Data Access in HEP
A tech overview

Performance and interoperability

Fabrizio Furano
Vincenzo Vagnoni

Outline

- The problem
- HEP-related sw architectural aspects
  - Offline part, jobs, data, bookkeeping etc.
  - Catalogues and metadata repositories
- Data access approaches
  - DBs, DFS, Web like, XROOTD
- "New" ideas to adapt to HEP
  - Direct access and proxying in WAN/LAN
  - Storage cooperation

The problem

- HEP experiments are very big data producers
- The HEP community is a very big data consumer
  - Analyses rely on statistics on complex data
  - Scheduled (production) processing and user-based unscheduled analysis
- Performance, usability and stability are the primary factors
  - The infrastructure must be able to guarantee access and functionalities
  - The softwares must use the infrastructures well
- Data access in NOT only storage
  - The functionalities interleave with the design of a computing model

Structured files

- In the present times most computations are organized around very efficient "structured files"
  - The ROOT format being probably the most famous
  - They contain homogeneous data ready to be analyzed, at the various phases of HEP computing
  - Centrally-managed data processing rounds create the "bases of data" to be accessed by the community
  - One site is not enough to host everything and provide sufficient access capabilities to all the users

Some architectural aspects

The minimalistic workflow

- …general, an user will:
  - …which analyses to perform for his new research
  - …e which performs it
    - …level macro or a "simple" plugin of some ex… …d software framework
- Ask a sy… …t the data requirements
  - Whi… files contain the needed information
- As… …ther system to process his analysis
- C… …esults

Typically an experiment-based metadata repository and/or file catalogue

Typically the GRID (WLCG) or batch farms or PROOF

This will likely become also his own computer as the hw performance increases and the sw uses it efficiently

Semi-automated approach

- One can:
  - Choose carefully where to send a processing job, e.g. to the place which best matches the needed data set
  - Use tools to create local replicas of the needed data files
    - In the right places
    - Eventually use tools also to push new data files to the "official" repositories
    - If overdone this can be quite time and resource consuming
- Any variation is possible
    - E.g. pre-populate everything before sending jobs

Direct approach

- We might want not to be limited by local access
  - I.e. not pre-arrange all the data close to the computation
- The technology allows more freedom if:
  - Everything is available r/w through URLs
    - proto://host/path
  - The analysis tools support URLs to random access files
    - The I/O is performed at byte level directly in the remote file
  - The analysis tools exploit advanced I/O features
    - E.g. ROOT TTree + TTreeCache + XRootd
- Again, if overdone, the WAN could become a limiting factor
  - Less and less… the bandwidth is increasing very fast
  - HEP data files are big and quite static, better to exploit locality if possible
  - Balance what needs to be pre-copied in the site with what can be accessed remotely
    - Perfect solution for the ALICE condition data

Direct approach: the pitfall

- To use URLs we must know them fully
  - Proto://host/path/filename
  - We may know the filename but not the hosting site
  - With the WWW we use a search engine (e.g. Google) for a similar problem
    - It can be seen as our unified entry point, to know where an information is (might be)
  - With HEP data the problem is a bit different: the matches must be exact, not just a good hint, moreover:
    - Replicas should have the same path/filename, eventually in different places
    - If not, "something" must keep track of this massive worldwide aliasing (very misfortuned idea)

Where is file X

- The historical approach was to implement a "catalogue" using a DBMS
  - This "catalogue" knows where the files are
    - Or are supposed to be
- This can give a sort of "illusion" of a worldwide file system
  - It must be VERY well encapsulated, however
    - One of the key features of the AliEn framework
  - It would be nicer if this functionality were inside the file system/data access technology
    - No need for complex systems/workarounds

Approaches to access data

Databases and data

- A DB as a "structured, heterogeneous base of data"
  - From the BaBar experience it became clear that putting *everything* in a relational or object-based orthodox DB was not a good choice
    - Versatility and expressive power comes at the expenses of performance, ease of maintenance and scalability
    - An insufficient performance also can cause big frustration and big system instabilities
    - Difficult to scale them at these extreme levels
      - Also the cost does not scale linearly

Databases and storage

- In very simple words, HEP data is composed by:
  - A bookkeeping repository able to do searches
    - A relational/OO DB is perfect for that (order of 10^8 files per experiment)
  - A file-based data repository
    - Served by an efficient and scalable data access system
      - ALL the performance of the hardware (disk pools) must be usable
        - And scale linearly with it
      - If an app computes 15MB/s the disk will have to 'see' 15MB/s for it, not more.
        - "Computes" means "computes", not "transfer" or "consume". This is very important.

