D$_{Aa}$M Jamboree,
Jun 17, 2010

GEMSS: GPFS/TSM/StoRM

Vincenzo Vagnoni
On behalf of CNAF staff and LHC experiment representatives at CNAF

DISCLAIMER

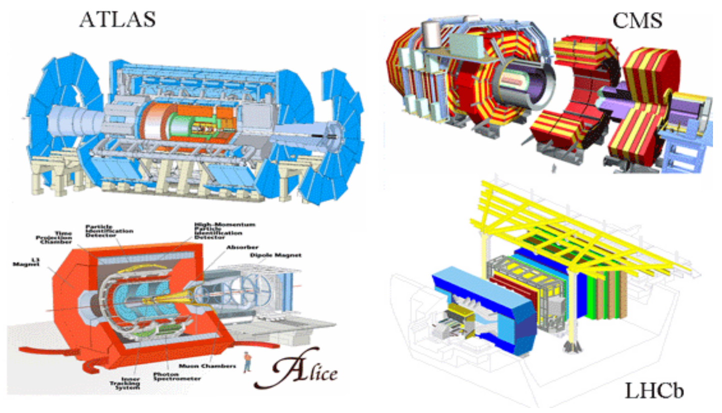
Who speaks is not an IBM stakeholder (nor his friends)

What IBM will gain or loose as a consequence of our work, is just their business…

INFN-CNAF

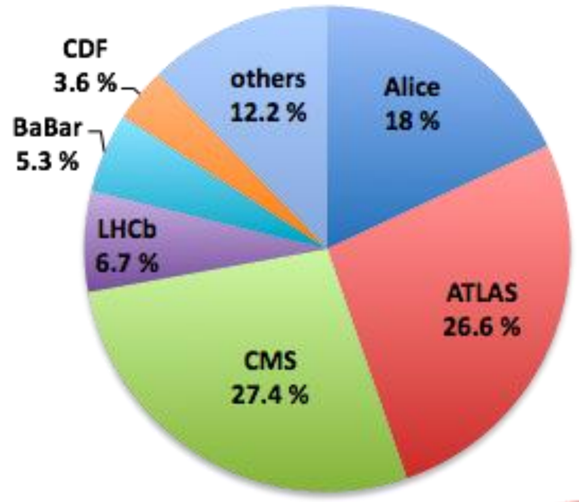
Year	CPU power [HS06]	Disk Space [PB]	Tape Space [PB]
2009	23k	2.4	2.5
2010	68k	6.8	6.6

[Not all 2010 resources available on Jan 1st]

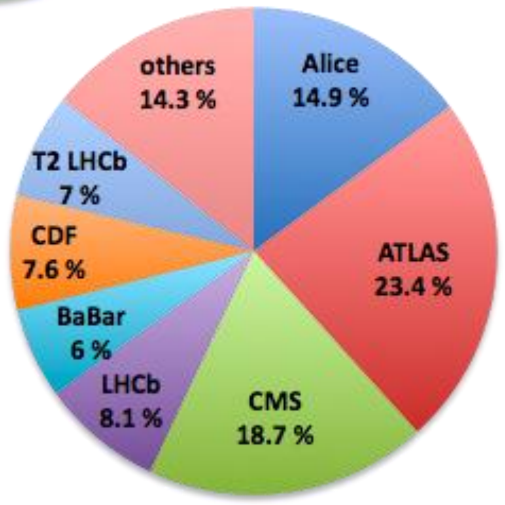


CNAF is the central computing facility of INFN

- Italian Tier-1 computing centre for the LHC experiments ATLAS, CMS, ALICE and LHCb...
- ... but also one of the main processing facilities for other experiments:
  - BaBar and CDF
  - Astro and Space physics
  - VIRGO (Italy), ARGO (Tibet), AMS (Satellite), PAMELA (Satellite) and MAGIC (Canary Islands)
  - More...



Disk shares



CPU shares

Mass Storage at CNAF

CASTOR was the “traditional” solution for MSS at CNAF for all VO's <u>since 2003</u>

Large variety of issues

- both at set-up/admin level and at VO’s level (complexity, scalability, stability, …)
- Yet, successfully used in production, despite some large operational overhead

In parallel to production, in <u>2006</u> CNAF started to search for a potentially more scalable, performing and robust solution

- <u>Q1 2007</u>: GPFS (from IBM) definitively chosen as the solution for disk-based storage after massive comparison tests (but it was already in use at CNAF much before this test)
- <u>Q2 2007</u>: StoRM (developed at INFN) implements SRM 2.2 specifications
- <u>Q3-Q4 2007</u>: StoRM/GPFS in production for D1T0 for LHCb and Atlas
  - Clear benefits for both experiments (very reduced load on CASTOR)
  - No major impact yet on CMS workflows (no large use of D1T0)

Then we started thinking to a complete MSS solution based on StoRM/GPFS + something

Mass Storage Systems at CNAF (2)

End 2007: a project started at CNAF to realize a complete grid-enabled HSM solution based on **StoRM/GPFS/TSM**

