(deprecated document)

Please refer to:

https://docs.google.com/document/d/1WslZOavtI1wruWzzpwIyqm3AcIeaGRiH1FbAFWFcee
o/edit?usp=sharing

Datalakes straw models and its building blocks

Nomenclature: archive, disk, staging areas, caches and buffers

Technological, funding and scientific prospects are forcing data management systems, storage providers and experiment computing models to evolve.

One-size-fits-all approach is no longer sustainable in future distributed storage. Optimization of resources is the main goal and comes with an engineering challenge in the way data is stored, organized and accessed.

Files continue to be the same immutable entity from client perspective but internally to storage files can have different characteristics and evolve:

- **On-demand redundancy**: file replication shall no longer be integer number of replicas but fractions (Erasure Coding)
- **QoS**: different types of storage with different properties in terms of reliability, price and performances
- **File workflows**: files will transition and flow with time among different states and properties

These three storage characteristics will be provided by a conjunction of storage specialized sites (**D**ata and **C**ompute **C**enters). A subset of DCCs will define a Datalake.

Data processing oriented sites (**C**ompute **C**enters) will be *connected* to the Datalake through an orchestrated mechanism of Staging Areas (latency hiding layer) and Caches.

Let’s define these terms:

Datalake

Set of sites, associated by proximity, providing together storage services to an identified set of user communities capable to carry out independently well defined tasks. Operationaly, is more efficient that VOs use the same Datalake boundaries. Proximity could be defined by geography, connectivity, funding or a shared user community. This requires that their combined storage capacity and bandwidth can meet the demands of the designated task and that usage of the different sites is transparent to the users. This means that some form of trust relationship has to exists and a way to locate data. This can range from a simple file catalogue to a full fledged namespace.
While access for users is transparent during job execution, the population and management of the storage within the Datalake is a planned and managed activity. This includes the transitions between QoS levels. These operations are done on the granularity of the Datalake. Data is moved to a Datalake, not to a specific site, the storage resource management within the Datalake is the responsibility of the Datalake.
Data access between Datalakes is possible, but will be done in a managed way, as and example: leveraging the population of data by data management frameworks and re-population of data after data loss.

Example: A group of sites, not further than a RTT of 50ms apart, well connected with enough storage to hold the working set (all Mini and Nano AOD) for analysis for an experiment. Using a simple file catalogue. The data for the ongoing processing campaign is moved there by central data management tools (ie. Rucio)

The advantage of Datalakes is in the reduction of replication of functionality, economy of scale of the large scale providers, reduction of complexity for users and experiment data management.

Staging Areas

This is storage that is used to give unimpeded access to workloads that are active or will become active very shortly. These Staging Areas are located at the edge of sites, data is moved and removed there by data management systems and discovery is through storage system and some form of simple file catalogue. Staging areas are a planned way of latency hiding and can be used as gateways to resources that are

not connected directly to global networks, like HPC centres or IaaS. It is a technicality whether the deletion of data is managed by the data management tools of the experiments, or by the store itself. The requirements for data reliability are low, due to the transient nature of the usage, access requirements in terms of I/O operations and bandwidth can be high, depending on the use case. The size can be limited to handle the amount of data needed for up to a week of operation. Read/write functionality is required in use cases where the edge service connects different authorisation domains. The implementation of the staging area is transparent to the users.
In areas with sites being located sufficiently close, in matters of bandwidth and latency, this staging area can be shared.

Benefits are that for a limited amount of storage data accessibility can be significantly improved. For the use of resources like HPC farms this can be mandatory, otherwise it depends on the scale of the Datalake whether these are beneficial.

Cache

A self-managed transient storage layer at the edge of a site. To be truly useful the Cache has to be streaming and providing read-ahead functionality. The content of the Cache is only known by the Cache while for users and data management services (ie. Rucio) the cache is fully transparent. The role of the local cache is twofold:

- a) Reduce the required wide area network bandwidth by holding frequently used files in the cache
- b) Ability to read ahead to reduce the impact of latency and peak bandwidth requirements for the first reading of the file.

The size of the cache depends on the data type being used at the site. From cache simulation studies it has become clear that the reuse rate depends on the data type. The optimal size is given by the frequently reused files and enough storage to serve the running jobs that process files only once. First studies, based on data popularity studies, indicate that a very modest size, compared to current T2 storages is sufficient.
The support of write access from the site would allow to also improve the performance of tasks that produce sizeable amounts of data. However, the same benefits can be achieved by using a Staging Area from where results are moved by the storage management services to the final destination.
These Caches can be implemented in many ways, as standalone services, distributed over all processing nodes at the site, etc. The different advantages have to be studied for the different use cases.

Cache activity is driven by client activity, with clients requesting data from a cache. If the requested data is stored locally then the cache will serve the client's request using the cached data. If the requested data is not stored locally then the cache will request the data from the remote source. Existing caching technologies (ie. XCache) provide a rich set to express preferences regarding the location of the data.

It has to be noted that on large sites with many active clients the I/O demands on cache nodes will be substantial.

Client Buffer

To achieve the behaviour of an intelligent (streaming, read-ahead) cache with respect of managing latency and bandwidth the application can manage the I/O in most cases. Investments by CMS and ALICE have shown that by employing advanced asynchronous I/O techniques and tools to select and predict those subsets of data that will be used by the application, the throughput/efficiency of the workload can be maintained with losses below 5%, despite significant latency. Most of these techniques are available and integrated in the analysis frameworks that we currently use (ROOT), however care has to be taken to configure them correctly for the situation at hand.

