EWDC 2009

12th European Workshop on Dependable Computing

Toulouse, France, 14 – 15 May, 2009

Experiences in testing a Grid service in a production environment

Flavia Donno

CERN, European Organization for Nuclear Research

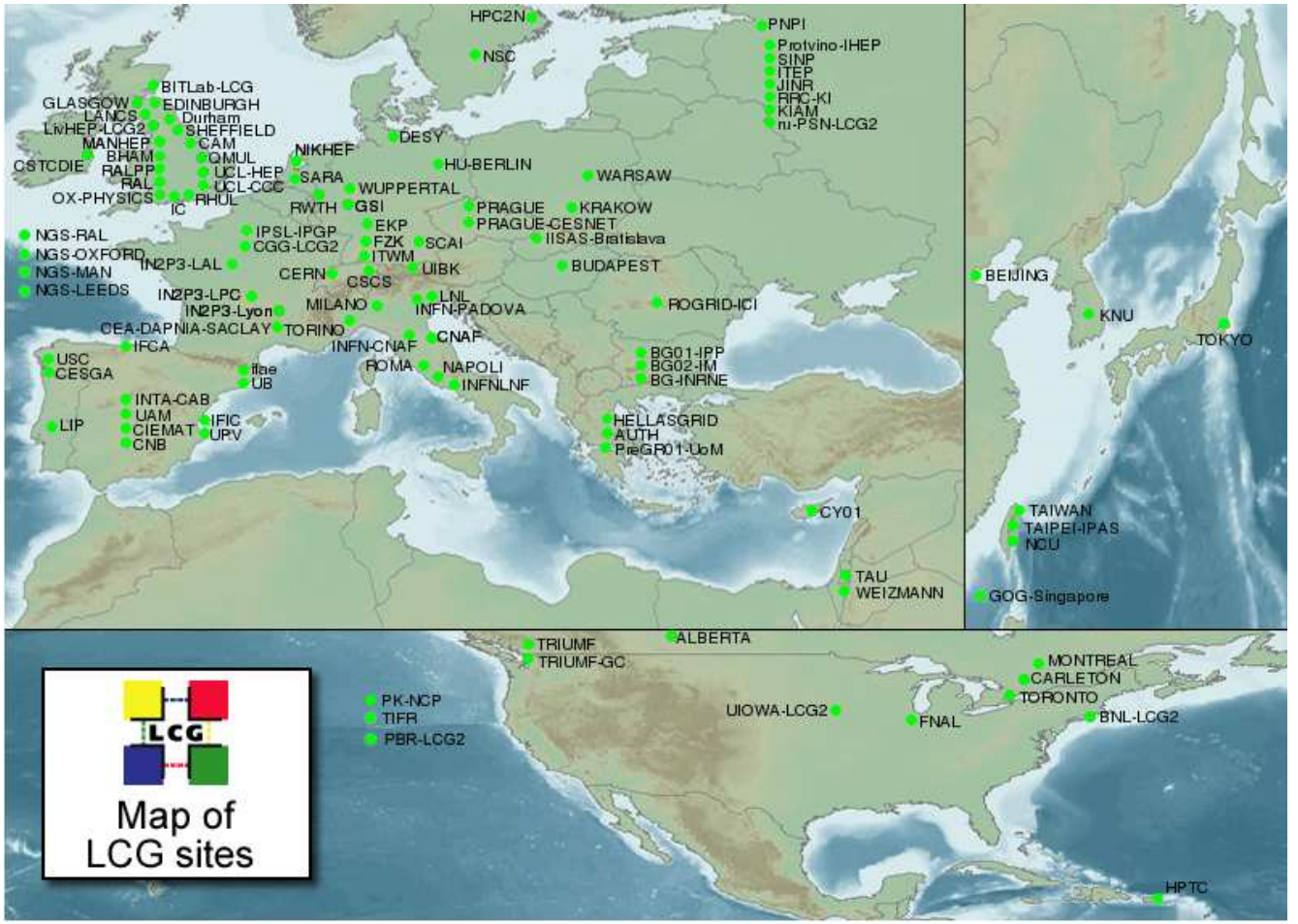
Flavia.Donno@cern.ch

Andrea Domenici

DIIEIT, Università di Pisa

Andrea.Domenici@iet.unipi.it

The Worldwide LHC Computing Grid



Grid sites in the world (Courtesy of CERN).

