A robust and GRID compliant system for Virgo data transfer

Leone Bosi [a], Giancarlo Cella [b], Alberto Colla [c][d],

Cristiano Palomba [c], Livio Salconi [e]

[a] INFN-Perugia - [b] INFN-Pisa - [c] INFN-Roma1

[d] Università di Roma Sapienza - [e] Ego-Virgo

VDAS meeting, November 10th, 2010

Outline

- Current status of Virgo data transfer
- Proposed Data Transfer (DT) framework guidelines
- DT overview
- Tests
- Data access with Virgo Database (VDB)
- Current data access at CNAF and in2p3

Current status

- Present status of Virgo data transfer:
  - ➔ Virgo/LIGO computing centers use different transfer/storage technologies
  - ➔ **SRB** (IN2P3), **bbftp** (CNAF), **LDR** (LIGO)
    - — **bbftp** is obsolete and no longer supported by CNAF!
    - — **LDR** is based on obsolete native Globus file catalog (RLS)
  - ➔ Does not push toward a **common infrastructure** for data analysis

- **data administration activities**, and consequently **data bookkeeping** difficult to perform
  - ➔ This is reflected to the end user difficulty to have an **easy and intuitive access to storage resources**

Wishlist

- **In-time** data transfer to permanent storage
- **Single interface** to data distributed in different computing centers
- Local and remote **Data integrity** checks
- End-user data access through **pure scientific metadata**

Solutions

- Transfer and storage engines → **GRID tools**
  - **Standard interface** to access and handle data among the **most important computing centres** worldwide
  - Developed, used, and supported by a **wide community of scientists** for the **next 10 years** at least
  - **Data handling functions** (copy, replica, etc)
  - **Data integrity checks**
  - World-wide available **Logical File Catalogue (LFC)**
    - allows **transparent access to the distributed data**, hiding the underlying complexity

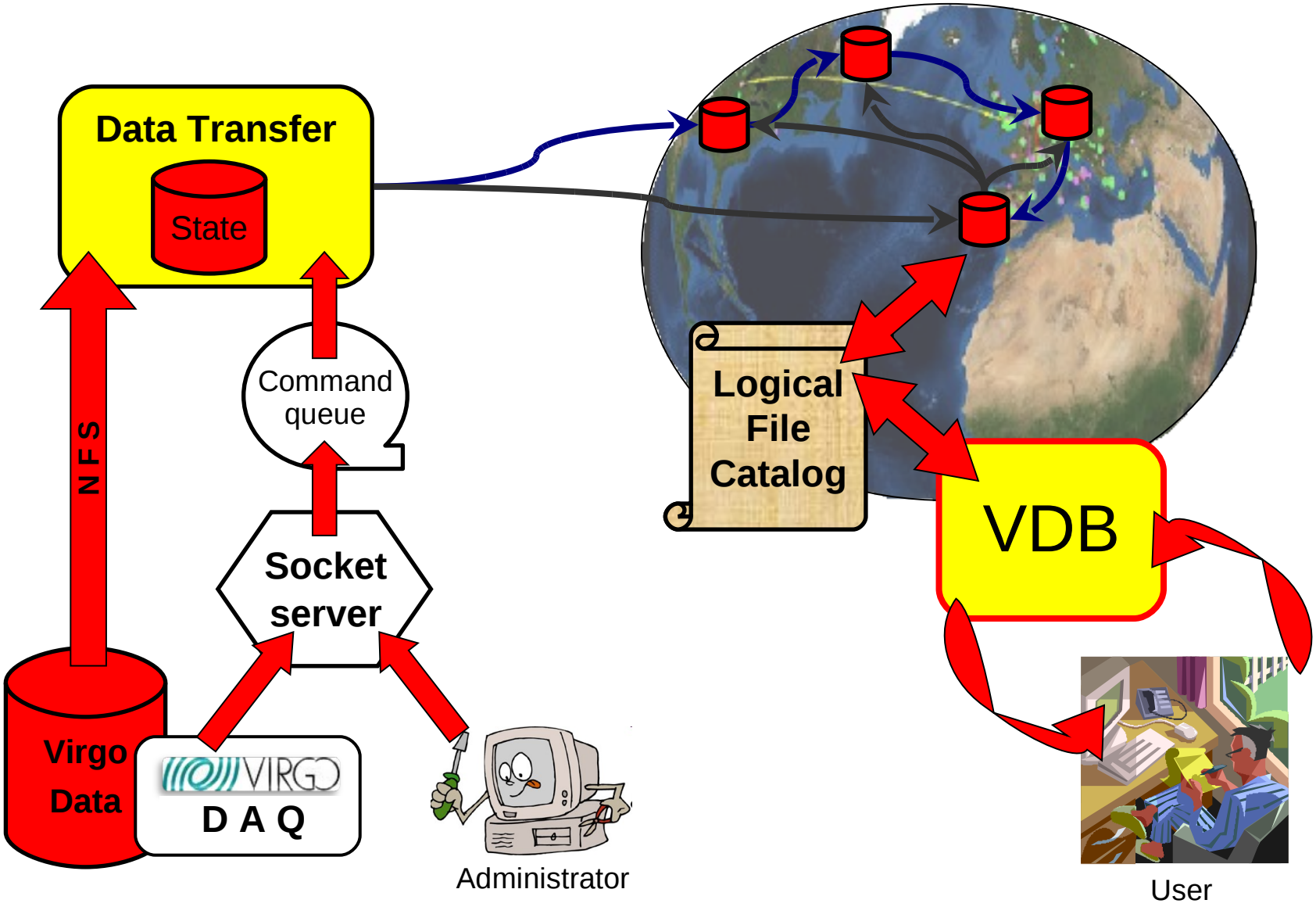
- Data bookeeping → **Virgo metadata catalogue (VDB)**
  - **Bookkeeping of experiment-specific metadata** allows to query data using **"physics" search criteria** only
    - e.g.: data taken in a given time interval, in specific science conditions, with specific quality flags
  - **LFC** provides transparent access to distributed data

Data Transfer (DT) framework

Main features:

- Code written in **Python**
- Synchronization with Data Acquisition (**DAQ**) system via **socket server/client**
  - ➔ New produced files are automatically added to the transfer queues
  - ➔ The same socket channel act as user CLI for manual intervention (add/remove files, open/close transfer streams, etc.)
  - ➔ Messages are put in a persistent **command queue** and periodically parsed by the DT
- Each file is associated to a **transfer task**, made of sequential steps managed by synchronized queues
  - ➔ **Local checksum** calculation (checksum queue)
  - ➔ **Transfer/replicas** to the remote storage elements (SE)
    + remote checksum calculation (transfer queue)
  - ➔ **Registration** in the **file catalogue + VDB** (registration queue)
- **Multi-threading**
  - ➔ Each step is associated to a (configurable) number of specific threads
  - ➔ Threads are run as sub-processes and monitored by the main program

