T-StoRM: a StoRM testing framework

Elisabetta Ronchieri[∗†]
INFN CNAF
E-mail: `elisabetta.ronchieri@cnaf.infn.it`

Michele Dibenedetto
INFN CNAF
E-mail: `michele.dibenedetto@cnaf.infn.it`

Riccardo Zappi
INFN CNAF
E-mail: `riccardo.zappi@cnaf.infn.it`

Cristina Aiftimiei
INFN, Sezione di Padova
E-mail: `cristina.aiftimiei@pd.infn.it`

Vincenzo Vagnoni
INFN, Sezione di Bologna
E-mail: `vincenzo.vagnoni@bo.infn.it`

Valerio Venturi
INFN CNAF
E-mail: `valerio.venturi@cnaf.infn.it`

StoRM, an implementation of the SRM interface, is a multi-service software subject to intense testing, validation and verification activities in order to guarantee high-quality services. Its characteristics of being usable on different file systems and of supporting several transfer protocols raise the need of StoRM to be validated on a variety of deployment scenarios with multiple machines. With this purpose in mind, T-StoRM is a StoRM testing framework that aims at improving and automating the service evaluation. It provides several abstract support classes that can simplify writing test suites, which are logically put into group of similar test cases amongst installation, configuration, conformance, system, regression and stress categories. This solution addresses the need of improving software development life cycle and optimizing the deployment of a new software release. In this paper, we describe T-StoRM and the supported tests. Furthermore, we present its current and near future usage.

EGI Community Forum 2012 / EMI Second Technical Conference,
26-30 March, 2012
Munich, Germany

∗Speaker.
†Correspondent author.

1. Introduction

StoRM is used as a Storage Resource Manager (SRM) endpoint in the context of the Large Hardon Collider (LHC) at European Centre for Nuclear Research (CERN) for the major High Energy Physics (HEP) experiments such as Alice, ATLAS, CMS and LHCb; in the context of non-LHC experiments including Babar, CDF, and SuperB, and astrophysics and space physics experiments namely VIRGO, ARGO, AMS, PAMELA and MAGIC. StoRM, implementing the 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 a collaboration between INFN CNAF[2] and IGI[3] (the Italian Grid Infrastructure), and part of the European Middleware Initiative (EMI) project[4].

StoRM is a SRM solution designed to leverage the advantages of cluster file systems, such as GPFS from IBM [1] and Lustre from SUN[5], 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 (v1.8.x) enables the management of hierarchical storage resources through a generic interface that is based on GPFS and TSM[6]: this configuration is used at the Italian INFN Tier-1 in Bologna. StoRM is also characterized by supporting several transfer protocols, such as gsiftp, file, https and http, publishing information by using the GLUE standard and supporting VOMS and GSI for authentication and authorization.

The increasing complexity of the 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 the SRM specifications and if their outputs are correct. The growing demand of the StoRM communities in terms of service level conveys the message that a more effective testing framework is essential in the life cycle of StoRM in order to quickly and reliably validate new StoRM software releases.

StoRM has a multi-layer architecture composed by two main stateless components, called FrontEnd and BackEnd, and one DataBase as described in Table 1.

The StoRM deployments, the examples of which are shown in the following Figures, can range from the simplest one to the most complex according to users' requirements. Figure 1 shows the standalone StoRM deployment where the main components are deployed on the same machine. Figure 2 shows the common StoRM deployment 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, that is called Dynamic Info Provider and is responsible for collecting and publishing status information on the Information Service, is installed and configured in the same machine where the BackEnd and DataBase are. On the left side of the Figure a list of the GridFTP [2] instances forming the GridFTP pool is shown. The files transfers performed via gsiftp protocol are handled

[1]The Worldwide LHC Computing Grid (WLCG), `http://lcg.web.cern.ch/LCG/`

[2]INFN CNAF, `http://www.cnaf.infn.it/en`

[3]Welcome to IGI, the Italian Grid Infrastructure, `http://www.italiangrid.it/`

[4]European Middleware Initiative, `http://www.eu-emi.eu/`

[5]Lustre, High Performance and Scalability, `http://wiki.lustre.org/index.php/Main_Page`

[6]IBM, Overview - Tivoli Storage Manager Supported Operating Systems, `http://www-01.ibm.com/support/docview.wss?uid=swg21243309`

Components	Descriptions
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 the authorization permissions and it can interact with other Grid services, such as the external authorization service and the mapping service. Moreover, BackEnd is able to use advanced functionalities provided by some file systems to accomplish the 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 the ongoing SRM requests; space metadata will be recreated at the next restart.