Distributed FSs

- Several sites chose to have local storage clusters managed by mainstream DFSs
  - E.g. Lustre/GPFS/NFS/ …
  - Good performance for local clients, generally bad for WAN random access
    - It relies generally only on sequential read ahead
  - No functionalities to aggregate sites
  - Need for a SITE gateway for other protocols
    - Very common architecture for large HTTP sites
    - Several (often good) experiences with XROOTD
  - Files are not stored as they are
    - DFSs use their own policies to distribute data chunks among disks
    - This can help performance in some cases
    - Questionable data management/disaster recovery. Need their software infrastructure working to do anything

Xrootd and Scalla

- In 2002 there was the need of a data access system providing basically:
  - Compliance to the HEP requirements
  - "Indefinite" scaling possibility (up to 200Kservers)
  - Maniacally efficient use of the hardware
  - Accommodate thousands of clients per server
  - Great interoperability/customization possibilities
- In the default config it implements a non-transactional distributed file system
  - Efficient through LAN/WAN
  - Not linked to a particular data format
    - Particularly optimized for HEP workloads
    - Thus matching very well the ROOT requirements

What XROOTD can do NOW

- Build efficient local storage clusters, virtually no limit to scalability
- Aggregating storage clusters into WAN federations
- Access efficiently remote data through WAN
- Push files SE-to-SE without datamovers or external systems
- Build proxies which can cache a whole repository
  - And increase the data access performance (or decrease the WAN traffic) through a decent 'hit rate'
- Build hybrid proxies
  - Caching an official repository while storing local data locally

Performance in data access

- The Scalla/xrootd project puts great emphasis in performance. Some items:
  - Asynchronous requests (can transfer while the app computes)
  - Optimized vectored reads support (can aggregate efficiently many chunks in one interaction)
  - Exploits the 'hints' of the analysis framework to annihilate the network latency
    - And reduce the impact of the disks' one by a big factor
  - Allows efficient random-access-based data access through high latency WANs

