹ (this is like having a unique challenge on the knowledge of the session key). This authenticates the client with the server because it is an evidence that who created the authenticator had knowledge of the shared key.
- For a mutual authentication, the server takes the time found in the authenticator, encrypts it using the session key and returns it to the client. The client decrypts it and compares it with the time he used and if they match he has an evidence that the server was able to decrypt the authenticator to obtain the time and therefore was able to obtain the session key by decrypting the ticket using the master key.



Figure 21: Example of a (simplified) mutual Kerberos authentication

This simplified description of how Kerberos authentication works (figure 21), triggers some comments that also allow the comparison between a PKI and a Kerberos infrastructure:

⁴¹ To avoid replay attempts and to ensure the uniqueness of the time challenge, the server should keep the history of all time requests used in the past (the history is called replay cache). To reduce the size of the replay cache, the server keeps only the history of requests received within a fixed time interval and compares the time in the authenticator with his present time. If the time is within the time interval and not in the replay cache, the authentication is validated. If the time used is in the replay cache the authentication is rejected. And if the time found in the authenticator is outside the time interval for which the replay cache is kept, the authentication is rejected with the hint of what is the server time so that the client can re-issue an authentication attempt with a synchronized time.

- All encryptions using Kerberos are symmetric and this gives a performance advantage over a PKI authentication, in particular all peer-to-peer communication between mutually authenticated identities can be done in a confidential and encrypted channel based on symmetric encryption using the shared session key. However, when using PKI, the problem can be mitigated by extending the PKI authentication with the negotiation of a shared secret to be then used for encrypting the communication channel symmetrically. For example, the SSL⁴² protocol widely used in the secure HTTP protocol (HTTPS)⁴³ uses a certificate-based authentication that is then used for negotiating a symmetrically encrypted communication channel.
- The Kerberos authentication requires the KDC to be online while the PKI authentication can be done with the certification authority being offline or unavailable⁴⁴. However, the Kerberos protocol allows several workarounds to cope with situations when the KDC is unavailable by allowing reusing older session keys between clients that have mutually authenticated in the past. The typical example of this feature is the 'logon with cached credentials' warning that is displayed in Microsoft Windows domains when users logon in disconnected mode or when the domain controller (the KDC in Microsoft language) is unavailable.
- As all keys are managed by the KDC, the implementation of a forwardable authentication is straightforward in a kerberized environment compared to a PKI one. Forwardable authentication is typically required in 3-tier applications when a client connects to a web server and the web server then executes a query in a database on behalf of the client. With 'forwardable authentication' the web server is able to impersonate the identity of the client when executing the database request, while in the scenario where 'forwardable authentication' is not possible the web server can only execute the request on behalf of the client with full privileges on the server, exposing some security risks if the web server application is compromised. Even in this case, while forwardable authentication is straightforward with Kerberos, there are workarounds with PKI to implement similar functionalities.
- In a Kerberos environment, identity management is centralized in a unique KDC while PKI allows native federation of identities from multiple and independent certification authorities. In both scenarios there are advantages and disadvantages (*) and even in this case, Kerberos offers some workaround to establish trusts between multiple Kerberos domain but this brings some extra complexity to the end-users who must understand that credentials are no longer limited to username/password pair but also the domain name becomes relevant. And concerning PKI, there is nothing that prevents an infrastructure to

⁴² See the specifications of SSL at <http://tools.ietf.org/html/rfc6101>

⁴³ See http://en.wikipedia.org/wiki/HTTP_Secure

⁴⁴ Strictly speaking, this is incorrect as a formal PKI authentication also requires a verification of the Certificate Revocation List (CRL) that must be downloaded in real time from the certification authority. In practice, in the trade between performance and security (*) this step is often either skipped or a cached copy of the CRL is used, limiting the download of the CRL to predefined intervals. To increase the scalability, the CRL can also be distributed through a cluster of web servers so that the PKI authentication to proceed event with the CA being unavailable.

accept certificates issued only by a unique certification authority which then allows a centralized identity management.

The storage services at CERN also support Kerberos-based authentication mainly to handle requests coming from inside the local campus intranet. CERN has been one of the early adopter of Kerberos when already in 1990 the 'Andrew File System (AFS)' was deployed. Now the technology has evolved and Kerberos is the base of many authentication used to access storage service. In practice, as described in the next section on 'mixing multiple authentication methods', even when the user is authenticated using a certificate, its identity is mapped to a Kerberos one.

In the specific CERN case, the Kerberos infrastructure handles an average of 4,000 Kerberos-based authentications/day from a pool of nearly 50,000 accounts registered⁴⁵.

8.2.4 Mixing multiple authentication methods

As the reader can imagine, the characteristics of the Kerberos approach (centralized identity management, online KDC, forwardable authentication, high performance) are typically well suited for a centrally managed infrastructure deployed in a unique location (typically the local area network of a research institute or university), while the PKI advantages (distributed identity management, offline authentication, highly scalable to large number of potential users) ensure excellent interoperability in distributed environments, especially when deployed over wide areas under independent management.

Unfortunately, as we do not live in a perfect world, the two approaches end up becoming mixed together with even the third approach of username/password authentication used by several web portals. In these scenarios, a solution is to deploy a 'single sign-on' infrastructure that allows to cross the boundaries seamlessly and to be authenticated on both sides whichever type of authentication was used.

For example, it is straightforward to have a Kerberos-based service that allows end users to connect either using Kerberos credential or using a PKI certificated that upon authentication is automatically mapped to a Kerberos credential.

However, while mixing multiple authentication protocol is possible and seems appealing, it introduces additional complexity:

- The infrastructure becomes more difficult to understand because the user is exposed to two authentication methods: username/password and certificate, and they have to be both managed. This adds to the complexity of local security policies that forces periodic password changes and certificate expirations.
- There are several scenarios where the security is weakened by the master-slave setup of the two authentication mechanisms: for example, it may be possible to reset the Kerberos password using the certificate or, vice versa to obtain a new certificate using the Kerberos based credentials.

⁴⁵ The CERN authentication service had 48,884 accounts on beginning of October 2013 and an average of 25 new accounts are created every day. The service is located at <http://cern.ch/account>.

- The way that the 'single sign on' is implemented is based on the trust that all users have in the KDC and/or the CA and relies on the KDC or the CA to 'leak' some secrets to the other environment considering it part of the trusted environment. This somewhat violates the definition of the original authentication protocol.
- When the mix of multiple authentications is motivated by the will to integrate new identities that have been certified by an external CA, it becomes necessary to create a Kerberos account for each certificate that will be used (each identity authenticated with a certificate must be mapped to a Kerberos account). Some simplifications to this issue are possible, for example doing a many-certificates-to-one-Kerberos-account mapping, but beware that this may add even more complexity as the identity verification can no longer be reversed, and therefore the system may lose the primary role of authentication on the 'Kerberos' side.
- In the scenario of multiple authentications with a one-to-one certificates-to-Kerberos identity mapping, the required Kerberos accounts can be created 'on-the-fly' whenever a new certificate is used or they can be 'pre-created' in advance (this requires user registration). In both cases it is essential not to underestimate the amount of development and additional identity management procedures to ensure that the two accounts database remains correctly synchronized. The identity management process must ensure that when a certificate is rekeyed or renewed that the new certificate is mapped to the same Kerberos account it was mapped before; and when the subject in a certificate change (people do change name) to ensure that the identity is preserved. In practice there are so many opportunities for problems that this should never be considered a 'zero-cost' enhancement.

