The StoRM Certification Process

Elisabetta Ronchieri¹, Michele Dibenedetto¹, Riccardo Zappi¹, Stefano Dal Pra¹, Cristina Aiftimiei², Sergio Traldi²

¹INFN CNAF, Viale Berti Pichat 6/2, Bologna I-40126, Italy, ²INFN Padua, Via Marzolo 8, 35131 Padova, Italy
E-mail: elisabetta.ronchieri@cnaf.infn.it

Abstract. StoRM is an implementation of the SRM interface version 2.2 used by all Large Hadron Collider (LHC) experiments and non-LHC experiments as SRM endpoint at different Tiers of Worldwide LHC Computing Grid. The complexity of its services and the demand of experiments and users are increasing day by day. The growing needs in terms of service level by the StoRM users communities make it necessary to design and implement a more effective testing procedure to quickly and reliably validate new StoRM candidate releases both in code side (for example via test units, and schema valuator) and in final product software (for example via functionality tests, and stress tests). Testing software service is a very critical quality activity performed in a very ad-hoc informal manner by developers, testers and users of StoRM up to now. In this paper, we describe the certification mechanism used by StoRM team to increase the robustness and reliability of the StoRM services. Various typologies of tests, such as quality, installation, configuration, functionality, stress and performance, defined on the base of a set of use cases gathered as consequence of the collaboration among the StoRM team, experiments and users, are illustrated. Each typology of test is either increased or decreased easily from time to time. The proposed mechanism is based on a new configurable testsuite. This is executed by the certification team, who is responsible for validating the release candidate package as well as bug fix (or patch) package, given a certain testbed that considers all possible use cases. In correspondence of each failure, the package is given back to developers waiting for validating a new package.

1. Introduction

StoRM is used as Storage Resource Manager (SRM) endpoint in the context of the Large Hadron Collider (LHC) at European Centre for Nuclear Research (CERN) for the major High Energy Physics (HEP) experiments such as Alice, ATLAS, CMS and LHCb; also in the context of non-LHC experiments like Babar, CDF, and SuperB, and astrophysics and space physics experiments like VIRGO, ARGO, AMS, PAMELA and MAGIC. StoRM, implementing SRM interface version 2.2, offers standard protocols to receive data and store them in different Tiers of Worldwide LHC Computing Grid (WLCG)¹. It is developed at INFN CNAF, the National Computing Center located in Bologna, Italy, of the Italian National Institute for Nuclear Physics (INFN). INFN CNAF is also called the INFN Tier-1 where StoRM is largely used by the LHC and non-LHC experiments. The increasing complexity of StoRM services makes necessary to ensure software quality, especially in the HEP domain where a fault in the software may lead to lose data. It is important to determine if the StoRM services meet specification and if their

¹ The Worldwide LHC Computing Grid (WLCG). http://lcg.web.cern.ch/LCG/
outputs are correct. The growing demand of the StoRM communities in terms of service level conveys that a more effective testing procedure is essential in the life cycle of StoRM in order to quickly and reliably validate new StoRM software releases. Software testing is a very critical quality activity performed in a very ad-hoc informal manner by developers, testers and users of StoRM up to now. However, because of the effort, it is interesting to know which are the most critical parts in the development and certification process to efficiently save effort. All tests that settle the certification process, are the collaboration result amongst the StoRM team, experiments and users. Furthermore, the certification testbed set up at INFN CNAF tries to be much similar to the production environment in order to responsively provide good answers to common issues. This paper is organized as follows. After a short overview of StoRM in Section 2 detailing StoRM architecture and the most complex StoRM deployment, we describe literately the certification procedure in Section 3. Then, in Section 4 we detail test categories used in the StoRM certification process. In Section 5 we focus on the certification testbed configurations providing details of tests implementation. Finally, Section 6 presents the concluding remarks as well as future directions.