- StoRM extended to include the SRM methods required to manage data on tape
- GPFS specific features (available since version 3.2) were combined with TSM (also from IBM) and StoRM
- An interface between GPFS and TSM implemented (not all needed functionalities provided out of the box)

Q2 2008: First implementation (D1T1, i.e. w/o user driven recalls) in production for LHCb (CCRC'08)

Q2 2009: "**GEMSS**" (StoRM/GPFS/TSM) supporting a full HSM solution ready for production at CNAF

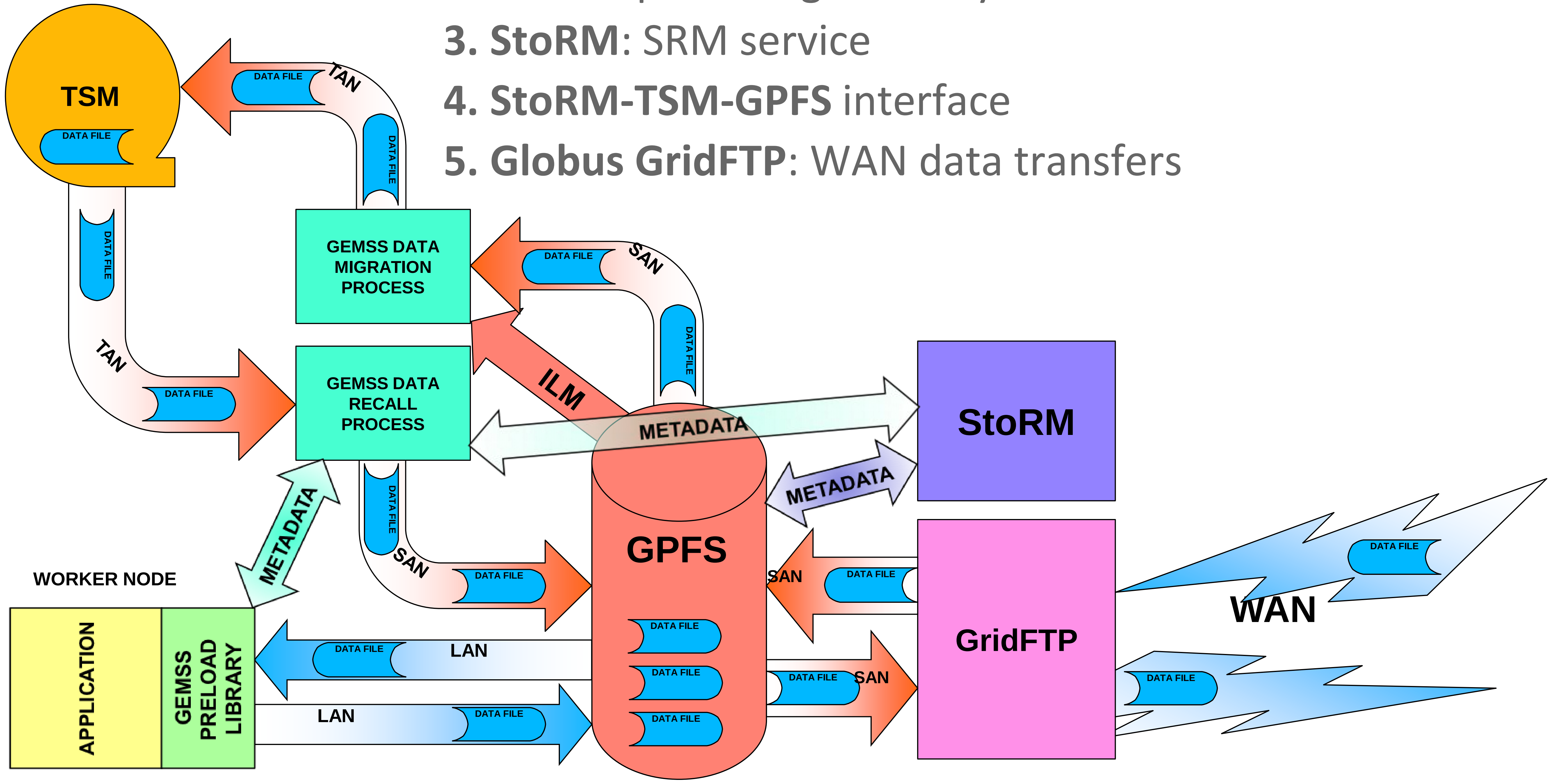
- Pre-production testbed built to accommodate the scaling needs of CMS

Q3 2009: **LHC expts finally started moving from CASTOR to GEMSS**

Building blocks of GEMSS system

Disk-centric system with five building blocks

1. **GPFS**: disk-storage software infrastructure
2. **TSM**: tape management system
3. **StoRM**: SRM service
4. **StoRM-TSM-GPFS** interface
5. **Globus GridFTP**: WAN data transfers

