Xrootd/Lustre comparison using CMS analysis jobs

Giacinto Donvito
INFN-BARI

Outlook

- CMS job description
  - Configuration on CMSSW framework
- Xrootd:
  - Features and configuration
  - Performance
- Lustre:
  - Features and configuration
  - Performance
- Miscellaneous test and interesting scenarios

CMS Job Description

- The job used is:
  - An "analysis" job (MTR3), which reads ~40 branches (PDFs, {Gen,Calo,PF}Jets, Electrons, Muons, Photons, Tracks) and performs basic computations (invariant masses, track isolation), and produces no output
  - This produces a lot of pseudo random seek and very small read operation (from 4k to 64k)
- CMSSW used:
  - CMSSW_3_9_0_pre5
- Dataset used:
  - O(10TB) of MC data written with an quite old CMSSW version (3_6_x)

This is simply a typical "non optimised scenario" that is quite common in every-day analysis

CMS Framework configuration

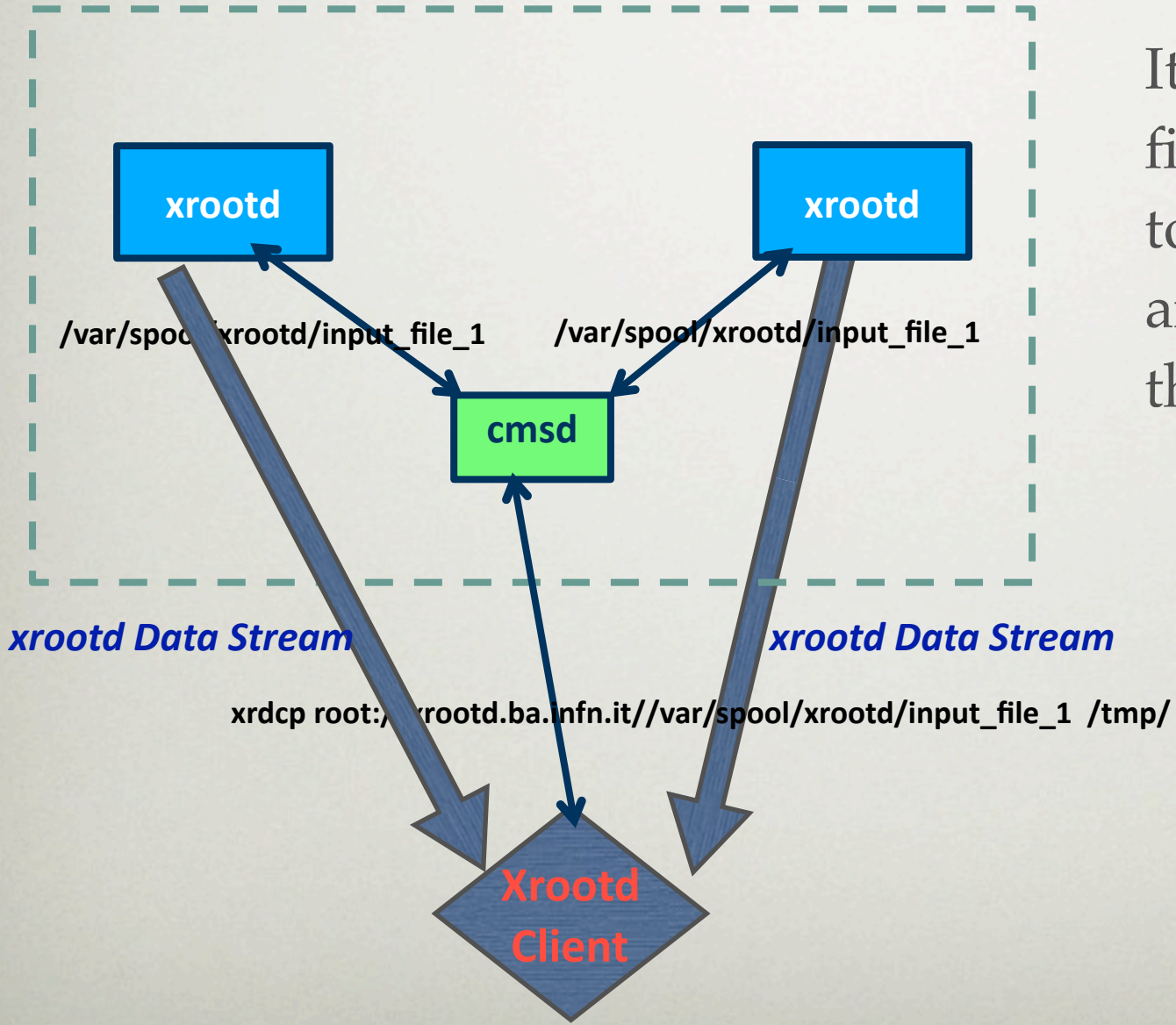
- Starting from the 3_8_x release, CMSSW allow few configuration in order to improve data access:
  - **cacheHint**
    - The cacheHint indicates how file caching requested in PoolSource.cacheSize should be implemented. Possible values are "application-only", "storage-only", "lazy-download" and "auto-detect".
  - **cacheSize**
    - Size of TTree read cache in bytes. If the value is the default zero, ROOT will not cache anything. If the value is non-zero, then the I/O layer caching options affect how the value is interpreted.
  - **readHint**
    - The readHint indicates how I/O reads should be performed. Possible values are "direct-unbuffered", "read-ahead-buffered" and "auto-detect".