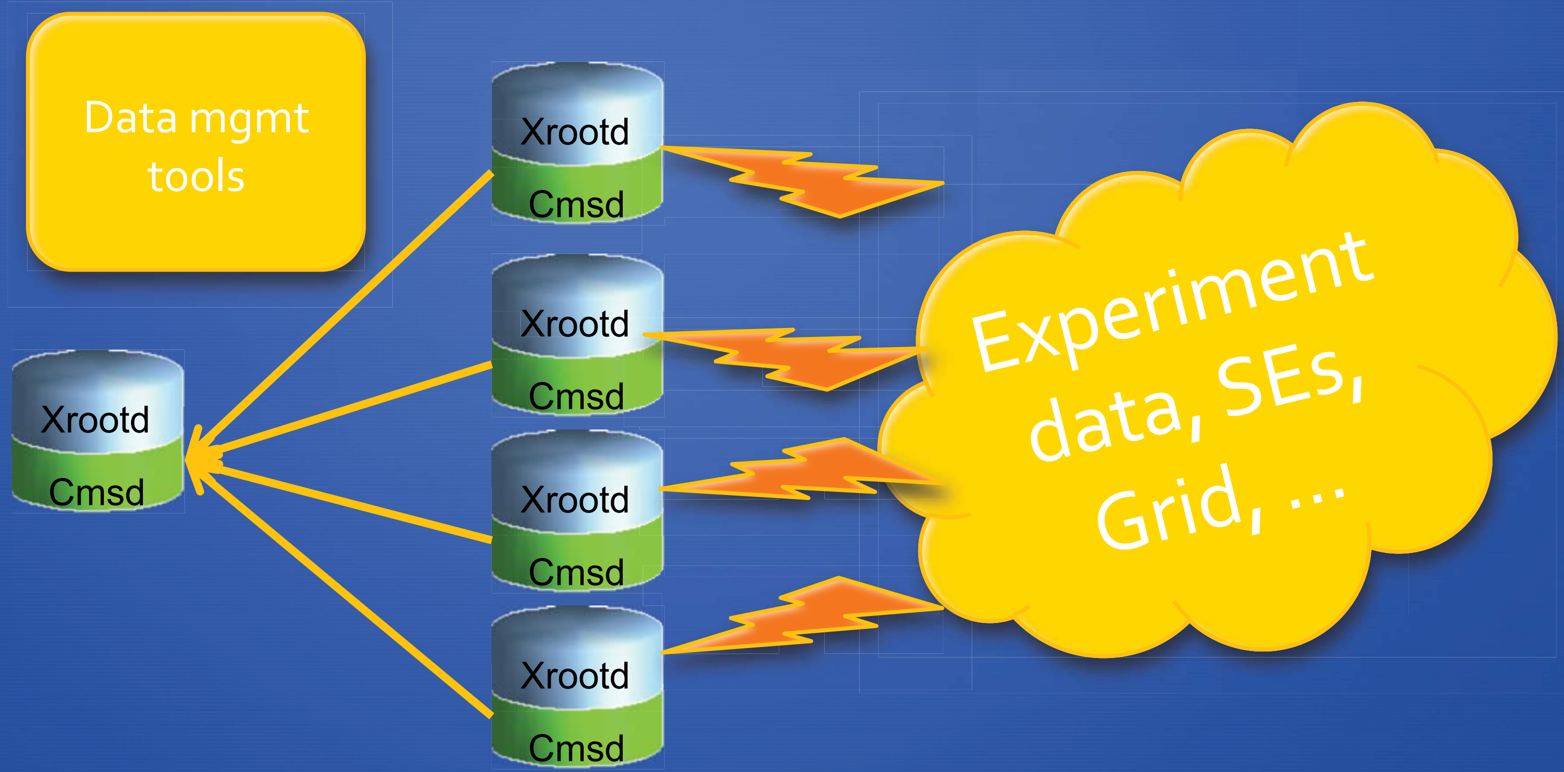
Current and future evolutions

HEP forward proxies

- Analysis clients work at a site
- The local storage, accessed through URLs acts as a proxy of the worldwide storage
  - A local r/w cache
  - In practice, if a file is missing, it is 'fetched' from an external system
  - Or a file can be 'requested' to appear
  - Must have a sufficient size to reduce the "miss rate"
  - Efficient data movement tools can populate it as well

The "*AF" storage

- Data is proxied locally to adequately feed PROOF
- Very generic design, ALICE@{CAF, SKAF, LAF}, ATLAS@{Wisconsin, BNL, lightweight T3 design}, ...