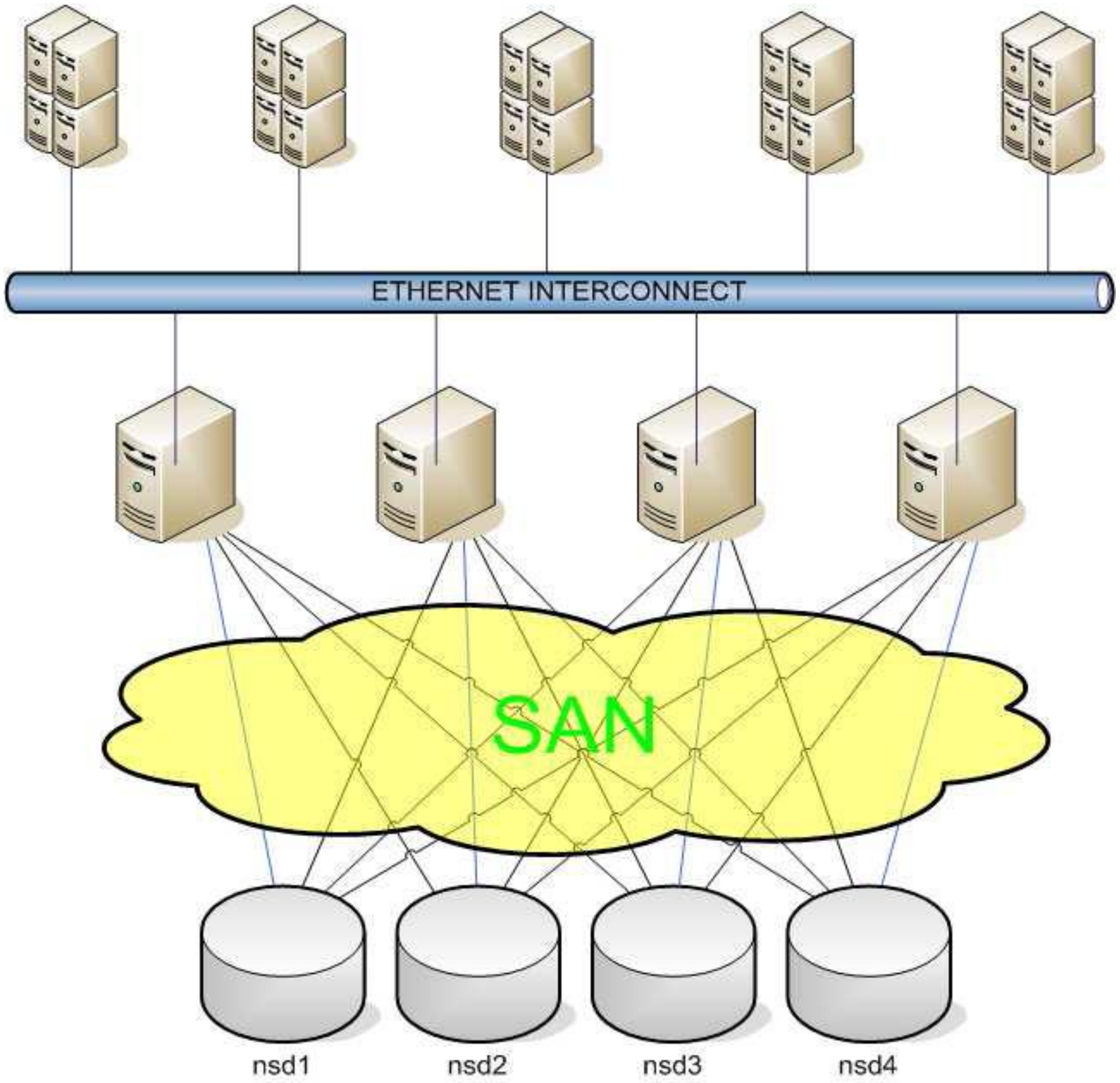


Why GPFS

- Original idea since the very beginning: we did not like to rely on a tape centric system
  - ✦ First think to the disk infrastructure, the tape part will come later if still needed
- We wanted to follow a model based on well established industry standard as far as the fabric infrastructure was concerned
  - ✦ Storage Area Network via FC for disk-server to disk-controller interconnections
- This lead quite naturally to the adoption of a clustered filesystem able to exploit the full SAN connectivity to implement flexible and highly available services
- There was a major problem at that time: a specific SRM implementation was missing
  - ✦ OK, we decided to afford this limited piece of work → StoRM

Basics of how GPFS works

- The idea behind a parallel filesystem is in general to stripe files amongst several servers and several disks
  - This means that, e.g., replication of the same (hot) file in more instances is useless → you get it "for free"
- Any "disk-server" can access every single device with direct access
  - Storage Area Network via FC for disk-server to disk-controller interconnection (usually a device/LUN is some kind of RAID array)
  - In a few words, all the servers share the same disks, but a server is primarily responsible to serve via Ethernet just some disks to the computing clients
  - If a server fails, any other server in the SAN can take over the duties of the failed server, since it has direct access to its disks
- All filesystem metadata are saved on disk along with the data
  - Data and metadata are treated simmetrically, striping blocks of metadata on several disks and servers as if they were data blocks
  - No need of external catalogues/DBs: it is a true filesystem