Xrootd: Feature tested

- We are testing a couple of very interesting features of this storage software:
  - parallel stream (using more than one source server)
  - automatic caching files

- General comments:
  - Parallel stream is easy to implement:
    - it is enough to have multiple copy of the same file on different servers
  - Automatic caching is quite easy to configure and very flexible

Xrootd: parallel streams

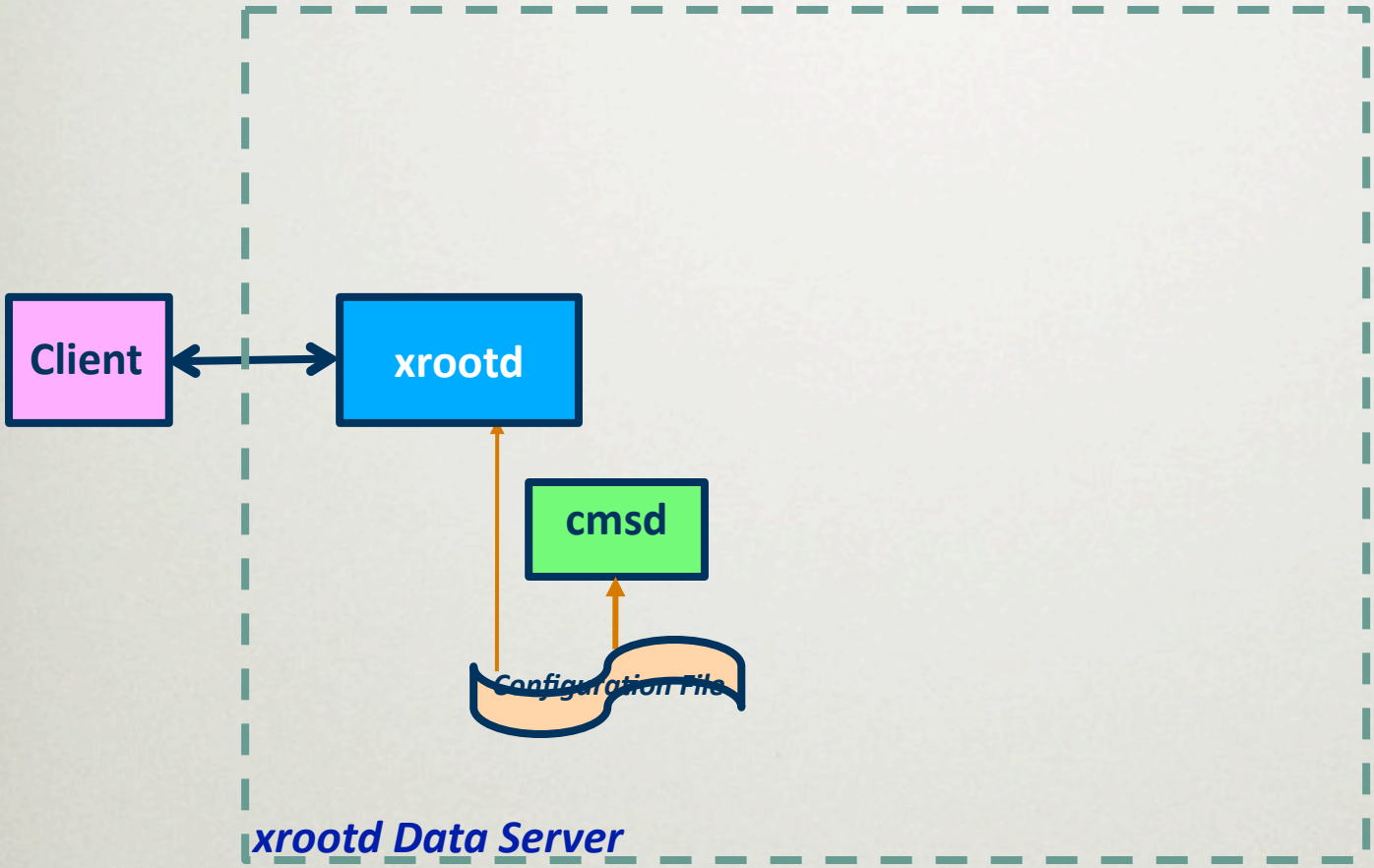


It is enough to have a file copied "manually" to more than one server and it will be used from the cluster