The Worldwide LHC Computing Grid (WLCG) is one of the largest Grid infrastructures dedicated to high-performance scientific computation, with more than 200 sites all over the world.

Mass Storage Systems

A robotic tape library.

The Grid uses heterogenous Mass Storage Systems (MSS), based on different technologies and with different capabilities and interfaces.

Storage Elements

A *Storage Element* (SE) is a Grid Service that provides:

- A mass storage system.
- A GridFTP service to provide data transfer in and out of the SE to and from the Grid.
- Local POSIX-like input/output calls providing application access to the data on the SE.
- Authentication, authorization and audit/accounting facilities.

The Storage Resource Manager

The *Storage Resource Manager* (SRM) is the common interface of the Storage Elements.

The SRM specification defines many service requests:

- *Space management* functions allow the client to reserve, allocate, release, and manage storage spaces, their types and lifetimes.
- *Data transfer* functions have the purpose of getting files into SRM spaces either from the client's space or from other remote storage systems on the Grid, and to retrieve them.
- Other function classes are *Directory*, *Permission*, and *Discovery* functions.

Testing for SRM compliance

The goals of testing are:

- Validating the SRM interface and protocol specification for adherence to the explicit and implicit user requirements, and against inconsistency, incompleteness, or inefficiency;
- validating the SRM implementations for compliance with the specification;
- checking the SRM implementations for performance and reliability.

Difficulties arise from:

- Large and complex set of service requests,
- informal specification,
- number of different implementations, and
- number of sites.

A typical SRM request

srmReserveSpace

Input parameters:

TRetentionPolicyInfo	**retentionPolicyInfo**
unsigned long	**desiredSizeOfGuaranteedSpace**
string	authorizationID
unsigned long	desiredSizeOfTotalSpace
int	desiredLifetimeOfReservedSpace
TTransferParameters	transferParameters

. . . *more optional parameters*

Output parameters:

TReturnStatus	**returnStatus**
string	requestToken

. . . *more optional parameters*

Space properties

The *retentionPolicyInfo* parameter specifies two properties of the requested space:

- **Retention policy**, likelyhood of file loss: REPLICA, OUTPUT, CUSTODIAL.
- **Access latency**, readiness of file access: ONLINE (e.g., disk), NEARLINE (e.g., tape).

A *storage class* is a combination of retention policy and access latency. In the WLCG, the following storage classes are supported:

- **Tape0Disk1**: *Replica*, *Online*;
- **Tape1Disk1**: *Custodial*, *Online*;
- **Tape1Disk0**: *Custodial*, *Nearline*.

A large test space

- The *srmReserveSpace* request has nine input arguments.
- Some arguments range over a finite set of values.
- Other arguments range over theoretically infinite sets of values.
- **Equivalence partitioning** enables us to reduce the number of values to consider

. . . **but** we are still left with some $\mathbf{20000}$ test cases.

And then we have the other 38 requests!

Use-case analysis

We may shrink the test space by pruning argument values and combinations that may be ruled out based on the actual operating conditions in the WLCG.

- The SRM specification is very general and flexible:
  - many negotiations are possible;
  - much leeway for implementation or site dependent defaults;
  - allowance for future requirements.
- The full power of the SRM is currently not used by the implementations...
- yet they are SRM-compliant.

A careful analysis of usage patterns and implementation constraints enables us to significantly reduce the size of the test space.

This requires *a close interaction* between testers, users, and developers.

Reshaping the signature

We prune the domain of an argument and eliminate some altogether:

retentionPolicyInfo: only a few of the possible values are in use.

authorizationID: unused, as in the WLCG credentials are not passed as parameters (certificates are used instead).

transferParameters: unused, as site dependent defaults are used.

Other parameters (not shown) are similarly ignored.

However, we consider the validity or absence of a user certificate as an extra argument.

We can then test the request with only five variable arguments, thus reducing the test space size to about $\mathbf{200}$ cases.

Modeling constraints and conditions (1)

Cause-effect graphing is used to derive test cases covering constraints and operating conditions, e.g.:

Causes:

1 retentionPolicyInfo is not NULL

2 **retentionPolicyInfo is supported by server**

. . .