ETHERNET INTERCONNECT
SAN
nsd1
nsd2
nsd3
nsd4

Some GPFS key features

Very powerful (only command line, no other way to do it) interface for configuring, administering and monitoring the system

- In our experience this is the key feature which allowed to keep minimal manpower to administer the system
  - 1 FTE to control every operation (and scaling with increasing volumes is quite flat)
- Needs however some training to startup, it is not plug and pray… but documentation is huge and covers (almost) every relevant detail

100% POSIX compliant by design

Limited amount of HW resources needed (see later for an example)

Support for cNFS filesystem export to clients (parallel NFS server solution with full HA capabilities developed by IBM)

Stateful connections between “clients” and “servers” are kept alive behind the data access (file) protocol

- No need of things like “reconnect” at the application level

Native HSM capabilities (not only for tapes, but also for multi-tiered disk storage)

GPFS policy engine

Information Lifecycle Management (ILM) is a programmable policy engine embedded in GPFS

- It is the key component used to implement tape HSM and multi-tiered storage
- ILM interprets an SQL-like language and performs a metadata scan to match files according to given rules
  - It is not a “find”, but an optimized sequential scan of metadata structures
  - The maximum rate we measured on a filesystem is about 100k inodes/s
- It can be used for a plenty of use cases, e.g.
  - Trigger automatic migration of less used files from a “gold” GPFS storage pool (maybe made of faster and more reliable disks) to a “silver” pool (maybe made of older or cheaper hardware) and then to a nearline pool, on the basis of arbitrary rules built on top of time and file attributes
    - GPFS “storage pools” are basically striping groups of disks within the same filesystem which can be composed of/served by different hardware
  - Currently at CNAF ILM is only used to identify candidates for migrations to tape, but it would be a valid tool for a multi-tiered storage system and we are thinking about it

File migrations from GPFS to TSM (I)

We implemented data migration from GPFS to TSM employing standard GPFS features

ILM performs metadata scans to produce the list of files eligible for migration

- the list is splitted in sub-lists with a configurable maximum number of files per sub-list, in order to allow for the parallelization of migrations on multiple processes and nodes
- the number of GEMSS migrators running on each node is configurable for fine tuning the desired degree of parallelism, hence maximum number of drives to employ

ILM triggers the startup of GEMSS data migrator processes on a set of dedicated “data mover” nodes

- each migrator receives as input one sub-list

GEMSS data migrators in turn invoke HSM-client TSM commands to perform file transfers to tape

File migrations from GPFS to TSM (II)

Files belonging to different datasets are migrated to different TSM tape pools

- datasets are in general identified in ILM by means of appropriate rules matching file paths (or other file attributes)

When the file system occupancy exceeds a (configurable) threshold, ILM triggers a garbage collector process

- contents of files already copied to tape are removed from disk in order to bring down the occupancy to the desired value according to ILM rules defined by the sysadm
- when the content of a file is removed, a so-called “stub” file is kept on disk
- the stub file in general contains no data and holds in its metadata the unique key needed to identify the file record in the TSM DB

GEMSS selective tape-ordered recalls (I)

Native tape-ordered HSM recalls from tape are missing at the moment in TSM, although they provide client tools to implement it

We implemented selective tape-ordered recalls in GEMSS by means of 4 main commands/processes