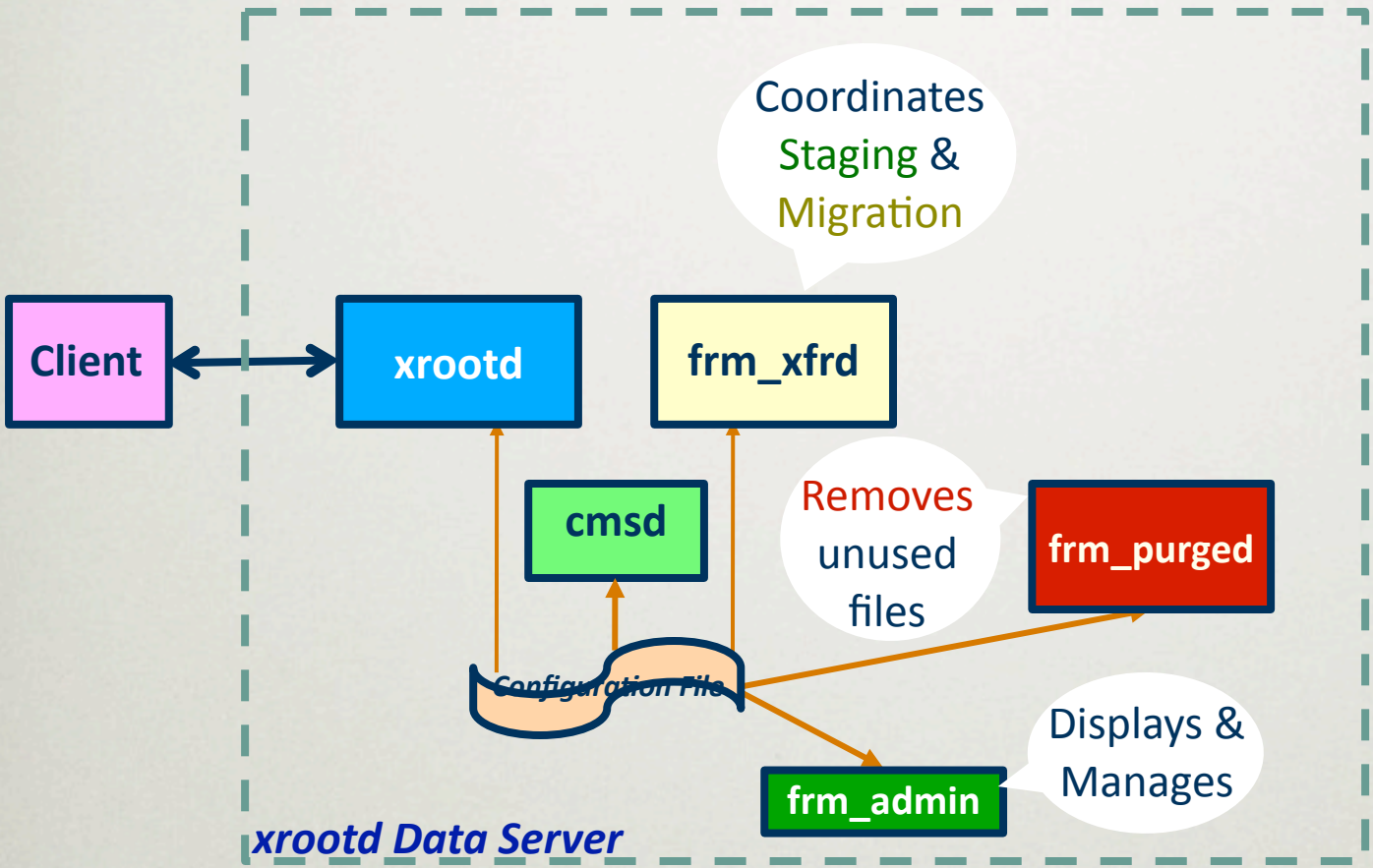
Xrootd: parallel streams

- *xrdcp -d 1 -f root://gridse14.ba.infn.it//mnt/sdh/0000/EC3C02B0-442C-DF11-97BB-000423D6BA18.root /tmp/*
  - ~20.1 MB/s
- *xrdcp -d 1 -S 12 -f root://gridse14.ba.infn.it//mnt/sdi/0000/EC3C02B0-442C-DF11-97BB-000423D6BA18.root /tmp/*
  - ~58.9 MB/s
- *xrdcp -d 1 -S 12 -x -f xroot://gridse14.ba.infn.it//var/spool/xrootd/mons.log1 /tmp/*
  - ~100.2 MB/s
- Using parallel streams increase the performance but requires much more CPU on the client side (typically a factor 3 in CPU utilization)
- "-x" allow reading from multiple servers
- this method could be used only with "xrdcp" command line