11 requestToken is returned **[11 and 12 mutually exclusive]**

12 spaceToken is returned **[12 requires 13]**

13 sizeOfGuaranteedReservedSpace and lifetimeOfReservedSpace are returned

. . .

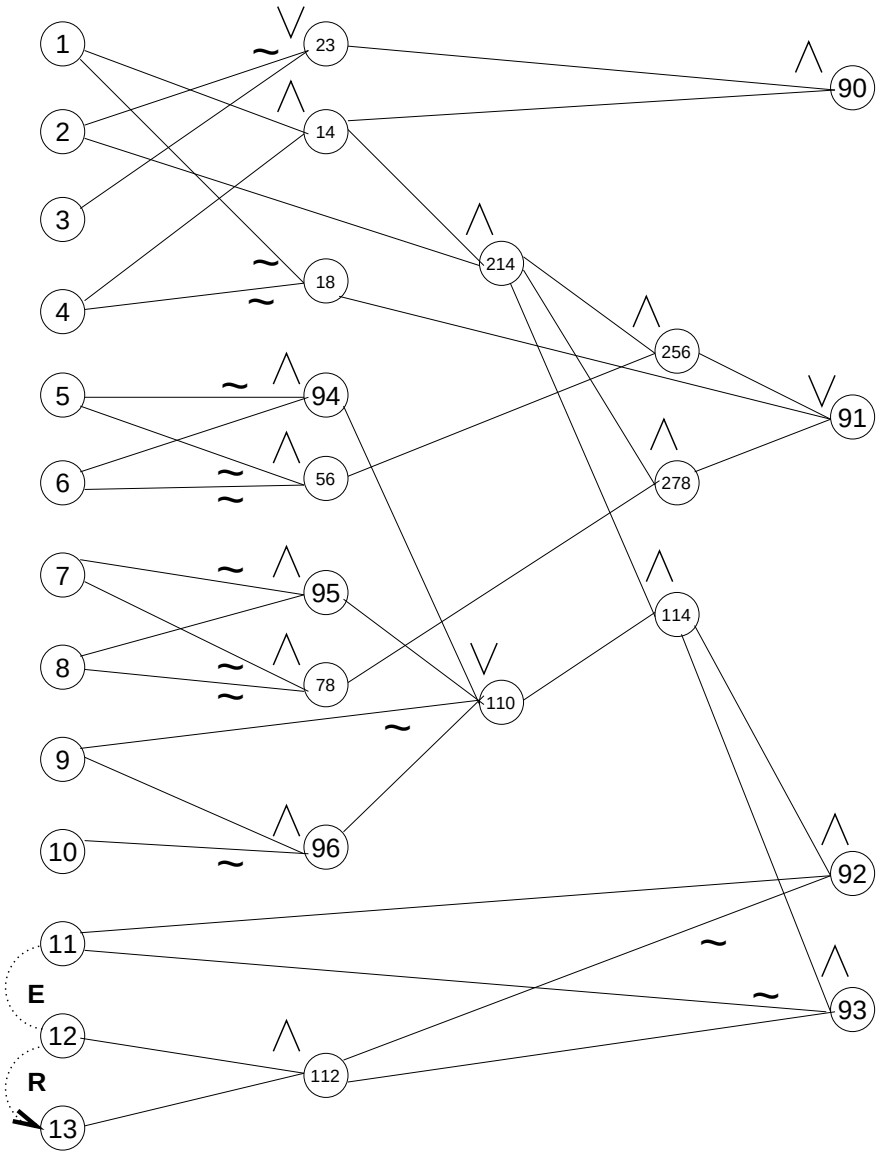
Effects:

94 sizeOfGuaranteedReservedSpace = default

95 lifetimeOfReservedSpace = default

96 transferParameters is ignored

Modeling constraints and conditions (2)

