SVDI Scalable Virtual Data Infrastructure

Summarized & translated by: Long Phan

1. Introduction and Organization

Today there are more and more data and information emerging, coming from many science fields (physic, climate, chemical, industry etc.) so that concept „Big Data“ is becoming popular. To analyze and process this huge data many tools and complex systems are (being) created to support scientists. This might causes other problems that scientist need to learn, know and overcome technical issues when using different tools for their research, especially in distributed environments where data is not often staying at local system, but propagated in many data centers/ systems in different location.

Target of our group is creating a tool data management which support scientists to work with different storage environments begin from local system to Grid/ Cloud environments. This tool should support different basic operations POSIX (read, write, modify, rename) or sharing data with other scientists, support search information, flag (meta) data etc. In a view of technical issues, this tool will have 3 fundamental legacy functions in data management regarding Management in **1. Replica Catalog, 2. Metadata Database, 3. Authentication and 4. Data transfer**. This tool will have to tackle many challenges and requirements in data transfer protocols in Grid and Cloud environments.

Our group is organized by 9 students, tutor by 2 teacher. We will continue to work in small groups from 2 to 3 students to warrant our team work. Development tasks are flexible, at home or in lab. But the most important thing to keep the tasks forward is communication. Result and exchanging information, ideas and approaches/ solutions will be reported every week at round-table-session. Email and Wiki (Redmine) are useful tools to keep updating task, observing efforts of individuals and whole group. This project has taken totally two semester at university.

2. Problem and requirements (Use cases)

The following scenarios will determine the requirements and functionalities of group's products.
Scenario 1:
Scientists want to access from his location (home) to Grid resource locating remotely through his personal Grid certificate and exchange data between local-system and remote-system.
Requirements are simple data transfer (upload, download) and authenticated by Grid standard certificate.

Scenario 2:
Scientists want to share their research results with colleagues in team. So that only data owner is allowed to change content of data, and also deliver write-access to other people on the same data.
Requirements are management of right access in group, simultaneous access mechanism on data, determine the most update version of data.

Scenario 3:
Scientists have data saved on distributed system/ resources so that it's feasible to find data quickly.
Requirements are simultaneous access on many resources and able to have search-function for data.

Scenario 4:
Scientists want to execute some complex operations on their data record, determine which data package has been used, able to sort and recognize these data packages in hierarchical structure.
Scientists are able to transfer data third parties, from resource to resource. Requirements are using metadata and applied it into search/sort-funtion, enable to transfer data between systems remotely.

Scenario 5:
Scientists often use different devices to access their data on remote resource. Requirements are integrity of (meta) data in case data is being updated and using program on (mobile) devices.

3. Approaches and Architecture

The main functionalities of program base on data management where data set are saved on file system, and information about data are located in database management system. Therefore, data often need to be queried by user, especially simultaneous access to database, lead to requirements about execution of transaction and logic in database. The consistent of data record in database need to be warranted while running time of current transaction, no other access on the same data are allowed.

3.1 Object relational database model

→ PostgréSQL: in order to save information of structured-data, PostgréSQL is first priority to take this position. PostgréSQL offers the functionalities of a object relational database system, which works with ability of relational database but also with additional functionalities like write/ read abilities from object information and their contents will be saved in corresponding database attributes. PostgréSQL is able to be integrated with Java easily by using adaptor JDBC. In current version (9.0 and higher) some functionalities like replication, distributed storage and synchronization of the same data are also supported. Another benefit when using PostgréSQL is the availability, so that it will increase the performance of all system. PostgréSQL is used for storage of data record, documents in project [pos].

3.2 Document oriented database model

→ MongoDB: is one typical database, belong to group of NoSQL-Databases, where database entries are called documents and includes pairs of Key-Values. The values can be not only numbers, strings but also all document. Without fixed schema, it's dynamical to add more Meta data to single document and can apply this properties by querying database. MongoDB is used for Management of Metadata in project [mon].

3.3 Some analyzes about distributed file system

One of ability or even goal of our program is regarding with properties of distributed file system, especially detachment management of metadata and data saving or high abstract layer of application view so that it's desirable to have a look on important concepts of distributed file system. The distributed file system plays important role in distributed system, is deployed in system like Grids. Distributed File System is deployed where big memory capacity and massive parallel access at high scalability, also stable data performance/ throughput are needed/ required.