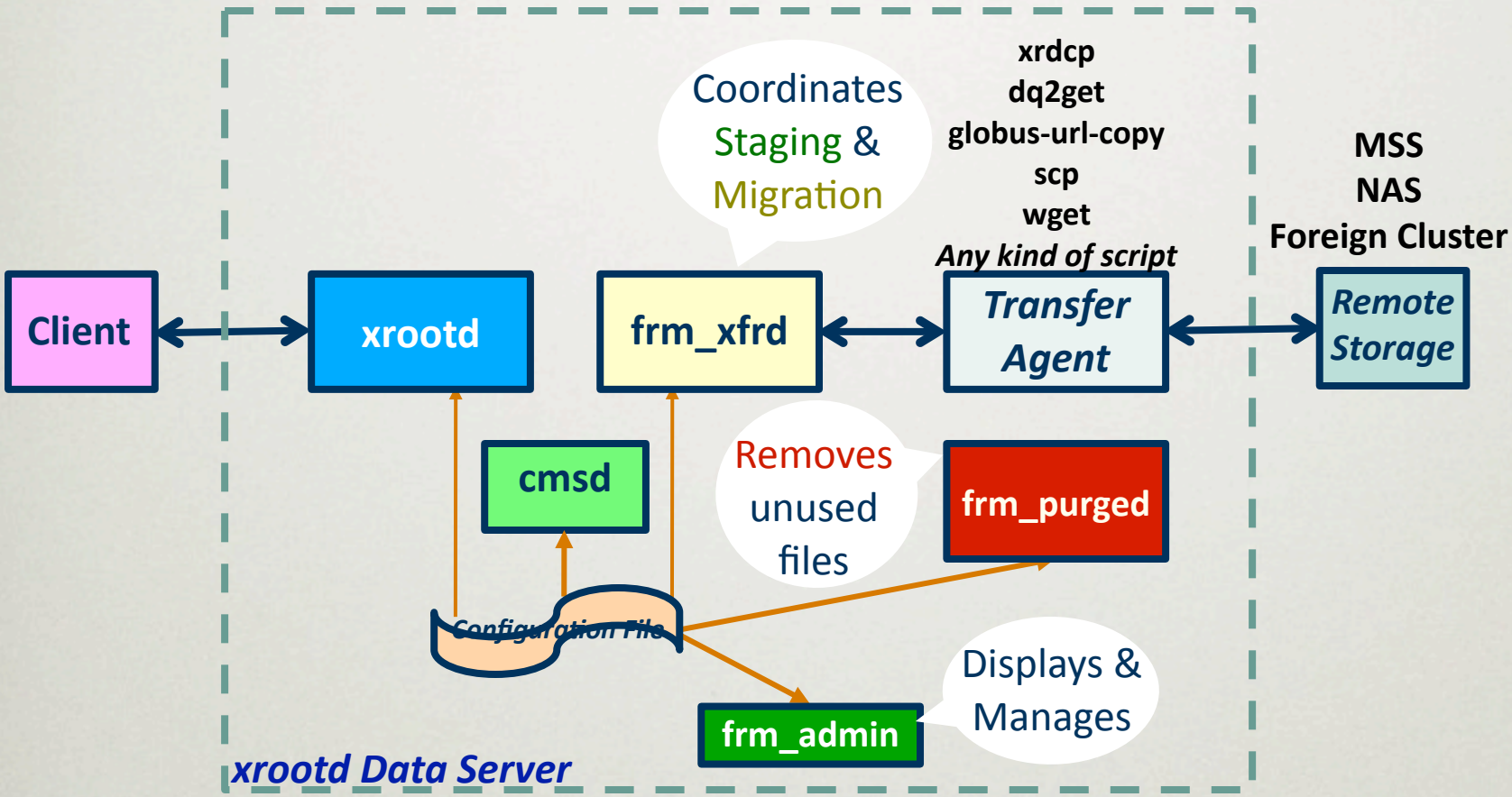
Xrootd: Automatic caching files



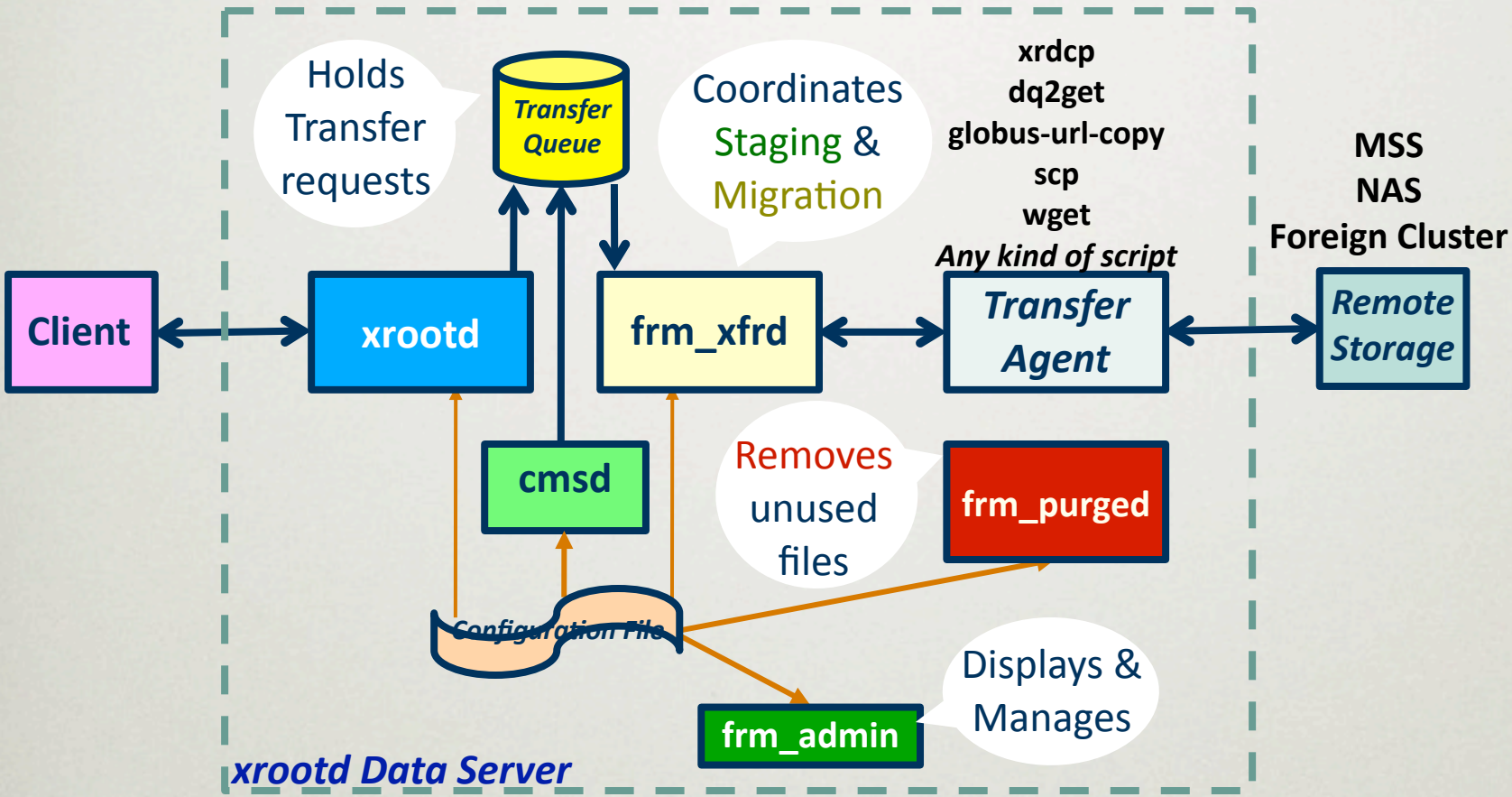
Xrootd: Automatic caching files



Xrootd: Automatic caching files



XROOTD: AUTOMATIC CACHING FILES



Xrootd: Automatic caching files

- You can easily specify (or customize) the command to be used, the source of the files:
  - mps.xfrcmd = /root/20100315-1007/bin/xrdcp %sfn %tfn
  - mps.mssdir root://origin_source.ba.infn.it/
- Each single request trigger a check on the local file-system:
  - cacheHIT: the file served immediately
  - cacheMISS:
    - an automatic copy is triggered while the client is waiting
    - as soon as the whole file is cached the client starts getting data