Straw Models for Datalakes

Abbreviations

TC = Tape center = location that has a tape archive.
Throughout we assume that the total data is available across all TCs. I.e. an individual TC does not have all the data.
DCC = Data and computer center
CCC = Compute center with cache.
CCNC = Compute center without cache. A CCNC relies on accessing all data via the network from either a CCC or a DCC.

Frank's Model

1) Distributed working set per lake with caches.
    a) Model components: TC, DCC, CCNC
    b) Assumptions:
        i) There are multiple data lakes in the world. One data lake per region.
        ii) The data lakes may have 0,1, or more TCs. I.e. the TCs are fundamentally not connected logically to a data lake as they need to feed all data lakes.
        iii) The data lake notion is useful only for analysis. Centrally organized processing should be thought of as an activity across data lakes.
        iv) Each data lake has a complete copy of the analysis working set distributed across a set of DCCs in that region. Each DCC within the region is assigned a subset of the total working set. Very popular datasets may be hosted in more than one DCC within some data lakes, but not others.
        v) Each TC has some modest size disk as tape buffer only to feed the DCCs.
        vi) A given center may be both a DCC and a TC, but this is not required from any center. It's possible to have a TC only center.
        vii) There are possibly three types of disk usage at the DCC:
            (1) Buffer space for processing campaigns where data is placed only for the duration of the processing, and placement is managed as part of the workflow.
            (2) Replica management system managed space. We need to make sure that one copy of the working set is guaranteed to be on disk. This guarantee implies that the storage system for this space is either replicated at the DCC, or that two copies are replicated across two (or more) different DCCs on non-replicated storage. The latter is probably the more effective use of disk space.
            (3) Cache space. Each DCC has a namespace attached to its cache space. These namespaces between DCCs may overlap in order to maximize the total CPU that can hit a given dataset within the namespace. Other than that, this cache space is not centrally managed. The namespace that is managed by a cache at a DCC may be changed dynamically by the operations team of the experiment.
        viii) It is an operational decision by the experiments to shrink or grow the cache space vs replica space vs buffer space. E.g. in general, processing campaigns are not running all year, and buffer space may thus be reallocated as cache space for part of the year to increase the availability of data to CPU.
        ix) A DCC may have multiple CCNC assigned to it. The CCNC have only compute power, and are configured to access data only from the DCC they are assigned to. The assignments of CCNC to DCC are based on some maximum RTT. Network bandwidth at the CCNC must be commensurate with the total processing power of the center. Network

bandwidth at the DNC must be commensurate with the sum of processing power across all assigned CCNC plus the processing power of the DNC itself.

x) The workload management system sends jobs to start at DCCs or their assigned CCNC based on the namespace the DCC is configured to serve. I.e. the RTT between DCCs is too large to efficiently run jobs with remote access.

xi) DCCs may be distributed. E.g. two physical centers that are close in RTT such that it does not matter which one hosts what data may configure their shared diskspace as one DCC.

xii) None of the above addresses the issue of user data.
   (1) User data is generally not expected to be backed up to tape. It thus needs reliable storage to host it, or be replicated across more than one DNC within a data lake.
   (2) Unless user data is registered with the replica management system, it would be confined to the data lake that was used to produce it, as it could not transfer across data lake boundaries.
   (3) Hm, this needs more thinking.

Markus-Xavi's model

Situation:

The world is divided in DataLakes with a radius of roughly 20ms in network latency. Each DataLake has a full set of Mini and NanoAODs of an experiment: O(50) and O(1) PB respectively.
The data is stored on 3-4 Data and Compute Centers (DCC) that will define a DataLake
Files are stored without internal redundancy.
As first approximation we assume there are less than 10 of these DataLakes.
The data can be located by using logical/trivial file catalogues. These allow the trivial mapping from fileName → DataSet → DataLakes → Site
All processing sites access data through a latency hiding cache. This cache has a few additional abilities to the current Xcache implementation (they could be added). In the following text this cache will be referred to as CACHE+ and will be located close to the processing nodes of the Compute Centers
The additional characteristics are:

- The CACHE+ can do the mapping of the trivial file catalogues

- The CACHE+ is aware of the surrounding DataLakes and knows the distance to them in units of latency and bandwidth
- The CACHE+ can redirect the access request of a file if it isn't accessible within the local DataLake
- The CACHE+ is stateless by definition. Impact of not having the cache (equivalent to a file miss) is only reflected in latency (jobs continue getting data)

Storage systems in Data and Compute Centers in the DataLakes have the following additional abilities:

- When a broken disk is found they create a list of lost files in a **standardised** format.
  - This ability is already present in the current systems, but it isn't standardised or automated
- Like the CACHE+, the storage system can do the trivial file catalogue mapping
- Like the CACHE+, the storage system is aware of the nearest neighbour DataLakes
- The storage has access to bulk data managing services that can do third party copies

We assume that the disk failure rate is 1%/year. Which is a conservative assumption based on the measurements at CERN (0.89%).

It is understood that there is a second layer of storage for software/ conditions data/ etc.