XtreemFS [xfs]: is object based distributed file system for grid, it can replicate object to increase the fault tolerance. System supports cache for data and meta data to decrease latency of access. Additionally, SSL can be used to encrypt connection in network. In current version, distributed file system enables parallel ability read and write of data over windows and linux client. Data and Meta data are detached, and distributed on many nodes to increase data throughput. It's also possible when exploiting method detachment of metadata to search data on a central metadata server. It's simple and efficient like database's query. Another benefit of XtreemFS is compatibilities with POSIX.

Client/ Access Layer: enables a interface between user process and file system, allow client to access on data of distributed system. Access Layer supports to interpret the task which's sent by Client into understandable commands for “Object Storage Device” (OSD) and “Metadata and Replica Catalog” (MRC). Access Layer is understood as Negotiation Layer between Client and Distributed File System.

OSD:
Object Storage Device is responsible for saving content of files. Files in file system are represented by one or many objects. In order to increase performance of parallel read-write rate, these objects can be distributed on different server nodes. Moreover, OSD need to ensure the consistenz of replica, that's why all objects are always watched and ruled to keep in updated state.

MRC:
MRC is used for purpose of management metadata of files and directories in file system. Again to increase performance of parallel queries, more Metadata and Replica Catalogs are given. This service save metadata into database and works on prinzip Master/ Slaves / filesystem. One change on Master will transfer to all Slaves automatically. Therefore, all nodes stay in consistent state.

RMS “Replica Management Service”
RMS decides when one replica will be created and removed. This service watchs all transactions of user and create replica base on number of accesses of user on files. RMS will correct state of data in case that node is corrupted or inconsistent is happening. This service decides number of replica with “striping-policy”.

XtreemFS is extended by different interfaces (ex. CORBA), Modules are written in C++ and Java.
→ Another file system, beside XtreemFS is Lustre
→ Amazon S3 (S3QL) saves data in buckets. These Buckets are applied like normal directories on local file system. One Bucket saves unlimited many data objects, contains one binary data object, name and its metadata.

3.4 Metadata Management
Metadata or MetaInformation are informations which filesystem needs to manage files, it describes the properties of files. Example, system metadata are informations about size, date of last change, file name, access right, path etc.
Management of Metadata includes the tasks regarding with typical operations like create, update, remove, search of saving data.
XML is powerful-tool to structure data, however there exists some limitation regarding description of metadata. To support XML to expresse the description, properties of resource by means of metadata information, more following tools are introduced:
(1) URIs Uniform Ressource Identifier: title of ressourcen, describt one concrete object in context of modelling.
(2) RDF Resource Description Framework describes metadata fomally
→ RDF Schema: contains many parts, begin with rdfs:Resource

rdfs:Resource
rdfs:Class
...

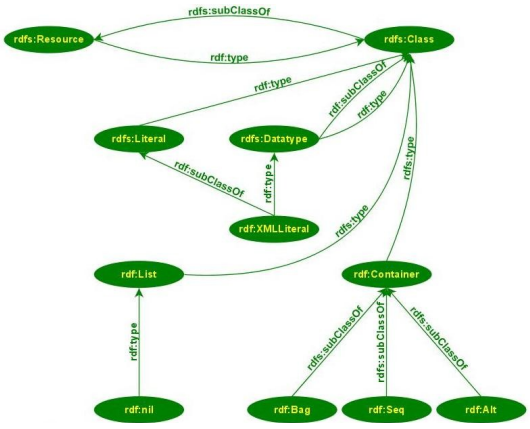

Figure: RDF-Schema

(3) OWL (Web Ontology Language)
OWL is extension of RDF and RDFS and is formulated in XML.
→ OWL Full → OWL DL → OWL Lite.

(4) SPARQL
SPARQL is officially Standard of W3C for query language in order to improve query an RDF. SPARQL supports description of query statement to get the concrete resource with certain properties.

3.5 Architecture: Approaches and Development Design
In order to design one architecture for our software, many models are being discussed and considered. Among of them there are 3 models which are considered to be able to describe the suitable specific properties of in our requirements. They are approaches of P2P, Fat-Client Server, Thin-Client Server.

Approaches:
(1) P2P approach
The idea is building one architecture that is viewing client will have the same function as server. So, in context of P2P, concept server and client are almost the same. One raising idea with P2P is applying one Super node to decrease network loading regarding query from Clients and this Super node should keep communicating with other Super node to keep the update state in case changing of new information. Anyway, to recognize and avoid Deadlock situations is not easy to solve. Another difficulty is database, what we need to distribute right access. To realize this kind of distributed database inside P2P requires more intensive knowledges and efforts what we could not accomplish in a short time.

