Proactive Data Management System

Abhishek Gaurav, Nayden Markatchev, Philip Rizk and Rob Simmonds
Grid Research Centre,
University of Calgary, Canada.

Contents

1 PDMS Overview

Proactive Data Management System (PDMS) is designed to manage large datasets within grid environments. PDMS is particularly useful in scientific environments where large amounts of data are often moved between computing and data archiving sites. PDMS facilitates management and movement of data using metadata, i.e., the data items are identified using their inherent properties and characteristics rather than the file names in which they are stored. The use of metadata abstracts away the physical location of a file allowing PDMS to transparently manage replicas of a file. It is intended to be used by groups needing to manage large data sets across several locations. PDMS utilizes well known Data Grid services. This allows it to interoperate with various workflow managers in use today.

Management of data using metadata allows the replication requests to PDMS to be specified in terms of metadata. For example, a replication request to PDMS can be "move the data generated by user A for project B within last three months to the site X". The metadata in the above example are - (i) generated by user A, (ii) belonging to project B and (iii) generated in last 3 months. The metadata in the above example correspond to some logical files in which the data is stored. The logical files can be physically present at multiple locations, in which case, PDMS locates all pieces of the dataset and initiates a transfer of all the pieces. Thus, with a given replication request, PDMS needs to perform two key tasks before initiating transfers - (i) use metadata to establish the logical names of the files that match the metadata query and (ii) select sources of replicas for those logical files not already at the destination. This management of replicas on the basis of metadata fills gap in the previously available Data Management services available.

PDMS is designed to restrict access to authorized and authenticated users who have permission to use the system. Files are stored in logical groupings referred to as *collections*. It also restricts users access to specific collections. These access restrictions resemble file ownership with ownership of collections as well as read and write privileges. A more complete description of the access control can be found in [3].

PDMS maintains the consistency of the data for a collection. This currently includes not allowing the same physical file to be registered twice (as two separate logical files). PDMS also ensures that users conform to the schema they include in their registration request. PDMS could be configured to enforce each collection to conform to a specific schema. This is particularly useful in large groups that need to be sure all metadata contains certain information and want to prevent buggy registration processes from introduction inconsistency or incomplete metadata. Consistency requirements for the PDMS system are intended to be configurable as consistency checking can be expensive.

2 PDMS Design

2.1 System Architecture

The system level architecture, depicted in Figure 1, shows the grid services used by PDMS. They are: the Globus Container, Replica Location Service (RLS), Reliable File Transfer (RFT) and Metadata Catalog Service (MCS). The role of each of the above services within PDMS are described next.

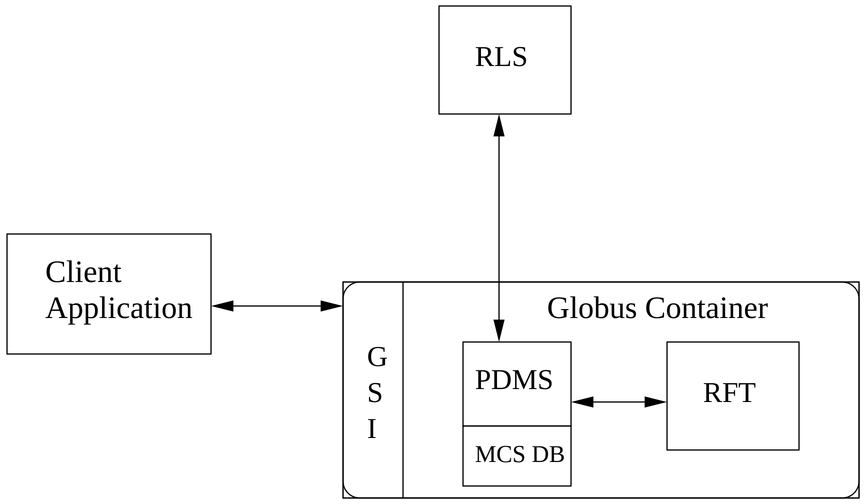


Figure 1: Third-party software dependencies for PDMS.

2.1.1 Metadata Catalog Service (MCS)

MCS is a system that allows management of metadata about logical files. It provides a means to define, store and access metadata of data. The users of MCS can query for data items based on their attributes. PDMS uses the MCS schema to manage the metadata of the datasets that it handles. PDMS acts directly on the database rather than going through the MCS service for performance reasons. The use of the schema allows the MCS service to be presented if any applications rely on it.

2.1.2 Replica Location Service (RLS)

RLS is a service distributed with the Globus Toolkit that is used to establish the actual storage locations of copies of a logical file. A logical file may be stored at multiple physical locations for faster access. RLS keeps track of all the physical locations at which a logical file is stored and returns them when queried for a particular file. PDMS uses RLS to retrieve the physical locations of the set of logical files that were obtained from MCS.

2.1.3 Reliable File Transfer (RFT)

RFT is a file transfer service within the Globus Toolkit that is used for reliably transferring files in a Grid environment. RFT maintains transfer states in persistent storage and can therefore, successfully recover from failures. PDMS uses RFT to transfer data from physical locations returned by RLS to the destination site. The current version of PDMS uses RFT, although different data transfer services could be developed as plugins. Various data transfer services could be developed with different features and semantics. One example of such a service could schedule User Controlled Light Path (UCLP) circuits. PDMS could also use multiple sources to copy files to a single destination. The transfer service would then have the option of using any of the sources or even all sources at the same time. This is the strategy used by BitTorrent [2]. It is not clear that BitTorrent can meet the security requirements, or that its strategies are suitable for the network and file characteristics in Data Grids.

2.1.4 The Globus Toolkit

Apart from the RLS and RFT services, which are part of the Globus Toolkit 4.0 (GT4), PDMS also uses the Globus Security Infrastructure (GSI) and WS-Core services. The GSI libraries included with the GT4 toolkit provide the security mechanisms used by PDMS. The security mechanisms enable client applications to authenticate with PDMS, check if a client application is authorized to perform an operations serviced by PDMS, and delegate a client's credentials to services like RFT. The Java WS-Core component of GT4 implements the Web Services Resource Framework (WSRF) standards. Following WSRF conventions makes it easier for other developers to ingrate PDMS into their distributed applications.