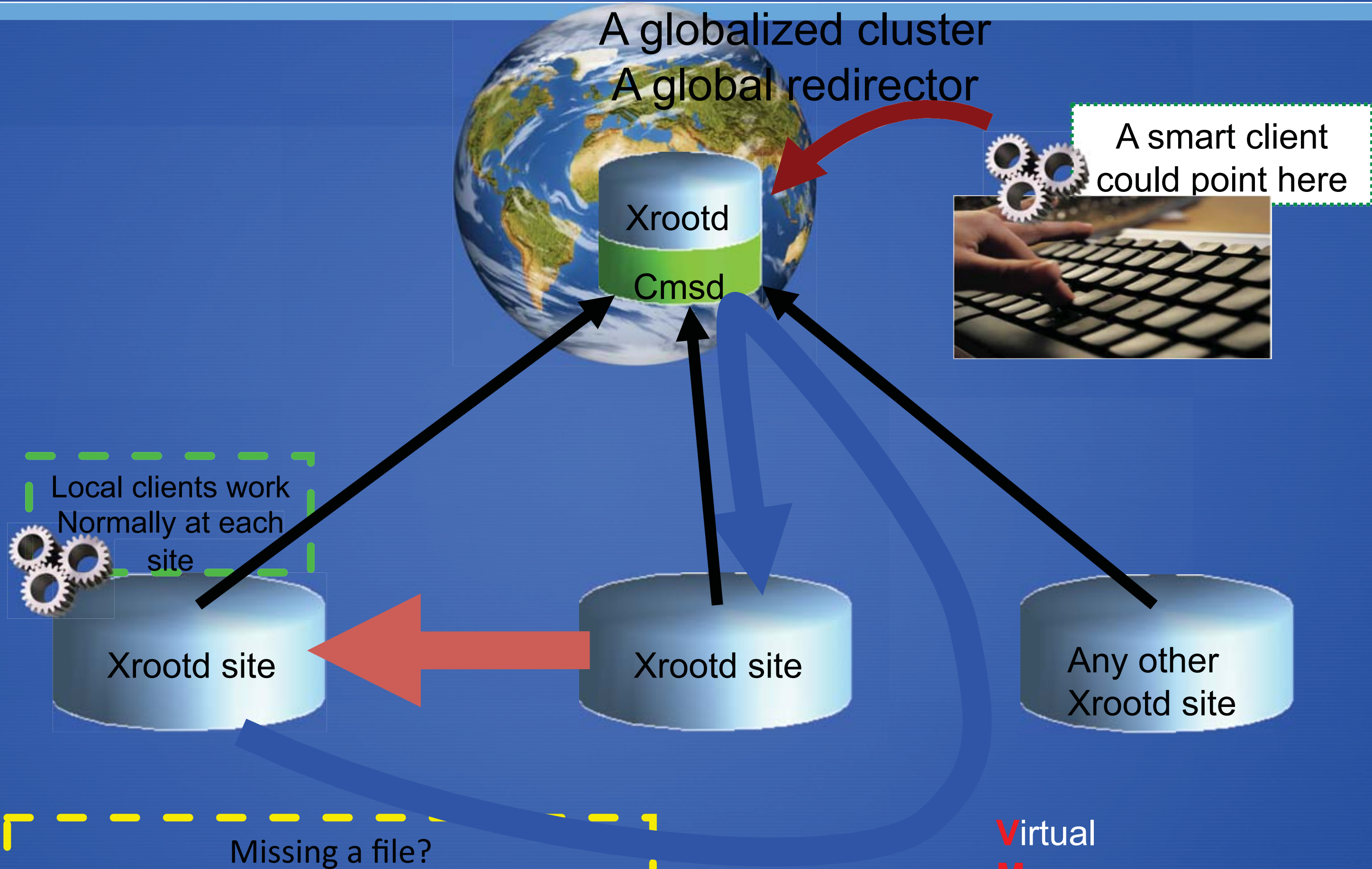
Federations

- Suppose that we can easily aggregate remote storage sites
  - And provide an efficient entry point which "knows them all natively"

- We could use it to access data directly
  - Interesting idea, let's keep it for the future

- We could use it as a building block for a self-referring federation
  - If site A is asked for file X, A will fetch X from some other 'friend' site, though the unique entry point
  - A itself is a potential source, accessible through the entry point

The VMSS



Missing a file?
The storage asks to the global redirector
Gets redirected to the right
collaborating cluster, and fetches it.
Immediately.

Virtual
Mass
Storage
System
… built on data Globalization

Looking forward

- Proxying is a concept, there are basically two ways it could work:
  - Proxying whole files (e.g. the VMSS)
    - The client waits for the entire file to be fetched in the SE
  - Proxying chunks (or data pages)
    - The client's requests are forwarded, and the chunks are cached in the proxy
- In HEP we do have examples of the former
- It makes sense to make also the latter possible
  - Some work has been done (the original XrdPss proxy or the newer, better prototype plugin by A.Peters)

An idea for an analysis facility

- It contains:
  - Institutional data: proxied
  - Personal user's data: local only
- Data accessible through:
  - FUSE mount point if useful
  - Simple data mgmt app
  - Native access from ROOT (more efficient)
- Accessible through WAN
  - Possibility to travel and still see the same repository from the laptop
- Federable with friend sites

Greedier data consumers

- In the data access frameworks (e.g. ROOT) many things evolve
- Applications tend to become more efficient (=greedier)
- Applications exploiting multi/many core CPUs will be even more
  - An opportunity for interactive data access (e.g. from a laptop)
  - A challenge for the data access providers (the sites)
  - The massive deployment of newer technologies could be the exciting challenge for the next years

Components

- The XROOTD protocol usage is gaining importance
  - Many kinds of components to design massively distributed data access systems
  - Born in BaBar, to support local site data access
  - Evolved to a high performance platform for globally distributed storage
  - Challenge: create/evolve the newer components, e.g. :
    - chunk-based and file-based proxies
    - What about a personal cache/proxy ?
    - Bandwidth/queuing managers
  - Goal: a better experience with data

Conclusion

- "The next level in Storage+Data Access" …
  - A Web-like functional level, tailored to the hyper-tough HEP requirements
  - Very good examples right now, many others are coming
  - Interoperability and performance

Thank you

Questions?

The Data Management

- Files and datasets are stored into Storage Elements, hosted by sites
  - The decision is often taken when they are produced
- Processing jobs are very greedy
  - Up to 15-20 MB/s now.
- The GRID machinery (ev. Together with some service of the experiment) decides where to run a job
  - The service can also be human-based (!)
- Matching the locations of the data with the available computing resources is known as the "GRID Data Management Problem".