Xrootd: performance consideration

- MTR3 CMS job looks like very random application:
  - Small read operation
  - quite random read seek operation
- We measure the CPU efficiency during the run (CPUTime/WallTime)
  - Used bandwidth is not a good metrics
- Surprisingly big RAID5 with Fiber Channel controller performs worst than simple single SATA disk for a single job
  - It was difficult to obtain >40% in cpu efficiency using raid5
  - While it was easy to got 90% with a single disk
- The problem seems to be correlated with IOPS and stripe size on the controller
  - The initial test point is 1MB of stripe size

Xrootd: performance consideration

- Reconfiguring the raid to 256kb of stripe size we easily got 86% of CPU efficiency for a single job
  - "cacheSize" value="20048576" ## "cacheHint" value="storage-only" ## "readHint" value="read-ahead-buffered"
  - looking to the used bandwidth: a single job is able to read at about 3MB/s constantly
- We tested: xfs, ext3, ext4
  - no mayor differences observed
- We tried to run up to 120 concurrent jobs against the same server:
  - 100MB/s of aggregated bandwidth at maximum
  - ~40% of CPU efficiency

Xrootd: performance consideration

- It is clearly limited by disk IO
  - High I/O wait on the server
- The network is not a big issue here
- Changing the IO parameters in CMSSW do not add big improvements
- The raid controller under test do not support smallest stripe size
- This gives a measure of the scalability in “job-per-server” of the disk sub-system
  - maybe a single-disk configuration could give better performances
    - more test are still needed using “JBOD configuration”

Lustre: Feature

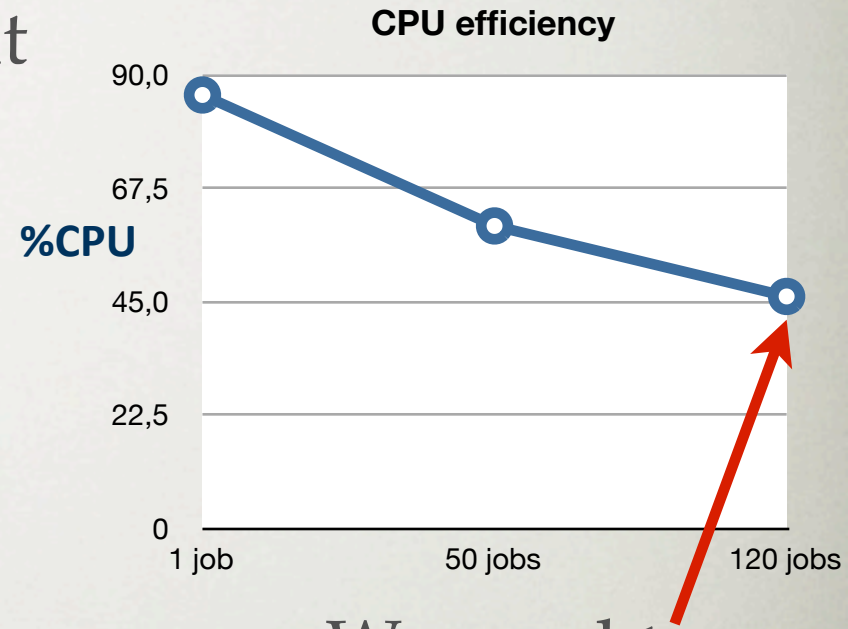
- Fully natively posix compliant
- Multi purpose file-system
  - experiments data and users data could share the same file-system
- Two different caches at Operative System level
  - on the server and on the client side
- Strong capability to re-order random I/O requests
- High performance on big files:
  - if needed a single file could be split on more than one server

Lustre: Performance

- Tuning a bit CMSSW parameters we easily got ~86% of CPU efficiency
  - "cacheSize" value="20048576" ## "cacheHint" value="lazy-download" ## "readHint" value="read-ahead-buffered"
  - Using a posix file-system the framework do not really download the files, but does only read-ahead-buffered
  - The configuration of the raid controller here do not affect to much the performance
- With this configuration a single job could read data with spikes of 50-60MB/s
  - there are, obviously, periods of time in which the job do not read data

Lustre: Performance

- In case of lustre, we observed that increasing the "cacheSize" could reduce the I/O on the disks
  - but this easily could become a bottleneck on the network
- For example running 120 jobs against a single disk server could require more than 250MB/s on the network
- If we reduce the "cacheSize" to 2MB this reduces the load on the network but increases the load on the disk subsystem



We need to repeat the test with a more powerful network infrastructure

Lustre: Performance

- Xrootd add a very small overhead in terms of amount of data transferred
- Lustre requires 2 time the same bandwidth