(2) Fat-Client Server approach
In this approach, Client will have all functionalities for direct data transfer on any desired resources and Server only manages global information about all metadata and user exclusive login information of user. In case that user needs access to information, client will need to get contact with Server to be authenticated. Server will have a module for saving all information management about metadata, resources, users and right access in its database. Client will keep a module for data transfer so that Server plays a role of manager and Client plays a role of executer.
This approach has advantages that all Clients could have accesses to many resources wherever directly. Client only need to be get acknowledges authenticated by Server. Other benefit is that Client can exchange files among each other without interaction with Server that could lead to bottlenecks on Server when many Clients are working at the same time.

(3) Thin-Client Server approach
In this approach, Client takes on less role and Server adopts more roles in management of Metadata and queries from Client, control information and access especially with operation download and update of data.
Client needs to be authenticated by Server firstly. Subsequently Client want to download data, it needs to send its request about one Metadata on Server what Client has legal right access. After getting this Metadata, Client is able to have one overview of requested data and resources/ locations of data also data (updated) status. Then Client can initiate download process data from resource.
In case Client has updated new information on data, role Update data will transfer into Server. Server will overwrite the old data on resource with new data from Client and update Metadata in database. To avoid inconsistent when download and update files, Write-access on data will be locked. This lock will prevent double accesses at the same time.
The approach for program could be (Thin)-Client Server because the variety of application server.

Application Server can offers many functionalities like abstract database access, application on web services and transaction manager. However, that indeed during development of program, one opening point considered as less efficiency functionality is that there is only one session for all Clients/ Users. All User/ Clients will use the same connection to Server. This fact is not acceptable so that development group has decided to turn back with approach (Fat)-Client Server.

Development Design

The following picture/ diagram shows overview of whole architecture of SVDI (Client-Server)

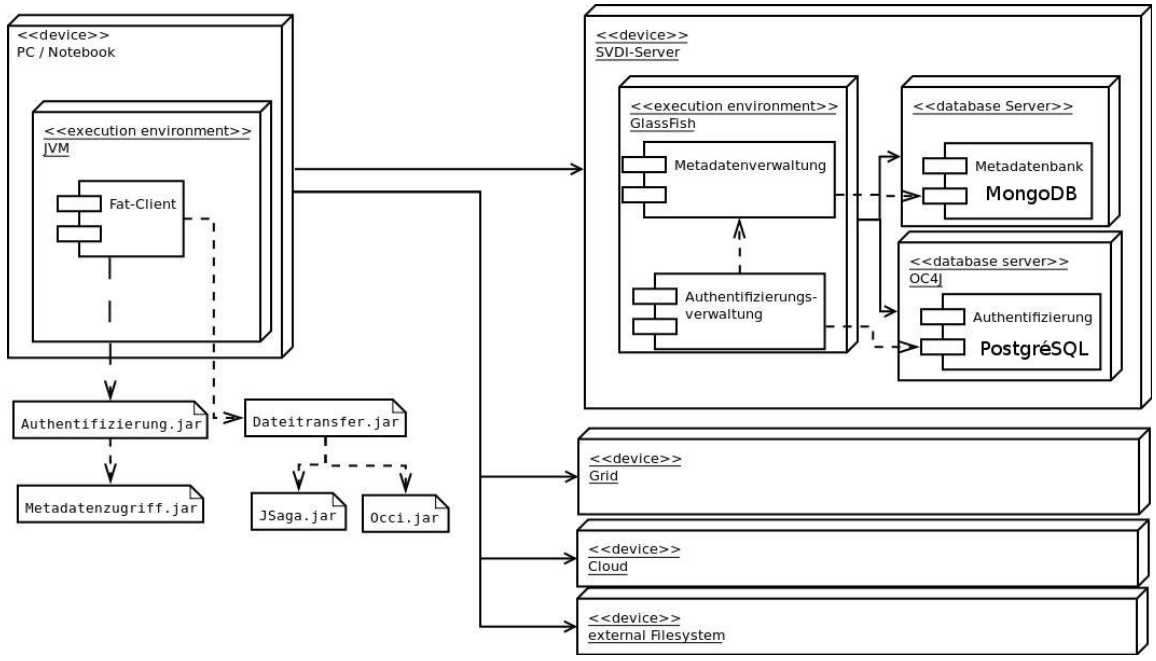

Figure: Fat-Client approach

(1) Database

PostgréSQL: offers Administrator possibilities to correct errors by means of SQL commands manually. Subsequently it's feasible to replicate saving information. PostgréSQL in our SVDI will save relevant information of user (Email, Session key, Login time, user name) Moreover, it also saves information about the state of available resources.