Figure 2 further shows the roles of MCS, RLS and RFT within PDMS. The figures depicts various stages in handling a replication request. Upon arrival of a replication request, PDMS queries MCS (1) to retrieve the logical names of the files (2). The retrieved list logical names are then submitted to RLS (3), which finds the physical locations of the logical files (4). As a logical file can be present at multiple physical locations, RLS returns a list of all locations for this file. For a logical file that is present at multiple locations, PDMS chooses a single source location to transfer from. PDMS then submits the source locations of the files and the destination locations to RFT (5) which performs the actual transfers.

2.2 PDMS Internals

2.2.1 Resource Management Component

Instances of RLS and MCS clients in PDMS are managed using a pool of clients that are created during the system initialization. The minimum and the maximum number of client instances is hard coded, but could be made configurable.

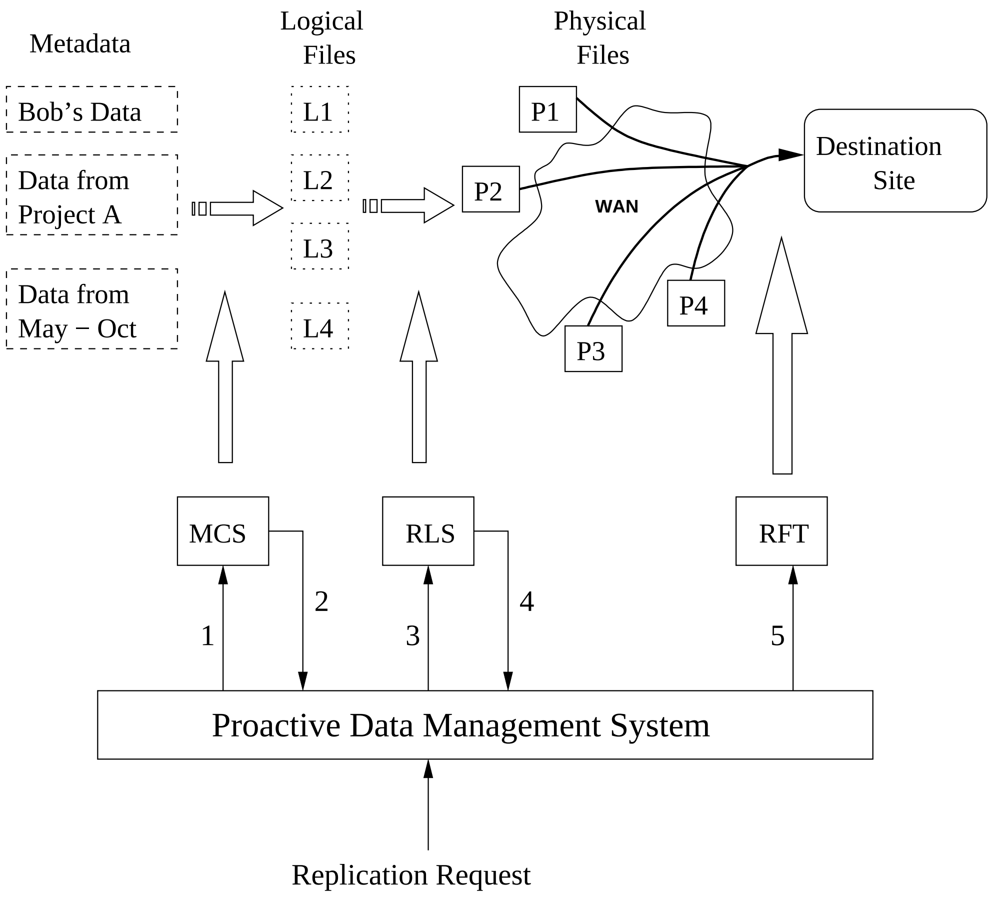


Figure 2: Roles of MCS, RLS and RFT in data replication.

2.2.2 Persistence Component

One of the fault tolerance mechanisms in the PDMS system is the ability to save the states of a replication job as well as the state of a replicating LFN. This is necessary in the case when an error occurs inside PDMS or inside any of the systems PDMS interacts with. If a replication job is in progress when transient errors occur, and if PDMS does not keep track of the replication process, the system may end up in inconsistent state. For example, if a hardware failure occurs during the replication of N number of files from site A to site B, and if PDMS does not keep track of the replication process, two problems arise. The first problem is that upon resumption of proper operation, the system has no knowledge that a failed replication request ever existed, therefore it would not be able to restart it automatically. The second problem is that a number of files have been transferred to site B before the failure occurs, but the system has no knowledge of that fact. This means that if the same replication request is manually restarted once the system is operational, the files which have been transferred to site B will be transferred again although they are already present there. In the case when the replication request is not manually restarted the system will not account for these files, which would result in a waste of disk space. Consequently, saving the state of a replication is a useful fault tolerance mechanism.

The fault tolerance of PDMS is limited to maintaining information about replication jobs in the face of a server shutdown. PDMS will continue to monitor the status of a replication job that has entered the REPLICATING state. This is the only state a replication job spends any considerable time in; failure during other operations can be recovered by simply restarting that replication job. If the PDMS server shuts down, upon restart PDMS will continue to monitor the replicating job. In addition to this awareness, PDMS will continue to attempt to make progress on a replication task after server shutdown. For this guarantee to be in place the replication itself is handled by RFT, which offers identical fault tolerance guarantees and also assures use of GridFTP's restart mechanisms to ensure previously transferred data need not be retransferred. There is an option to not use RFT as the transfer mechanism, but this option does not have the same guarantees that a job will continue to make progress.

At the time of writing, PDMS implements the fault tolerance guarantees by saving the status of a replication job and the status of a replicating LFN in MySQL database. In order to abstract this implementation detail the persistence module implements a factory Data Access Object (DAO) class, which contains the implementation specifics for connecting to a MySQL database server. The component also contains two other classes that are responsible for implementing the implementation dependent details that deal with saving the state of the replication job and the state of a replicating LFN.

2.2.3 Replication Component

The replication component is responsible for the replication of collections. A collection is a set of files that are grouped together because of a logical dependency amongst them. The logical dependency could be that the files are part of the same experiment, or that the files are part of the same observation. One of the logical subcomponents of the replication module is the Replication Manager. It is responsible for overseeing a replication and for reacting to unusual events such as errors. The replication task is divided into a number of subtasks in order to make the design more modular and the code more manageable. The subtasks are as follows: Replication Job, MCS query job, PFN query job, PFN to PFN job, create vault directory job, and RFT job. The execution of the subtasks is further subdivided in two queues. The first queue is called Work queue and it holds all the jobs that are not involved in the physical movement of data. The second queue is called the Transfer queue, which holds the rest of the jobs.