DT workflow


Data Transfer
State
Command queue
NFS
Socket server
Virgo Data
VIRGO
D A Q
Administrator
Logical File Catalog
VDB
User

More on transfer process

- Data transfer process does not make use of unnecessary Grid services (Information System, file catalogue, …)
  - ➔ "Brute force" **gridFTP** (lcg-cp)
  - ➔ **Minimizes** possible points of failures
  - ➔ Remote checksum calculation comes for free
  - ➔ SRM interface hides the complexity of the underlying remote SE's architecture
    - — A SRM endpoint is fully identified by a small set of parameters (see later)

- Same command (**lcg-cp**) for local→remote and remote→remote replicas
  - ➔ "Third party" (remote→remote) replicas done from the same transfer server at Cascina
  - ➔ Eases transfer schema configuration (e.g. "Star" vs "Daisy chain" …)

- **Load balancing**
  - ➔ The system keeps track of the number of incoming and outgoing streams in each endpoint, and chooses the following transfer endpoints consequently
  - ➔ Limits can be set on the number of concurrent incoming/outgoing streams in each endpoint

“Robustness” features

- DT processes (local checksum, lcg-cp, etc) are run as **sub-processes** monitored by the main thread
  - ➔ Kill in case of timeouts
  - ➔ Failure tracking
- DT retains the status of each task in a persistent local database
  - ➔ Task processes are retried a (configurable) number of times before being marked as “failed”
  - ➔ Failed tasks can be re-run manually by the operator, through the socket CLI
- The system keeps track of the status of each transfer endpoint
  - ➔ An endpoint is automatically closed if too many transfers from/to it consecutively fail
  - ➔ It can be re-opened manually by the operator through the command queue
- The command queue is persistent
  - ➔ In case of crash of the DT process the command aren’t lost
- The DAQ infrastructure keeps track of the messages in case they don’t reach the socket server
  - ➔ Safe also against socket server crash

DT configuration

- Parameters for Data Transfer protocol are in a single **`.ini`** file

```
[PATHS]
outputBasePath: /grid/virgo/TSCascina

[SOURCE]
maxOut: 4
maxFailures: 5


[SE1]
hostName: storm-fe-virgo.cr.cnaf.infn.it
hostPort: 8444
srmVersion: srmv2
srmPrefix: /srm/managerv2
srmPath: /virgo3/TSCascina/
maxIn: 6
maxOut:4
maxFailures: 3

[SE2]
hostName: ccsrm02.virgo.in2p3.fr
…
```

Base path in the LFN

source parameters

Remote storages' parameters

Max n. of in/out transfers

Max n. of consecutive failures

```
[TPARAMETERS]
Timeout: 3600
NReplicas: 2
NMaxTransferRetries: 3
NMaxRegisterRetries: 3

ChecksumThreads: 4
ReplicaThreads: 4
RegisterThreads: 4
…
```

Running parameters

Monitoring web interface

Transfer Status Mon Nov 8 16:06:18 2010

SE Status

SE hostname	Status	Data IN	Data OUT	Consecutive failures	Next status switch	Comment
storm-fe-virgo.cr.cnaf.infn.it	True	3	2	0	Not set	None
ccsrm02.in2p3.fr	True	4	1	0	Not set	None
local	True	0	4	0	Not set	None

Queue Status

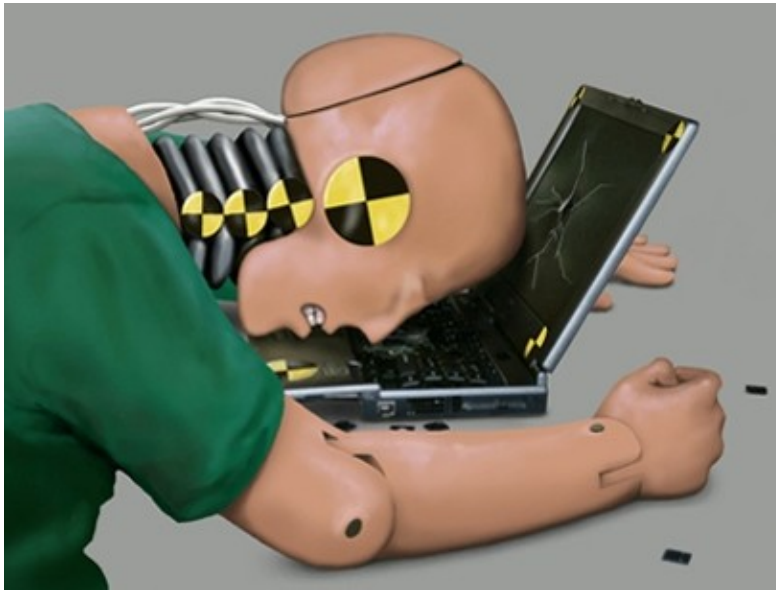
Checksum	Transfer	Register	Done	Delayed	Dead
10	2	0	7	0	0

File Status (total files in DB: 30)

File name	Size	Current status	Local Checksum	Remote checksums	Transfer status	Transfer failures	Register status	Register failures
V-973087200-06-Nov-2010-15h00-720F.50	1200534077	Done	21DD1A11	[None, '21DD1A11']	['Done', 'Done']	[0, 1]	['Done', 'Done']	[0, 0]
V-973094400-06-Nov-2010-17h00-720F.50	1201243217	Done	3327D773	[None, '3327D773']	['Done', 'Done']	[0, 0]	['Done', 'Done']	[0, 0]
V-973116000-06-Nov-2010-23h00-720F.50	1200546935	Done	7513FB21	[None, '7513FB21']	['Done', 'Done']	[0, 0]	['Done', 'Done']	[0, 0]
V-973137600-07-Nov-2010-05h00-720F.50	1200649749	REPLICA[0->2]_START	5051BC2A	[None, None]	[None, None]	[0, 0]	[None, None]	[0, 0]
V-973144800-07-Nov-2010-07h00-720F.50	1201040263	REPLICA[1->2]_START	15978406	[None, None]	['Done', None]	[0, 0]	[None, None]	[0, 0]
V-973195200-07-Nov-2010-21h00-720F.50	0	Waiting	None	[None, None]	[None, None]	[0, 0]	[None, None]	[0, 0]
V-973224000-08-Nov-2010-05h00-720F.50	0	Waiting	None	[None, None]	[None, None]	[0, 0]	[None, None]	[0, 0]
V-973231200-08-Nov-2010-07h00-720F.50	0	Waiting	None	[None, None]	[None, None]	[0, 0]	[None, None]	[0, 0]
V-973245600-08-Nov-2010-11h00-720F.50	0	Waiting	None	[None, None]	[None, None]	[0, 0]	[None, None]	[0, 0]
V-973252800-08-Nov-2010-13h00-720F.50	0	Waiting	None	[None, None]	[None, None]	[0, 0]	[None, None]	[0, 0]
V-973260600-08-Nov-2010-15h10-60F.50	0	Waiting	None	[None, None]	[None, None]	[0, 0]	[None, None]	[0, 0]
V-973261200-08-Nov-2010-15h20-60F.50	0	Waiting	None	[None, None]	[None, None]	[0, 0]	[None, None]	[0, 0]
V-973080000-06-Nov-2010-13h00-720F.50	1199989141	Done	F00334DF	[None, 'F00334DF']	['Done', 'Done']	[0, 1]	['Done', 'Done']	[0, 0]
V-973101600-06-Nov-2010-19h00-720F.50	1200942907	Done	2E39F961	[None, '2E39F961']	['Done', 'Done']	[0, 1]	['Done', 'Done']	[0, 0]