MongoDB: NoSQL Database will save information Metadata of files. MongoDB enables rapid Update of Data and Metadata and solve the conflicts through transaction, create easy MySQL-alike queries, support Administrator to search and correct errors. Besides, MongoDB offer Map-Reduce functionalities to evaluate big data set.

GlassFish ApplicationServer: is implementation of Oracle, light-weight Application Server for JavaEE-Standard. With function “Persistence Framework TopLink”, enables good connection to applied database products.

(2) Server functionalities

- Approach (Thin)-Client Server got difficulty when library for management connections of Client can not be applied in distributed environments so that Modules ex. Login Management of user and service for data transfer are unable to keep going on.

- Replacement is approach (Fat)-Client Server. Fat-Client becomes more important when Client

keeps connection to Resource every time when user connects to his resource where he's allowed to access. For every resource, there is one field triple <user id, resource id, user info> symbolizes that user/ Client uses this resource for his work. After user removes this data record, it means that user has been disconnected from this resource at his local system, but this resource still exists for other user and meta data certainly.

Functionalities management Login-information and data transfer will be moved to Client's functionalities. However Server will support Client to resolve conflict inconsistence of data when 2 users are trying to access to the same data record on same resource. It must be sure that one of user is faster than other to get the data record so that one Metadata will be marked in database (mongoDB) of Server to notify the other (later) user that data right now are (being) changed and write-access or Update ability is not possible. To be able to update this data, (lately) user need to wait the next notification of (earlier) user with confirmation that Metadata is now unblocked. He should also get the newest version of data from Server instead of working on downloaded version at local-system, anyway this step is desired, not a must. Only data owner can block/ unblock this metadata to let other user modify his data.

Server also offers Client the insertation of user-defined Metadata in form Key = Value. It also gives policy that who are owner of Data will be able to delete or modify this Metadata, this function is also being used to share data/ information about state of data with other user but still keeps right of data owner. Using this (user-defined) Metadata, Data Owner can create a group of users who are allowed to have read/ write access on his data.

(3) Client functionalities
Like above information, Client will take care of data transfer directly to resource without problem of connection management with library. This way also releases high-load of server to avoid problem bottleneck. After user has changed his data, Metadata on Server will be updated when user uploads his data back to Resource but need to go across the Server. Connection between Client and Server will be transferred using SOAP-format and the connection will be encrypted with HTTPS to ensure that no one can read out (or change) information Metadata of data.

(4) Metadata
There are two groups of Metadata
The first one is System-Metadata. This information are standard for every saved files, to let file system identify the presence of file in its storage system. They includes ex. File name, size, location resource, path etc.

The second one is User-defined Metadata, is being created using Key-Value. User can create as many as possible Key-Values pair in database MongoDB. It's also possible to let MongoDB apply both of Metadata saving into one object. Other useful functionality is using Metadata to search data in Database (mongoDB).

The following table shows a survey of (System)-Metadata saved in database (mongoDB)

Id	*: internal ID of Metadata,*
Name	*: path of file on resource,*
ResourceID	*: Resource ID where files are being saved,*
Size	*: size of file,*
DateCreated	*: time/ date when file is created,*
Categories	*: categories where file are being appertained,*
Locked	*: state of file (block/ unblock changing file),*
customMetadata	*: Key-Value pairs user-defined Metadata,*
lastchanged	*: last changing date (pro user) to file.*

The Metadata will be saved not only on Server but also on Client and the synchronization process will be performed from Client to Server since user has changed data. Metadata will be packed in format Key-Value in Object and transferred to Server. Searching will go similarly. User will put search-information into search-field, this information will be sent to Server per invoke-Webservice that will perform search in Metadata-database. In case that connection between Server and Client is broken, one Error-message will be output. Otherwise, Metadata on Server will be updated as well or one list of search-result will be returned. This list can also be empty in case no information were found.

4. Implementation/ Tools

4.1 Implementation
The implementation in general are organized in two parts: Client and Server.

Client Implementation:
The architecture pattern MVC is applied mainly to demonstrate Model as processing main Unit of Client, View as Graphic User Interface, and Control is middle-component between View and Model.
Moreover, two extra modules are created: Proxy-Module and Start-Module.
Start-Module undertakes role of start-configuration, contains functions for user, resource at local system. Proxy-Module undertakes role of interface for Client to connect to Server and transfer data to remote-resources.