The replication process is orchestrated by the Replication Manager where the first task of the Replication Manager is to initialize a Replication Job. Once the Replication job is recorded into the persistent database, a directory is created on the destination host by the create vault directory job. Next, the Replication Manager handles the MCS query to select LFNs to be replicated. The MCS query job takes a query array, which is an array of conditions, selects an MCS client from the pool of MCS clients and places itself in the Work queue. The result of the MCS query job is an array of LFNs. That array is passed to the PFN query job, which queries the RLS database for the physical locations of the LFNs. The result is a hash table that has an LFN as a key and a list of PFN locations as a value. The reason why the value is a list of locations is that physical files that correspond to a LFN could reside in multiple physical locations. This is the case when a collection has been replicated. Depending on whether RFT is selected as a transport mechanism or not, the next step could be one of the following. In the case when RFT is selected, the RFT subcomponent is executed. The RFT job creates and initializes an RFT client and the actual job of physically transferring the files between sites is delegated to the RFT server. In the other case, the physical transfer of files is initiated by the PFN to PFN job. The PFN to PFN job selects the source locations from the hash table, sets up a third party transfer, and places itself into the Transfer queue. Once a job is in the transfer queue, the actual movement of data can start to take place.

2.3 Replica Selection Strategy

RLS is used to retrieve the physical locations of replicas. As copies of data items can be physically present at multiple sites, RLS returns a list of all locations. PDMS uses only one of the locations to transfer the data from. PDMS selects the first location from the list of retrieved locations. However, in order to optimize the transfer costs, more sophisticated strategies could be employed to select the location from which to start the transfer. This requires a definition of a cost model and an algorithm for selecting the source location.

2.4 Network Overhead

PDMS uses several grid services that may be available on remote servers. Calling those services will involve network costs. Moreover, those grid services use databases, which again can be on remote hosts and incur network costs when database operations are performed. The network cost could be a function of the usage pattern and/or network conditions. This subsection identifies the various network costs in PDMS design.

Figure 3 shows the interactions that can happen over network between various components of PDMS. As PDMS is deployed as a Web service, the clients call the server over the network. PDMS does not require a full deployment of MCS; a MySQL server with a database configured using the MCS schema is enough. The MCS database can exist on a remote host and any interaction with MCS would incur network costs. RFT and RLS are deployed as Web services and hence, these services are also being invoked over network. PDMS, MCS, RLS and RFT in turn use databases which may be present on remote hosts and would incur network costs. All the above interactions are shown as a dotted line in the Figure 3

2.5 Scalability

The MCS and RLS, used by PDMS, are single instance services, i.e., the MCS stores the metadata information in a database on a single host and the local replica catalog (LRC) component of RLS maintains information about replicas at a single site. Due to this fact, MCS and RLS are the potential single points of failure. If the MCS database is unreachable and/or if the network path to the single LRC component of RLS becomes unavailable, PDMS

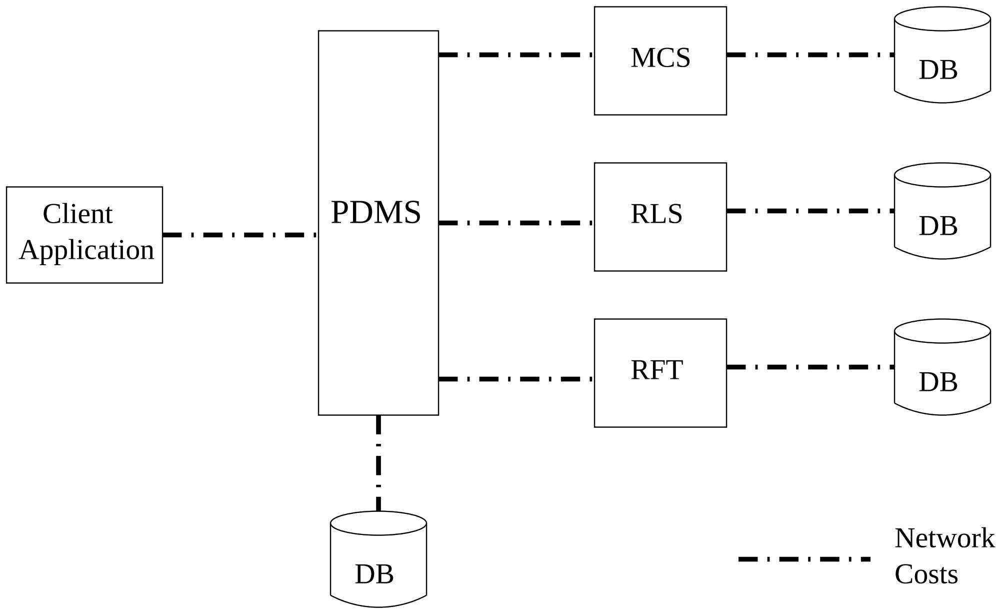


Figure 3: Network Costs involved in PDMS.

would fail in finding the information required to replicate data. It is unlikely either of these services will become performance bottlenecks. In tests the time taken for queries or additions to either service is small in comparison to time taken by the GSI handshake of the PDMS service itself.

The RLS system need not represent a single point of failure. The RLS service is designed for fault tolerance as an RLS server can have multiple redundant replica location indexes (RLI) which point to local replica catalogs (LRC) that claim to have knowledge of the location of a particular logical file. A higher degree of fault tolerance is available by using a larger number of LRCs and RLIs.

MCS is a single point of failure in the system. One way to keep some functionality when the MCS database is not available is by creating several slave read only replicas of the MCS database. This will allow replication requests to continue in the event that the master MCS is not reachable. Given the complexity involved in dealing with inconsistent modifications to MCS, it does not seem useful to attempt to allow MCS modifications to a non-primary MCS database.

2.6 Support for Different Replication and Data Movement Strategies

PDMS has been primarily designed as a tool to replicate datasets in distributed environments. However, in the current design, the decisions of when and where to replicate the data items are not made or supported by PDMS. Datasets can be replicated using a number of strategies. For example, replicate data based on the usage history or replicate data based on the present/future system conditions. The PDMS design could be extended provide mechanisms to support different replication strategies.