An example

My distributed sites
With pre-filled storage
With computing farms

My data greedy jobs

The “Line of Fire”

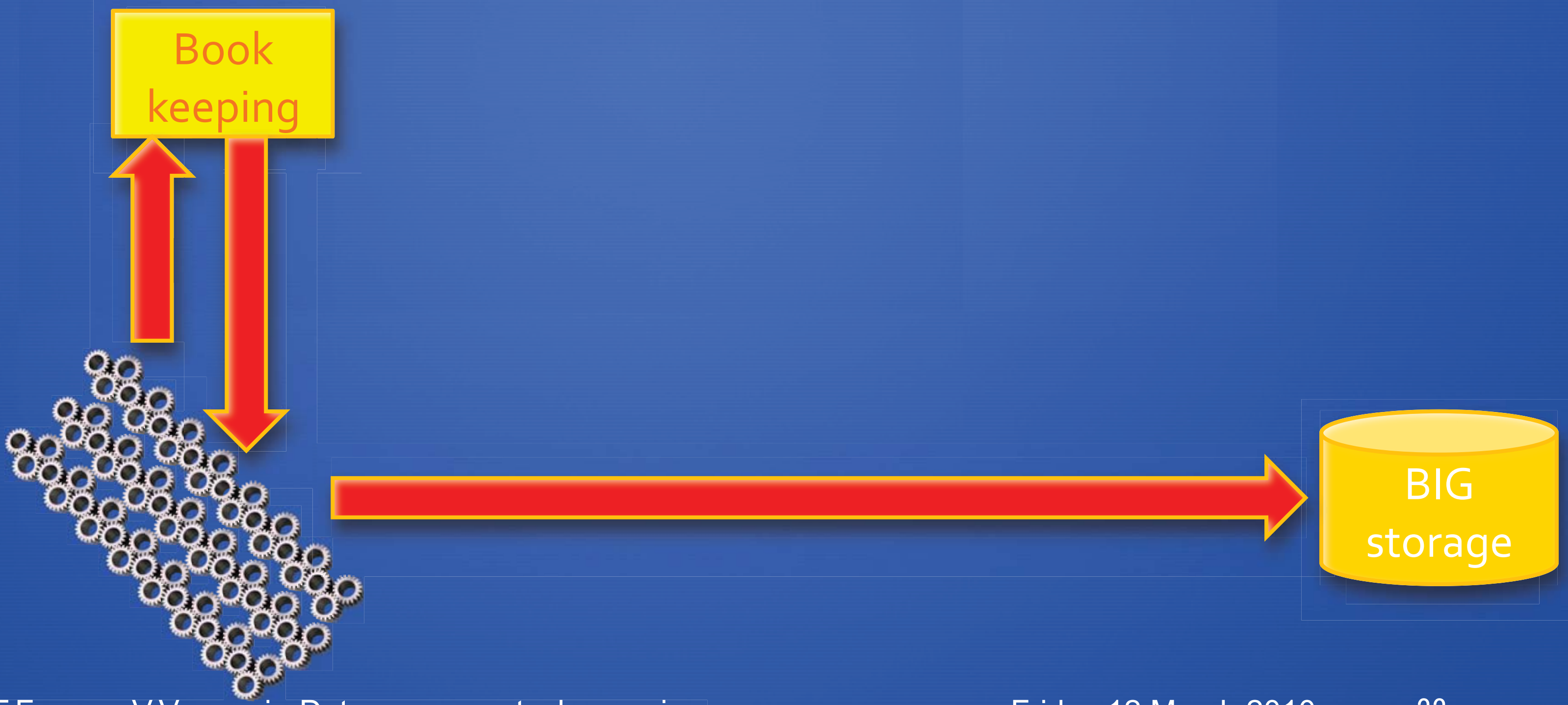
- A very common pitfall: “we can translate all the requests towards the SE as they come, so we can implement a relational DB-based system which spreads the load through N data servers or N storage systems”
  - In practice, it stores the exact location(s) of each file
  - It may work in principle, but it may be as demanding as serving the data. Very difficult to accommodate in sites with varying service levels.
  - Also, such an external system cannot reflect unexpected changes (e.g. a broken disk)

The "Line of Fire"