2. StoRM Overview

StoRM is a SRM solution designed to leverage the advantages of cluster file systems, such as GPFS from IBM [1] and Lustre from SUN\textsuperscript{2}, and standard POSIX systems in a Grid Environment. It is characterized by being a SRM service for different disk-based storage systems, easy-to-configure after an initial effort, efficient and secure. The latest stable version of StoRM (v1.5.x) enables the management of hierarchical storage resource through a generic interface, used in this configuration at the Italian INFN Tier-1 in Bologna managing a hierarchical system based on GPFS and TSM\textsuperscript{3}. StoRM has been included in the European Middleware Initiative (EMI) project\textsuperscript{4}.

2.1. StoRM Architecture

StoRM has a multi-layer architecture composed of two main stateless components, called FrontEnd and BackEnd, and one DataBase as shown in Figure 1.

---

\textsuperscript{2} Lustre, High Performance and Scalability, \url{http://wiki.lustre.org/index.php/Main_Page}

\textsuperscript{3} IBM, Overview - Tivoli Storage Manager Supported Operating Systems, \url{http://www-01.ibm.com/support/docview.wss?uid=sug21243309}

\textsuperscript{4} European Middleware Initiative, \url{http://www.eu-emi.eu/}
FrontEnd exposes the SRM Web service interface, manages user authentication, stores SRM requests data into Database, retrieves the status of ongoing requests from Database, and interacts with BackEnd. BackEnd is the core of StoRM service since it executes all synchronous and asynchronous SRM functionalities. It processes the SRM requests managing files and space, it enforces authorization permissions and it can interact with other Grid services, such as external authorization service and mapping service. Moreover, BackEnd is able to use advanced functionalities provided by some file systems to accomplish space reservation requests. BackEnd uses a plug-in mechanism to easily extend new support for different file systems. Database is used to only store SRM requests data and space metadata. It does not hold any crucial information but only transient data. An accidental loss of the full database simply leads to failing ongoing SRM requests; space metadata will be recreated at next restart.

2.2. StoRM Deployment
Figure 2 shows the most complex deployment of StoRM. In the center of Figure, it is possible to observe the main StoRM components where multiple FrontEnd instances are deployed on separate machines. All these FrontEnds are configured to work on the same Database and with the same BackEnd service form the StoRM FrontEnd pool. In addition, another StoRM component, called Dynamic Info Provider, responsible to collect and publish status information on Information Service, is installed and configured in the same machine where BackEnd and Database are. On the left side of the Figure a list of the GridFTP [2] instances forming the GridFTP pool are shown capable of calculating checksum for files transferred using the GridFTP servers, whilst on the right side a list of the Checksum instances forming the Checksum pool are highlighted responsible to calculate checksum for files on demand. Moreover on the right side it is shown YAMSS [3] a service that provides StoRM BackEnd with all need to interact with tape. As consequence of the complexity of the StoRM system and the growing number of communities requirements, the activity of verification and rigorous validation has become more and more important to ensure quality and guarantee that the StoRM implementation is consistent with the specification. Validation and verification represent a new activity in the overall software development life cycle of StoRM where most effort is needed and most of the effort may be gained.

3. Certification Procedure
In this section we explain the procedure that is used to certify StoRM software, specifying how the certification is modelled and highlighting what actors (i.e. developers, testers, pre-production sites, and release manager) are involved in each decision as described in Figure 3. The procedure is part of the StoRM software release life cycle composed of five phases and three decisions: phases show the maturity of software detailed in Table 1, whilst decisions, commonly a Passed or Failed test (or a Positive or Negative question), determine a change in the progressing phases (Certification Testing when tester performs tests; Pre-Production testing when sites involved in the pre-production session provides information for feedback analysis; Feedback Analysis

![Figure 3. Workflow representing certification procedure.](image)
when release manager evaluates software in manufacturing release establishing if it can be made available to the community.