In the specific implementation used at CERN, figure 22 shows how the multiple authentication method can be seamlessly integrated. The interface shows that the user can type the credential or use an existing Kerberos ticket or a PKI certificate to authenticate.

The screenshot shows a login window titled "Sign in with your CERN account". It has two main sections: "Use credentials" and "Use one-click authentication".

Use credentials

Username or Email address: Password:

☐ Remember Username or Email Address [Need password help ?](#)

Use one-click authentication

 [Sign in using your current Windows/Kerberos credentials \[autologon\]](#)
 Use your current authentication token. You need Internet Explorer on CERN Windows or Firefox on SLC (Firefox help here).

 [Sign in using your Certificate \[autologon\]](#)
 Use a EuGridPMA trusted certificate. Don't forget to first map your Certificate to your CERN Account.

Figure 22: Example the interface at CERN which offers the choice between Basic/SSL, Kerberos or Certificate-based authentication

8.3 Identity federations

When a storage services requires authentication of generic users from the internet, there are two opposite approaches in account management that are possible:

- The first supposes that every user must obtain a set of credential in your infrastructure (either a Kerberos account, a Certificate issued by your own certificate authority, an email address with a password, or whatever authentication system has been setup). In this case the service must foresee a registration portal which offers all proper tools to manage the

account. This first option has the advantage that the identity management process is entirely under the full control of the service.

- The second relies in either accepting certificates issued by other (trusted) certification authorities or to accept Kerberos authentications from account attached to other domains with which the service has established a trust relationship. This has the advantage that users do not have to register again and can directly use their existing credentials.

Both approaches are perfectly valid and safe. However, it is essential that the two approaches are not mixed together by allowing end users a free choice between a local account or an external account as this will give the users the possibility of having two independent identities. Multiple identities for the same user becomes a problem in the case (typical of a storage service) because it makes impossible to map resource ownership information or permissions to a person in the existence of multiple accounts that can be difficult to differentiate.

8.4 Identifying users

Independently, from the type of authentication method chosen, it is also important to understand how the users are identified and referred within the storage infrastructure.

One option is to identify them by the 'login name' (or 'electronic mail') string of the account or by the 'subject' string found in the certificate which is the unique 'Distinguished Name (DN)' of the users.

This approach is simple and easy to understand as there is no translation necessary between what the infrastructure uses and the human readable unique identifier of a user. However, there is the drawback that changing the 'login name' or the 'email' or the 'subject' in the certificate means changing the identity with the consequence that the user loses all permissions, quotas and even ownership of his own data.

As an alternative, which is used by nearly all security implementations, users should be identified by a unique globally unique identifier (GUID)⁴⁶ or Virtual ID. The login name or the certificate 'subject' becomes only an attribute of the account named after the GUID (figure 23).

This approach gives the flexibility of renaming accounts and allow users to change their certificate subject (for example they change the issuing certification authority or they change personal name) while preserving their identity and permissions. In addition, this flexibility comes with a performance gain as it is much faster, to parse GUID (it can be treated as a 128-bit integer) in access control lists (See 8.7) to resolve permissions compared to parsing variable-length text string.

It is important to remember that the GUID associated to an account is specific to a site or even specific to a particular instance of a storage service. This means that authorization information (access control list in particular) need to be recalculated with the correct GUID when the data is moved from one site to another or between different instances whenever the accounts used do not share an identical GUID mapping.

⁴⁶ See http://en.wikipedia.org/wiki/Globally_unique_identifier

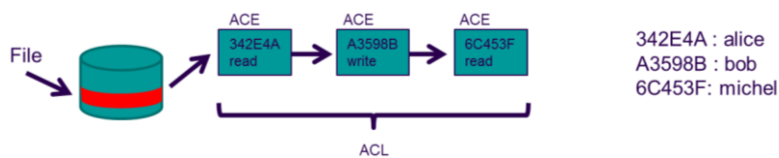


Figure 23: Access control lists containing only GUID mapped to subjects in a separate table

8.5 Integrity

Despite this topic having already been partially discussed in the case of data synchronization (See 5.9), integrity of the data itself is also an essential requirement of any storage service. It measures the ability to ensure that for some reasons the data stored has not involuntarily changed due to hardware failure, software bugs, human error, or during an I/O operation like reading, writing, moving, copying.

Integrity is part of the service reliability requirement and all the techniques used to ensure high reliability are available to guarantee data integrity.

In data management, loss of data integrity can be the worst nightmare. When this happens, the most important requirement is to know exactly where, how and which data has been affected. The golden technique to guarantee this awareness is to calculate checksum or hash information for every piece of data that is under custody.

Two important comments:

- When used for verifying integrity, some of the cryptographic properties of the ideal hash function are less relevant: we no longer require to be impossible to find 'collisions' (two different files generating the same hash) or to calculate some data that matches a given hash. Therefore, the choice must favor computational performance and this is why simple checksums instead of cryptographic hashes can be used to verify the integrity.
- Multiple levels of checksums are often useful and sometimes even necessary. In particular, if a file is stored in chunks, it is necessary to have a checksum for every chunk as well as a checksum for every file. Similarly, for a directory, it is important to have the checksum of every file as well as the checksum of the entire directory content, including subdirectories. If the requirement of having multiple checksum levels is important, then the choice of a checksum algorithm that allows incremental calculations and calculation of the checksum of concatenated data from the sub-checksum of the components rather than from the concatenated data becomes essential⁴⁷.

Why are multiple levels of checksums necessary? If this information is available in the metadata, this allows syncing application to be extremely effective as they do not need to scan for all metadata (or even data content) to know which files have been modified.

Integrity is taken very seriously by the two CERN storage implementations (EOS and CASTOR). Checksum are calculated and recorded for every file and any subsequent partitioning of the file (in

⁴⁷ For example, the CRC algorithm has this property: If a message M is a concatenation of messages M1 and M2, with the CRCs of M1, M2 (computed with some initial values v1,v2 respectively) are known, the CRC of M can be computed without reading the contents of M. See <http://crcutil.googlecode.com/files/crc-doc.1.0.pdf>

case it is cut in chunks or in multiple tape segments) and recalculated at every data movement and at every verification.

8.6 Confidentiality and Encryption

Encryption is the golden standard to ensure confidentiality. However it is important to differentiate between ensuring the confidentiality of the data transfer and communication across the network from the confidentiality, protection and encryption of the data itself on the storage service.