Hepix Tests

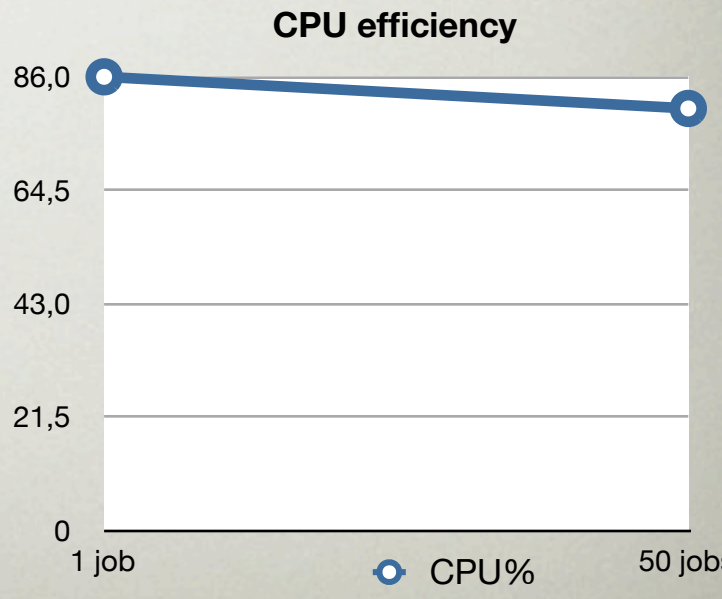
CMS

AFS	140 MB/sec	155 MB/sec	153 MB/sec	146 MB/sec
chu16.4G	262977 evs	277992 evs	267193 evs	252982 evs
NFS	56 MB/sec	[illegible]	77 MB/sec	86 MB/sec
tund.lazy	336000 evs	439417 evs	[illegible]	510183 evs
AFS/VIL4M	375 MB/sec	618 MB/sec	744 MB/sec	802 MB/sec
chu16.05	291375 evs	460075 evs	540152 evs	556916 evs
LUSTRE	168 MB/sec	170 MB/sec	170 MB/sec	
a_4M	[illegible]	[illegible]	679249 evs	
Xrootd	70 MB/sec	82 MB/sec	88 MB/sec	
auto	563798 evs	651225 evs	686875 evs	
Xrootd	69 MB/sec	81 MB/sec	88 MB/sec	
tuned	558414 evs	647442 evs	683015 evs	

- Also in this test the disk subsystem is the real bottleneck
  - The number of event is quite the same with lustre or xrootd

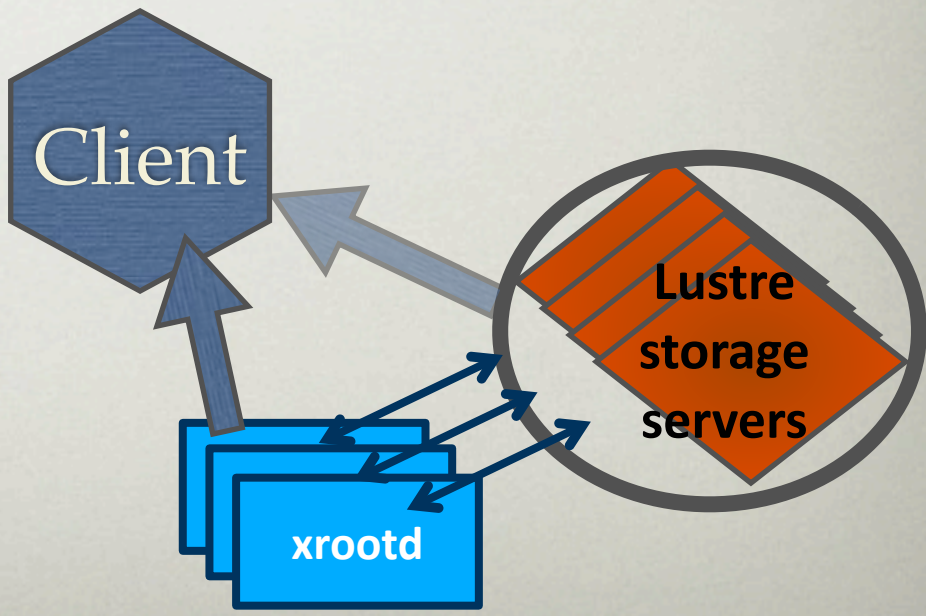
SSD test

- In order to be sure that we have a limitation on the storage sub-system we tested an SSD disk with an Xrootd server
  - a single MLC SSD (256GB) is able to provide data to 50 concurrent jobs without losing in CPU efficiency



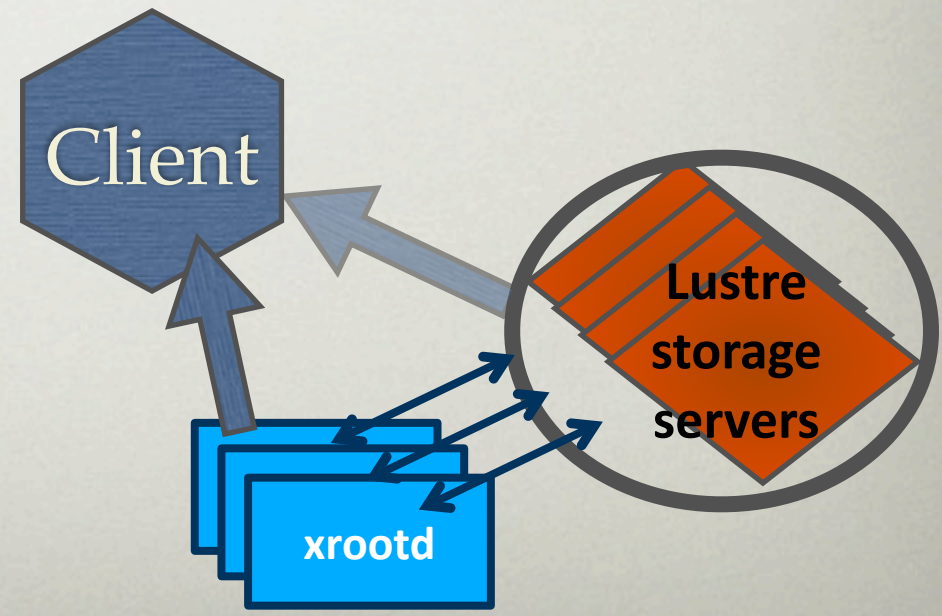
Xrootd over Lustre

- Xrootd over Lustre could be interesting as lustre adds some *missing feature* to Xrootd like:
  - *transparency of data access* (one xrootd server can go down and the data are still available)
  - *more uniform storage management* in a multi-VO computing farm
  - *posix compliance*