- EnqueueRecall
- Monitor
- ReorderRecall
- ProcessRecall

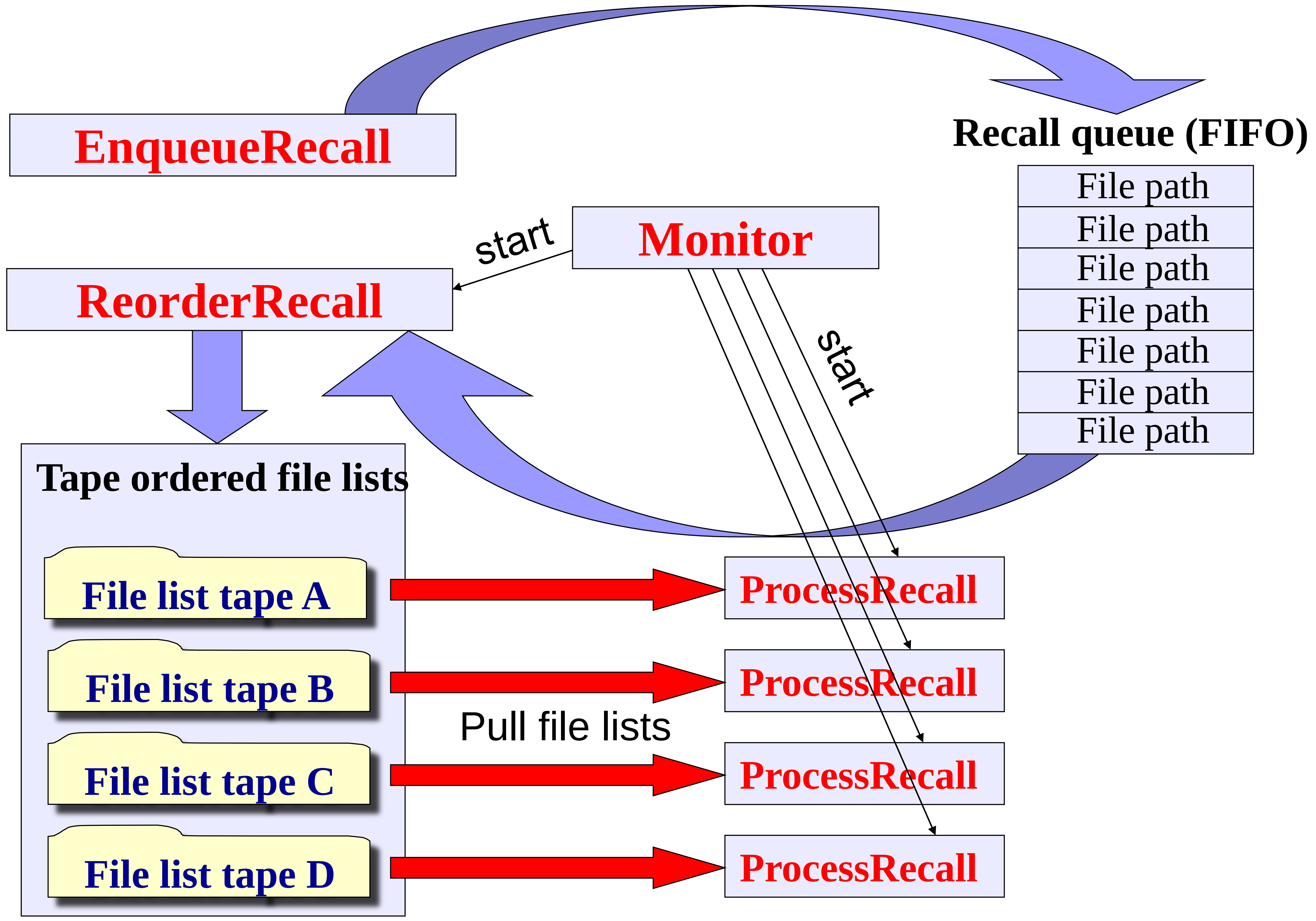
EnqueueRecall is a command used to insert file names to be recalled into a FIFO (alternative is direct access to file or SRM)

ReorderRecall is a process which fetches files from the queue and builds sorted lists with optimal file ordering

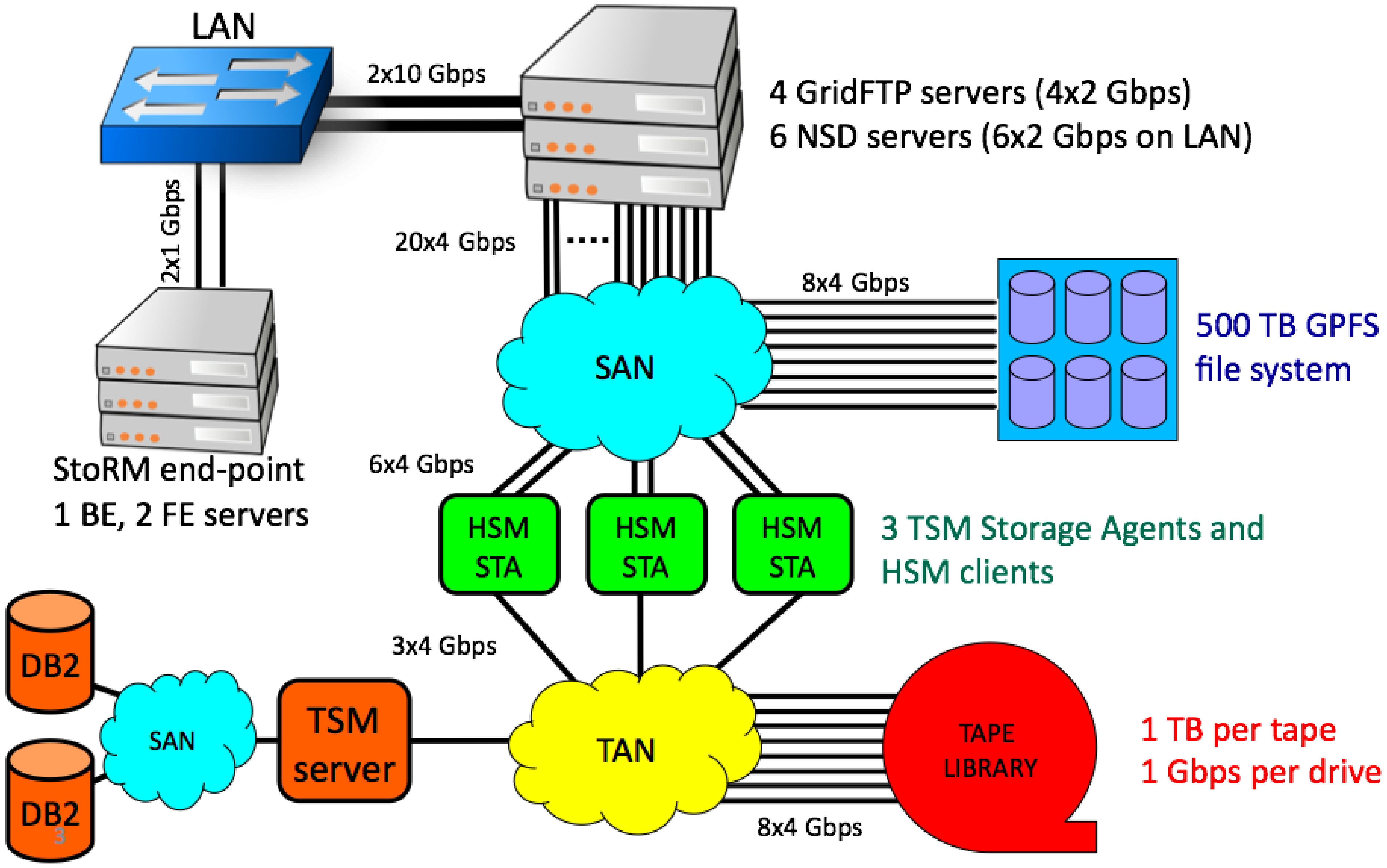
ProcessRecall is a process which performs actual recalls from TSM to GPFS for one tape by issuing HSM-client TSM commands