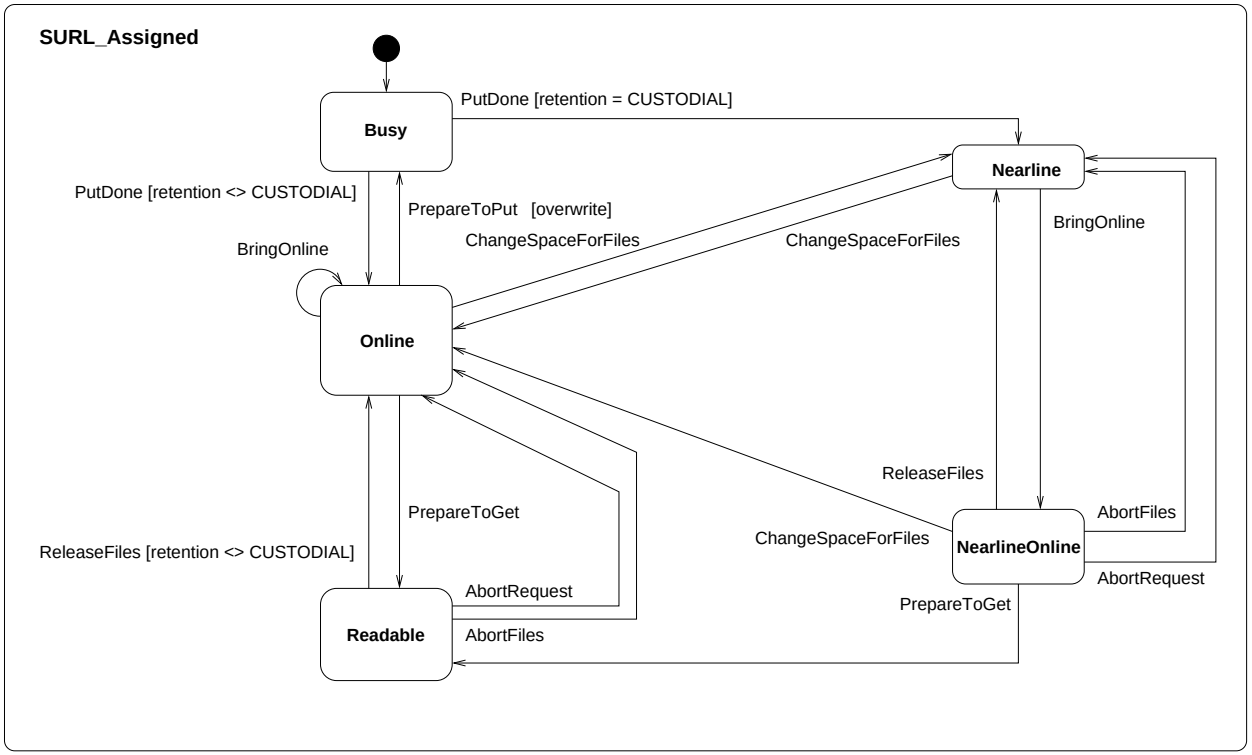


Cause-effect graph for the *srmReserveSpace* request.

Error guessing

Error guessing $=$ pragmatic knowledge $+$ formalization.

Example: formalization of behavior by state machines led to discover unexpected interactions.



Partial state machine for a file.

Test case families

Five families of test cases have been designed:

Availability to check the availability in time of the SRM service end-points.

Basic to verify basic functionality of the implemented SRM APIs.

Use Cases to check boundary conditions, use cases derived by real usage, function interactions, exceptions, etc.

Exhaustion to check “extreme” values and properties of input and output arguments such as length of filenames, URL format, etc.

Stress tests to stress the systems, identify race conditions, study the behavior of the system when critical concurrent operations are performed, etc.

The SRM testbed

The following SRM implementations are being tested:

CASTOR developed at CERN, uses tape libraries with disk servers as front-end caches. SRM 2.2 implementation developed at RAL (UK) (**4 Tier-1 sites**).

dCache developed at DESY (Germany), uses multiple MSS backends, both custom and proprietary. SRM 2.2 implementation developed at FNAL (USA) (**7 Tier-1 sites**).

DPM developed at CERN, a disk-only MSS. SRM 2.2 implementation developed at CERN (**6 Tier-2 sites**).