Normal Operation

The jobs on the WNs at a site access files through the CACHE. The ability of caches to hide latency has been studied and results can be found here Ref [Laura and Corentin].
From cache studies based on data popularity we learned that for AODs and DAODs small caches (<<10% of SE size) already cover a significant fraction of the active working set. Studies in line with the FW model have to be done, but require new popularity data.

Direct access to remote sites 20ms away leads without latency hiding techniques, dependent on the workload, to significant degradation of the throughput. Measurements have shown that client site approaches currently are capable up to 10ms.

Data not found in the local cache can be (where *n* is the number of DCCs in the lake) :

- Read from the closest DCC (<20ms) in streaming mode while it is persisted at cache level with 1/*n* probability

Disk Failure

CERN measured in 2017/18 a failure rate of 0.86 %/a for the disks in EOS. A rate of 1% /a can be used as a conservative approximation.

Potential strategies to deal with disk failures:

Storage System:

The storage systems in this scenario can deal with failed disks in the following way:

1. After detection create the list of lost files using their namespace
    a. A 100PB storage system will lose, on average 1PB per year. On a daily basis this corresponds to 2.75 TByte. With an average file size of 1.5GB the system has to do close to 2000 namespace queries in addition to the normal operation. EOS namespace query rates are in the O(kHz) range. Within 10 minutes the list of lost files can be produced with less than 1% of the query rate.
2. The list has to run through the mapping: fileName → DataSet → DataLakes → Site
    a. This is a simple string based manipulation, or a database query. Implemented in the least efficient way a query rate of 10Hz should be possible. This step adds less than 10 minutes.
3. This list has to be given to a storage management system, like FTS or RUCIO, to copy the lost data to the local storage system from the remote site.
    a. The time to copy depends on:
        i. the bandwidth that can be allocated to this task. In a large site the I/O rates are frequently O(10GB/sec). Assuming 0.5 GB/sec, to limit the interference with normal operations, the average amount that has to be copied will take ~1.5 hours before it has been copied back.

The number of sites contributing to the recovery, if we assume 10 DataLakes each one with 3 DCCs, this means 29 sites potentially contributing to the recovery. If we assume ⅓ of the sites hold part of the lost files, then the recovery time is x10 faster, ie. 10 mins with same level of background recovery throughput (0.5GB/s) This means that the average time between data loss and restoring the data by replication from another DataLake will take about 2 hours. If we take into account the granularity of disks, assuming modern disks of 10TB, this means that for a 100PB system the time to restore a lost disk will take ~7.3 hours (30 mins with the above described overall contribution by the whole DataLakes system). This will occur every 3.6 days (on average).

How many/which fraction of jobs will be affected?

During the first 73 minutes of the recovery all jobs that require the data will not be able to access the data directly. From then on every hour 1/6th of the data will be restored. Without some assumptions on the distribution of files on disks and the data demands of the jobs running at the site it isn't possible to calculate the number of jobs affected.

Approach B) Dealing with absolute numbers of files.

All data is read over a certain time, the files are distributed randomly. During the restoration of the disk a number of files will be accessed on the broken disk that is proportional to the restoration time and

the fraction of files that are accessed on the failed disk/hour. This has to be compared to the total number of files accessed during the same time. This makes the ratio independent of the assumed processing time of the whole data set since it takes into account that the failed disk holds a fraction of the data and that only a fraction will be accessed.
In our example, with the 100PB site build of 10TByte disks we will have a miss rate during reconstruction of the disk of 0.53 ‰ . This corresponds to an average file access miss rate of 0.036‰. 1 access in 27780 will request a file currently not present, in worst case the file will be back in 6 hours. This has to be compared to the normal job failure rates of the experiments. From Panda log file analysis we know that for production jobs is O(10%).

For more details and experiments :
https://docs.google.com/spreadsheets/d/1qKY6LfE8Uyr-Uc9li0iYNi9pe-Z7Re4uBHfElVostgw/edit?usp=sharing

Input parameters:

fail rate/year [1/a]	0.01
storage size [PB]	100
size of disk [TB]	10
Average file size [GB]	1.5
NameSpace operations/sec [OP/s]	1000
Fraction of OP/s for restore	0.01
Trivial Catalogue, mappings/sec [1/s]	10
Bandwidth for restoring data [GB/s]	0.5
time for all data being accessed [d]	10onts

1 disk failing, time to restore:

This will happen per year [1/a]	100
Per day [1/d]	0.27
Data loss/day [TB/d]	2.7
Number of files per failed disk	6666
Number of files per day [1/d]	1826
Time to build lost file list for 1 disk [s]	666
Time to do the mapping for 1 disk [s]	666
Time to copy data for 1 disk [h]	5.5
Total time to restore 1 disk[h]	5.9

Impact of lost disk, file misses, relative impact:

Total Number of files on site	66666666
number of files accessed during 1 hour [1/h]	277778

fraction of total number of files accessed in an hour [1/h]	0.0042
fractional size of the failed disk	0.0001
number of files accessed on the failed disk per hour [1/h]	28
files missed during locating replicas	10
files missed during replication (files are gradually moved)	77
total number of files missed during recovery (6h)	**87**
total number of files accessed during recovery	1646090
Fraction of files missed during reco period	0.000053
Above in ‰	0.53
Average file miss rate in ‰	**0.036**

Strategies to deal with missing data

Ignore and try again later:
This will be handled by the re-submission mechanisms of the experiments. Since they clearly can handle O(10%) of failed jobs, an addition 0.036‰ will not be noticable. If this rate increases to 1% (=factor 1000) it will be still barely noticeable.