Monitor starts one ReorderRecall and as many ProcessRecall processes as specified in configuration files

GEMSS selective tape-ordered recalls (II)



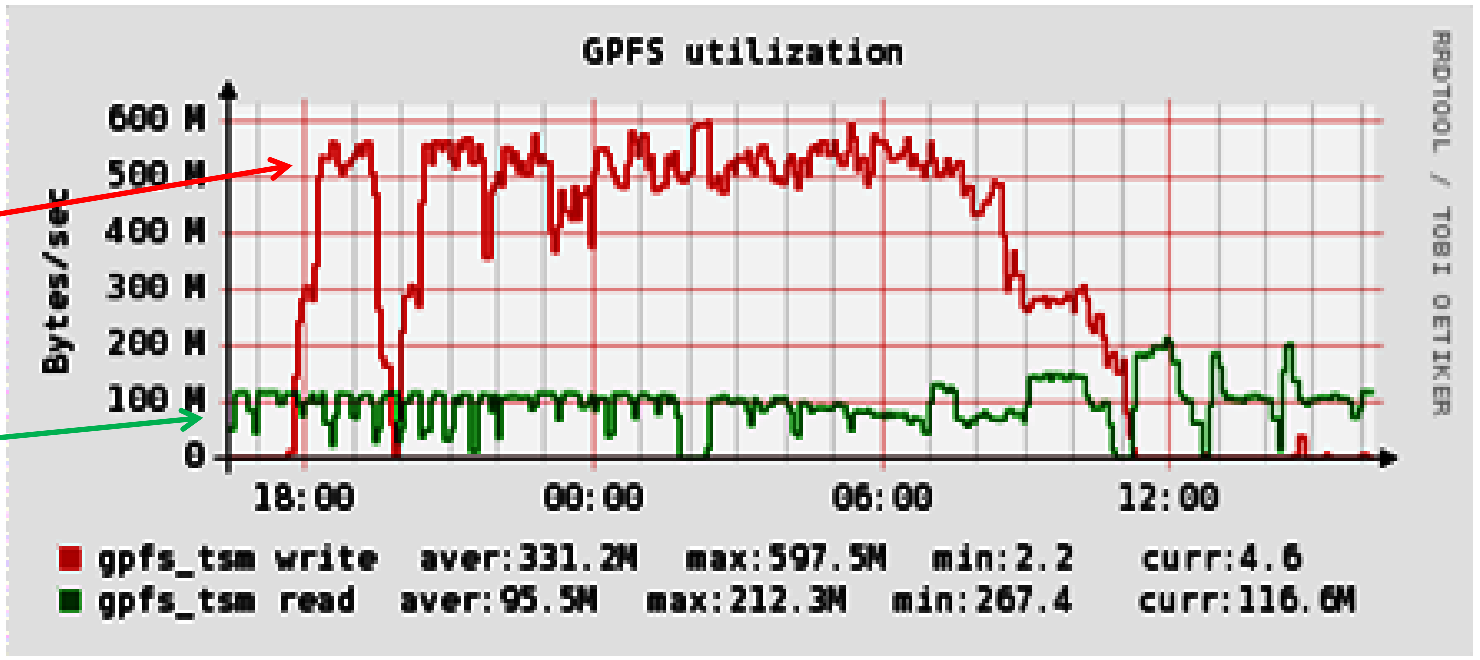
GEMSS production layout for CMS (being replaced these days)