Confidentiality of the data transfer and communication across the network is easily inherited from the security framework being used. For example, if the data is exchanged using the secure HTTP protocol (HTTPS) it is protected from eavesdrop and tampering. Similarly in Kerberos there is a unique session key between any two identities that allows secure and confidential communication and when using PKI, encrypting an information with the public key of the recipient ensures that he is the only identity who can access it.

The problem becomes more important when addressing the confidentiality, protection and encryption of the data itself in the storage service. The first two options differentiate between data stored in clear or data stored encrypted.

If the data is stored in cleartext, confidentiality relies on the authorization and the access control architecture of the storage service. If the data is encrypted, the protection relies also on the knowledge of the decryption keys and can be completely independent from the access control architecture of the storage service or managed by it. This gives three scenarios:

- Data in clear, data accessible by the service management, security ensured by the access control of the storage service.
- Data encrypted with keys managed by the storage service. The data accessible by the service management (allowing data recovery when keys are lost), security ensured by the storage service who distributes and manages appropriate decryption keys to authorized identities.
- Data encrypted with keys managed by the data owner. Only the encrypted data is accessible by the service management (this still allows backups and replication without knowledge of the data), and the security ensured by the storage owner who distributes and manages appropriate decryption keys. In this scenario, the data cannot be accessed nor recovered by the storage service management if the end-user loses knowledge of the decryption keys.

In the two scenarios when data is stored encrypted, the implementation requires the usage of hybrid encryption (See 8.1.3) as it allows the encryption of the data only once with a unique decryption key which is then encrypted multiple times, one for each identity requiring access. The inclusion or the exclusion in the list of decryption identities of the 'recovery agent' identity of the storage service management is an optional but important decision when encryption storage solutions: This significantly affects the security of the data because it allows the data owners to position themselves on the 'highly secure' extreme, compared to a 'less secure but recoverable' option.

In high energy physics there are no strong requirements for high confidentiality of data. For this reason the implementations at CERN do not encrypt the data when storing it. Within this scenario the data protection is ensured only by the authorization process of the storage service described in the next section.

8.7 Authorization

Once the authentication component has been implemented, the authorization process implements the access control which has the responsibility to decide to execute or reject every requests by verifying if the remote user has the permission to access a given resource or execute the request.

The information describing what end-users can do on the storage resources are represented by the association of a right (use, read, modify, delete, open, execute, etc.), a subject (person, account, computer, group, etc.) and a resource (file, computer, printer, room, information system, etc.). In addition, the association can also be time-dependent: either it is valid only during particular time intervals (example: working hours only) or it could be time-limited (expiration).

This association is recorded in a data structure known as access control lists (ACL) managed by the data owner (figure 24). The ACL, which can be considered as a part of the metadata, can be stored with the data or within the metadata of the name server. From the functional point of view, the three options are identical, but performance can be different because affects the performance of the access control verification: if the ACL is part of the metadata, the performance of the name server will be reduced, but ACL verification will be faster. On the other hand, the ACL can be stored with the data when the service wants to boost performance of unprotected read requests which do not have an ACL and gain in scalability of the name server.

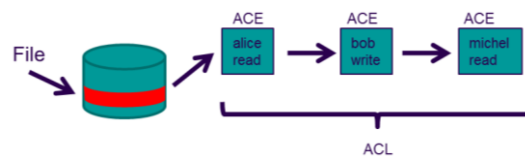


Figure 24: An ACL example (where GUID have been resolved to usernames)

Another option in implementing authorization is to consider ‘delegating’ this task to the operating system: the ACL are set on the file system and the process accessing the file impersonates the credential of the user. The drawback of this option is that a local account must be created for every potential user connecting to the system. If this option is possible, then it is likely to offer an excellent performance compared to the alternative option where the authorization is managed within the application: The permissions of the request owner must be verified by the data management software on every request.

The ACL is a linked list composed by a number of ‘Access Control Entries’ (ACEs) that enumerate permissions of identities. When designing a storage system it is important to remember that the ACL must be parsed sequentially when calculating effective permissions and this has can have an increased access latency cost for data that has been discretionally protected.

It is within the ACE that is written the information representing the ‘subject’ (the identity to which the permission is granted) and the ‘right’ (read, modify, delete, open, execute, ...) that the subject has on the ‘resource’. Eventually also the ‘time’ during which the ACE is valid.

8.7.1 Access control entries (ACE) rights

The type of rights that can be set in the access control entries can typically be of two types: ‘Allow’ and ‘Deny’ (figure 25). And for both types there are literally dozens of very high granularity rights that can be thought of. Note that the more granular is the access control system, the more flexible

and specific the access control can be. However, this increased flexibility has an important drawback: if the security model is too complex, the opportunities for misconfigurations and security incidents increase.

For example, in an environment with 'Allow' and 'Deny' ACLs that supports assigning permissions to groups, it is possible to define contradictory results and to resolve these conflicts sometimes the order of the ACE within the ACL becomes important as it relies on the order in which ACEs are processed.

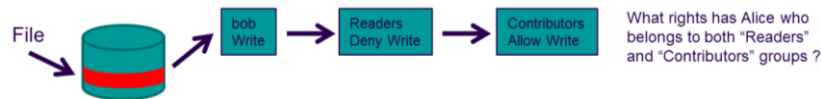


Figure 25: An ACL referencing groups with both 'Allow' and 'Deny' ACEs

Among the possible simplifications here are some examples:

- Consider a security model where only 'Allow' permissions are supported, starting from a well-defined default permission.
- Consider of supporting ACLs only on directories rather than on individual files.
- Consider of limiting the possible right list to a minimum

As an example of possible list of rights, we have:

- 'Read', which grants the permission to read/download a document when applied to files. When applied to directories, they set the default read permission to be inherited by files without ACLs.
- 'List' which grants the permission to read/download the metadata of a document when applied to files or directories. 'List' also grants the permission to obtain the directory content which is also considered metadata information.
- 'Write' , which grants the permission to write / modify / upload / delete a document when applied to files. When applied to directories, it set the default read permission to be inherited by files without ACLs. Note that many implementations prefers to split this permission between 'Modify' and 'Delete'.
- 'Change Permissions', which grants the permission to create / modify / delete the ACL of a files or directory.

8.7.2 Default permissions and permission inheritance

It is important to define what is the default permission of a document without ACL or with an empty ACL. Note that some implementation can even make a difference between these two cases by considering public a file without ACL and protected a file with an empty ACL.

A common possibility, in the absence of an inherited permission, is to define the default permissions to something like 'deny-all, except owner', meaning that any new document without permissions inherited will be accessible only by its owner. This default approach can be combined with an 'allow-only' rights model.

Defining individual permissions on each individual file can be time consuming, thus permission inheritance should be supported on every node (folders) of the file structure. Inheritance flags and

binary masks must be foreseen in case the requirement is to control which rights are propagated from a parent folder to subfolders and files.