Table 1: Some of the SRM operations with the supported SRM clients.

by those GridFTP servers that are able to calculate file checksum on the fly. Figure 3 shows the most complex deployment of StoRM. In addition to Figure 2, on the left side of the Figure a list of the GridHTTP instances that forms the GridHTTP pool are shown: they handle file transfers performed via the http or https protocols.

StoRM is a multi-service software subject to intense testing, validation and verification activities. T-StoRM is a StoRM testing framework that is able to optimize the deployment of a new StoRM software release to be certified; to simplify the reproduction of environments where the error events raised to react as quick as possible; to provide users that experience criticalities with suitable support in reasonable time; to delegate validation and verification of remote sites to their administrators; to reduce time and effort spent on users' support; and to improve its software development life cycle. It has been designed considering the need of the StoRM team in order to simplify the integration of new test suites and the verification of the performed tests. T-StoRM performs automating SRM testing and discourages manual testing that are time consuming, inconsistent to be effective, error prone, and inaccurate to cover all cases.

The rest of the paper is organized as follows. Section 2 details T-StoRM, whilst Section 3 provides the list of testing levels that T-StoRM supports. Section 4 details how to manage a test in T-StoRM. Section 5 describes the current and near future usage of T-StoRM. Finally, Section 6 provides future work, while Section 7 concludes.

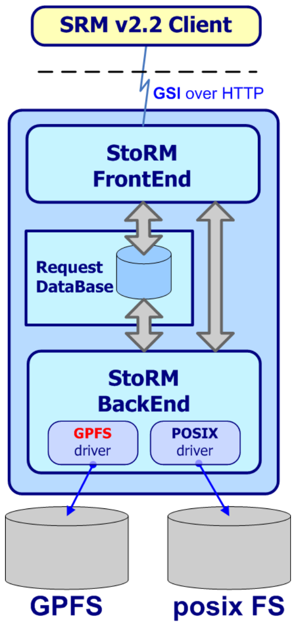

Figure 1: Standalone StoRM deployment.

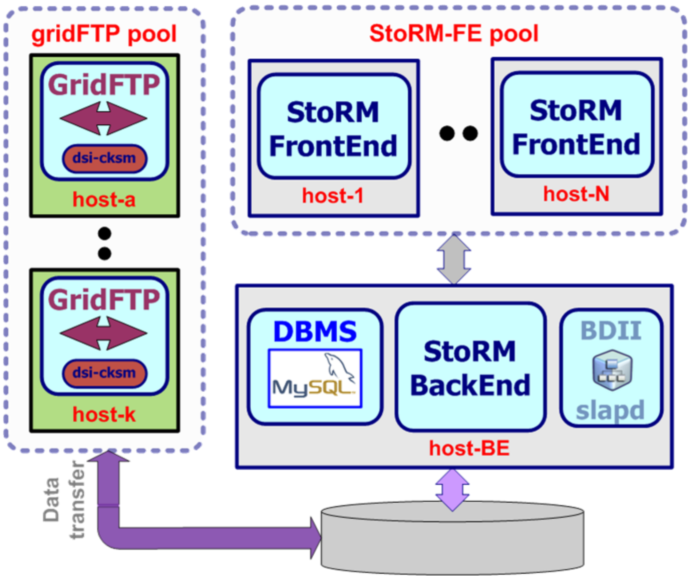

Figure 2: Common StoRM deployment.