Crash tests

- Data transfer framework **tested** with “fake” frame files (in Rome) and with real **raw data** (at Cascina)
- Crash tests (incomplete list):
  - **Kill** ongoing transfer process
  - **data corruption** (file modified/removed after checksum calculation)
  - **Grid proxy expiration**
  - **Unmount** local data partitions (NFS)
  - **Switch off** remote endpoint
  - **Overfill** data partitions (local and remote)
  - Attempt of copy on **already existing/corrupted remote files**
  - Unavailability of **LFC**
  - Data replication up to **5 different** remote storage servers



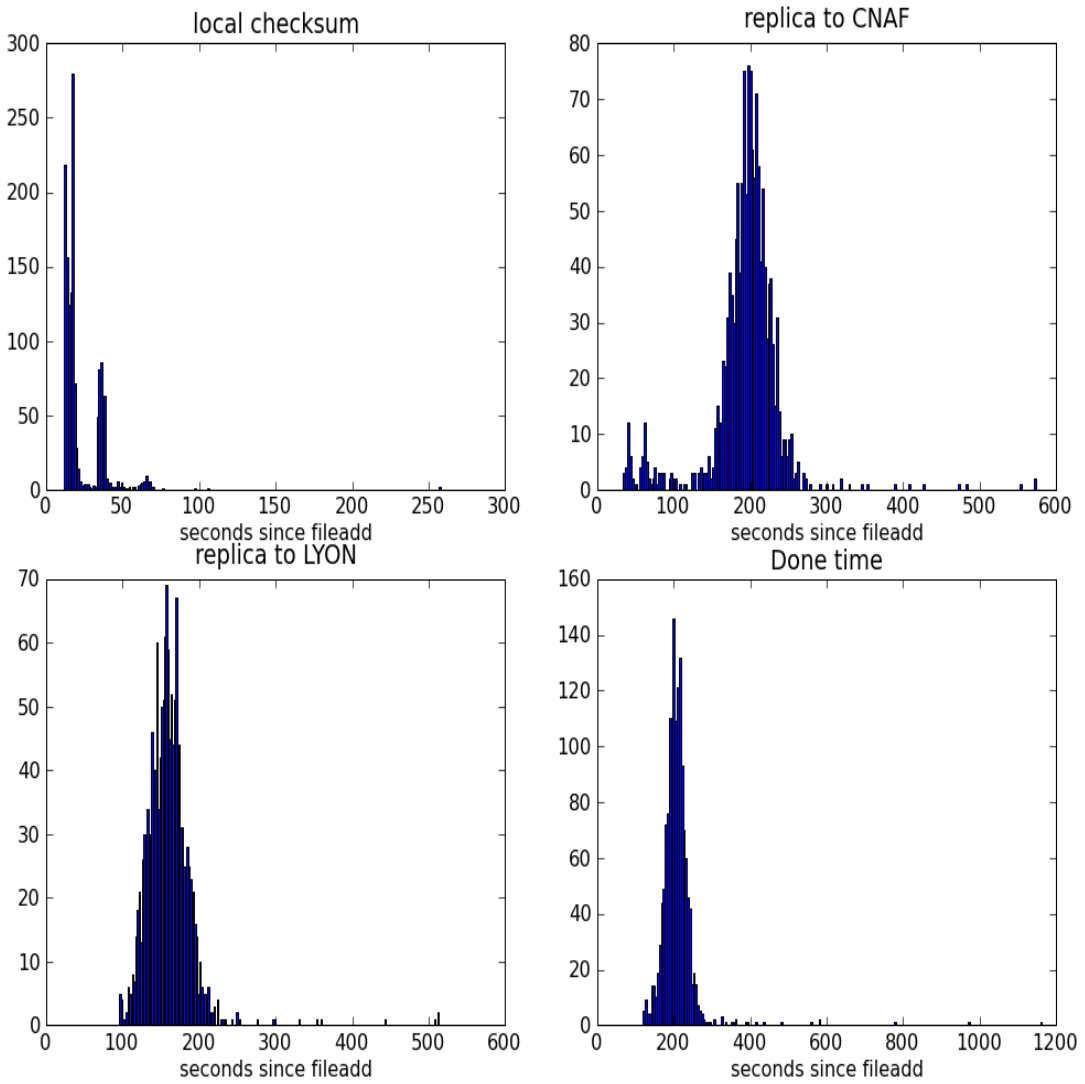
The tests served to enhance and confirm the **robustness** and **scalability** of the system and the **error recovery** capabilities

Test Cascina → CNAF & in2p3

- "Real" raw data transferred from Cascina to Cnaf (disk/GPFS, StoRM SRM interface) and in2p3 (tape/HPSS, dCache SRM interface)
- Synchronisation with DAQ
- Setup at Cascina:
  - ➔ Single core, 4 GB RAM machine installed and configured as **Grid (gLite 3.1) User Interface**
  - ➔ Robot Grid certificate, Grid proxy **automatically renewed**
  - ➔ **NFS mounted** data disks (same setup as current transfer framework)
  - ➔ Maximum available bandwidth: **~40 MB/s**
- 5 days test, ~ 3 TB transferred
  - ➔ **bandwidth shared** with official data transfer! (no ideal conditions)

Test results

- 1.75 GB raw data files
- Completion time from “fileadd” to Done: ~200 s
  - ➔ dominated by pure transfer times
- Stable operation
  - ➔ remember: test done during official data transfer!
- In general files are first copied from local to in2p3, then from in2p3 to Cnaf
  - ➔ just a consequence of the SE order in the configuration