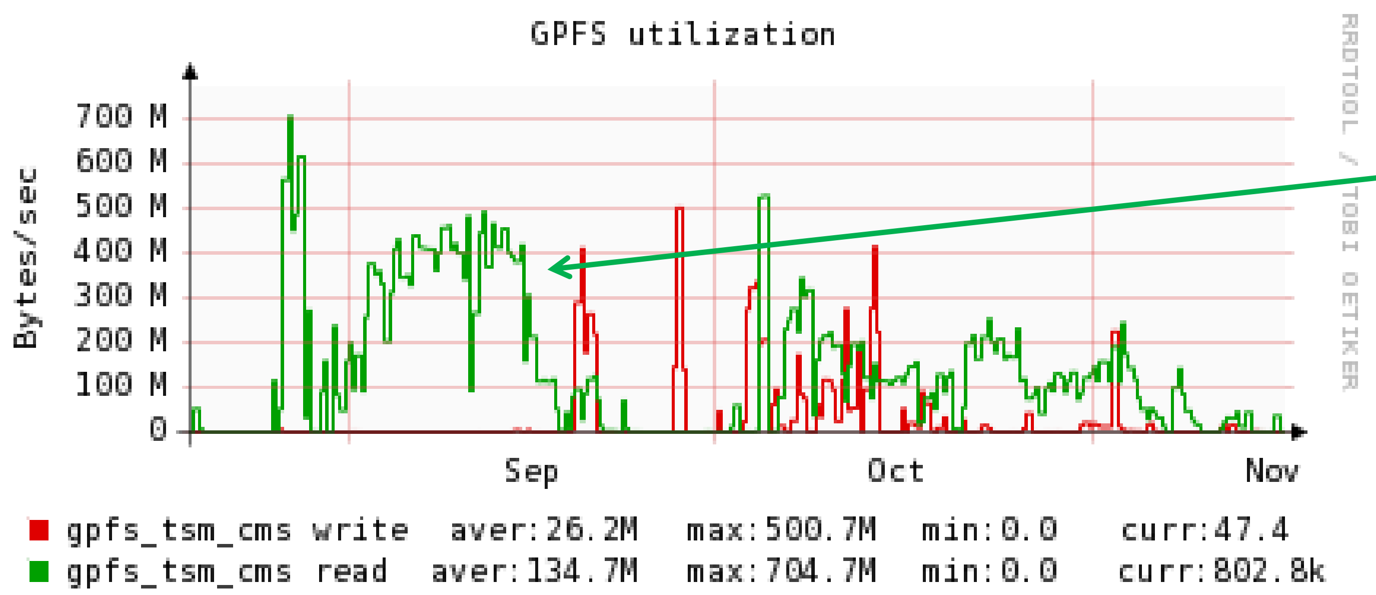
CMS tests for local access to TSM

Summer 2009 tests

- Manual recall from tape
  - **550 MB/s**
- Migration to tape
  - **100 MB/s**
- Local access to data from the batch farm nodes
  - 400 MB/s (not shown in the plot)



🟥 gpfs_tsm write aver:331.2M max:597.5M min:2.2 curr:4.6
🟩 gpfs_tsm read aver:95.5M max:212.3M min:267.4 curr:116.6M



🟥 gpfs_tsm_cms write aver:26.2M max:500.7M min:0.0 curr:47.4
🟩 gpfs_tsm_cms read aver:134.7M max:704.7M min:0.0 curr:802.8k

From tape — To tape

Migration of data from Castor to GPFS/TSM

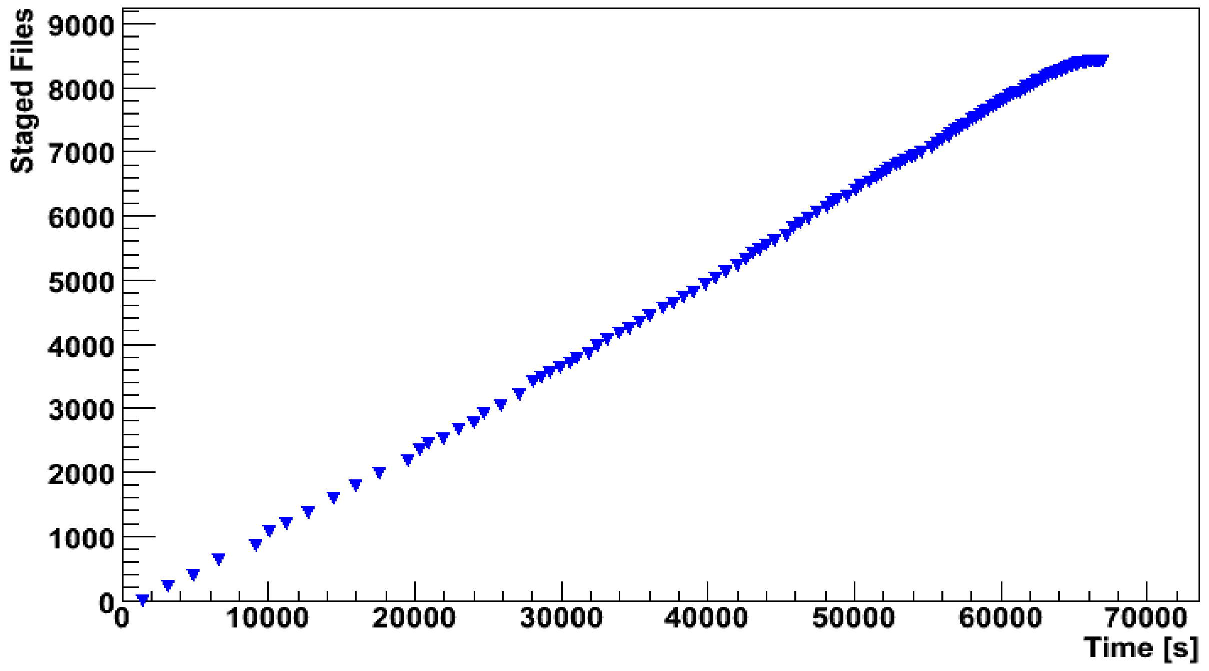
- several hundreds of TB
- In parallel to production activities