Proceed and suffer from extra latency:
Since all reads will happen in this model through the CACHE, it is the CACHE noticing the absence of data. Equipped with ability to map the missing file via the trivial file catalogue to an instance of the file(s) in a neighboring DataLake and access it from there. The increased latency will have an impact on the throughput of the running job. The scale will depend on the type of job, measurements of this impact of latency beyond 20ms on CACHE performance are currently done. With 20ms latency and an empty cache we measured a maximum degradation of throughput of less than 10%. Given the small fraction of jobs affected, the main advantage of this approach is that for a small loss in efficiency of the running job no re-submission is needed.

N.b. Usage of buffering on the client side is encouraged. It has demonstrated the ability of efficiently hide latency on jobs running in streaming node (start processing data on the fly) with no special (or very little tuning) on the client.

Strategies to deal with temporary unavailabilities

Data can be temporary unavailable without implying irrecoverable data loss at the site. This type of incidents does not require data re-replication.
Possible cases can be: diskserver maintenance (OS upgrade, security patch), PDU failure, broken motherboard, networking incident (NIC, cable, switch, ToR,...)

Hence we deal with scheduled and unscheduled situations with a common signature: big fraction of data is not available at the corresponding site for a short period of time, ranging from a full diskserver node to a complete rac.

Measures should be in place to distinguish this recoverable incidents from the the unrecoverable incidents where data re-replication will be triggered. Regarding scheduled interventions we know how to deal with in WLCG operations using different means, while for the unscheduled situations we should put in place mechanisms to alleviate it: reading data from a neighbour lake as first measure and schedule new jobs accordingly.

The solution to overcome this situation sits in a delicate terrain. Being too much reactive can easily clog the system in case of small glitches or flapping situations. The risk is that the system notice diskserver A is down, start triggering actions, and shortly after diskserver A is back. With the number of nodes we will have even 1% of this “false” alarms can induce some chaos in the scheduling and the state of the whole system. We need to take this in mind and put resilience to glitches (usually means add more time until taking a decision).

The impact on the running jobs is being estimated in the spreadsheet below.

Strategies to deal with data centers failing

In addition to the incremental failures of individual disks we have suffered in the past and will suffer in the future from events that render complete centres unavailable for several months.

Following the described scenario this means that within a DataLake ¼ - ⅓ of the data will become unavailable for several months.

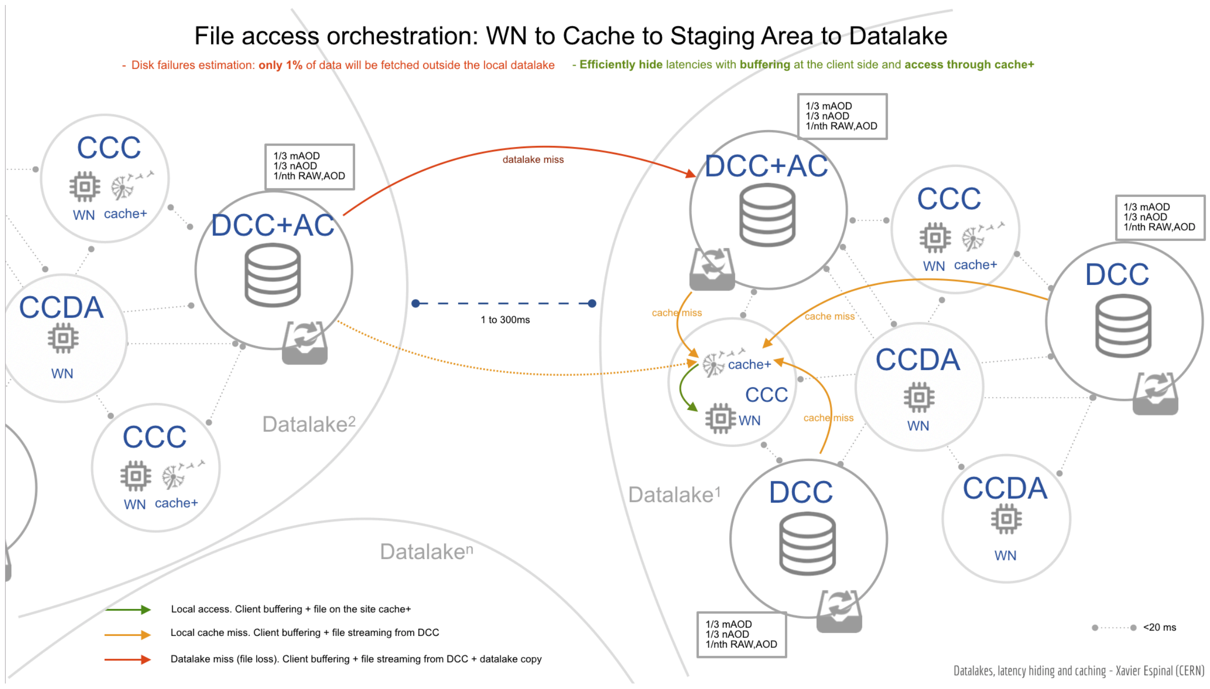
“Ignore and try again later” is no option in this scenario, however, the CACHEs can be re-directed for the missing data to use the data from the closest DataLakes. Since not all sites within the DataLake will access the same neighbouring sites the impact can be split between several places. This might be best done by using a special mapping during the rebuilding of the site.
Over the last 10 years WLCG has experienced a few events that made complete sites inaccessible for extended periods or destroyed significant fractions of storage. Tracked over the years the impact in volume is comparable to the “statistical” loss of individual disks.