Data access with Grid tools

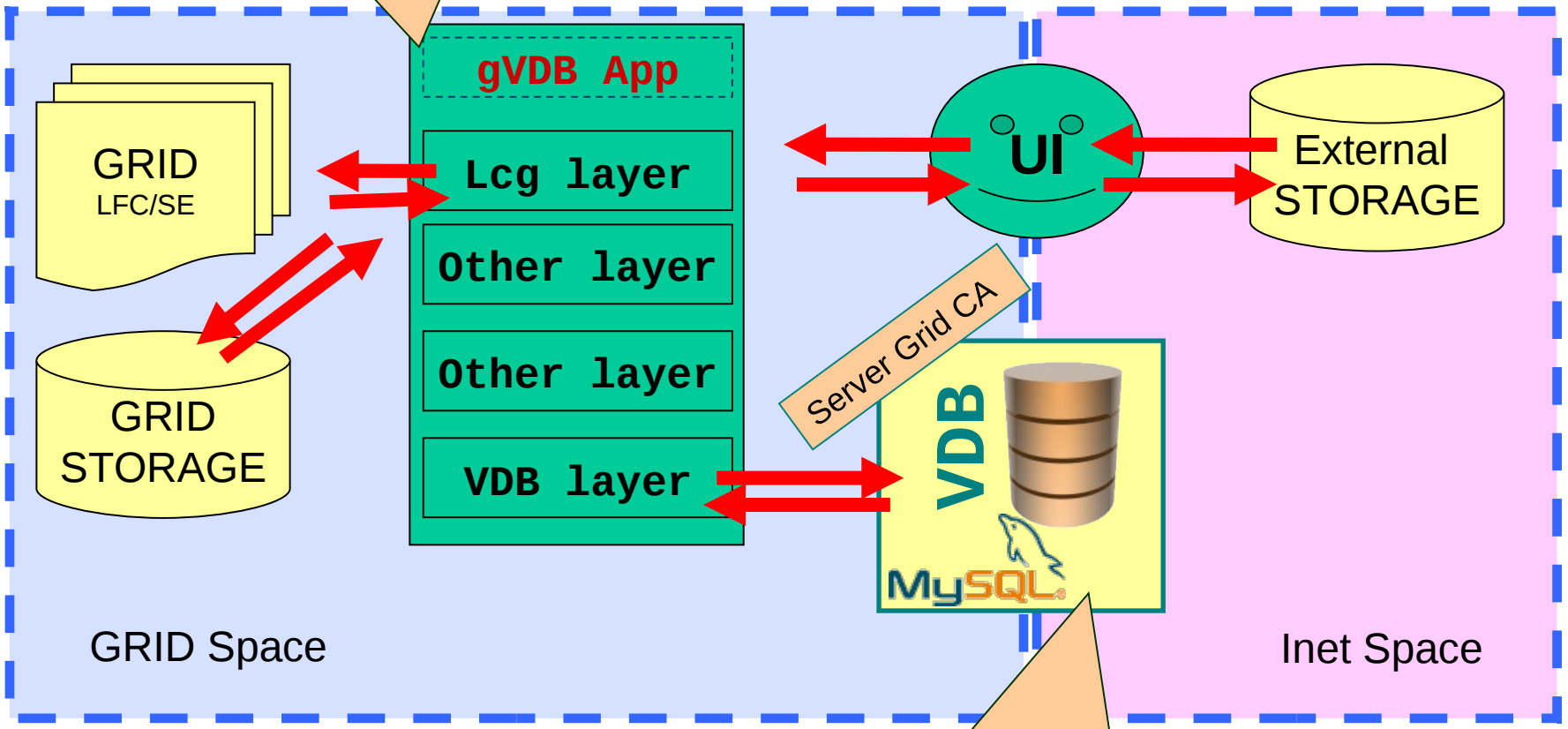
- The most direct way to access data transferred with the DT framework is to use the same Grid LCG tools
- Some examples (works from any Grid User Interface):

```
> # List files in LFN
> lfc-ls /grid/virgo/TSCascina/50Hz/0/

V-973080000-06-Nov-2010-13h00-720F.50
V-973087200-06-Nov-2010-15h00-720F.50
...

> # Download file to local disk
> lcg-cp -v lfn:/grid/virgo/TSCascina/50Hz/0/V-973080000-06-Nov-2010-13h00-720F.50
file:`pwd`/test.50
Using grid catalog type: LFC
Using grid catalog : lfcserver.cnaf.infn.it
...
1195376640 bytes  10808.86 KB/sec avg  10241.77 KB/sec inst
Transfer took 109070 ms

> # List replicas
> lcg-lr lfn:/grid/virgo/TSCascina/50Hz/0/V-973080000-06-Nov-2010-13h00-720F.50
srm://ccsrm02.in2p3.fr/pnfs/in2p3.fr/data/virgo/tape/TSCascina//50Hz/0/V-973080000-
06-Nov-2010-13h00-720F.50
srm://storm-fe-virgo.cr.cnaf.infn.it/virgo3/TSCascina//50Hz/0/V-973080000-06-Nov-
2010-13h00-720F.50
```

Data access with VDB

•General gVDB application example
•Can run everywhere (UI & WN)

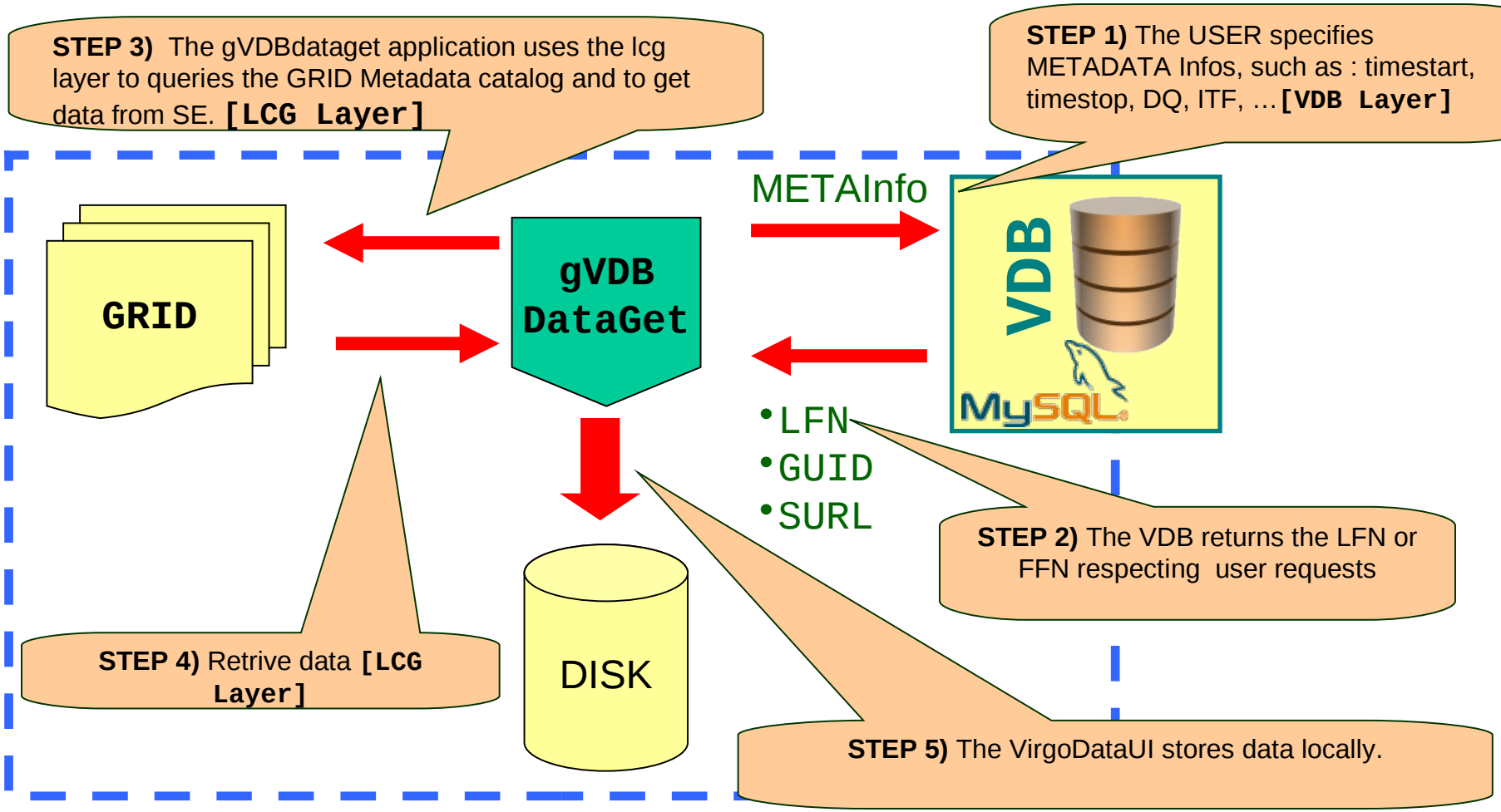
(Slides by Leone)