If the storage service requires permission inheritance, we face another opportunity to trade performance between read and write operations as we have the option to:

- Calculate the 'resultant permissions' in real time when the file is accessed by reading all the parent permissions up to the root of the file system. This option is possible but requires low-level code optimization, otherwise the storage service will suffer from an heavy performance hit when accessing (reading) files stored deep in the folder structure. On the other hand, writing or modifying a permission is very efficient as it is done locally on the file or directory.
- Compile the 'resultant permissions' (file permissions + the inherited permissions) within the file. This approach can be very inefficient when changing permissions that affect a large number of files (as all their ACLs need to be modified) but gives the maximum possible speed when reading the files as resolving permissions requires only one lookup.

In practice, this forced choice between optimizing read or write speed can be mitigated by implementing a mix of the two approaches, which consists of compiling the resultant permissions asynchronously and being able to invalidate them in large number when the permissions are changed. This gives a storage service that normally is optimized for read (permissions are pre-compiled) and also for write (permissions are changed only within the resource and compiled asynchronously). The drawback is when, after a permission change that has an impact on a large number of files, a read request is executed before the permission have been recompiled. In this case, good software would revert to 'slow mode' by traversing the permissions of the entire path, but this inefficiency should be statistically confined to a very small number of rare access requests.

Whatever method is chosen for the implementation of the permissions inheritance, the model for the end user should remain simple to explain. As an example, this could be: if the ACL on a file/directory is missing or empty, the parent path is escalated until a valid ACL is found. When an ACL is found, the subsequent parent ACLs are ignored. If no ACLs are found after escalating to the root, the default permissions are used.

As a final comment, it can be noted that the hierarchical folder structure is in general a feature that can hinder the scalability of the storage system. For this reason cloud storage solutions that are designed for high scalability often drop the 'POSIX' compatibility and do not support directory hierarchies anymore, but only a flat container for all files. In this case, nothing prevents to put forward slash '/' characters in filename to simulate directory names and structure, but aggregated operations on directories or permission inheritance is no longer possible. Aggregated operation can only be done on the base of other metadata attributes.

8.7.3 Access control entries (ACE) subjects

The access control entries (ACE) can be granted to

- individual identities (accounts) recognized either by the local system or by the identity federation network.
- Generic (and often unreliable) network properties: IP v4/v6 addresses or contiguous address class (with masks), DNS domains obtained by reverse lookup, location identified by DNS geo-

location, connection properties that the storage service can detect: 'Authenticated users', 'Everyone' (literally any request), 'Anonymous users' (users that have not authenticated).

- Attributes that are attached to the identity of the authenticated user. These attributes could be 'personal' (nickname, gender, nationality, age, email), 'Group Memberships' (organizational units, user-defined groups) and 'roles' (supervisors, administrators, accountant, ...) and are typically available to facilitate permission management by granting rights to classes of accounts (that are maintained by the identity management process) rather than to constantly changing list of end-users (list managed by the resource owner).

Note that attributes attached to the identity of the authenticated users may be of variable quality. Some attributes may be verified and guaranteed by the identity management process and can be used to enforce a reliable access control. Other attributes could be self-declared and self-managed by the end-user and should be used only to profile as hints to enhance the user experience, but not for access control as their value can be unreliable.

8.7.4 Authorization at CERN and in high energy physics implementations

Given the multiple solutions of storage used in high energy physics we can only say that nearly all possible options described in the authorization sections above are currently in use.

At CERN we have multiple implementations (EOS and CASTOR⁴⁸, for example) that have different set of authorization rights with the authorization metadata stored in the name server. This adds with an existing Andrew File System (AFS)⁴⁹ infrastructure and a Distributed File System (DFS)⁵⁰ that have different and incompatible security model for authorization. This diversity exists mainly for historical reasons and it is source of confusion for end-users and increased complexity. The most important drawback is that when data is moved from one type of storage to another, permission and authorization information are not always preserved. For this reason, as part of the on-going storage renovation process foreseen in the years to come, a simplification and an uniformization of the various authorization models is currently being planned.

8.8 Group and roles based authorization

The typical sets of attributes attached to the identity of authenticated users are group memberships and roles and in many frameworks are essential architectural components to effectively manage security and permission because they allow a separation of responsibilities between:

- Defining a permissions to a resource in terms of classes of users (responsibility of the resource owner)
- Maintain the list of users who belong to this class (responsibility of the identity management process)

For example, if the author of a scientific paper (the resource owner) wants to restrict access to his document to the 'members of his collaboration' (class of users), he or she just sets the access control

⁴⁸ See <http://cern.ch/eos> and <http://cern.ch/castor>

⁴⁹ For a description of the AFS infrastructure at CERN, see <http://cern.ch/afs>

⁵⁰ DFS is the Microsoft based 'Distributed File System' based on NTFS permissions. For a description of the DFS infrastructure at CERN, see <http://cern.ch/dfs>

list to allow access to the named group. The identity management process, then maintains the correct group membership which indirectly keeps the permission up to date.

Note that this role separation between assigning permissions and maintaining them can also happen between end-users when the identity management service supports the creation of 'user maintained groups'. One user create and maintains groups of users and these groups are then used to set permissions by other users who trust the owner of the group to correctly maintain the group membership.

Despite the group and roles-based permission model seeming very attractive, it has, however, some issues that hinder the scalability of the entire storage solution. The first is to understand when the group membership resolution is done:

- at runtime, when a resource is accessed and the associated ACL contains permission assigned to a group. At this point, the process requires a lookup in the identity management database to verify if the user originating the request is member of the allowed group. This lookup must be repeated for every ACE found in the ACL and each these requests to a central database will make the architecture non scalable.
- at authentication time all the active roles and the groups to which the user is member are added to the user authentication token as extended properties. When a resource is accessed and the associated ACL assigns permissions to a group, the group membership is searched within the authentication token and the access granted accordingly.

This second approach gives a better scalability to the storage architecture and is the preferred one. However, resource owners should be aware that as the group resolution is done at authentication time, changes in group membership requires a re-authentication to be effective and in an environment where authenticated sessions do not expire, there is no guaranteed consistency between permission assigned and effective rights. It is a typical trade between scalability and consistency.

Another factor limiting scalability is that when an identity is member of a large number of groups or has cumulated a large number of roles the attributes of the authentication token can grow to the point that every authorization verification becomes slower. To mitigate this limitation, groups can be defined as belonging to a particular 'scope' and their validity (and propagation) is then limited to a local storage element or to a site instead of becoming a global attribute. This adds quite some complexity but is unavoidable in a global environment with large number of users, groups and roles that are often entirely specific to a site or to a service.

This means that in a federated identity environment, groups and roles are often managed locally within the site and the number of 'global groups' managed by the identity federation is either very limited or deliberately not implemented to ensure scalability.

Finally, another reason against using a large number of global groups is that the group membership information may contain personal information or details on personal responsibilities and/or organizational structures that may be considered confidential. For example, do we need to have nationality and age information of end user so that permissions can be set on these two attributes? This is an interesting trade between security and personal privacy.