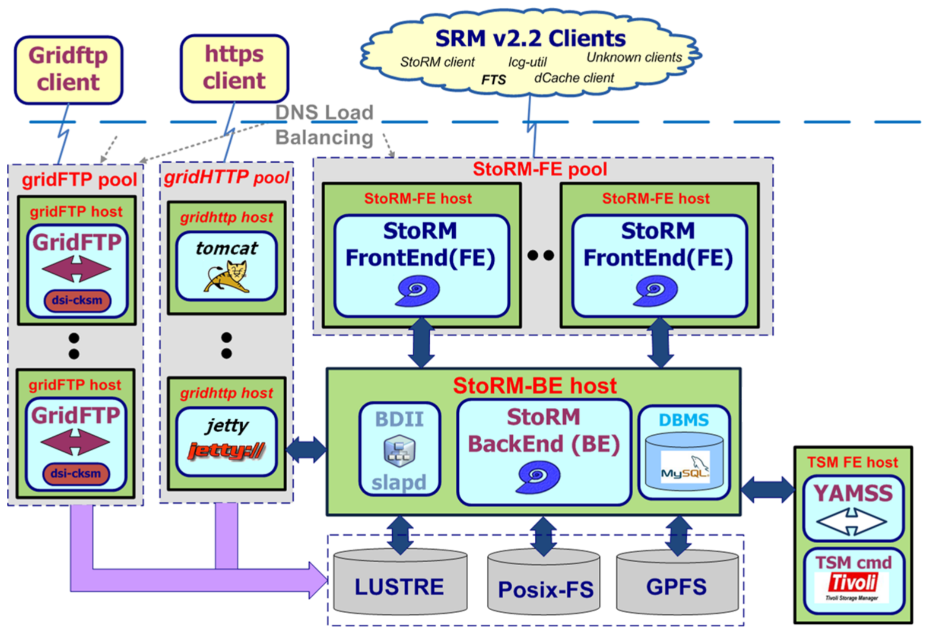

Figure 3: Complex StoRM deployment.

2. T-StoRM

T-StoRM is a StoRM testing framework that orchestrates tests by using a proper deployment and a test engine fed with a pre-built configuration file. During the validation and verification activities of StoRM, T-StoRM supports various levels of types of testing that are found in a software development life cycle:

atomic testing that verifies any SRM operation;

functional testing that verifies any SRM functional specification;

sanity testing, a StoRM specific testing level, that verifies the correctness of the installation and configuration of StoRM;

regression testing, a StoRM specific testing level, that verifies the correctness of each bugfix in StoRM;

integration testing that verifies the behaviour between different SRM implementations;

performance testing that determines how StoRM performs in terms of responsiveness and stability under a particular workload. This is correlated with a set of sensors such as the usage of memory and cpu, the StoRM logging file size, the usage of memory of the StoRM processes to better understand the behaviour of the StoRM services;

conformance testing that determines whether the SRM implementation agrees with the SRM specification and the SRM memorandum of understanding.

For each StoRM release T-StoRM verifies the StoRM implementation of the SRM interface by using all the SRM clients, such as `arc*`, `lcg-*`, `srm*`, `clientSRM` that are provided by all the storage elements providers involved in the EMI project (i.e., DPM, dCache, StoRM). Furthermore, it tests the SRM calls one by one, providing a set of atomic tests that can be arbitrarily arranged to build more complex tests. T-StoRM produces a nicely formatted report file for all the executed tests. This development is ongoing in collaboration with IGI. T-StoRM contains especially SRM tests, however its design can be reused for other types of software just including other tests.

T-StoRM is written in Python and runs implemented tests by using the unittest Python package. T-StoRM is composed of three components, called `tstorm-common`, `tstorm-sanity` and `tstorm`.

`tstorm-common` contains common utilities and the pre-built configuration file of the whole testing framework based on JSON. It is a software dependency of the `tstorm-sanity` and `tstorm` components;

`tstorm-sanity` contains sanity testing level and the configuration file of the SRM endpoint to be tested. It verifies the correctness of the installation and configuration of a given StoRM component in relation with Figures 1, 2 and 3. It must be installed where the StoRM components are installed;

`tstorm` contains atomic, functional, regression, integration, performance and conformance testing levels and the configuration file of the SRM endpoint to be tested. It verifies the StoRM functionalities, the StoRM's compliance with its specified requirements, the system whenever program changes and new bugs are fixed, the correct interaction of StoRM with other SRM implementations, the StoRM stability. It is typically installed on a user interface machine.