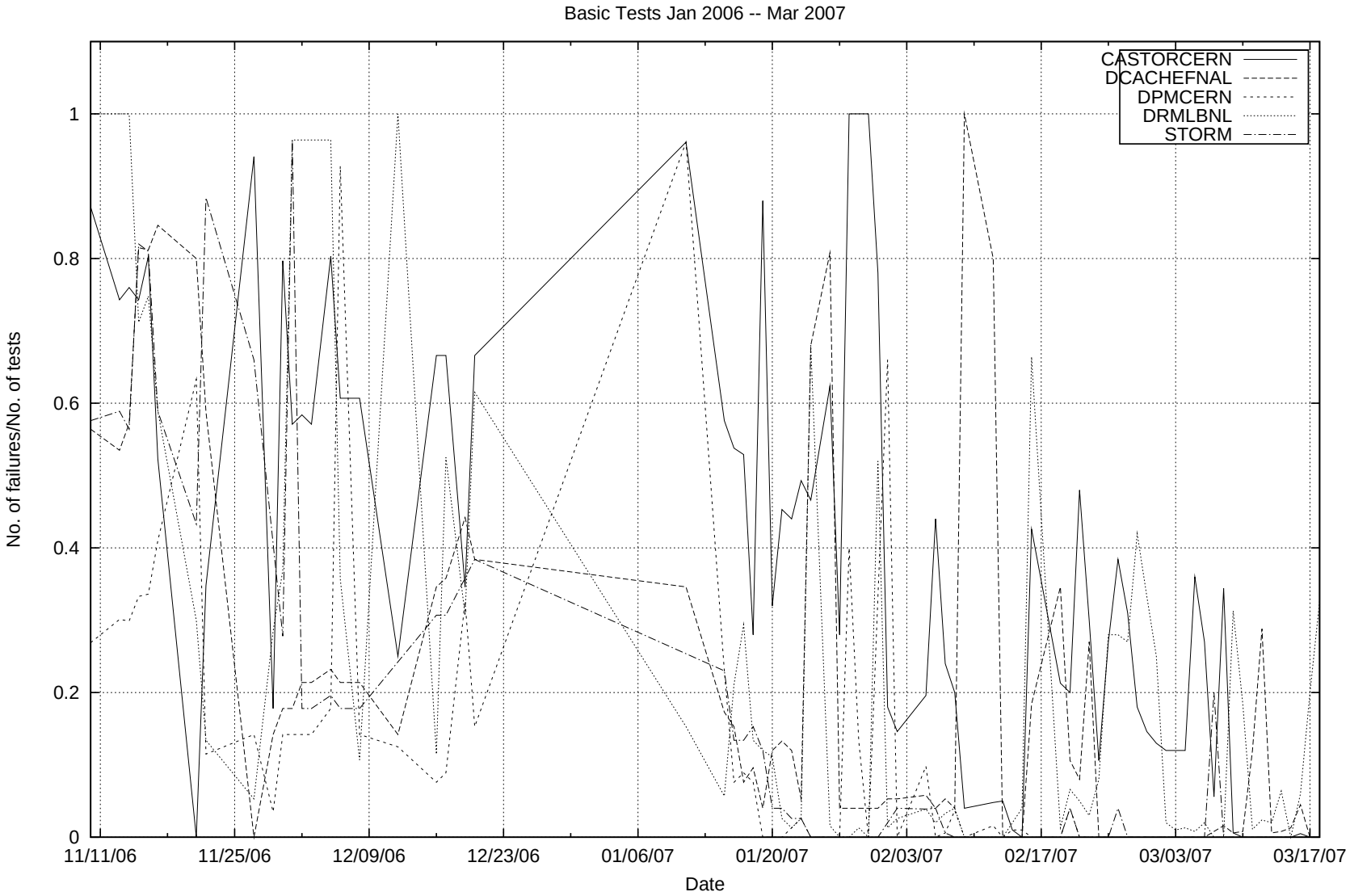
DRM/BeStMan is the LBNL (USA) disk-based storage system. LBNL has been the first promoter of SRM, and this storage system was the first prototype on which SRM has been tested (**1 Tier-2 sites**).

StoRM developed at CNAF (Italy), uses parallel file systems such as GPFS or PVFS. (**4 Tier-2 sites**).