•VDB becomes a certificated Server
•It is reachable from everywhere inside and outside grid

Data access with VDB (2)

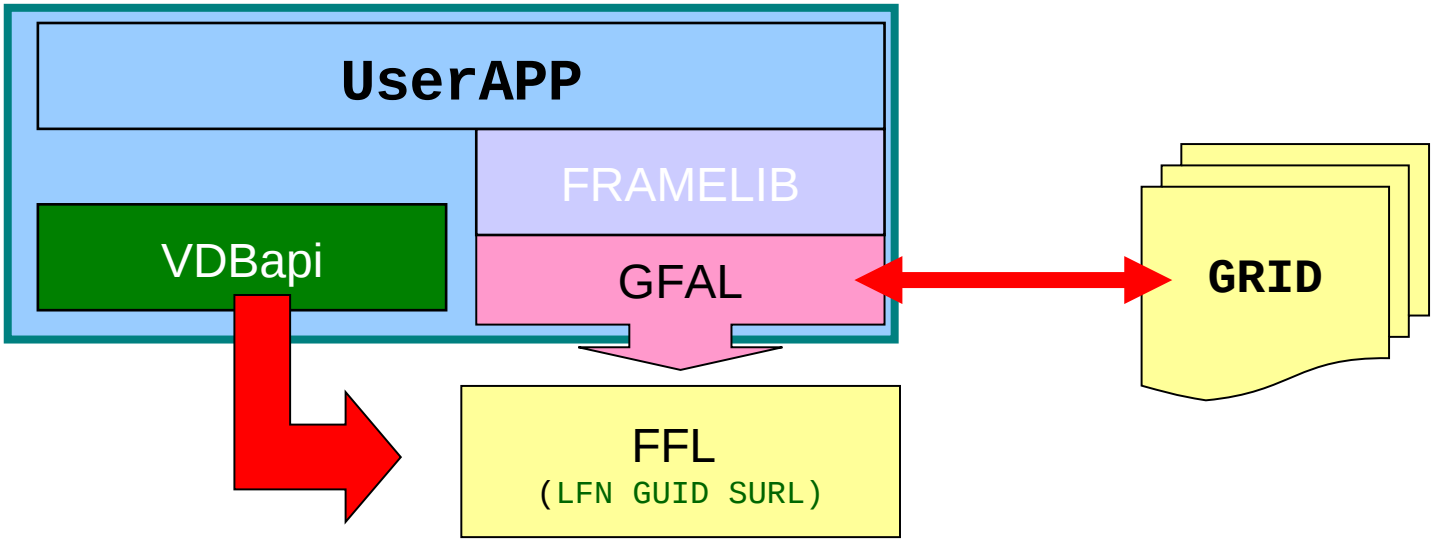
(Slides by Leone)



Data access with VDB (3)

(Slides by Leone)

Data processing: inside GRID with GRID tools@UI&WN



CNAF data access

- Current storage architecture: **Disk** (**GPFS**), **Tape** (**CASTOR**, archive only)
- Data accessible **locally** from the worker nodes and UI
- SRM backend: StoRM
- Past experiences of data access via Grid (CW group)
  - ➔ Analysis done on **pre-processed** data, very small sizes involved (~7000 files, 300 GB for VSR2)
  - ➔ Data **manually registered** on the LFC and downloaded to the WN's with **LCG commands**
  - ➔ In this case the challenging part is the concurrent request of thousands of files from different locations
  - ➔ Input from the analysis is the **file logical name** only (underlying complexity hidden to the user)
- Future: Migration to **HSM** (**GEMSS**)
  - ➔ data is stored on disk and automatically backup on tape
  - ➔ Old data (removed from disk) automatically staged back to disk when requested
  - ➔ Stage-in and -out operations are transparent to the user
  - ➔ User needs to know the path of the files on disk only
    - — Even in case files have been removed from disk, a pointer to their original path is kept, and they are staged back to the same position

in2p3 data access

- Storage architecture: **Tape** (**HPSS**)
- SRM backend: **dCache**
- Virgo data is currently transferred with **SRB** and accessed with **xrootd**
  - ➔ this must be maintained
- It is not straightforward to use xrootd to access data stored to HPSS through dCache
- Possible solutions: install an "**xrootd door**" on the dCache instance, or an "**xrootd-dCache**" interface
  - ➔ xrootd interface adopted by the **ATLAS** experiment at in2p3
  - ➔ Problems: **building** and **maintaining** the interface
    - — lack of manpower in Lyon
    - — our request is considered of low priority
  - ➔ Needs a strong support by the Virgo collaboration