Figure 4 shows the execution of the two components against a StoRM instance, while Figure 5 shows the execution of the `tstorm` component against any SRM implementations.

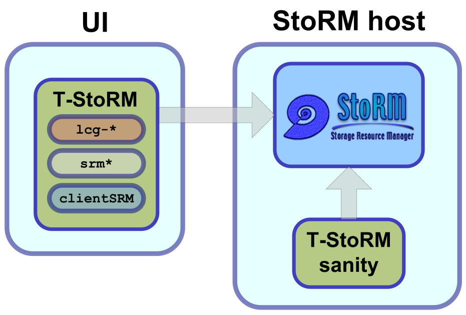

Figure 4: The T-StoRM deployment against StoRM.

Each test is structured as a `(key, value)` pair, where `key` is the name of the test, whilst the value is a list containing the following information: the name of the test; a basic test description; the unique test identifier as specified in the Test Plan; the rfc information as specified in the Test Plan; the StoRM release range in which the test is valid (i.e., the superior extreme of the range

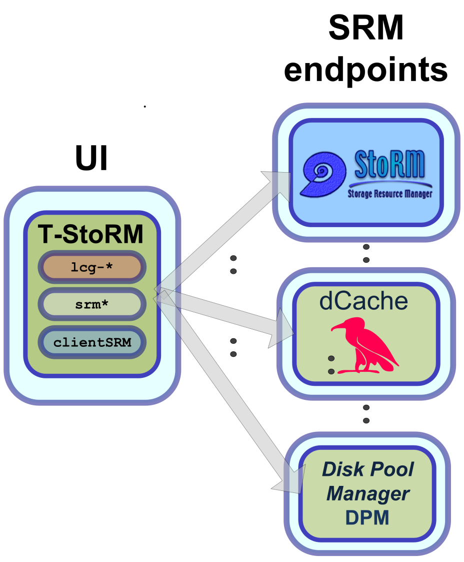

Figure 5: The T-StoRM deployment against any SRM implementations.

can be `]` or `)`, whilst the inferior extreme of the range can be `[` or `(`. The inferior and superior values can be $\star$ or a StoRM release in the form `x.y.z-w`); the testing level amongst atomic, load, system, and deployment; a boolean value to express if the test is a regression one; a boolean value to express if the test is an idempotent one; the test signing to be called to execute the test. The range element of the list allows T-StoRM to exclude tests that are not valid for a given StoRM release.

3. Testing Levels

3.1 Atomic Testing Level

The atomic testing level verifies any SRM operation. This level contains a set of tests that are also used in functional, performance, regression, and integration levels. Table 2 details the SRM clients that are included in T-StoRM for some of the SRM operations.

3.2 Functional Testing Level

The functional testing level verifies any SRM functional specification by reproducing a certain use case. Each use case can be implemented by using different SRM clients, such as dCache client, lcg-utils, and StoRM client. Some examples are: move a file from the local node to the storage element by using the gsiftp protocol or the https protocol; move a file between the local node and the storage element by using the gsiftp protocol.

3.3 Sanity Testing Level

The sanity testing level verifies the correctness of the StoRM installation and configuration. It contains tests that are specific to StoRM. It is recommended to execute these tests before those of the other levels.

SRM operations	SRM clients
Ping	clientSRM, srmping
PtP	clientSRM, lcg-cp
PtG	clientSRM, lcg-cp
Rf	clientSRM
Gsm	clientSRM
GtP	clientSRM
Rm	clientSRM, srmrm
Ls	clientSRM, lcg-ls
RmDir	clientSRM, srmrmdir
MkDir	clientSRM, srmmkdir

Table 2: Some of the SRM operations with the supported SRM clients.