PDMS uses the RFT service to perform the actual data movement. However, RFT is just one mechanism of doing the transfers; data transfers can be performed using several other replications services. For example, PDMS can make use of a GridFTP overlay network that splits TCP connection for improved performance [4]. The design of PDMS allows for the integration of new data movement mechanisms.

3 Functional Interface

The PDMS system is a Web service and its interface is formally described in Web Service Description Language (WSDL). This section describes the functionality provided by the PDMS.

3.1 Register Collection

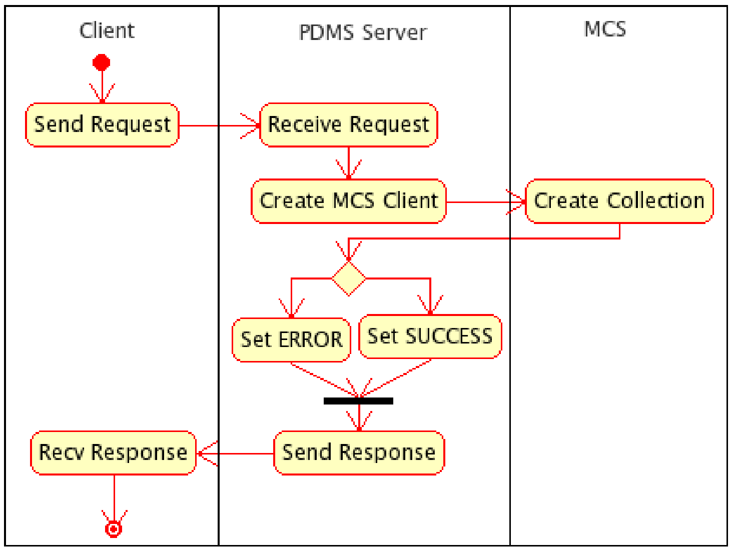


Figure 4: Activities for a register collection operation.

A collection within MCS is a user-defined aggregation of logical file names and/or other collections. Collections in MCS are conceptually similar to directories on the filesystems. Collections aid in supporting authorization on groups of files rather than on individual files. PDMS requires that all the logical files for a certain dataset belong to an MCS collection. The ‘register collection’ operation within PDMS registers the specified collection with the system. The following input parameters are accepted by the ‘register collection’ operation:

- Collection Name - The name of the collection.
- Collection Attributes - An array of attributes for the collection.
- Owner DN - The distinguished name of the owner of the collection.
- Manager Members - An array of the distinguished names of the users who have management privileges on the collection.
- Read Members - An array of the distinguished names of the users who have read privileges on the collection.

- Write Members - An array of the distinguished names of the users who have write privileges on the collection.

The following output parameters are produced by ‘register collection’:

- Return Code - The return code of the operation.
- Error Message - The error message if the collection could not be successfully registered.

Figure 4 shows the activities that are performed within client, PDMS server and MCS when a register collection operation is invoked. Client sends a request to the PDMS server, as seen in the figure, which then creates a MCS client and sends the collection creation request to MCS. The result of collection creation, obtained from the MCS, is send as the response to the client.

3.2 Register LFNs

A set of metadata uniquely identifies a logical file name. ‘Register LFNs’ in PDMS allows to register a set of LFNs to a collection. It accepts a set of metadata corresponding to certain dataset and creates and registers the logical file names within MCS. Following input parameters are accepted by ‘register LFNs’:

- MetaData - The metadata of the data in XML format. The XML representation of the metadata should conform to the PDMS schema, explained in Section 3.2.1.
- PartialRegistration - A flag that indicates whether the failure of registration of a single logical file fails the entire registration request.

The following output parameters are generated by ‘register LFNs’:

- Return Code - The return code of the operation.
- Error Message - The error message, if any.
- New LFNs - An array of the new logical file names created by PDMS upon registration of data with MCS.
- Old LFNs - An array of the logical file names that were already registered and were tried to register again.

Figure 5 shows the activities that are performed within client, PDMS server and MCS when a ’register LFNs’ operation is invoked.

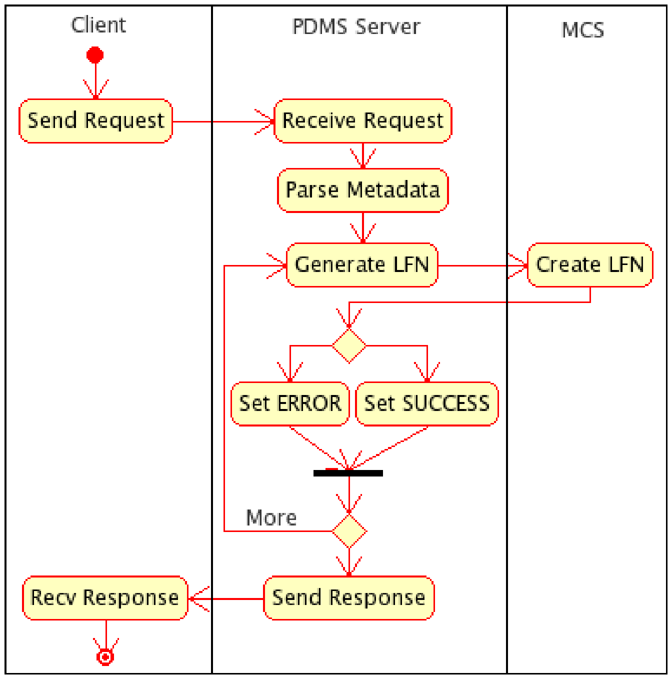


Figure 5: Activities for a register LFNs operation.

3.2.1 Register Logical File List Schema

PDMS allows registration of datasets. A dataset can be registered by specifying the metadata of the data. The metadata is specified in XML format. In order that PDMS understands the XML representation of metadata, the XML document should conform to a pre-defined schema. The description of XML schema is as follows:

- The root entity of the XML document should be a logical file list. A list contains **at least one** logical file entity.
- A logical file entity is a sequence of **at least one** physical location entity and **at least one** file attribute entity.
- A physical location entity is a standard XML URI.
- A file attribute entity contains **exactly one** attribute name and a choice of the attribute type from - STRING, INTEGER, FLOAT, DATE, TIME or DATETIME.