Impact on different workflows without using a CACHE

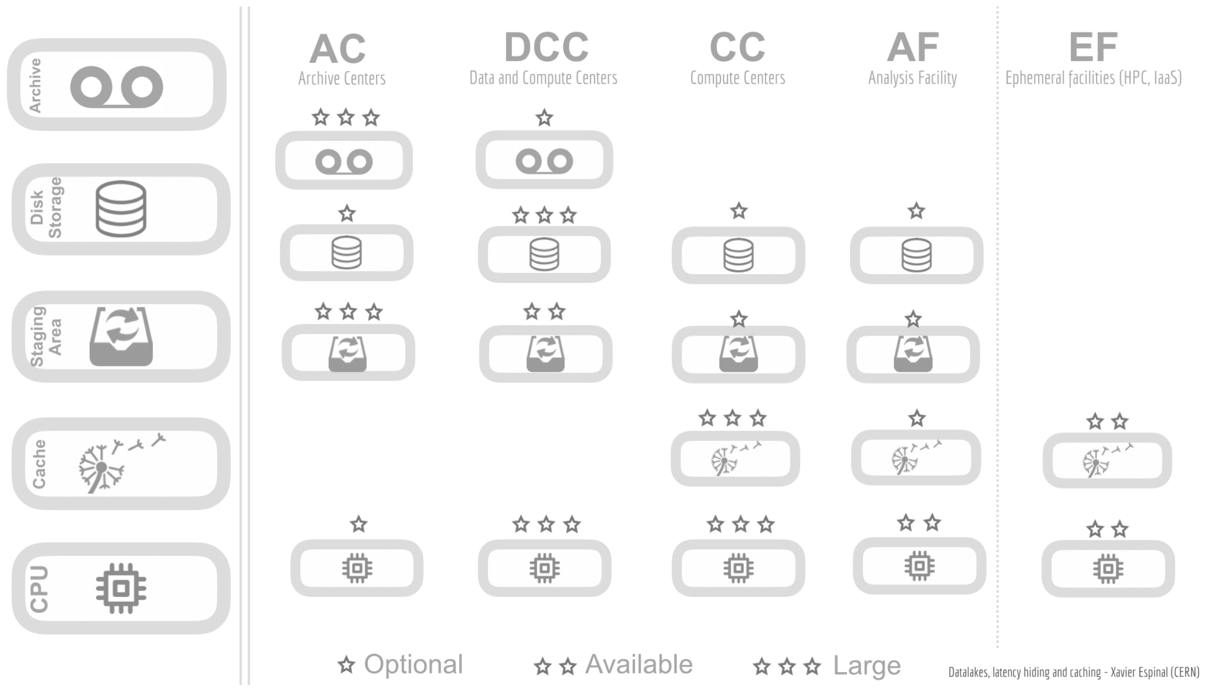
Based on the impact of latency on different workloads studies by our summer students we added some scenarios for two studied workloads. The summary for one example is summarised in the table below. Even workflows that suffer from more than an eightfold increase in wallclock time when the latency increases to 20ms, the average impact of reading data during the recovery periods is only 2.8 10^{-4}, even when the required data is located 60ms away.

Latency	1 ms	20ms	40ms	60ms
A1 relative wallclock increase	1.095	1.9	3.8	5.7
same for A2	1.0425	8.5	17	25.5
Average increase A1	1.04E-06	2.09E-05	4.18E-05	6.27E-05
Average increase A2	4.67E-06	9.34E-05	1.87E-04	2.80E-04