- Backup slides

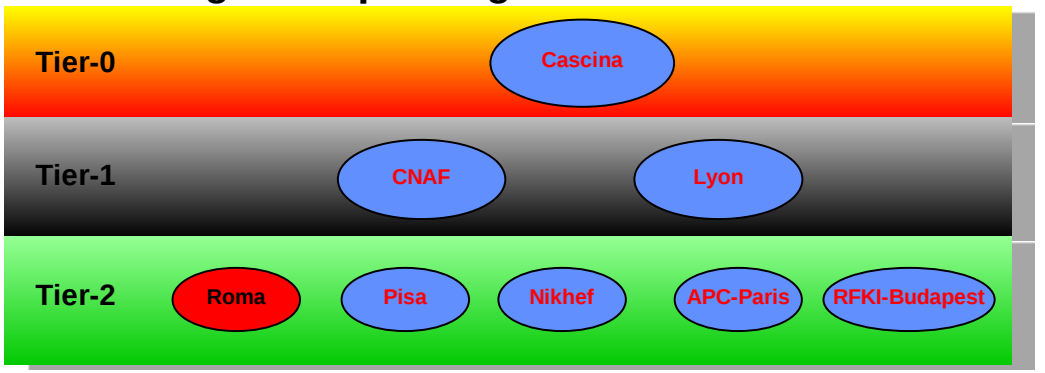
Talk outline

- Present transfer procedure drawbacks
- Proposed Data Transfer Framework guidelines
- DT overview
- Data Transfer robustness tests
- Conclusions & work in progress

DT framework: guidelines

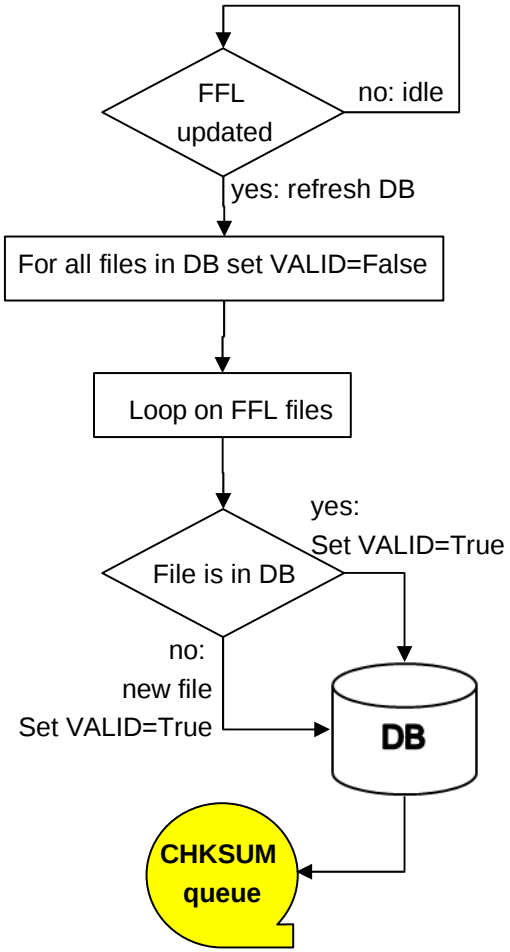
- Critical point: data **copied** to the Virgo remote computing centers (aka Storage Elements, **SE**) **as it is produced** (in-time mode)
- **Data integrity** checked
- Data **published** in the **LFC** and in the **VDB**
- Automatic **error recovery**
- Code **modularity**
- **Parallel processing**
- **Written in Python**

The Virgo computing "Tier" structure

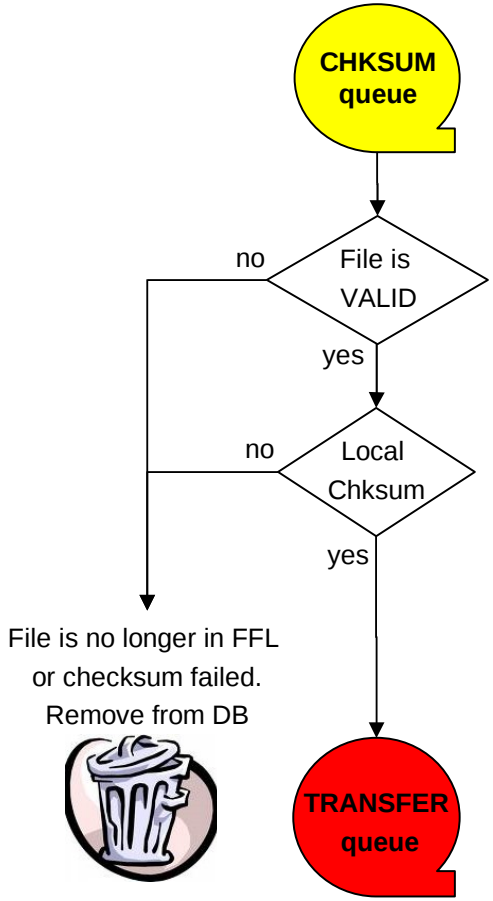


DT flux (1)