In practice, in high energy physics the middleware implements a very small number of global groups that are shared among all the institutes participating in the LHC computing grid: these are called 'Virtual Organizations'⁵¹ and are limited to a handful of very large groups that typically identify the members of an experiment. For example we have virtual organizations for the members of the 4 LHC experiments ALICE, ATLAS, CMS and LHCb. All these global groups contains thousands of members and are globally known to all middleware software worldwide.

As this approach of global groups using virtual organization does not scale, they are kept to a very limited number. This makes necessary to delegate groups-based authorization to individual sites which create local groups known only within the scope of the site.

At CERN, the service to manage local groups is part of the identity management process and offers the possibility to each user to own groups of which he can manage the membership or groups that have a membership automatically generated from properties derived from the identity management database. To give the reader an idea of the size of this service, the CERN identity management service currently manages approximately 35,000 local groups

8.9 Accounting

After the authentication and the authorization components, accounting represents the third pillar of data management security.

Accounting is the process that remembers the history of all requests executed on the storage infrastructure (who, when, what, where) that enables traceability of all changes and, in some cases, transactions rollback.

There are always multiple levels of accounting possible. The simplest is to log all requests in order to be able to know the history of what happened in the past. This allows to understand past sequences of events and identify responsibilities, errors and failures in incidents or failed processes.

Accounting can go beyond a simple log and includes a journal of all transactions. In this scenario, it is possible to know the history of changes and rollback the data content to past moments in time. This is a useful tool for recovering from data corruption, failures and security incidents.

It is important to note that a good accounting is a valid alternative to a strict authorization scheme to ensure an effective security. Within a strict authorization scheme, every actor is given only the minimum amount of permissions to execute his task. In a strategy to 'minimize the surface of attack' the strict permission system prevents the end-user from accidentally losing or corrupting data.

Depending on the technology used to record the transactions, changes can be rolled back only to predefined times (when using snapshot techniques) or to any arbitrary time when using an incremental journal.

If the system has a solid accounting service, then end-users can be allowed an authorization that goes beyond their responsibility, but they are held responsible of their actions. The security, in this case, is ensured by the traceability and the possibility to roll back errors and abuses.

⁵¹ See [http://en.wikipedia.org/wiki/Virtual_organization_\(grid_computing\)](http://en.wikipedia.org/wiki/Virtual_organization_(grid_computing))

However, this trade between strict authorization and detailed accounting is only a valid option to ensure data preservation and avoid corruption: accounting cannot be used to ensure the confidentiality of data for which access control remains the only possibility. Once the data has leaked, it is gone and there is no accounting mechanism that can rollback information leakage.

In high energy physics, every participating institute has his own account policy. At CERN we record more than 10 GB of accounting data every day and we have a retention period which varies depending on the service level provided. This retention period varies from few months to more than a year.

8.10 Quotas

Storage quotas in a service is an (optional) feature which allows the administrator to define a maximum usage of resources. Resources can be 'storage space', 'transfer bandwidth', 'requests per seconds', or any other resource usage indicator.

The main reason for having quota is to prevent involuntary abuse of resources even if they are also effective to prevent voluntary abuse and to bill resource usage.

The quota can be applied to restrict resource usage from a particular 'account' identity (for example a user is allowed a maximum storage space) or can be applied to a resource itself to restrict its size (for example the quota systems defines the maximum size of a pool, a volume or a directory). Beware that when both options are used, this can be source of confusion for end-users as the ability to execute a request is the result of the intersection of the permissions between multiple quota systems (one on the identity and another on the resource) that is difficult to understand. The typical example is the 'access denied' error message returned by many file access protocol which end-users have difficulties to understand and identify the exact origin of the failure of their requests.

There are several possible choices of quota enforcement architectures that allow trading of scalability versus consistency. Quotas can be enforced in real time, synchronously with the request (hard quotas). In this scenario, the system is guaranteed to be consistent and it is impossible for a user to abuse resources. But verifying the quota is another database operation 'in the line of fire' of the request, which breaks the service scalability! As an alternative, quotas can be enforce asynchronously (soft quota) by an independent process (linked to the accounting system), allowing to maintain the full performance and scalability but allowing temporary over-quota inconsistencies.

The architecture can deliberately link quotas and physical resources allocation. When this link exists, quota becomes a mechanism to guarantee the availability of resources to a particular owner, avoiding any possible shared resources. It is equivalent to a 'resource reservation' implementation where unused resources cannot be reused (and may be perceived as inefficient use of resources). On the opposite extreme, quotas and physical resources can be independent and the sum of quotas allocated can be orders of magnitude higher than the resources available. In this scenario quotas become a mechanism to 'control' resource usage and keep it under a reasonable level leading to potential 'overbooking' problems.

As a comment, all these choices lead to different architectures that have specific advantages or disadvantages. The way forward is always to choose the architecture that best fulfill the main needs and then define procedures and tools to handle the issues foreseen (as the overbooking example).

9. Monitoring

Monitoring is an essential component of any data management infrastructure, because it measures the key performance indicators of the service. Several parameters need to be monitored, one for every potential bottleneck of the service, for example:

- Network performance, memory and CPU load of every server
- I/O rates for every disks
- Popularity of every file, to identify hot spots
- For every existing cache: hit and miss analysis
- I/O rates per user
- Length of the pending request queue
- Access latency

All these parameters must be measured and then aggregated under multiple perspectives. The 'service' perspective that needs to identify and compare behavior of independent servers, pools, storage elements and calculate aggregated performances, statistics, global indicators in order to intervene in areas which misbehave. And the 'users' perspective that ensures that the agreed quality of service is respected for all users.

The first perspective, which detects individual component failures, also allows optimization and reduction of service cost in general, while the second is essential to maintain the quality of service contract with every user. For example, end-users will never accept for a very low transfer rate the explanation that there is no problem because 'on average' the aggregate transfer rate of the service meet the agreed quality of service. He is only interested in his own transfer rate.

This means that the monitoring infrastructure (figure 26) must be able to measure not only the service local averaged and aggregated values but also all the individual end-user indicators (queues waiting time, latency, throughputs, etc.).

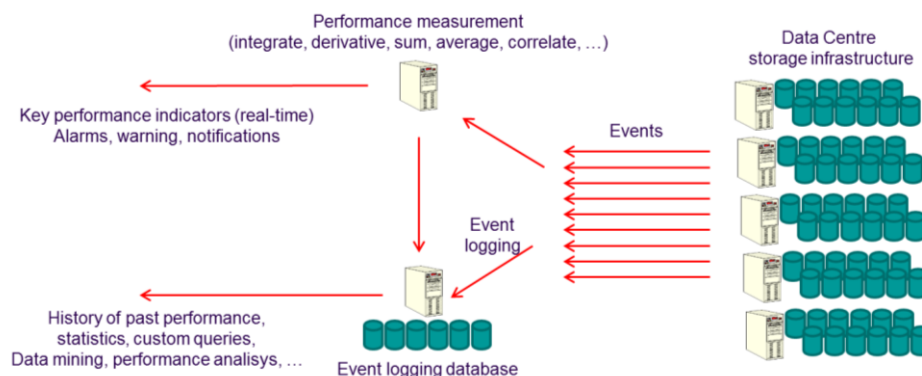


Figure 26: Workflow example of monitoring information

9.1 Alarms

The complexity of the multi-perspective service monitoring also requires that misbehaviors can be identified by comparing a particular measure with the performance of other identical components at the same time (spatial analysis) or by comparing it with the past performance of the same component (time analysis). Data must be aggregated in two dimensions in order to provide, in real

time, the maximum, the minimum, the average, the current value in time and all comparisons with past performances as well as comparison with performances of other identical instances.