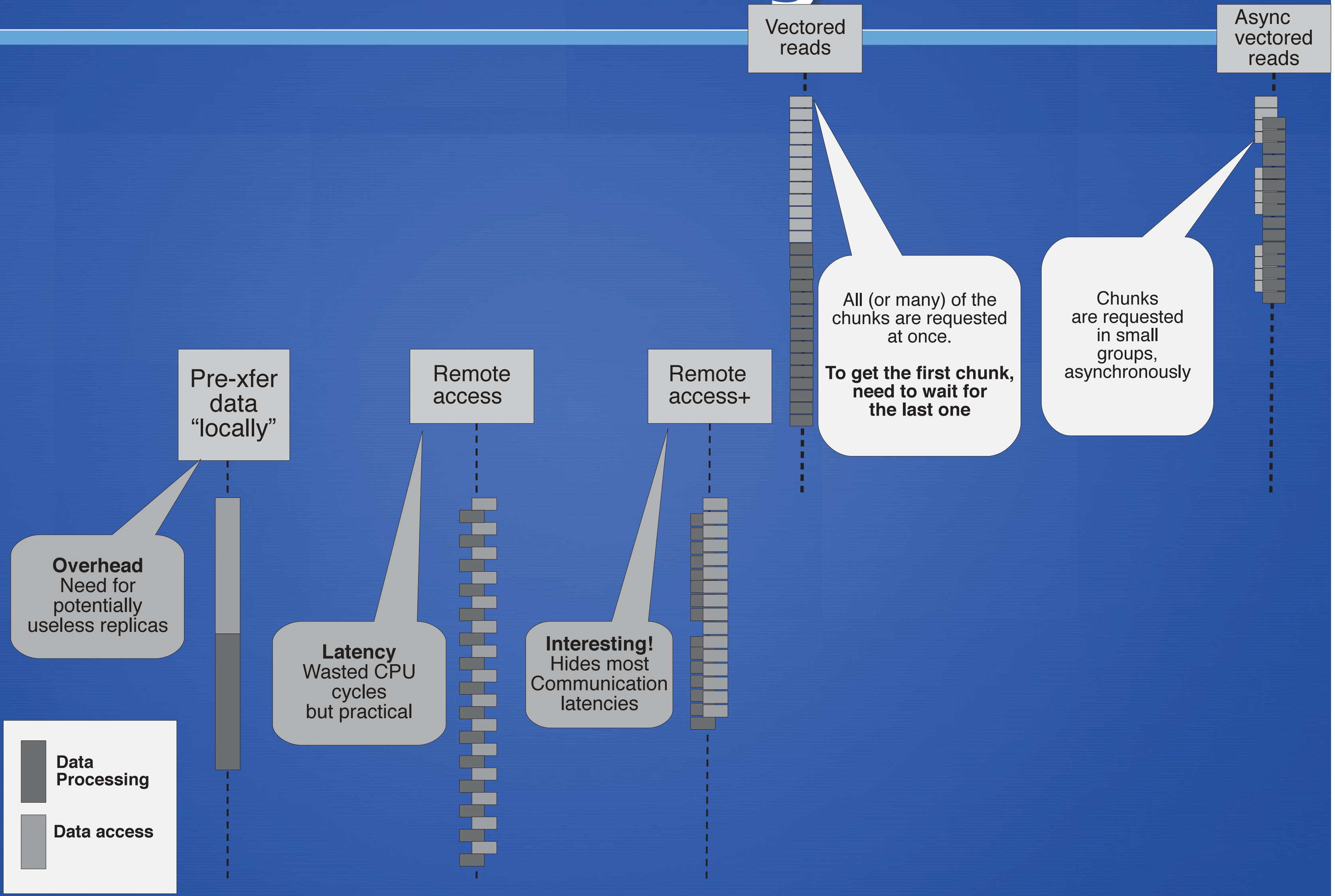
- Designs, informally, the architectural position in the front of a storage element, directly exposed to the load coming from the processing jobs
  - Very delicate position where to put any system
  - The transaction rate (open) can get up to 2-3K per second per site
  - Eventually, the load will accumulate until…

Avoiding troubles

- A slightly different architecture which makes the difference
  - In the "Line of fire", the simpler, the better
  - Processing jobs pre-prepare themselves before accessing the (worldwide or local) storage