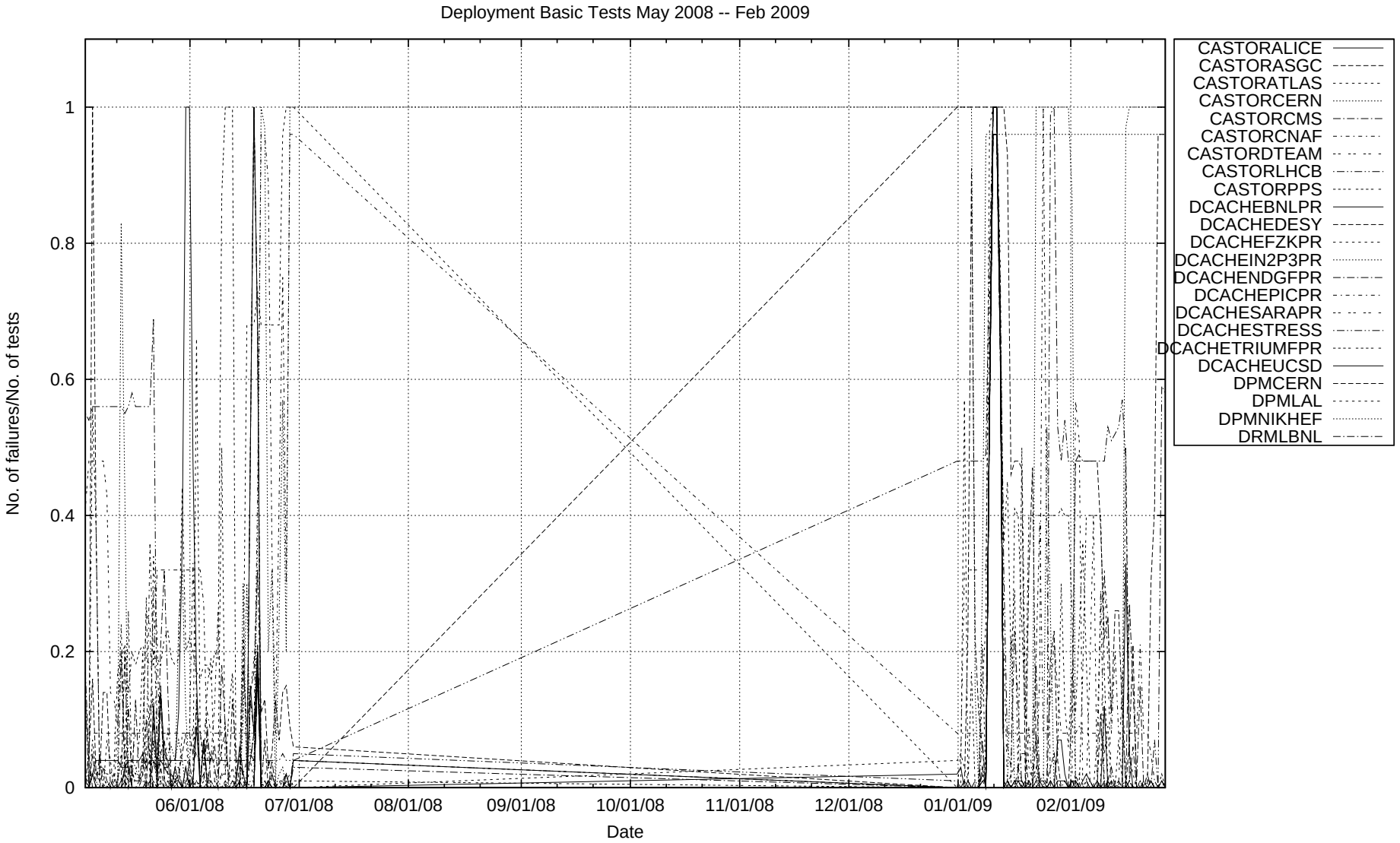
Test execution and analysis

- Execution framework based on S2 and shell scripts.
  - invoke SRM requests;
  - make checks on return codes;
  - define complex test actions.
- Automatic execution and result logging six times a day.
- Monthly plots for each test family.

Pre-production testing



In-production testing



Conclusions

- A complex Grid service such as the SRM poses a challenge to testers.
- Standard testing techniques are fundamental. . .
- but cannot be applied mechanically.
- Testers, users, and developers cannot live on different planets.
- The development of a (semi)formal model has helped design a few families of tests.
- The testing campaign itself has motivated the developers to reconsider many of the initial assumptions and decisions, leading to solutions that seem to better satisfy the needs of the users.

Thank you

Merci