The Alarm system compares all aggregated values and performance indicators with the 'Service Level Agreement' parameters and raises alarms whenever necessary from the large number of events constantly generated by the monitoring service.

Once the alarm has been raised, this is fed as an incident in the service management workflow which then triggers the appropriate incident response.

9.2 Monitoring and Alarms in high energy physics implementations

Every institute participating in the LHC computing grid has its own infrastructure and policy for monitoring and handling alarms. Similarly, CERN has developed its own monitoring infrastructure that takes into account the new requirements of a multi-site data center located in both Geneva in Switzerland and Budapest in Hungary.

This monitoring infrastructure will not be discussed further as it goes beyond the scope of this document⁵².

10. Performance

As mentioned when defining quality of service (chapter 3), performance in general is difficult to define because there are many indicators influencing it and rarely is the starting point of the design of a storage architecture.

The design typically starts from designing an architecture that satisfies the reliability and security requirements and then hoping that performance will be satisfactory.

It was briefly mentioned in sections 6.6 and 6.7 that the various way of storing the data was affecting performances. In general, a particular infrastructure can be performance optimized for a given access pattern, but this optimization needs to be reviewed every time the application, the workflow changes the way data are manipulated.

What follows in the performance chapter are some generic considerations.

10.1 Parallel access, single user

As a general rule, when a single user reads or writes data to disk, the more disks in parallel you have for your data, the faster you can read or write. And this applies to both disk striping (RAID 0) or chunk/block based storage (figure 27).

⁵² Some information can be obtained from the CERN monitoring pages available at <http://itsb.web.cern.ch/> and <http://cern.ch/sls/>

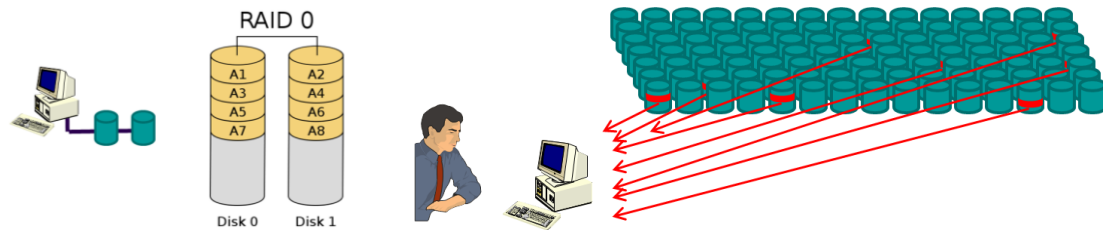


Figure 27: In single user mode, data is served in parallel, ensuring high performance

You can think that the performance scales linearly with the number of disks involved, and imagine that with the chunks-based approach the performance can reach the aggregated performances of all disks you have in the storage system. For example, to read a 1,000 MB file from one disk supporting 100 MB/s you need to wait 10 s. If you read it from a RAID0 stripe with two disks, both disk will be used in parallel and it will only take 5 s. Now, what about reading our file stored in 100 chunks of 10 MB each from 100 disks? The 100 disks will be able to deliver all the 1,000MB data in 0.1 s, but it is our client that will not be able to handle it. Therefore, you can conclude that parallel I/O is always effective to increase the speed of a particular bottleneck, but parallelizing an area which is not the bottleneck won't help.

This is another application of Amdahl's law⁵³ summarized by 'the overall system speed is governed by the slowest component'. Therefore after you have improved the parallel disk I/O, on both client and servers, you need to start parallelizing network interfaces, then CPUs and then the next bottleneck, whatever this is.

10.2 Parallel access, multiuser

A scalable and performing architecture in single-user mode can become totally ineffective in multiuser mode. For example, in the scenario of disk striping, two disk are synchronized and all read and write operation are serialized (figure 28). The resulting throughput should be comparable to a non-RAID configuration where the two disks are used independently, but the latency will be higher as there is a unique queue through which all I/O is serialized.

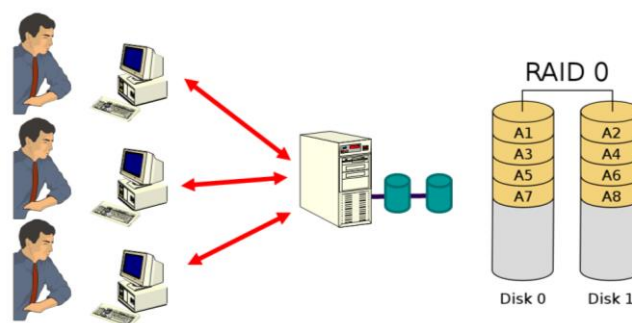


Figure 28: Multiuser access to striped disk, with performance drop due to the large number of I/O operations

An even more significant latency increase happens when using RAID-5 or RAID-6 configurations compared with simple mirroring, and demonstrates that the optimal storage organization strongly depends on the type of access patterns.

⁵³ http://en.wikipedia.org/wiki/Amdahl's_law

The final example is the usage of tape media, which gives an excellent throughput in streaming unique data streams, but which would give a catastrophic high latency if used in multi user random access mode.

10.3 Separating read and write operations

Another important performance factor is when there is simultaneous read and write activity on the same disk which triggers performance drops due to the constant disk head repositioning. In this case, if it is possible to separate the data-in and -out flows to different hardware, the content is written to write only disks and read from read-only disks, giving higher performance.

Typical the possibility of doing this optimization is related to the data workflow of the application. In the example of scientific computing, often the data is moved in and out of pools following organized and predictive ways and this gives plenty of opportunity for optimization, especially when two data flows are crossing each other in the same pool.

10.4 Cache implementation

The cache⁵⁴ is a 'fast' storage area where frequently accessed data can be temporary stored for rapid access: The cache algorithm tries to guess which data may be used in the future and to avoid re-fetching or re-computing the requested data, it keeps it in the cache.

A cache is effective whenever the correct requests that can be served from the cache (hits) outnumber the amount of data that need re-fetching (misses). The usage of a cache can dramatically increase performance, but here again it depends on using the correct algorithm for predicting future access which depends on the access pattern.