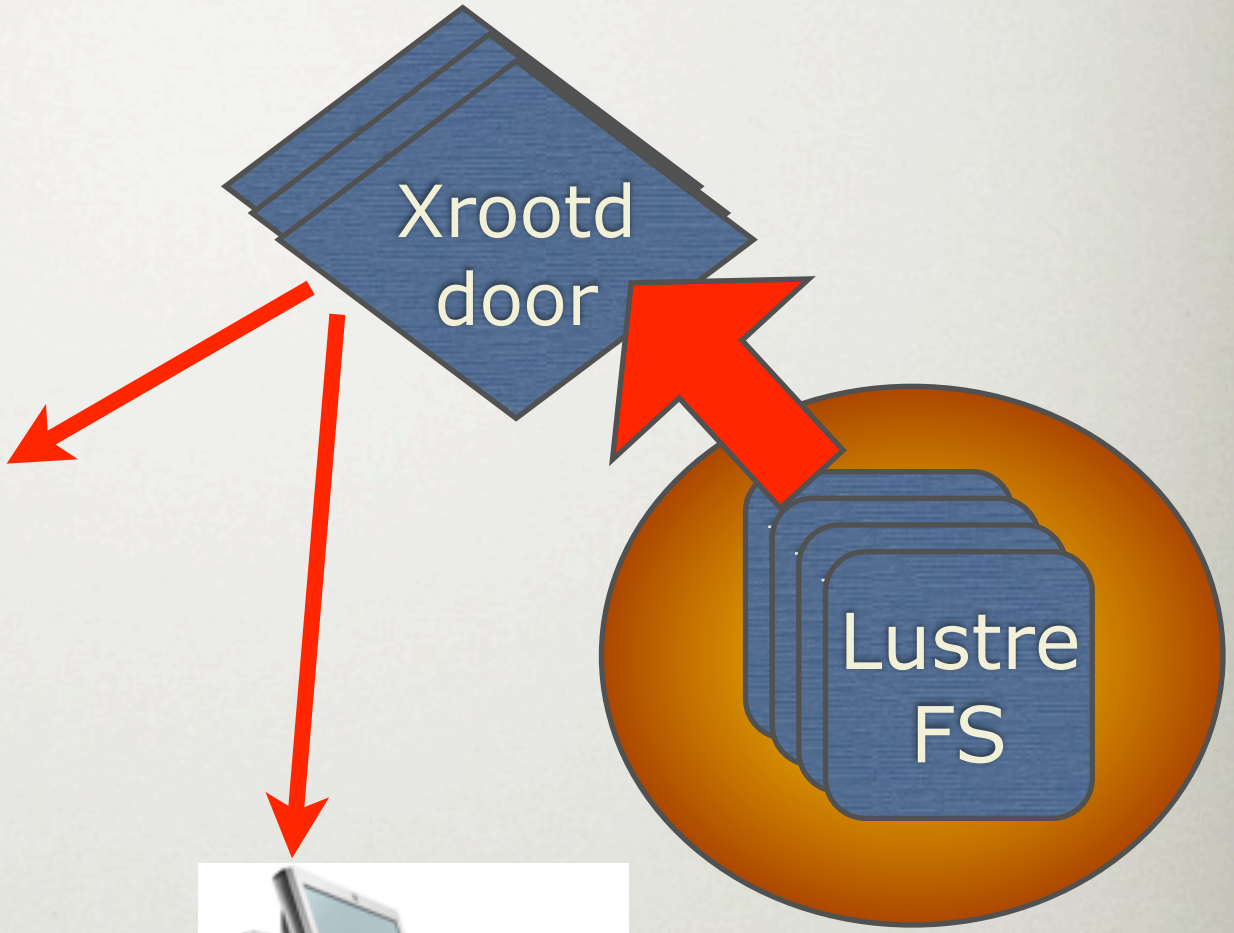
Xrootd over Lustre: performance

- Xrootd give at worst the same performance if the infrastructure is correctly tuned:
  - the network bandwidth among the xrootd doors and the lustre servers should never be a bottleneck
  - The "lustre read-ahead" on the xrootd machine should be tuned carefully looking at the real use case
    - as this could easily overkill the lustre servers