In the above schema, a file attribute represents a single attribute or characteristic of the data. Combination of file attribute(s) and physical location(s) represent the location(s) where the data corresponding to the attribute(s) is present, the logical file entity. A collection of logical file entities, the logical file list represents a dataset. A specific group of users may have more restrictive requirements on a logical file in their data set. For example they may insist that all logical files have a start date associated with it. PDMS will enforce the consistency of other schemas, but all schemas must be a subset of this one for PDMS to function.

The XML schema is included in Appendix A.

3.3 Replicate

Replicate operation is used to replicate a collection. This operation uses the Reliable File Transfer (RFT) service to perform the transfers. The following input parameters are accepted by the 'replicate':

- Query Array - Query Array is an array of conditions on the metadata that uniquely identifies a dataset. A condition is of the form <*attribute_type*> <*attribute_name*> <*condition*> <*attribute_value*>.
- Destination hostname - The hostname of the destination site.
- Vault - The location in the filesystem on the destination site.

The following output parameters are produced by 'replicate':

- Return Code - The return code of the operation.
- Error Message - The error message, if any.
- Replication Id - The unique id for the replication job. This id can be used to retrieve the status of replication.

Figure 6 shows the activities that are performed within client, PDMS server, MCS, RLS and RFT when a 'replicate' operation is invoked.

3.4 Replication Status

Replication Status returns the current status of a previously submitted replication task. It accepts the following input parameters:

- Replication Id - The unique id of the replication job, got from the 'replicate' task.

The following output parameters are produced by the operation:

- Replication Id
- Status - The status of the replication job can be either of:
  - Received - The replication request has been received by PDMS.
  - LFNs Retrieved - The logical file names for the data to be replicated have been retrieved from MCS. This corresponds to the step 2 in Figure 2.

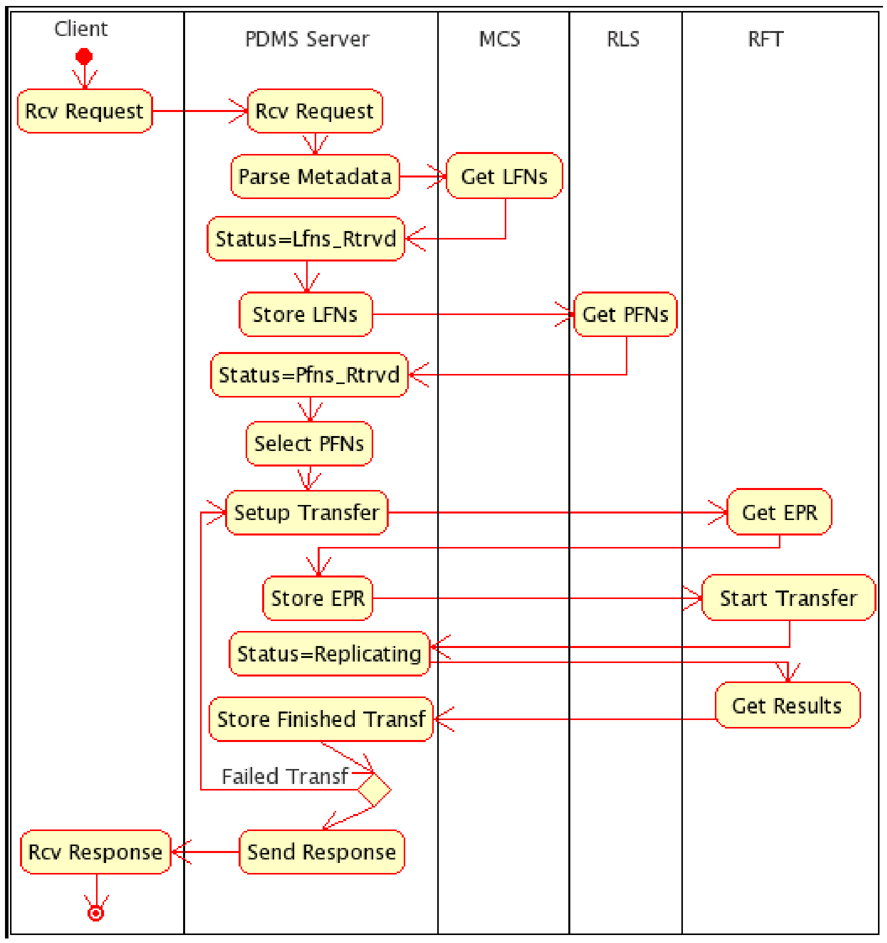


Figure 6: Activities for a replicate operation.

  – PFNs Retrieved - The physical file names for the data to be replicated have been retrieved from RLS. This corresponds to the step 4 in Figure 2.
  – Transferring - The data is being transferred from the source locations to the destination site. This status corresponds to the step 5 in Figure 2.
  – Done - All the transfers have been completed.

- Number of Transfers Finished - The number of files that have been already transferred.
- Number of Transfers in Progress - The number of files that are currently being transferred.
- Number of Transfers Pending - The number of transfers that have still not been started.
- Number of Errors - The number of transfers that have failed.
- Estimated Completion Time - The estimated time in which all transfers will be completed.

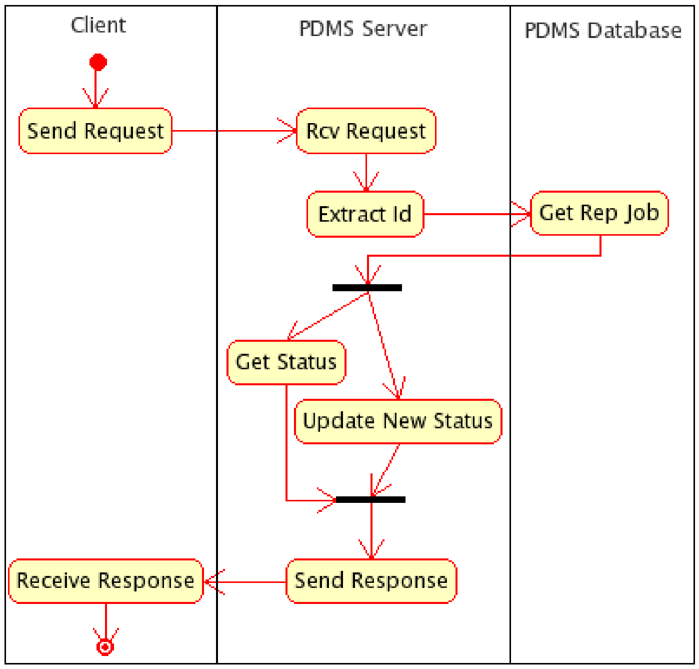


Figure 7: Activities for retrieving replication status.