Access strategies



WANs are difficult

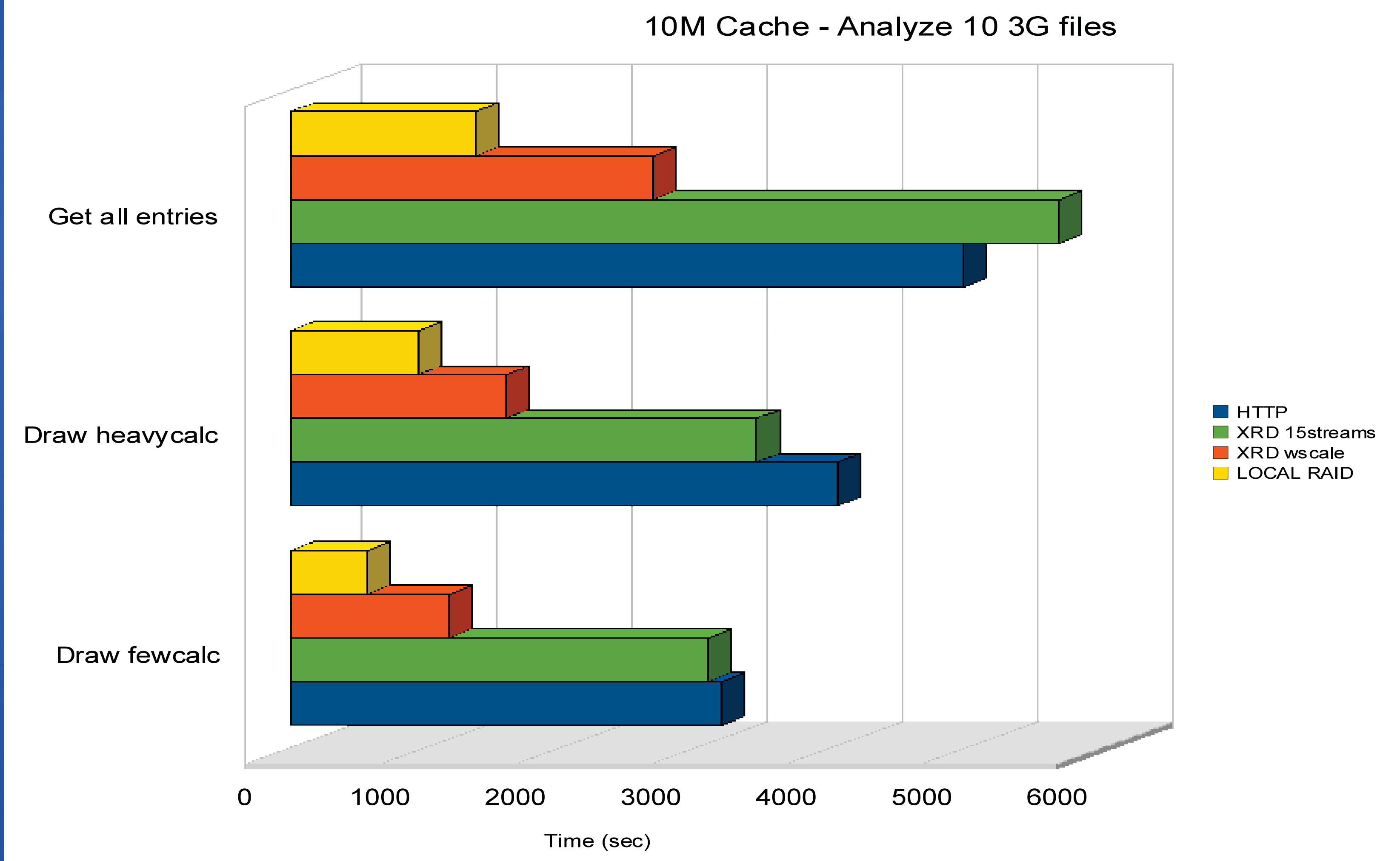
- In WANs each client/server response comes much later
  - E.g. 180ms later
- With well tuned WANs one needs apps and tools built with WANs in mind
  - Otherwise they are walls impossible to climb
    - I.e. VERY bad performance... unusable
  - Bulk xfer apps are easy (gridftp, xrdcp, fdt, etc.)
  - There are more interesting use cases, and much more benefit to get
- ROOT has the right things in it
    - If used in the right way

Exercise (from CHEP 09)