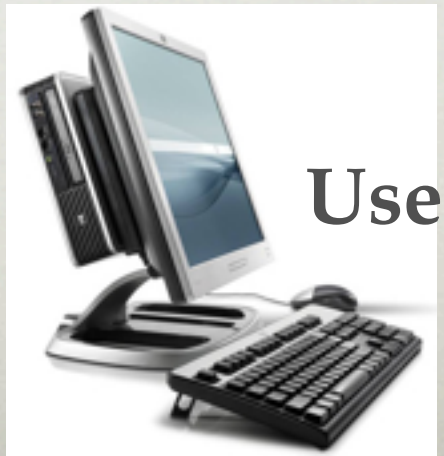


Tier2 - Tier3/Desktop data serving

Small Tier3

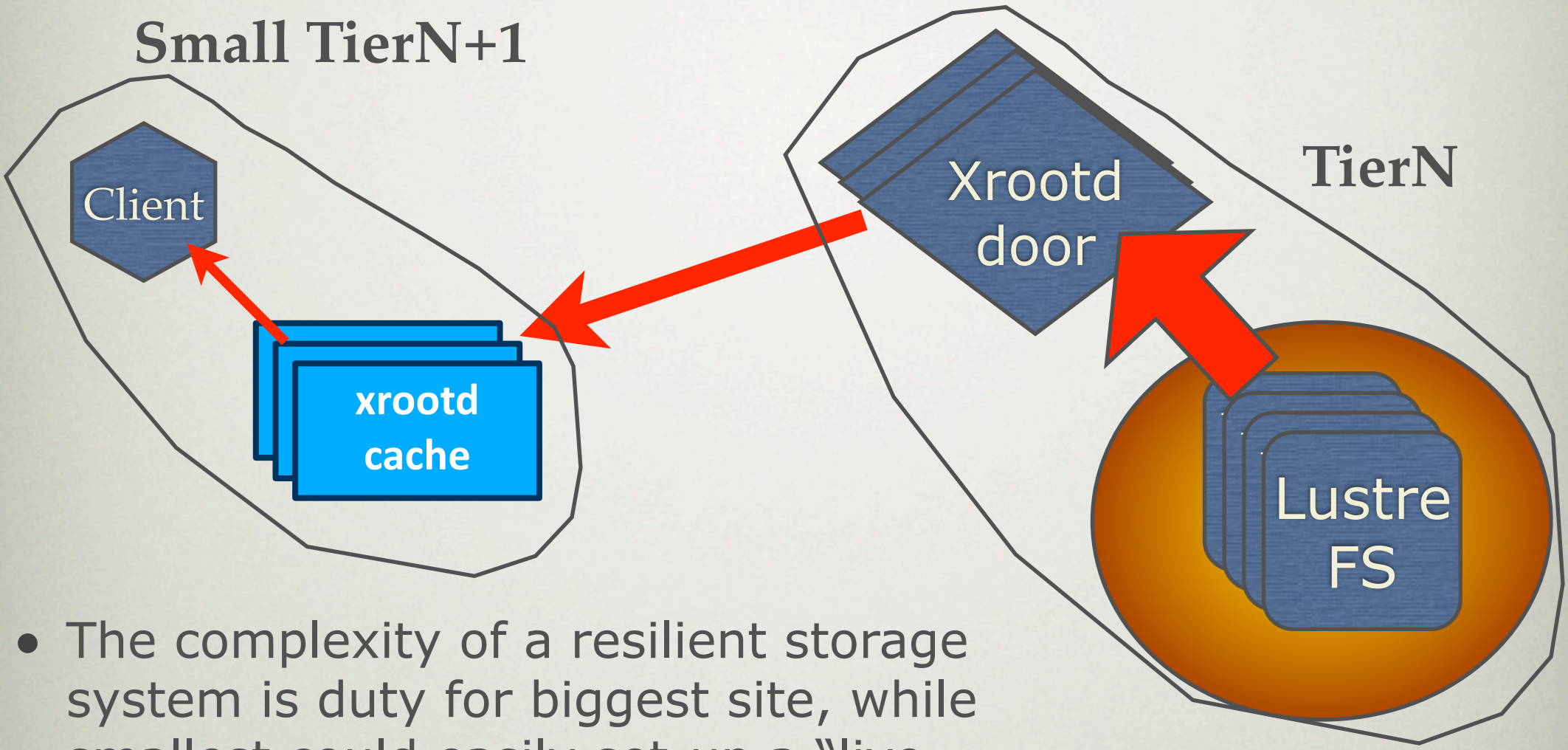


- The data hosted at T2 could be accessed by Xrootd remotely
  - Small Tier3 or Desktop could read data without a storage installation



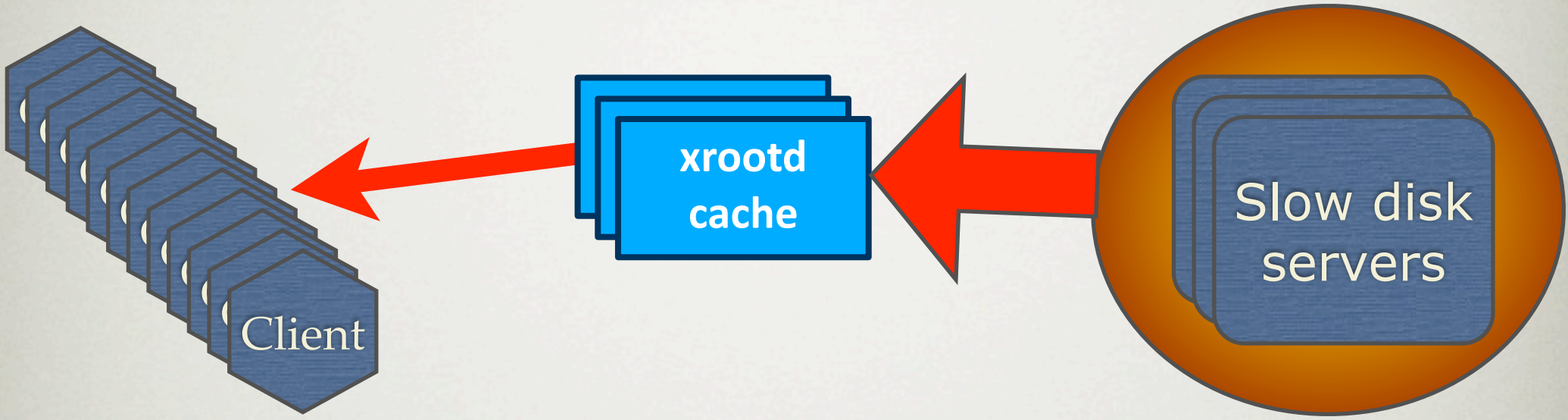
Users Desktop

Xrootd cache in multi-site environment



- The complexity of a resilient storage system is duty for biggest site, while smallest could easily set-up a “live-cache” of the “hot-data” required by the user, dynamically managed by the system

Xrootd cache in the same site



- SATA disks are becoming bigger but not faster while SAS/ SSD are getting cheeper
- while we cannot use the “Tiered storage” paradigm
  - to cache data depending on the requests of the users