<1> Model:
Class UsernamePasswordToken : manages basic information when connection like user name, password, Email, Session-Key .
Class UserCache: saves information about Resource, projects, certificates in dynamic data struture.
(abstract) Class Resource: as head-class for child-classes regarding to specific connected remote-resources (SFTP-Resource, FTP-Resource, Grid-Resource). With method connect(), every remote-Resource will be connected and enabled data transfer, especially when every remote-Resource has different data transfer protocol.
Class Project: manages projects of user.
Class FileHeader: are links on data, contains name, metadata, resource where saves this file, local path on this resource.
Class Certificate: contains information about created (proxy)-certificate and path to original certificate.

<2> Controller:
Controller is organized into 3 classes FileController, MetadataController and Controller.
FileController: contains all basic operations regarding to file like Download, Upload from remote-Resouce, operations (Remove, rename, Save etc.) at local and remote-system.
MetadataController: contains all basic operations for management Metadata (Add, Remove, Change, Synchronization with Server)
Controller: contains all of rest functions to control Client regarding Login-information, create (proxy)-certificate, user-information, project-information and encryption of Password with MD5-Hash algorithms.
Extra: InitController: to initiate instance of all above Controller-classes and create a common interface for View-component.

<3> View:
View describes standard-GUI of Client, contains classes and methods to demonstrate Client UI (user interaction).
Tasks on Client:
(1) Management of Project and files in project at local system: user cannot create more projects/ files with the same name. Directory “user-home/SVDI_HOME/SVDI_Downloads” is created to save the created projects and downloaded files from remote-connected-resource.
(2) Management of remote-connected-resource: list of files on remote-connected-resource will be displayed, also metadata of file will be downloaded. Other manipulation operations with file is also feasible. The main display for this function is Resource-Browser.
(3) Annotation of file with user-defined Metadata

Experiences and problems in Client implementation:
Problems come from experiences of group in development of distributed application. The requirements to architecture during implementation phase are sometimes not so clear, need more discussion and communication between group's members to resolve. The new problem without solution lead to changing of architecture, causes more times in development.
Other technical problems:

(1) Problem with Serialization of UserCache

(2) Problem with display of different kind of data types

(3) Problem of Model converting between classes (ex. Model classes and DTO classes)

Server Implementation:

The management of data transfer stays in Server, also control the access to data.
The module on Server are organized:

(1) User management: user information, functionalities for registry and login, session
(2) Metadata management
(3) Data transfer service
(4) Login Information management: <user, resource-server, login-data>
(5) Resources management: resource information ex. Location. User can remove information resource at Client meaning disconnect from Resource, however resource still keep existing for other user and visible remotely.

<1>-Tasks on Server:
Server keeps role of information control between Client and Databases. Server manages information of resources-information, access to Metadata, informations which User needs for their accesses. Server also enables User to execute Client on different end-devices.

<2>-Development of Server:
Modules on Server were developed by 4 people Team. Every member-team get area task with responsibility, define specification of work-package. Communication in Team are demonstrated through working together on UML-diagrams, discussion, review use-cases and implementation weekly. After specification of Server, it's needful to have one Web Service Interface to get message from Client and also inform Client how is the result (success or fail). It's necessary to give Client's team one description in detail about this Web Service Interface and Use-cases.

<3>-Server's Architecture:
Architecture is organized in 3 layers. The applied patterns are mainly applied on MVC, and other patterns like “Adapter” and “Singleton” in order to get high maintenance and enable the extension.

→ (1) Web Service Interface Layer : defines the methods of Web Service as interfaces which Client need to communicate, to get data input. This Layer also interacts with under-Layer Business Logic, it also controls the exceptions, the possible errors (message) and process Client's invoking. WSDL Document is used for Software interface with Client.

→ (2) Business Logic Layer: contains all server logic which execute method of above Interface Layer. This Layer is responsible to communicate with Interface Layer and under-Data Access Layer using DTO “Data Transfer Objects”,

→ (3) Data Access Layer: This layer defines some corresponding Annotations for Database accesses. Access to Database is realized by Java Persistance API (JPA). The entities from Database will be read out and transferred to above Business Logic Layer per DTO. This layer doesn't create database exeception but only forward error to above Business Logic Layer.