Table 1. Phases.

| Name               | Description                                                                 |
|--------------------|-----------------------------------------------------------------------------|
| Alpha              | It is when software is ready to be tested and usually software is said to be feature complete. Software in alpha phase is generally unstable, therefore it is uncommon to have alpha software available externally. |
| Release Candidate  | It is when software is potential to be a final product and ready to be released unless fatal bugs emerge. At this time all product features have been designed, coded and tested. |
| Manufacturing Release | It is when software is ready for or has been delivered or provided to the customer for installation and distribution. Software in this phase has a sufficient quality for mass distribution. |
| General Availability | It is when software have been made available to the community through a dedicated mailing list. At this point software is ready for the production that represents the final version of a particular product. |
| Rejected Release   | The rejected release phase is when wrong behaviour is found in software. |

Figure 3 shows the workflow followed by the StoRM team to certify its product. As soon as developers deliver software in alpha phase, testers start the certification testing decision: if they evaluate that tests are passed, testers deliver a release candidate. At this point some sites start the pre-production testing decision: if after the pre-production session they are able to provide all the information required for feedback analysis, sites deliver a manufacturing release that is the Release Candidate plus the produced feedback material. At this time the release manager start feedback analysis decision: if he or she establishes that manufacturing release can be made available to the community, the general availability is announced through a dedicated mailing list by the release manager. In case of failed decision software is rejected going back to developers who will evaluate what actions need to be taken to restart the workflow in order to certify the StoRM product. Certification testing is based on test scenarios detailed in the StoRM Certification Test Plan. Test scenarios take into consideration non-cluster and cluster StoRM configurations shown in Figure 4 and in Figure 5 respectively, and include tests relate to disk and tape storage, bug fixes and patches. The testing process evolves over time, therefore the StoRM Certification Test Plan is constantly updated to reflect new functionalities. If the StoRM Certification Test Plan is updated, certified software products may have to be re-tested in accordance with the new StoRM Certification Test Plan. In addition, certification is for specific product version.

4. Test Categories
Typically the development process involves various types of testing. Each test type addresses a specific testing requirement. There are many different test categories in literature [4] but we have selected three of them for the StoRM certification process: sanity, system and stress response metrics.

Sanity Testing is conducted to ensure known and predictable results and to assure that the installation and configuration of the StoRM work as expected prior to a more exhaustive
round of testing. For example, we have included in this category the following tests: verify if a given software release is well installed (e.g., checking list of rpms), check installed release version, and check the StoRM daemon status. Other tests included in this category consider that the StoRM services are configured using yaim tool\(^5\) that identifies each of the StoRM service with a given node that is installed and configured via yaim. For example, we have added the following tests: verify if the StoRM attributes in the yaim definition node file are well set according to the agreement node configuration (e.g., the \texttt{STORM\_STORAGEAREA\_LIST} attribute in the \texttt{lcg-siteinfo.def} file), verify if StoRM properties and attributes in the StoRM configuration files are well set according to the corresponding node-type configuration, verify if yaim configuration is finished with success (e.g., checking the \texttt{/opt/glite/yaim/log/yaimlog} file).

**System Testing** is conducted on a complete integrated StoRM system to evaluate if the StoRM system is compliant with its specified requirements. For example we have included tests that also consider the SRM specification \cite{SRMSpec}: check the behaviour of the StoRM system when the storage-type value is permanent, or when the SURL (Site URL) value contains several ‘/’, or when the file has a nearline or online accessing latency. System testing ensures that entire integrated software system meets requirements. This type of test falls within the scope of black box testing, and as such, requires no knowledge of the inner design of the code or logic.

**Stress Testing** is conducted to determine the stability and robustness of the StoRM system or a StoRM component. It evaluates the StoRM system or a StoRM component at or beyond the limits of its specified requirements to determine the load under which it fails and how, the memory usage, the number of required FrontEnds to obtain a given request rate, the number of required GridFTP to obtain a given transfer rate, and the number of required Checksum to obtain a given result rate. A graceful degradation under load leading to non-catastrophic failure is the desired result. This type of test put a greater emphasis on robustness, availability, and error handling under a heavy load, than on what would be considered correct behavior under normal circumstances.