- Errors Encountered - An array of error messages indicating the problems occurred during the transfers.

Figure 7 shows the activities that are performed within client and PDMS server when status of a replication job is retrieved.

3.5 PFN Lease

PFN lease is used to acquire unique namespace for a set of files on a particular storage location. This operation is used when checking in a dataset into PDMS. This operation guarantees unique physical file names for the files within dataset. The interface of this operation contains following input parameters:

- Number of Files
- Length of lease required
- List of possible sites

The output parameters are:

- Return Code
- Error Message

- Expiry Time
- Array of PFNs that can be used

3.6 Checkin Data

Checkin is a client side operation that moves a dataset to PDMS aware storage location(s) and subsequently registers the dataset with the PDMS. This operation is particularly useful when the dataset lies within a firewall and an outside PDMS server cannot access it. The client, inside the firewall, uses the PFN Lease operation to acquire a set of unique physical file names at the storage location(s). The dataset is then transferred to the storage locations by the client which uses GridFTP for the data movement. The checkin operation differs from the replicate operation in a sense that checkin is handled by the client whereas replicate is handled by the server. It is to be noted that subsequent to the checkin operation, the PDMS server is not aware of the source location(s) of the checked-in dataset.

3.7 Checkout Data

Checkout is a client side operation that copies a dataset from PDMS aware storage location(s) to a local system. The checkout operation first queries the PDMS server to retrieve the physical file locations of the interesting dataset and then initiates a GridFTP transfer of files from those locations to the local system. Subsequent to this operation, the PDMS server is not aware of and does not track the checked-out dataset. This operation is particularly useful when a user wants to retrieve a personal copy of the dataset from the PDMS system.

3.8 Grant Permissions

PDMS provides varying amount of authorization on the data and the meta-data that the system handles. This section describes the authorization mechanisms for the meta-data. The authorization on meta-data is handled at the collection level. When a collection or a file gets created, by default the user creating the object gets the read, write and manage permissions on it. This user a.k.a the owner can further distribute appropriate permissions to the other users using the grant permission operation. The input interface of this operation contains following parameters:

- Requesting user
- Set of target users
- Permission values
- Collection id

The output parameters are:

- Return Code
- Error Message

3.9 Deletion

Deletion provides mechanisms to delete objects from PDMS. The candidate objects are collections, logical files and the actual physical files. Deleting a collection can mean deleting just the collection or the collection and the logical files beneath it or the collection and the logical and physical files beneath it. Similar semantics exist for deleting the logical files. The deletion interface is as follows: The input parameters are:

- Deletion object - collection or lfn or file
- Collection identifier
- Lfn identifier
- File identifier
- Recurse Collections - boolean flag
- Recurse Lfns - boolean flag
- Recurse Files - boolean flag

The output parameters are:

- Collection deletion succeeded/failed/skipped
- Logical files deletion succeeded/failed/skipped
- Physical files deletion succeeded/failed/skipped

4 PDMS Monitoring

As PDMS is designed to handle large amounts of data, located at multiple storage sites and multiple logical collections, it becomes important to provide a monitoring service that aids in keeping track of the details of data that it manages. PDMS includes a GUI based tool that provides such a service for presenting live information of PDMS state. The monitoring tool is based on client-server model in a sense that it interacts with the PDMS server over the Web services to gather the information.

Within PDMS, logical collections are the primary objects for categorizing related data. The PDMS monitoring tool provides a per collection based monitoring. The tool monitors two key aspects of a collection - (i) catalog based summary information that details the specifics of collection contents and distribution of collections across various sites, and (ii) information about replication jobs, details of the migration of data associated with a collection. Changes to the information presented by the PDMS monitor is event driven and a change on the server side is reflected immediately on the client side. The WSRF notification facility from the Globus toolkit has been used to provide even driven updates.

The PDMS monitor can be used in two ways. First, as a standalone Java swing application that can be started using a command line interface and second, as an applet that is deployed over Internet into a browser. In order to access the monitored information in a portable manner, Gridsphere portal framework [1] is used to deploy the applet version of the monitoring tool. Gridsphere is a framework that allows easy deployment of third-party portlet web applications. Use of Gridsphere to deploy the monitoring tool enables the PDMS information to be presented along with information other systems that produce or consume the data managed by PDMS. For example, information about the PDMS managed data, required or produced by a workflow, can be presented along with the information about the job executions of that workflow for better understanding of the workflow processes. Figure 8 shows the PDMS monitor deployed within Gridshpere portal framework. The following subsections provide the details of the monitored information.

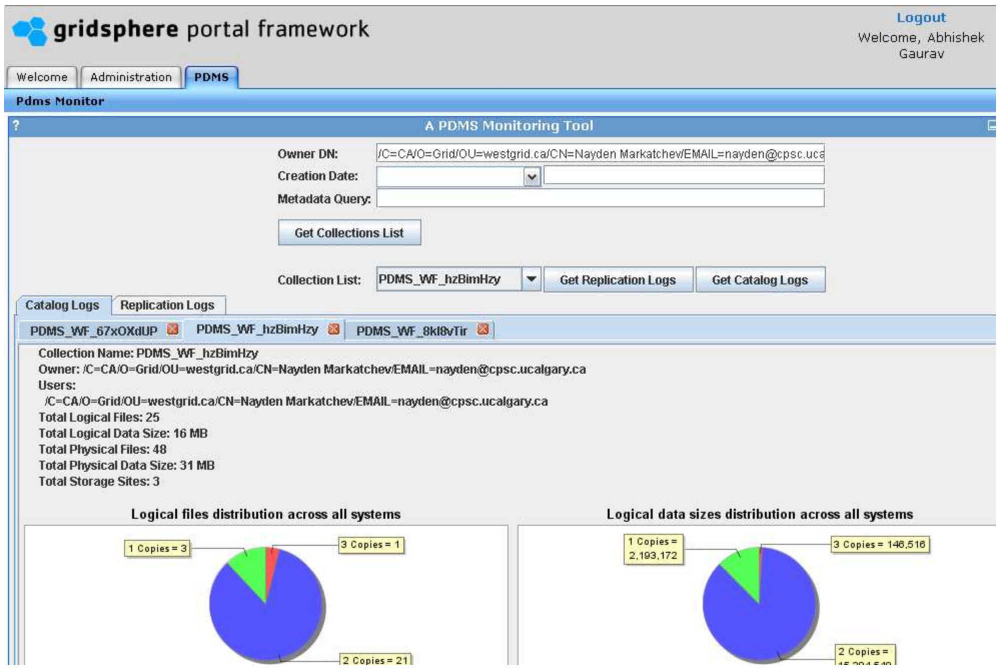
4.1 Catalog Based Information

The catalog based information includes details of the collection's contents and the spread of the collection across storage sites. Specifically, it presents the following information:

- Distinguished names of the owner and users of the collection. The user that creates the collection is the owner of the collection and the users that store data within that collection are termed the users of that collection.