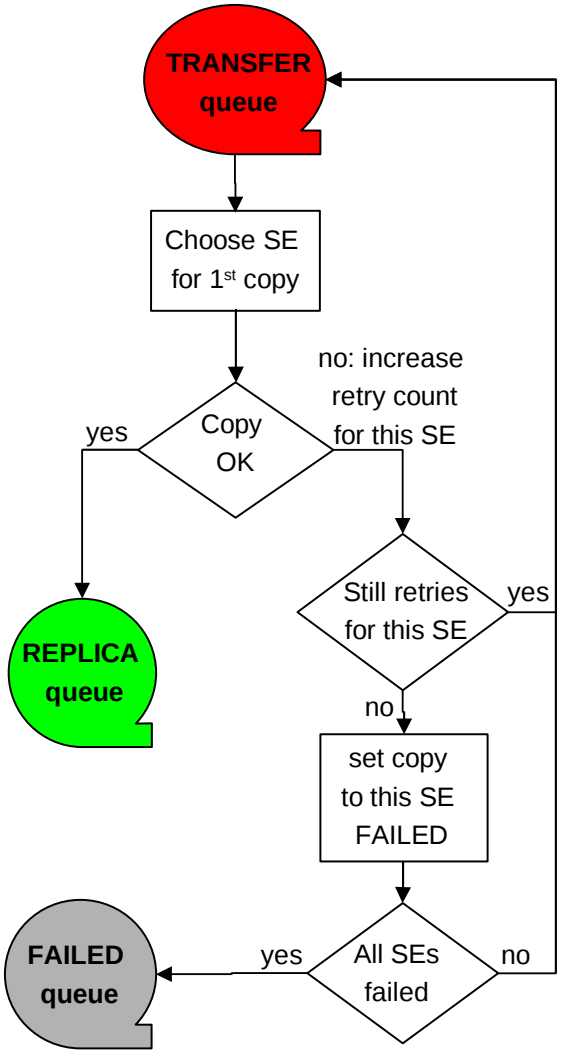
FFL Synchro thread



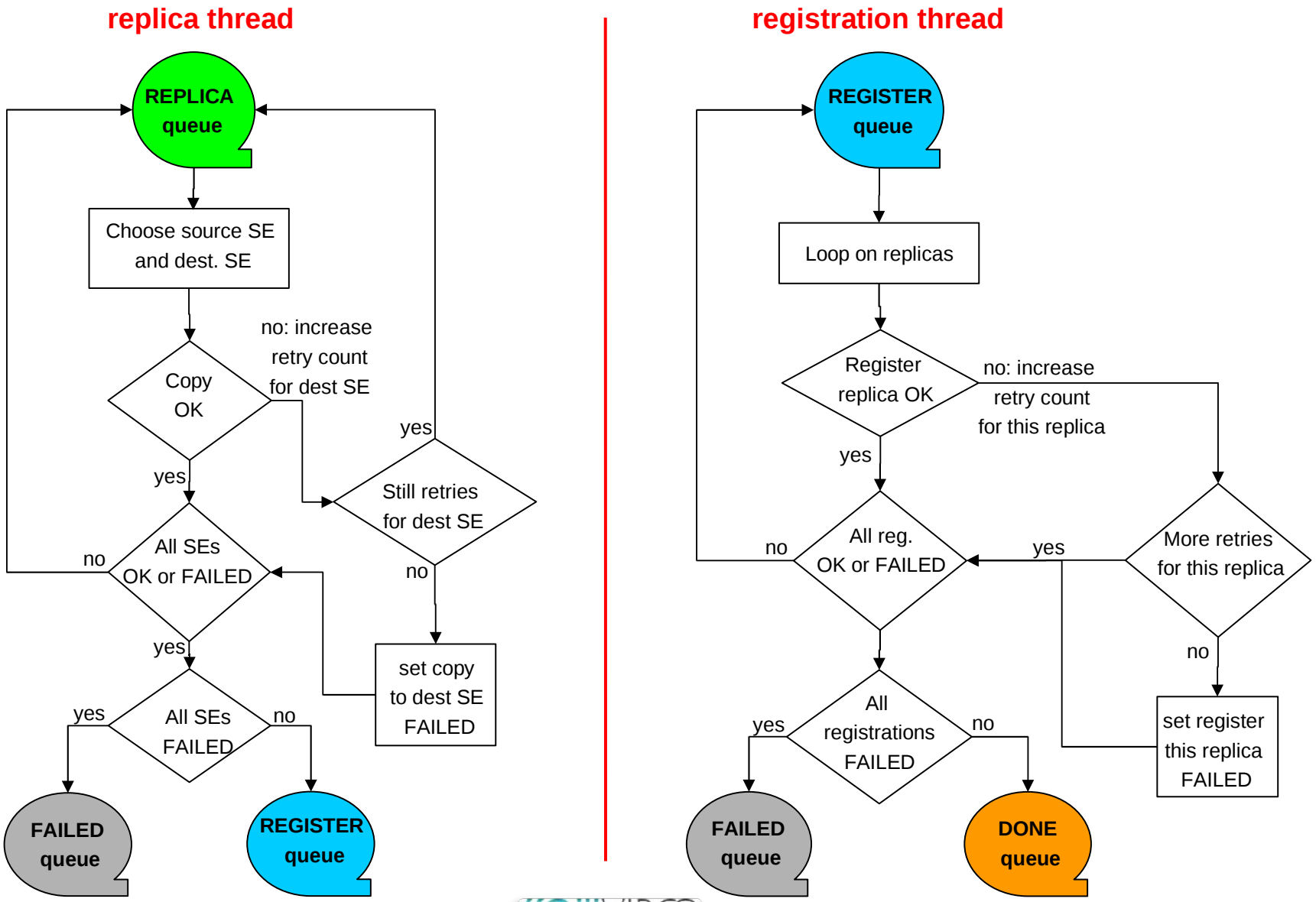
Checksum thread



1st transfer thread

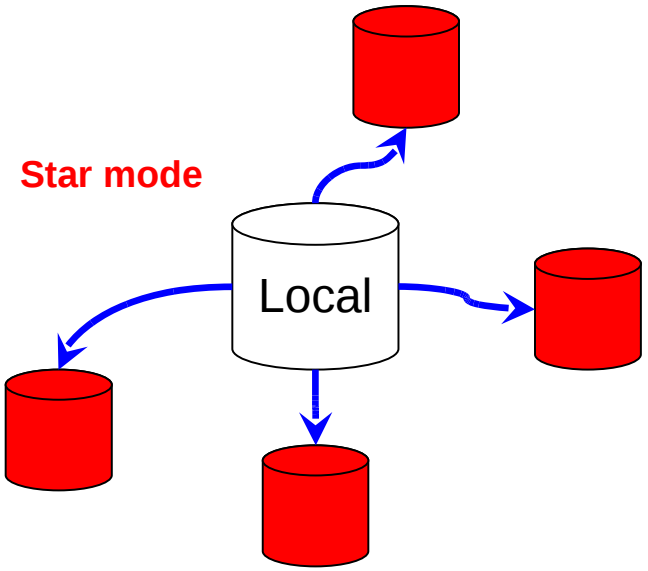


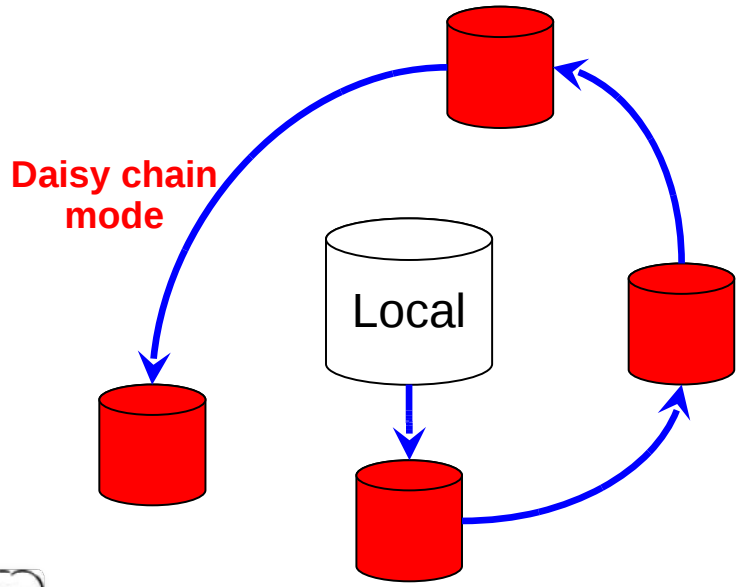
DT flux (2)


replica thread
REPLICA queue
Choose source SE and dest. SE
Copy OK
no: increase retry count for dest SE
yes
Still retries for dest SE
yes
no
All SEs OK or FAILED
no
yes
set copy to dest SE FAILED
All SEs FAILED
yes
no
FAILED queue
REGISTER queue
registration thread
REGISTER queue
Loop on replicas
Register replica OK
no: increase retry count for this replica
yes
All reg. OK or FAILED
no
yes
More retries for this replica
no
All registrations FAILED
yes
no
set register this replica FAILED
FAILED queue
DONE queue