5. **Certification Testbed**
The StoRM certification process is experimented and verified on top of a well-defined testbed that provides rigorous, transparent and repeatable testing always closer to the production

\(^{5}\) YAIM, https://twiki.cern.ch/twiki/bin/view/EGEE/YAIM
environment. According to what has been detailed in Section 3 for test scenarios, testbed considers at this point the following StoRM configurations also schematized in Figure 6.

In all the StoRM configurations all test categories detailed in Section 4 are executed. The pre-production sites are two: the first is the INFN Padua site supporting a non-cluster configuration based on ext3 as filesystem; whilst the second is the INFN CNAF Bologna site supporting two non-cluster configurations based on ext3 and GPFS as filesystems respectively, and a cluster configuration based on GPFS as filesystem enabled to handle access to disk or to tape.

Figure 6. Certification Testbed Overview.

5.1. Implementation Status
The sanity and system tests are implemented using the Python language\(^6\) with its unit testing framework. We have decided to use Python in order to reduce development time, improve program maintenance being the code extremely readable and reduce training effort being language easy to learn. These tests are easily configurable using JSON\(^7\) as lightweight data interchange format. Each test category has two configuration files: the first contains unchangeable information related to software to be tested, like the list of rpms that forms a general availability release or a release candidate; instead the second contains mandatory changeable information related to the test scenarios. Some stress tests are implemented using bash shell-scripting language. They will be ported in Python.

6. Conclusions
This paper shows the certification process defined and used by StoRM. We strongly believe that a more effective testing procedure to quickly validate new StoRM General Availability release is fundamental to increase the robustness and reliability of StoRM services. The presence of an automated configurable test system is also crucial to provide turnkey solutions in the design, manufacture and support of the StoRM tests, therefore we are sure that the effort spent in its implementation will be re-paid in the next years. Until now applying this certification process we have been able to: find bugs before users, improve code, and guarantee a better quality of the General Availability release; and define bunch of tests closer to the real production activity. Validation and verification represent a new activity in the overall software development life cycle of StoRM where most effort is needed and most of the effort may be gained. Our future work will follow two avenues. The first one will integrate StoRM testsuite with an ad-hoc Infrastructure like the one provided by the EMI project. The second one will provide a software solution to automate the testsuite execution given a certain StoRM configuration.

Acknowledgements
This work was partially funded by the Italian INFN - National Institute of Nuclear Physics - CNAF, and by EMI EU project INFSO-RI-261611.

References
[1] Schmuck F and Haskin R 2002 Gpfs: A shared-disk file system for large computing clusters USENIX FAST 2002 Conference on File and Storage Technologies URL http://www.usenix.org/events/fast02/full_papers/schmuck/schmuck.pdf
[6] Python Documentation Index, http://www.python.org/doc/
[7] JSON, http://json.org/
[2] Allcock W 2003 Gridftp: Protocol extensions to ftp for the grid Global Grid Forum GFD-RP vol 020 URL http://www.ggf.org/documents/GWD-R/GFD-R.020.pdf

[3] Andreotti D, Bonacorsi D, Cavalli A, Pra S D, dellAgnello L, Forti A, Grandi C, Gregori D, Gioi L L, Martelli B, Prosperini A, Ricci P, Ronchieri E, Sapunenko V, Sartirana A, Vagnoni V and Zappi R 2011 First experiences with cms data storage on the gemss system at the infn-cnaf tier-1 Data Driven e-Science, Use Cases and Successful Applications of Distributed Computing Infrastructures (ISGC 2010) ed Simon C L and Yen E (1st Edition., 2011, X, 540 p., Hardcover)

[4] McCaffrey J D 2009 Software Testing: Fundamental Principles and Essential Knowledge [Paperback] (BookSurge Publishing)

[5] Sim A, Shoshani A, Badino P, Barring O, Baud J, Corso E, Witt S D, Donno F, Gu J, Haddox?Schatz M, Hess B, Jensen J, Kowalski A, Litmaath M, Magnoni L, Perelmutov T, Petravick D and Watson C 2009 The Storage Resource Manager, Interface Specification, Version 2.2 Open Grid Forum URL http://www.ggf.org/documents/GFD.129.pdf