3.4 Regression Testing Level

The regression testing level verifies the correctness of each StoRM bugfix. It contains tests that are specific to StoRM. Each test verifies bugfixes for the current X.Y.Z-W and previous versions. If the StoRM version under verification is lower than X.Y.Z-W, the test should show the issue. Some tests can be only executed in a given version range. The test is executed on top of a test environment similar to the one that showed the bug. These tests cover atomic, functional, and sanity levels.

3.5 Integration Testing Level

A new testing level has been added to satisfy a request coming from the EMI JRA1. The integration testing level verifies the transfer functionality between different storage element providers such as DPM, dCache, StoRM and ARC-SE included in the EMI project, three of which are managed via the SRM interface. They handle the operations of putting and getting a file by using the GridFTP service after having interacted with the SRM service through calls srmPtP-srmPutDone and srmPtG-srmReleaseFile respectively. ARC-SE works without SRM and must be configured in order to allow requests that come from users belonging to the "testers.eu-emi.eu" Virtual Organization to be read and written. Even if it is out of the T-StoRM original scope, it well fits in its architecture.

Figure 6 and Figure 7 show two interesting use cases: in the first one T-StoRM will test the file transfers between 16 combinations of the specified storage elements; in the second one T-StoRM will test the file transfer functionality between the StoRM storage element and the other storage elements that are supported in EMI.

3.6 Performance Testing Level

The performance testing level determines how StoRM performs in terms of responsiveness and stability under a particular workload. The tests are not specific to StoRM. Load and stress tests are currently supported:

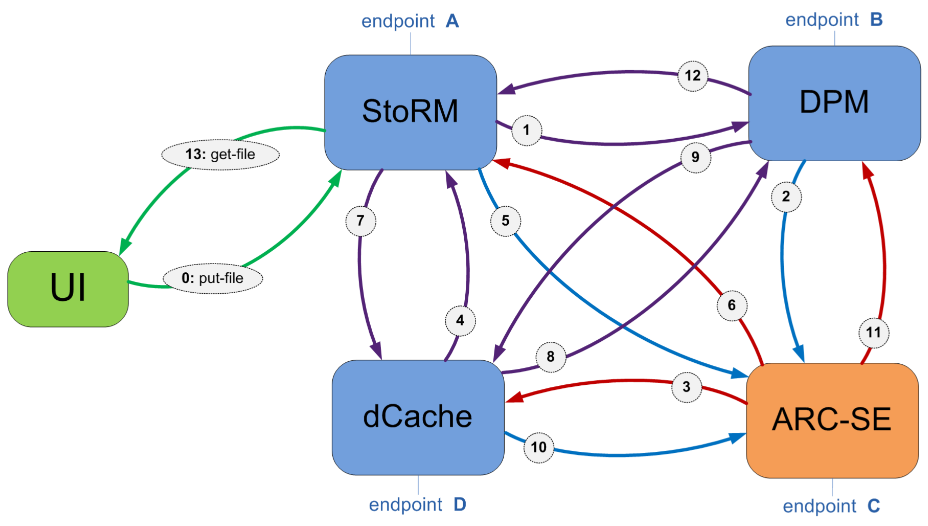

Figure 6: Use Case 1.