CMS test: recalls from tape

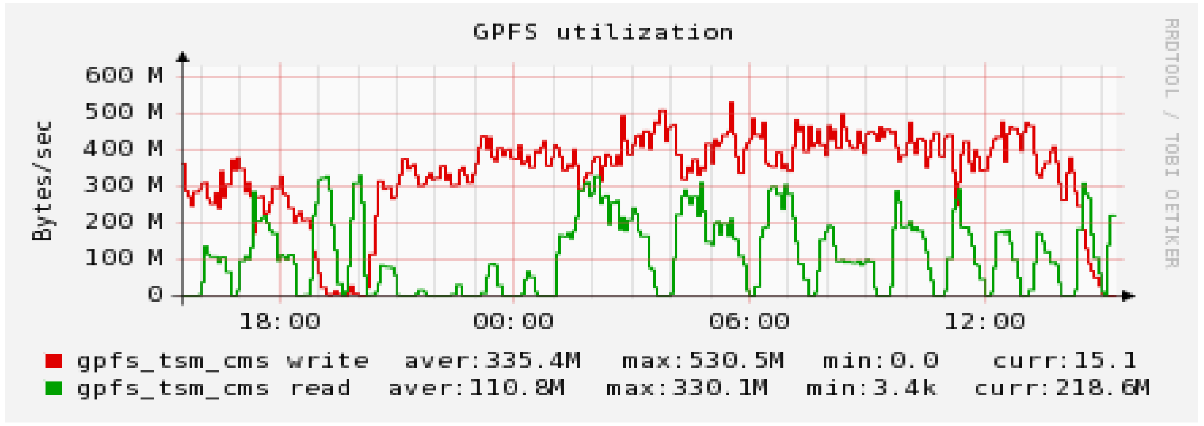
- Triggered by SRM BringOnline massive request (simulation of massive prestage)
- 24 TB randomly spread over 100 tapes (1 TB each) recalled in 19 hours
  - Peak measurements done with no overlap with other recalls
  - Quite some other activities running at the same time though (see plot below)

- 400 MB/s average throughput
- Peak at 500 MB/s
  - 70-80% of nominal tape drive throughput achieved in this test (6 drives)

Number of staged files as function of time



Net GPFS disk throughput on the GEMSS data movers



```
🟥 gpfs_tsm_cms write   aver:335.4M   max:530.5M   min:0.0    curr:15.1
🟩 gpfs_tsm_cms read    aver:110.8M   max:330.1M   min:3.4k   curr:218.6M
```

A practical example: the new ATLAS storage infrastructure

Limited amount of hardware is needed in this model(in terms of individual units)

- 8 data servers (NSD servers in GPFS terminology, 10 GbE)
- 2 metadata servers (NSD servers, 1 GbE)
- 3 GRIDFTP servers (10 GbE)
- 2 TSM data movers
  - network not used, LAN-free data movement from disk to tape and viceversa
- 2 DDN 9900 systems
  - Measured 11.5 GB/s of streaming throughput on a 1.6 PB GPFS filesystem

+ 5 StoRM nodes (no throughput)

Few machines to maintain also means reduced manpower

Note: this is a T1 scale

Last slide

ALICE (with a dedicated xrootd plugin for triggering tape ordered recalls --- note that ALICE at CNAF runs xrootd on top of a GPFS filesystem), ATLAS, CMS and LHCb are using GPFS/TSM/StoRM for today for every TxDx storage class
→ Castor fully dismissed for LHC expts

First 9 months of T1Dx production very promising

- In terms of performance and stability

GPFS was already used for T0D1 at CNAF before TSM integration (and also for T1D1 for LHCb pioneers)

GPFS is a very complete and robust solution. We are still not exploiting all its native features, e.g. tiered disk storage

StoRM is performing well and exposes now an interface to support interactions with tape backends

Data Management API

GPFS provides an implementation of the DMAPI open interface

DMAPI is widely used by the plain TSM HSM system, for example to react when a file is accessed but is still nearline

In a few words, DMAPI allows to write specific applications which can register for an I/O event on a filesystem (e.g. OPEN, READ, WRITE, etc.) and be notified at any of such operations, for files which have so-called managed regions set on these events (portions of files triggering a given reaction when accessed)

The application may then take actions accordingly, e.g. recall a file or transfer the content of a file on the fly from a given tier of storage before giving the client the green light to access it

This is in practice an advanced tool for customizing accesses to the filesystem