- The logical and physical file counts and the corresponding data sizes. A logical file F of size S can be physically present at multiple locations M. In this case, logical file count is 1, physical file count is M, logical data size is S and physical data size is M*S.

- The redundancy level or the extent to which the collection is replicated, i.e., how many files in the collections have how many replicas.



Owner DN: /C=CA/O=Grid/OU=westgrid.ca/CN=Nayden Markatchev/EMAIL=nayden@cpsc.uca
Creation Date:
Metadata Query:
Collection List: PDMS_WF_hzBimHzy

Collection Name: PDMS_WF_hzBimHzy
Owner: /C=CA/O=Grid/OU=westgrid.ca/CN=Nayden Markatchev/EMAIL=nayden@cpsc.ucalgary.ca
Users:
/C=CA/O=Grid/OU=westgrid.ca/CN=Nayden Markatchev/EMAIL=nayden@cpsc.ucalgary.ca
Total Logical Files: 25
Total Logical Data Size: 16 MB
Total Physical Files: 48
Total Physical Data Size: 31 MB
Total Storage Sites: 3

Figure 8: PDMS Monitor deployed within Gridsphere portal framework.

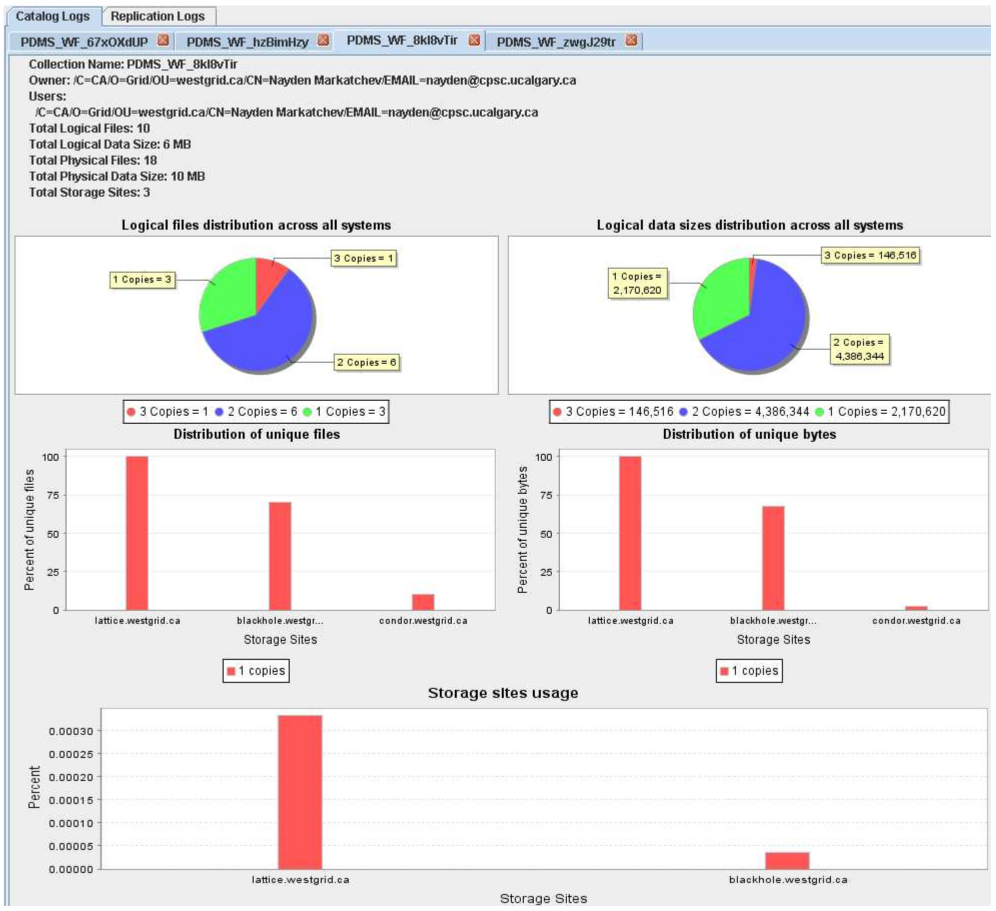
- The storage sites used by the collection and the percentage of the collection files present at each of those sites.
- Usage of a particular storage site, i.e., what percent of the storage site is occupied by the collection. In order to get this information, the PDMS monitor queries MDS to retrieve the total storage capacity of the storage site.

Figure 9 shows the catalog based information as presented by PDMS monitor.

4.2 Replication Information

The replication information includes details of the replication jobs associated with a collection. PDMS records the details of the replication jobs for fault tolerance purposes, as mentioned in Section 2.2.2. The PDMS monitor queries PDMS for presenting the replication information. Specifically, the following information about replication jobs is presented:

- user who started the replication job,
- current status,
- times at which the replication job was requested, started, and completed or last reported,
- the replication destination,