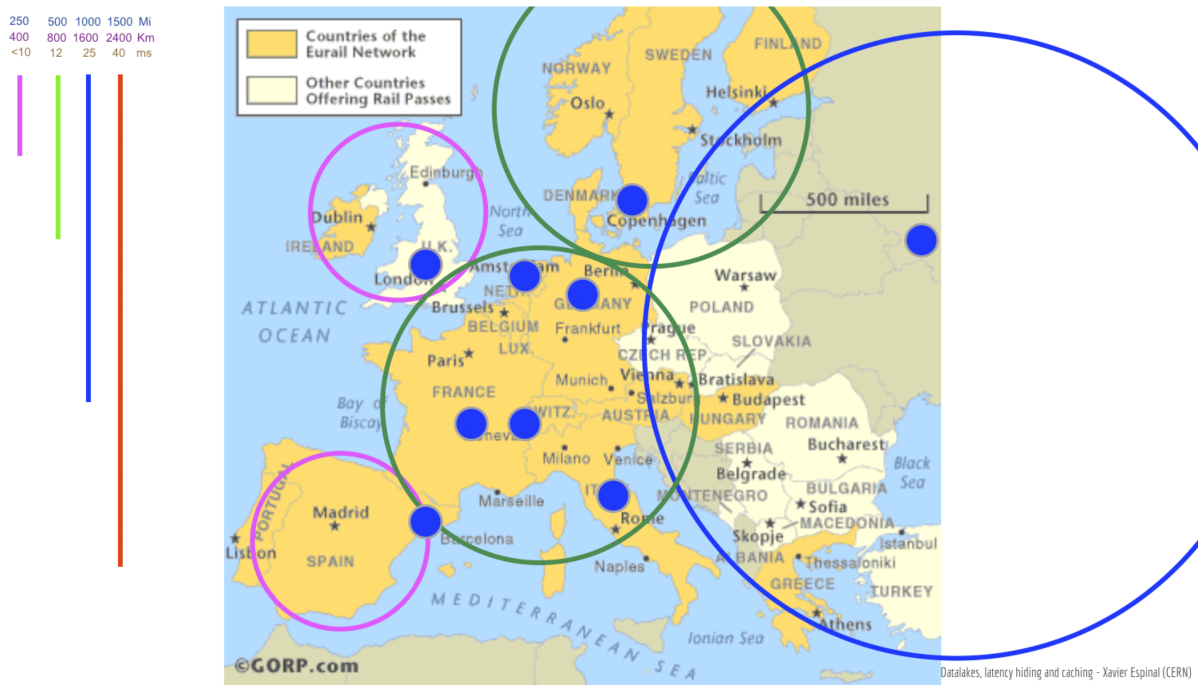
Data lake straw model sketch


File access orchestration: WN to Cache to Staging Area to Datalake
- Disk failures estimation: only 1% of data will be fetched outside the local datalake
- Efficiently hide latencies with buffering at the client side and access through cache+
CCC
WN cache+
CCDA
DCC+AC
DCC
1/3 mAOD
1/3 nAOD
1/nth RAW,AOD
datalake miss
cache miss
1 to 300ms
Datalake²
Datalake¹
Datalakeⁿ
Local access. Client buffering + file on the site cache+
Local cache miss. Client buffering + file streaming from DCC
Datalake miss (file loss). Client buffering + file streaming from DCC + datalake copy
<20 ms
Datalakes, latency hiding and caching - Xavier Espinal (CERN)