<4>-Processing Exceptions:
The processing of Exception and errors happens mainly inside Business Logic Layer, especially errors from Database are converted to user-defined errors to hide causes error of database so that these errors will not be exploited to attack Server and also clarify message for user. The Exception's type for notification of errors are:

- FaultBean: means in case of general errors
- InvalidInput: means in case of false input from Client (ex. Session-ID)
- MetadataTooOld: means when other Client has changed data in database so that metadata need to be updated.
- NoConnectionToResource: means in case of File Transfer Service
- NotExceptedException: not only for Jsage-Exception NotImplementedException but also for other APIs-Exception which have not been processed by SVDI.
- NotSufficientRights: means in case User does not have correct right to execute one action on Server.

<5>-Server-Development:
Team groups are divided into 2 groups Client and Server. To be able to test module on Server, the development on Client goes parallel with Server's team so that the unclear/ new problem could be corrected. Client's team also needed time to join in Server's team to get an overview on functionalities of Web Service Interfaces which Client needs to transfer Object (DTO) to Server. DTO: Data Transfer Objects. This object contains simple data of Database's objects, it has one field for Database-ID, and Strange-Key related with other Objects. This DTO is used for communication over Server's boundary. DTO which has been transferred to Server, has fields needed for calling methods. However not all of fields need to be fill in. Example: Registry field can have empty Session-ID, User-ID and Login-Time because Client will get this parameters after successful Login.

<6>-Tools

- Java JDK
- IDE Eclipse
- Maven: Build Management Tool (validate, compile, test, package, verify, install, deploy)
- (J)SAGA-API for Java-Version
- MyProxy (jglobus-core.jar, myproxy-logon.jar) [myp]

<7>-Test:
Requirements:

- Reliability
- Correctness
- Robustness and errors's behavior
- Exception for error's processing

Tools:

→ Junit (integrated with Maven)
→ SoapUI (Test Framework for Webservice on Server)

5. Software's functionalities compared to Scenarios

- Registry new User
- Login with User Name and Password
- Connect to Resources (using User's data, certificate) (SFTP, FTP, Grid)
- Data transfer (Local – Remote)
- Data Transfer (Remote – Remote)
- Create and Process of Metadata
- Search Files using Metadata
- Copy the found data to other Resource
- Replication of Data (manual partly → To Do: automatical fully)
- Building Right Access Management for User depend on different Resources (ToDo)

6. Conclusion (assessment, challenges and approaches in the future)

Project SVDI Scalable Virtual Data Infrastructure has been finished as well. Team group has achieved great goal of knowledges and team work spirit. Every member of team has collected for himself the necessary experiences in Software Development Life Cycle from scratch to product. Most functionalities have been developed corresponding with requirements and scenarios. However, some functions has not been developed fully when it should have needed more time to learn and understand different technologies, sort out and choose the correct technologies for development process. Using technique PBL (Problem-based learning) team group were able to work together and reduced many time in preprocessing learning and understand problem. Concept Scalability and Virtual Data Infrastructure are very abstract concepts and were understood in many different ways. In context of this work and insight, SVDI has been demonstrated through model Client-Server-Databases as well. Team group has reached the first step on the way to understand and continue to develop the other Softwares in the future.

References

[svdi] https://eldorado.tu-dortmund.de/bitstream/2003/29852/1/PG%20553%20Endbericht.pdf

[pos] PostgreSQL Overview. Webpage. http://www.postgresql.org/about/. # last called 30.08.2012

[mon] MongoDB Overview. Webpage. http://www.mongodb.org/. # last called 30.08.2012

[xfs] Hupfeld, Felix ; Cortes, Toni ; Kolbeck, Björn ; Stender, Jan ; Focht, Erich ; Hess, Matthias ; Malo, Jesus ; Marti, Jonathan ; Cesario, Eugenio: The XtreemFS architecture - a case for object-based file systems in Grids. In: Concurr. Comput. : Pract. Exper. 2060. 20 (2008), Dezember, Nr. 17, 2049-2006 http://dx.doi.org/10.1002/cpe.v20:17 - DOI 10.1002/cpe.v20:17. # ISSN 1532#0626

[jun] jUnit Overview. http://junit.sourceforge.net/doc/cookstour/cookstour.htm
 # last called 30.08.2012

[myp] MyProxy Overview. http://grid.ncsa.illinois.edu/myproxy/
 # last called 30.08.2012