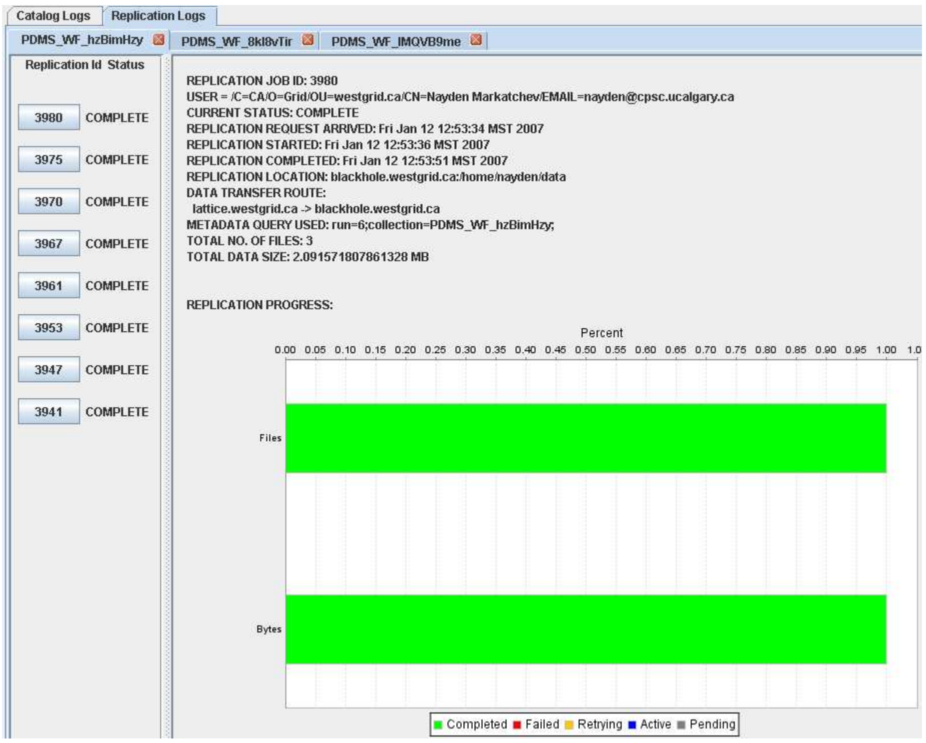


Collection Name: PDMS_WF_8kl8vTir
Owner: /C=CA/O=Grid/OU=westgrid.ca/CN=Nayden Markatchev/EMAIL=nayden@cpsc.ucalgary.ca
Users:
/C=CA/O=Grid/OU=westgrid.ca/CN=Nayden Markatchev/EMAIL=nayden@cpsc.ucalgary.ca
Total Logical Files: 10
Total Logical Data Size: 6 MB
Total Physical Files: 18
Total Physical Data Size: 10 MB
Total Storage Sites: 3

Figure 9: Catalog information page of the PDMS monitor.

- the metadata query used to invoke the replication job,
- the data transfer route, i.e.. the hosts involved during the data transfer; this may include any host(s) that may have been used to split the network connection,
- the total number of logical files and the associated data size,
- the current progress in terms of number of files and associated data sizes completed, failed, active, pending and being retried.

Figure 10 shows the replication information for a collection as presented by PDMS monitor.



Replication Id	Status
3980	COMPLETE
3975	COMPLETE
3970	COMPLETE
3967	COMPLETE
3961	COMPLETE
3953	COMPLETE
3947	COMPLETE
3941	COMPLETE

```
REPLICATION JOB ID: 3980
USER = /C=CA/O=Grid/OU=westgrid.ca/CN=Nayden Markatchev/EMAIL=nayden@cpsc.ucalgary.ca
CURRENT STATUS: COMPLETE
REPLICATION REQUEST ARRIVED: Fri Jan 12 12:53:34 MST 2007
REPLICATION STARTED: Fri Jan 12 12:53:36 MST 2007
REPLICATION COMPLETED: Fri Jan 12 12:53:51 MST 2007
REPLICATION LOCATION: blackhole.westgrid.ca:/home/nayden/data
DATA TRANSFER ROUTE:
 lattice.westgrid.ca -> blackhole.westgrid.ca
METADATA QUERY USED: run=6;collection=PDMS_WF_hzBimHzy;
TOTAL NO. OF FILES: 3
TOTAL DATA SIZE: 2.091571807861328 MB


REPLICATION PROGRESS:
```

Figure 10: Replication information page of the PDMS monitor.

A Register Logical File List Schema

```
<?xml version="1.0"?>
<xsd:schema
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:pdms="http://telesim.cpsc.ucalgary.ca/wsrf/services/PDMS"
                elementFormDefault="unqualified"
        attributeFormDefault="unqualified"
        targetNamespace="http://telesim.cpsc.ucalgary.ca/wsrf/services/PDMS">

        <xsd:element name="logicalFileList" type="pdms:logicalFileList"/>

        <!-- a pdms:fileAttribute consists of a name, and an element
        of one of the MCS types(excluding spatial) this ensure the
        parser can deal with the values.
        ie. if the MCS_INTEGER type is used, what is actually included
        is an integer-->
        <xsd:complexType name="fileAttribute">

                <xsd:choice>
                        <xsd:element name="MCS_STRING" type="xsd:string"/>
                        <xsd:element name="MCS_INTEGER" type="xsd:integer"/>
                        <xsd:element name="MCS_FLOAT" type="xsd:float"/>
                        <xsd:element name="MCS_DATE" type="xsd:date"/>
                        <xsd:element name="MCS_TIME" type="xsd:time"/>
                        <xsd:element name="MCS_DATETIME" type="xsd:dateTime"/>
                </xsd:choice>
        <xsd:attribute name="att_name" type="xsd:string" use="required"/>
        </xsd:complexType>

        <!--a logical file consists of one or more locations and one or
        more fileAttributes-->
        <xsd:complexType name="logicalFile">
        <xsd:sequence>
                <xsd:sequence minOccurs = "1">
                <xsd:element name="physicalLocation" type="xsd:anyURI"
                 minOccurs="1" maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:sequence minOccurs = "1">
                <xsd:element name="fileAttribute" type="pdms:fileAttribute"
                 minOccurs="1" maxOccurs="unbounded"/>
                </xsd:sequence>
                </xsd:sequence>
        </xsd:complexType>

        <!--a logical file list contains of one or more logicalfiles-->
        <xsd:complexType name="logicalFileList" >
        <xsd:sequence>
                <xsd:element name="logicalFile" type="pdms:logicalFile"
                 minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="collection" type="xsd:string" use="required"/>
        </xsd:complexType>
</xsd:schema>
```

References

[1] Gridsphere portal framework. http://www.gridsphere.org.

[2] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.

[3] U. of Calgary Grid Research Centre. PDMS user management document. GRC Internal report.

[4] P. Rizk, C. Kiddle, and R. Simmonds. A GridFTP overlay network service. In *7th IEEE/ACM International Conference on Grid Computing*, 2006.