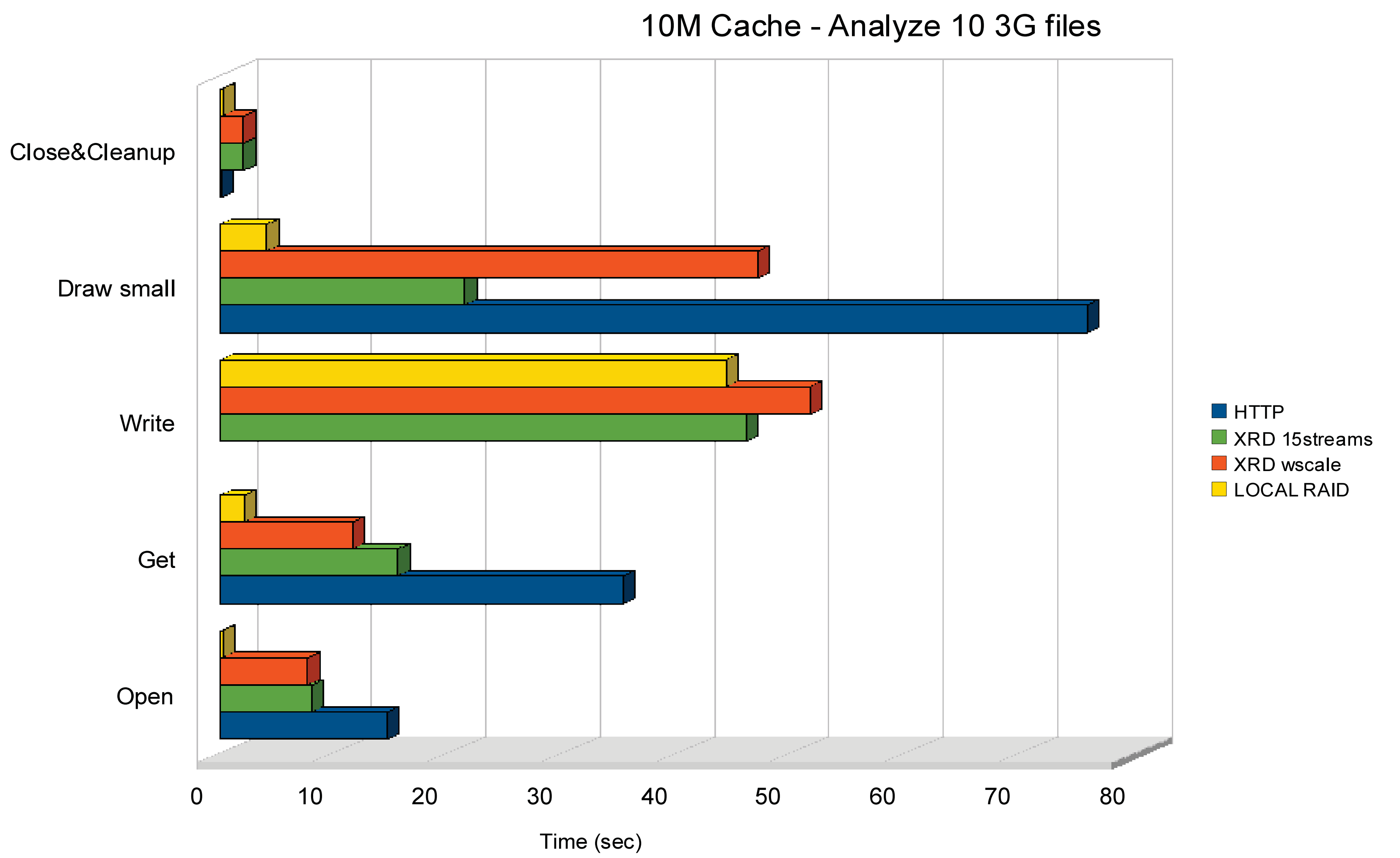
- Caltech machinery: 10Gb network
- Client and server (super-well tuned)
  - Selectable latency:
    - ~0.1ms = super-fast LAN
    - ~180ms = client here, server in California
      - (almost a worst case for WAN access)
- Various tests:
  - Populate a 30GB repo, read it back
  - Draw various histograms
    - Much heavier than the normal, to make it measurable
    - From a minimal access to the whole files
    - Putting heavy calcs on the read data
    - Up to reading and computing everything
      - Analysis-like behaviour
  - Write a big output (~600M) from ROOT

Thanks to Iosif Legrand and Ramiro Voicu

10Gb WAN 180ms Analysis



10Gb WAN 180ms Analysis



An estimation of Overheads and write performance

Comments

- Things look quite interesting
  - BTW same order of magnitude than a local RAID disk (and who has a RAID in the laptop?)
  - Writing gets really a boost
    - Aren't job outputs written that way sometimes?
    - Even with Tfile::Cp
- We have to remember that it's a worst-case
  - Very far repository
  - Much more data than a personal histo or an analysis debug (who's drawing 30GB personal histograms? If you do, then the grid is probably a better choice.)
  - Also, since then (2009), the xrootd performance increased further by a big factor for these use cases