Transfer process

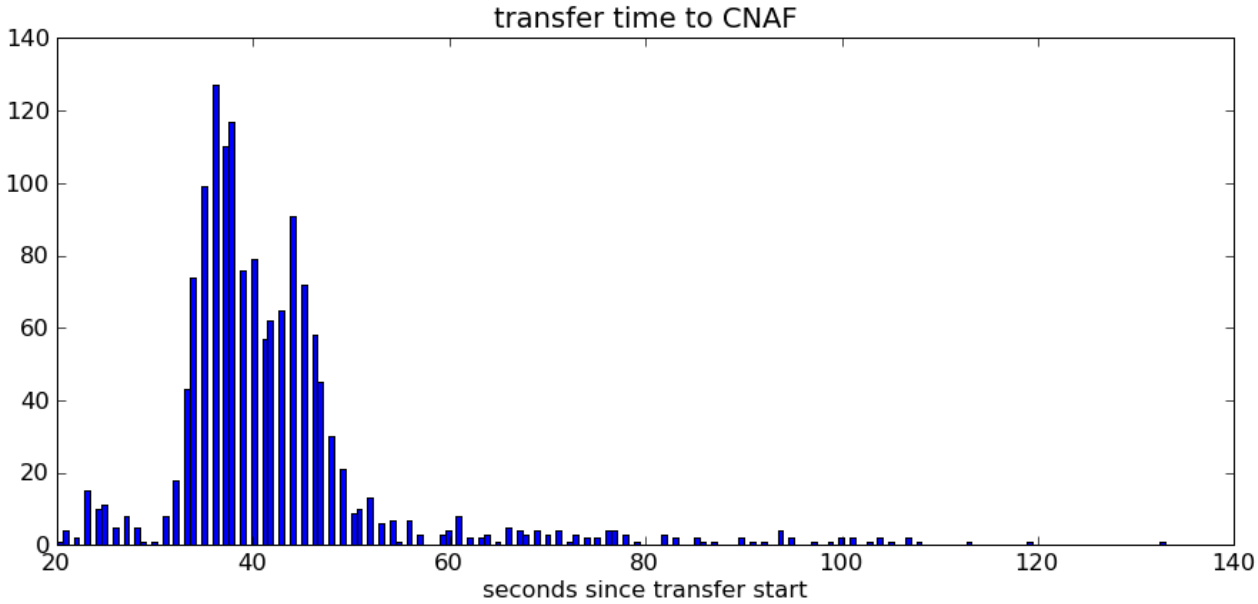
- Data transfer processes do not make use of unnecessary Grid services (Information System, file catalogue, …)
  - ➔ “brute force” **gridFTP** (lcg-cp)
  - ➔ **Minimize** possible points of failures
- “**Star**” vs. “**daisy chain**” transfer modes easily switchable (even in configuration)
- Current implementation:
  - ➔ **random choice** of source and destination SEs
  - ➔ “**daisy chain**” transfer mode

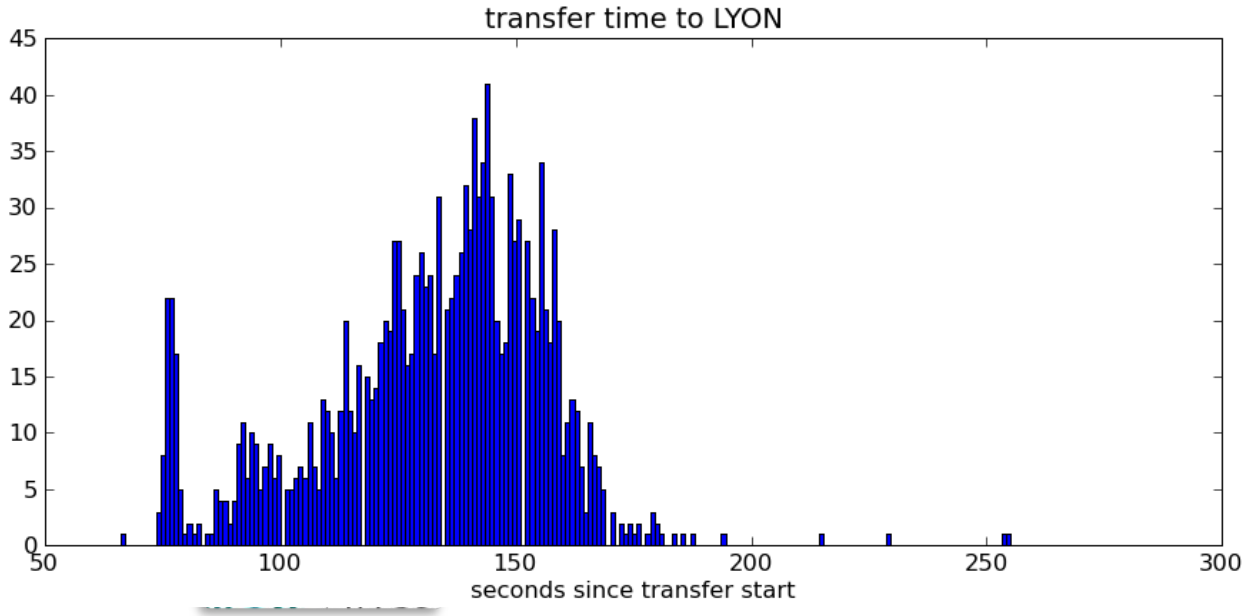




Test results (2)

- Distributions of pure transfer times





Conclusions & Work in progress

- We believe that the big part of the work is done:
  - the transfer system **works**
  - it is **robust, reliable, easily configurable and flexible**

- Work in progress:
  - Test connection with the **VDB**
  - **Queue optimization**
    - **prioritization** would allow better handling of file transfer
  - Some **benchmarking** to evaluate the **optimal transfer parameters**
    - e.g: number of parallel threads for each step
  - **"Watchdog" system** for external monitoring of DT framework:
    - Controls duration of external processes, kill them in case of stale
    - Controls general status of DT framework, restart it in case of stale
    - In case of unrecoverable problems (e.g. internal database corrupted) resets the database and restarts DT from the last "good" file
  - External monitoring requires some local **system information** functionality in the DT framework:
    - writing dynamic information such as process id, process elapsed time, etc. on temporary text files
    - Writing of a text "summary file" with the final status of each file, to allow intervention in case of failures and to set the recovery point in case of disaster
  - **Mailing service** to the DT administrators

Logical file catalogue

- If the file is replicated on **at least one SE** (and all the others are DONE or FAILED) the system registers its replicas on the **LFC**
  - ➔ Registration logic is similar to transfer one:
    - (configurable) number of retries for each replica
    - registration OK if at least one replica is registered, and no unprocessed replicas exist
- In the LFC all the **replicas** of a file are mapped to a **single logical entity**
- The **VDB** keeps the **logical name**