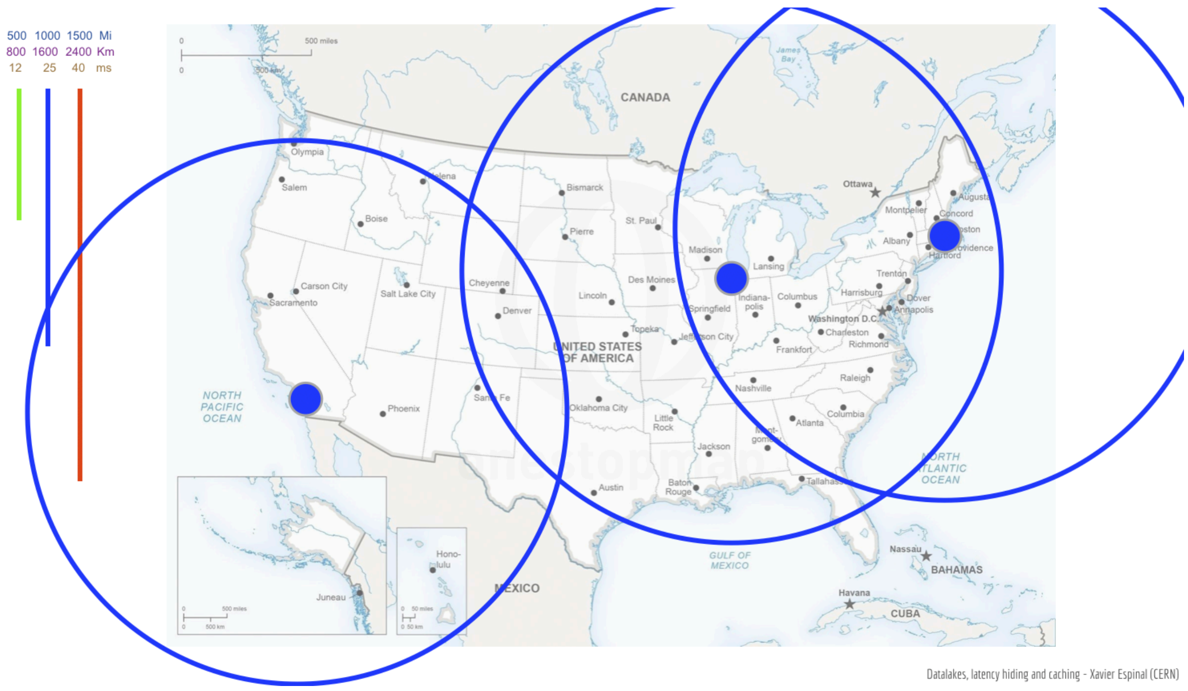
Site Classification



Aerial view of several latency areas



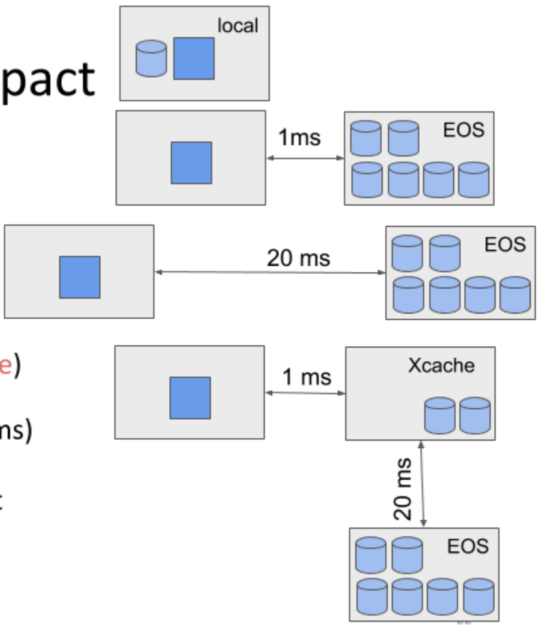
250 500 1000 1500 Mi
400 800 1600 2400 Km
<10 12 25 40 ms
Countries of the Eurail Network
Other Countries Offering Rail Passes
500 miles
ATLANTIC OCEAN
MEDITERRANEAN SEA
©GORP.com



500 1000 1500 Mi
800 1600 2400 Km
12 25 40 ms
CANADA
UNITED STATES OF AMERICA
NORTH PACIFIC OCEAN
NORTH ATLANTIC OCEAN
GULF OF MEXICO
MEXICO

Data Caching: Xcache impact

- Jobs:
  - ATLAS digi-reco 28GB input
  - ATLAS derivation Job 45GB input
- Setup
  - Data on WN (local)
  - Data on the same site (remote close)
  - Data at Wigner (remote far)
  - Xcache server at the same site (<1ms)
- Goal
  - Measure the impact on throughput



Measurements (Preliminary!!!)

Job type	Run's conditions	Run time	Relative Run time
Atlas-mcdigi-reco	Local Data	240m19s	1.00
	Remote Far	480m28s	**1.9**
	Empty Cache	261m39s	1.08
	Populated Cache	249m43s	1.04
	Local Data	147m19s	1.00
Atlas-derivation	Remote Far	1217m14s	**8.26**
	Remote Close	151m24s	**1.02**
	Empty Cache	155m17s	**1.05**
	Populated Cache	152m44s	1.03

Lesson learned: xcache can, for the tested workloads, hide latency efficiently

Missing:

- Extending the model to cover AOD and DAOD (+ntuple) based analysis
- List of functionality that is used in the model, but currently not implemented

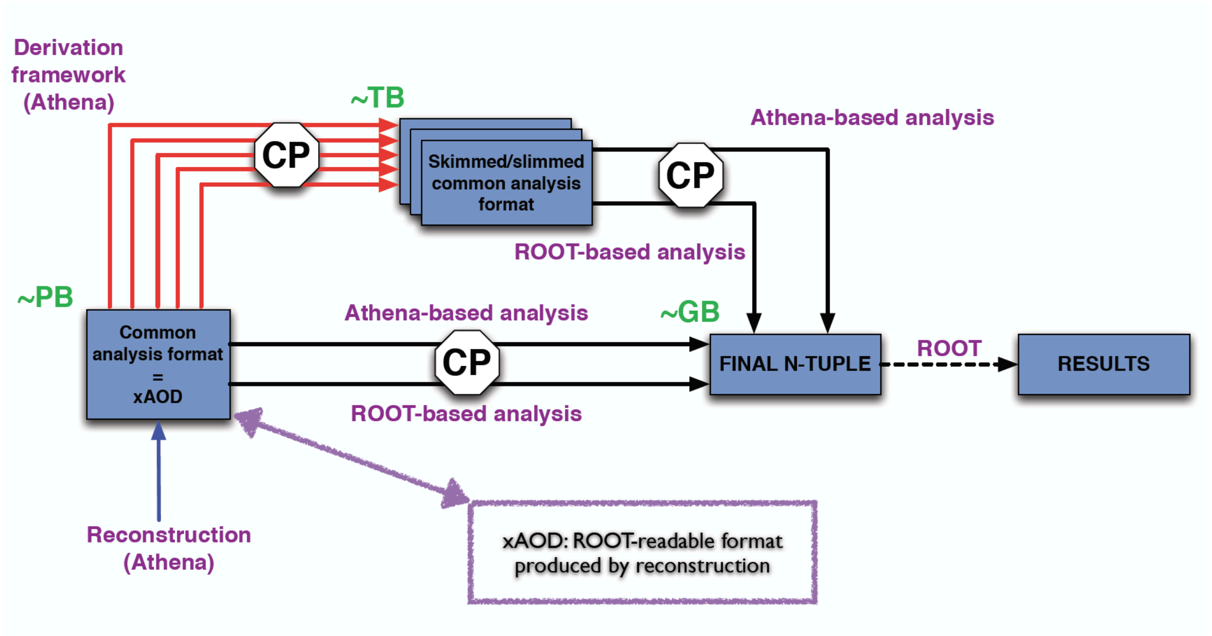
AOD/DAOD + ntuple based analysis in the Data Lake