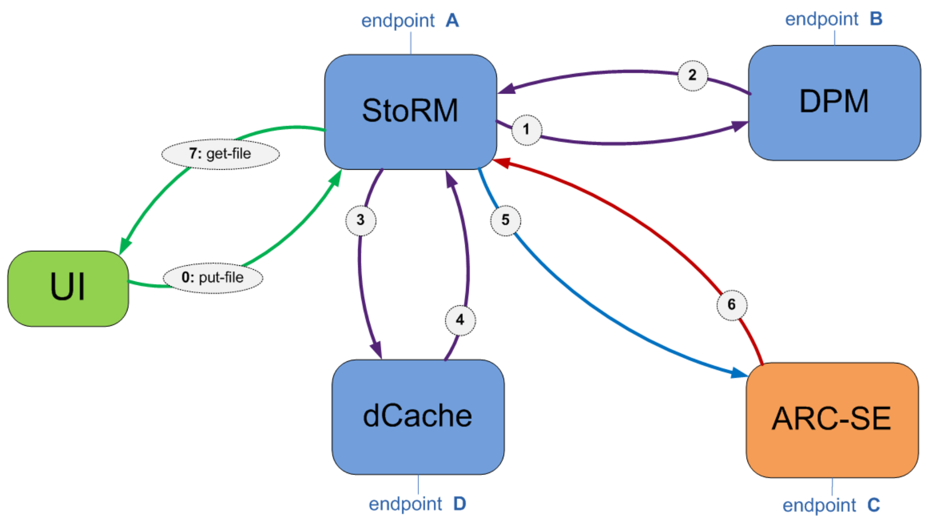

Figure 7: Use Case 2.

Load Tests are executed to understand the behaviour of the system under a specific load. Metrics to be gathered are for example: response time of the service by using an SRM client on a user interface and service resource usage such as CPU and memory.

Stress Tests are executed to understand the upper limits within the StoRM system, and its robustness in terms of load. They are also useful in production deployment to determine if the system satisfies expected bust load.

3.7 Conformance Testing Level

The conformance testing level determines if the SRM implementation agrees with the SRM specification (`https://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.html`) the SRM Memorandum of Understanding. Up to now, the more significant tests have been developed, such as srmPing, srmPtP, and srmPtG. One of the examples consists of verifying the srmPing behaviour: having authorizationID as input and returning the versionInfo of the SRM specification that is implemented and extraInfo of the SRM implementation such as the StoRM version as output.

4. Managing Tests

The production of a new T-StoRM test follows the five steps described below as schematized in Figure 8:

1. the test is identified with the pair (id, name), where id is a unique identifier of 6 characters;
2. it is defined and described in the Test Plan document where test id is included
3. it is developed in the correct test level
4. it is included in the test suite
5. finally, it is released in T-StoRM

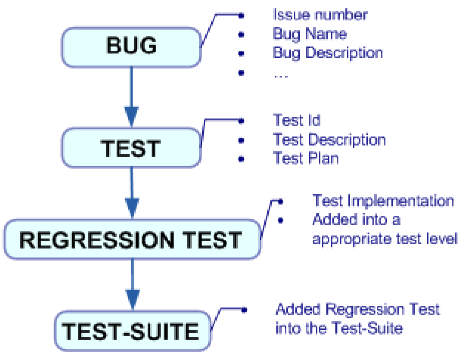

Figure 8: Test Production Procedure.

5. T-StoRM Usage

Currently T-StoRM has used by the StoRM team and the IGI-RTC (Release, Testing, Certification) group in order to certify the StoRM release, to verify the correctness of a StoRM installation and configuration in the StoRM testbed, to check new functionalities and fixed bugs, and to verify the StoRM stability. In the near future T-StoRM will be used by site-administrators, SRM certifiers in order to validate a StoRM installation and configuration, to certify SRM software products, to certify other software products containing tests suitable for other software, and to monitor SRM instances

Up to now all the detailed testing levels have been implemented with the exclusion of the stress tests. The results of sensors activated on the StoRM nodes during the execution of load tests are hand manipulated by users.

6. T-StoRM Future Plan

The scheduled activities on T-StoRM are related to the performance and conformance testing levels: the former will be correlated with a web-based monitoring to simplify the understanding of the whole system by publishing measures of the supported metrics, and generating suitable charts; the latter will be extended to other SRM operations. Furthermore, the commands python module, becoming obsolete and not being very safe, will be substituted with the subprocess python module. A T-StoRM integration with a Continuos Integration Framework will be also considered to automate tests execution, to immediate tests results reporting, taking advantage from virtual technologies, to automate deployment and configuration of StoRM.

7. Conclusions

T-StoRM is making faster and simpler the StoRM certification process. It has shown to be able to cover several test levels even not considered at the design time. It is designed to be easily extendible in order to be used for the certification of other software.

8. Acknowledgements

This work was performed as part of the EMI project and partially funded by the European Commission under Grant Agreement 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`

[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`