We can define the efficiency of a cache memory as the ratio between the 'number of hits' and 'number of total requests' (figure 29). In general, adding a cache memory to a storage infrastructure:

- always increases the cost
- never increase the usable storage capacity
- can increase the performance (depending on the access pattern and efficiency of the caching algorithm)
- can decrease the consistency (depending on the access pattern and efficiency of the caching algorithm)

One can see that if the prediction algorithm is not effective, the presence of the cache increases the cost without affecting performance. For example, imagine a data streaming application requiring a 'First In, First Out (FIFO)' circular buffer. In this case, the whatever the cache size will be, the efficiency will be 0% because the data will be always be requested again after it has been flushed from the cache.

Similarly, in a scenario where data is accessed in true random mode, the cache efficiency will increase linearly and be proportional to the cache size.

⁵⁴ See [http://en.wikipedia.org/wiki/Cache_\(computing\)](http://en.wikipedia.org/wiki/Cache_(computing))

In practical implementations, the efficiency varies depending on how well future data requests can be predicted from past data operations. And it is important, after having designed an algorithm to measure its efficiency and exploit the nonlinear behavior of the efficiency with respect to the cache size. If the algorithm matches the access pattern, the measure should show that the cache efficiency increases rapidly and then stop increasing after a certain size. This gives you the optimal value of the cache size for that algorithm.

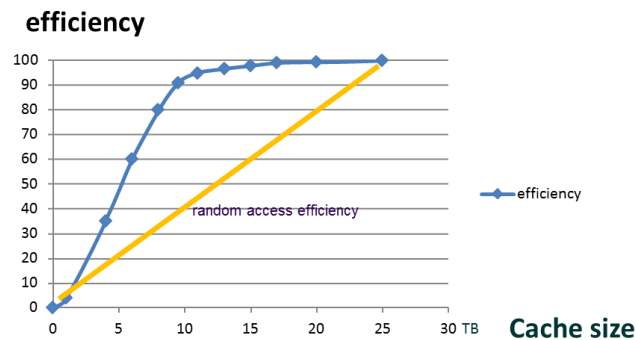


Figure 29: Example of cache efficiency measurement where efficiency does not increase after 10 TB of cache

Additional optimization can be done by performing a sensitivity analysis:

- Start measuring in real time hits, misses and response latencies.
- Measure the variations of the caching efficiency towards all possible small changes of adjustable parameters (cache size, cache algorithm, requests size, etc.).
- Finally tune the variable parameters to the desired compromise between efficiency and cost.

When the cache memory is 'volatile', meaning that in case of a power failure the content is lost, all write operation must be flushed to the 'slow' memory. The safest approach is to flush data synchronously when it is written, but this voids entirely the cache performance gain when writing reserving the cache usage only for read operations.

If caching is also necessary in writing data to maintain a minimum level of performance, data must be flushed to the slow memory asynchronously. This can be done with a thread pool with fixed number of threads which guarantees a constant data throughput or with a thread pool with dynamically changing number of threads to guarantee that all data is flushed within a maximum amount of time. Note that it is also possible to have both policies and dynamically change the flushing algorithm according to the flush queue length.

The asynchronous cache flush of the fast memory is another architectural choice that allows gaining performances at the expenses of reliability and consistency of the service.

10.5 Cloud storage

Cloud storage represents the set of technologies that have been used in recent years to address scalability issues in large storage environments. As large storage environments based on traditional file systems were suffering from reduced performances due to architectural limitations, the improvements in scalability of cloud storage have a direct consequence on performances.

There are several limiting factor within the infrastructures based on traditional file systems:

- Some metadata needs to be modified every time a document is accessed: The 'Last Access Date' is the notable example that, for a correct implementation would put an enormous load on the name server in a large multiuser environment.
- The hierarchical folder structure requires inheritance of metadata information, for example permissions and 'Last Modified Date' of folders and other attributes.
- The usage of metadata can be very intense and create severe performance bottlenecks. For example listing folder containing a large number of files can generate millions of metadata requests on the name server with no data transferred.

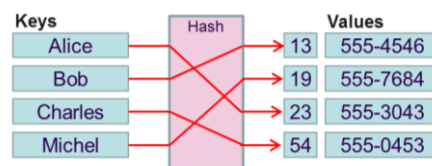
The Cloud storage solution and the associated access protocols (HTTP Get/Put or Amazon S3 in particular) address this issue in retaining all known scalable features of file systems and dropping all known bottlenecks. This provides a scalable and performing architecture but depending on the needs, some of the storage requirements may be missing and must be re-implemented within the application. Non-linearities affecting performance will reappear but will have to be addressed within the customer applications and are not perceived as problems of the storage service.

Cloud storage is generally hosted by third parties on the Internet which offer commercial storage service based on virtual pools with a common standardized quality of service. If this can meet the requirements of scientific data analysis, it represents a very simple and effective solution to consider: all data in a single logical pool, which is reliable, fast and outsourced. An additional simplification for small and medium data sizes is that no upfront investments are necessary as only variable costs are exposed.

Cloud storage is often implemented using 'Distributed Hash Table (DHT)',⁵⁵ that enables scalability as it spans multiple clusters/multiple data centres and is federated under a unique name space (figure 30). The building block is the 'Hash table',⁵⁶ an implementation of an associative memory which uses a data structure that map keys (ex: a person's name) to their values (ex: the telephone number).

In cloud storage, an hash function is used to transform the keys into the indexes (the hash) of an array element where the corresponding value is stored. The hash table brings perfect scalability because the cost (number of instructions) for lookups, insertions and deletions of key-values is independent of the number of elements stored in the table and can be done at constant cost per operation.

This ensures the scalability. And after looking at the architecture, we can also understand why some functionalities (like enumerating files in folders) are not directly possible and require separate indexes and software to be implemented.



⁵⁵ See http://en.wikipedia.org/wiki/Distributed_hash_table

⁵⁶ See http://en.wikipedia.org/wiki/Hash_table

Figure 30: A hash table stores key/values pair. The hash of the key is the index to retrieve the value

The type of hash function used defines the name space and the maximum number of entries in the hash table. For example, if the algorithm chosen is CRC16 the hash will be 16 bit, leading to 2^{16} entries (65536) in the hash table. If SHA1 is chosen the number of hash values will raise to 2^{160} entries (more than 10^{48}), but even if some hash functions have a low probability of collisions, these must be foreseen and handled (figure 31).

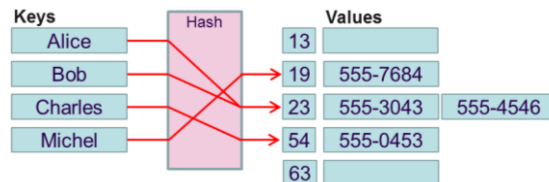


Figure 31: In this example, there is a hash collision between Alice and Bob.

There are many ways to handle collisions, and the most common is to use the next available hash location to store the value and it is essential to ensure that a value really corresponds to the original key used. Given that the hash function cannot be reversed, this makes necessary to save the cleartext key with the value.

Another interesting parameter of a hash table is the 'load factor' (or fill factor) which is the ratio between the number of stored entries (4 in the example above) and the size (00 – 99 in the example above) of the table's array. The probability of collisions and the cost of handling them increases with the load factor that must be kept low. On the other hand, when the load factor is low and approaches zero, the proportion of unused areas in the hash table increases resulting in wasted memory.