The current Straw Man model is based on the CMS assumptions of data rates and Mini/Nano AOD sizes.
ATLAS analysis uses a combination of AOD/DAODs and recent (2017) monitoring data indicates that there is roughly the same space needed for AODs and dAODs. Together they use about 90PByte of disk space. This includes already the space needed for the replicas, with a replication rate for AODs/dAODs of (I have to look it up, but I remember that for 2018 data it was close to 2). HL-LHC will provide 5-7 times the nominal luminosity, but the trigger rate is expected to be increased by a factor of 10. In addition the increased pileup will have an impact on the size of AODs for this 2.5 is assumed. Applying these scaling factors based on the increase in luminosity and pileup we can assume that this style of analysis would require for the AOD/DAODs about 2300 PB. In this AOD based analysis model the AODs aren't accessed very frequently. The N-Tuples used in the final steps of analysis aren't included in the accounting. Given the indicated data volumes of the different data types the final N-Tuples are 10^{-6} the size of xAODs (GB/PB). While between each reduction step a factor 1000 is gained, the number of different formats has to be taken into account. For the intermediate analysis format (TB scale) ~100 formats are produced and for the final n-tuple (GB scale) ~1000. While the intermediate formats can be seen as transitory the full space doesn't need to be provided at the same time. In any case this would add at most 10% to the storage needed for xAODs (~50PB). For the n-tuples we can assume that ~2.3PB should be sufficient.
Given the uncertainties of conditions of our current understanding the space needed for these final analysis objects can be ignored at the moment and due to the small size and specificity to different analysis

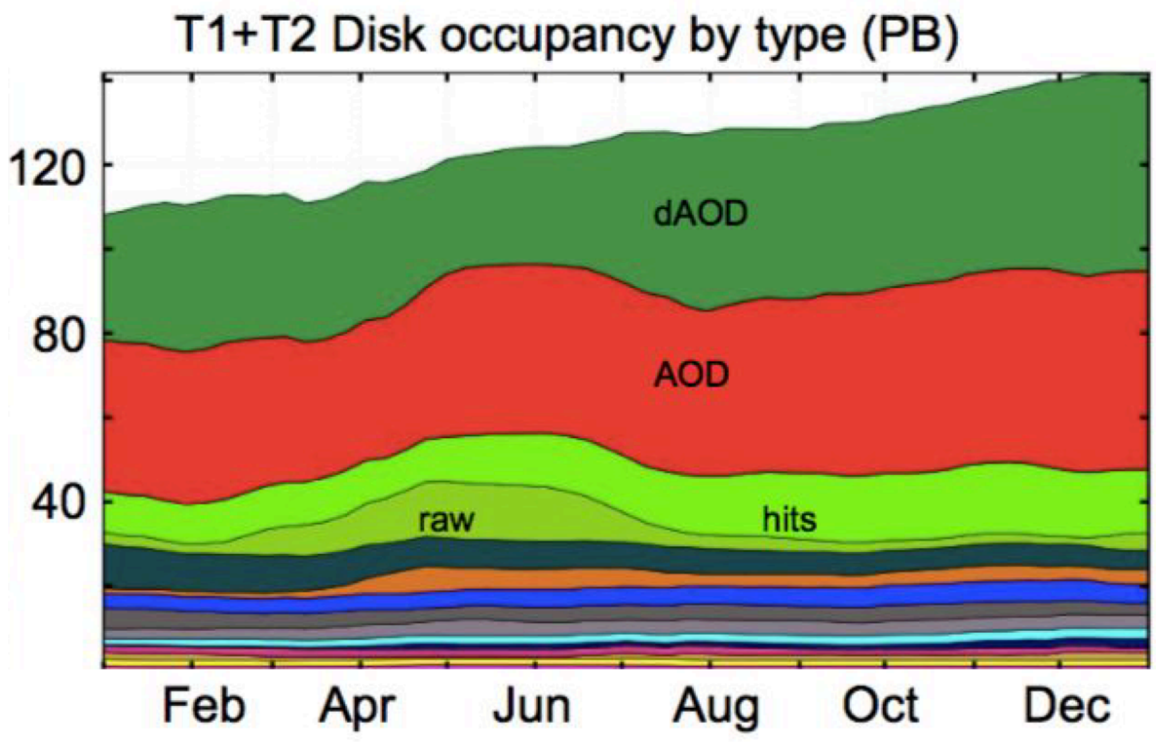
teams they should be probably stored outside the data lake with a very high (local) QoS level.

A complete working set of AOD/DAODs will not fit within a single Data Lake. The experiment that follows this model will have to split the data between different Data Lakes and schedule the Derivation and ROOT-based analysis accordingly. Having a distributed replica, the approach with no local data redundancy will still be feasible. In the Derivation step events are dropped (Skimming), objects from the event data tree (Thinning) and within the objects variables are removed (Slimming). While this is an I/O intensive process, data can be streamed to hide latency during access. Since the data is stored in ROOT format data from different events is stored intertwined and event's can't be skipped trivially. Test with an ATLAS derivation job run through an Xcache instance indicate that the impact of 20ms latency can be limited to less than 10%.

The modified spreadsheet indicates that the effect of data loss and non-availability …. Is different???? (I assume yes, one replica only→ increased restoration times…. Other effects should be close to the Mini/Nano use cases……….


Derivation framework (Athena)
~TB
CP
Skimmed/slimmed common analysis format
Athena-based analysis
CP
ROOT-based analysis
~PB
Common analysis format = xAOD
Athena-based analysis
~GB
CP
FINAL N-TUPLE
ROOT
RESULTS
ROOT-based analysis
Reconstruction (Athena)
xAOD: ROOT-readable format produced by reconstruction

Fig[ANT] AOD and N-Tuple based analysis model



2017 disk space usage by different data types, N-Tuples aren’t included