When the load factor is too low and close to 0 (waste of memory) or too high and close to 1 (too many collisions), the hashtable implementation becomes inefficient and requires a full or incremental re-hash of all keys becomes necessary to restore the design scalability. But re-hashing all keys can be an extremely long process which breaks the scalability itself.

The most well-known workaround to this problem is to choose a hash function that preserves the key hashes when the table is resized. When the hashing algorithm satisfies this property, it is called 'consistent hashing'⁵⁷.

10.6 Distributed Hash Tables in cloud storage

A solution relying on a hash table to locate its data supposes that all the storage is local to one server. We are very far from having a distributed cloud solution. To be able to (equally) distribute the data across multiple servers, the next requirement is to partition the keyspace of all possible hashes into independent storage servers.

⁵⁷ See implementations details at http://en.wikipedia.org/wiki/Consistent_hashing

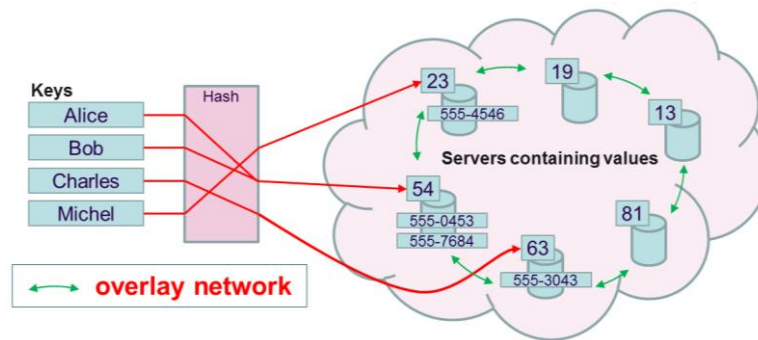


Figure 32: A distributed hash table with the keyspace partitioned across 6 servers.

Again, to avoid recalculating all hashes (and therefore re-reading all data) when a server is added or removed in the infrastructure a ‘consistent hashing’ algorithm becomes essential. This allows the removal or addition of one server by affecting only adjacent nodes.

It is important to remember that there is no ‘authoritative’ name server that a client can contact to obtain the address of the server which holds a particular partition of the keyspace. Therefore, a client will send its request to a random node (and randomness is important to ensure scalability) and, if the client is lucky, it may be served the requested data directly. Otherwise, its request must be redirected to the correct server with a minimum amount of network roundtrips. Routing a data request to the correct server owning the data corresponding to a given key is the role of the ‘overlay network’ displayed in green in figure 32.

As a comment, to be able to route the request only in one redirection, every server requires the knowledge of the entire namespace partitioning and this requires an unpractical reconfiguration of all servers whenever a node is added or removed. To avoid this drawback, sometimes a node maintains knowledge of only a subset of the keyspace partition of some nodes (adjacent and opposite) and the redirection is designed to rapidly converge within a maximum guaranteed number of hops.

An important option in designing a cloud storage concerns the ‘values’ stored in the distributed hash table: This could represent directly the end-user data (for example the key is the ‘filename’ and the value is the ‘file content’) or it could be a storage based on ‘chunks’ (or ‘blocks’) and with the keys being the ‘chunk hashes’ and the value being the ‘chunks content’.

High reliability, fault tolerance and high availability can be implemented by using the chunk-based approach and store an arbitrary number of additional chunks that contain error-correction information (as described in chapter 6.4). In a file-based DHT, there is also the option of replicating values (file contents and/or chunks) in an arbitrary number of adjacent nodes in the overlay network ring. This means that in a cloud storage infrastructure, implementing arbitrary reliability using chunks or replica is easy as it is a simple configuration option, which even allows mixing of the two

approaches (using chunks with error correction and replicate them). Note how similar this scenario is to the bittorrent protocol⁵⁸ architecture.

Finally, if nodes are distributed over wide area, we can attempt to calculate the overall reliability by supposing that probabilities of failures are independent, but this requires a full trust on the quality of the software which is deployed across multiple sites and remains the most important factor of inter-dependence of failures because a software bug can affect multiple sites simultaneously.

As a conclusion on this topic, the most important goal in the design of cloud storage remains scalability. This allows us to have arbitrary performance, as we have an architecture which delivers a service that scales linearly with the sum of the individual hardware resources. Typically, scalability can be verified by ensuring that the quality of services (discussed in chapter 3) does not depend on the storage service size. Once verified, we have an infrastructure that supports 'arbitrary performance' which combined with the techniques discussed in chapter 6 can also have an 'arbitrary reliability' levels.

11. Final considerations and conclusions

Storage in scientific computing is a complex environment and there are many attempts to simplify the approach with generic or commercial solutions with fixed balances between performance/reliability and cost, which can and should be used whenever the generic solution matches the requirements.

The generic/commercial solutions can also be of interest when the size of the storage is small. In this case, the inefficiency of the requirement mismatch can be compensated for by more storage investment, which being a small fraction of the overall project cost, may be acceptable.

Things can be different in a research project where the overall storage costs represent a large investment. In this case, a mismatch in requirements which triggers cost variations of only few % can generate significant additional budget requests. This makes the 'generic' storage solution with fixed quality of service not simpler, but simplistic.

If the 'generic' storage solution cannot be used, the alternative can be a custom design where an infrastructure is designed ad hoc for a particular application. This ensures that requirements are met and that the custom design minimizes the resource investment and the operational cost. However, the upfront development cost for a custom infrastructure may be enormous both for the initial design and for the later software maintenance. In addition, the development of an optimized solution may become inflexible to adapt to changing requirements of scientific researches.

Between these two extremes, owning and mastering a set of technologies that allow the integration of storage components and to deliver 'arbitrary performance' and 'arbitrary reliability' at an 'optimized cost' becomes of strategic importance. These components, represented by storage pools combined with data transfer and synchronization tools, can be quickly assembled to create a data workflow where different and specific requirements in the data processing application are quickly

⁵⁸ See [http://en.wikipedia.org/wiki/BitTorrent_\(protocol\)](http://en.wikipedia.org/wiki/BitTorrent_(protocol)) for more details on the bittorrent protocol.

matched by tuning few appropriate configuration parameters. In addition, the flexibility available in the initial design is again available during the subsequent operation and production phases. This is useful in environments where requirements are frequently changed (typical of a scientific research environment) and the architecture is constantly adapted to ensure that the optimizations are maintained at all levels of the deployed solution.

This strategy has been the main driving force behind all software developments of the last years to design a scalable and flexible data storage infrastructure for the CERN LHC scientific research worldwide. This architecture has proven to scale effectively beyond the 100 petabyte (100'000'000 gigabyte) size and provide a reliable and cost effective service. We firmly believe that this model for scientific data storage would be useful to other sciences and we do not see any limitations (yet) that would reduce the planned growth that is expected in the coming years.