XSEDE Data Management Use Cases: Architectural Response

Version 1.1

Revision history

Version	Date	Summary of changes
1.0	2015-08-19	Initial version
1.1	2015-10-16	Inserted/updated GFFS material

resource has migrated while they are interacting with it, or whether a particular resource instance has failed and the user has been redirected to a replica. Thus, the GFFS implements six of what are commonly referred to as the Golden Transparencies, access, location, migration, failure, replication, and implementation transparency.

The GFFS namespace and resources are accessed primarily via three mechanisms: a shell-command line tool (e.g., cp <pathname> <pathname>), a graphical user interface, and via operating specific file system mechanisms that effectively "mount" the GFFS into the local file system. For example, one can FUSE mount the GFFS in Linux, or use an SMB server to provide GFFS access in Windows. Once "mounted" GFFS files and directories can be used by users and applications without any knowledge of the GFFS or distributed computing. It (the GFFS), looks and feels like a file system. The figure below is an illustration of how the FUSE driver works in Linux.

Figure 1. Access to the GFFS via a Linux FUSE driver. The GFFS is “mounted” and then accessed as a local file system.

1. Introduction

This document explains how the XSEDE system architecture supports XSEDE's data management use cases. The use cases are described in the document, *XSEDE Data Management Use Cases, Version 1.6*, dated March 20, 2014 [UCDM].

This document refers frequently to *XSEDE Architecture Overview 2.0 [ARCH]* and *XSEDE Architecture Level 3 Decomposition [L3D]*. Readers should have both of these documents available for reference and should already be somewhat familiar with their contents. It also relies heavily on architectural responses to XSEDE's canonical use cases. (See the [UCCAN-*n*] references below.)

1.1 Structure of this document

This document is organized as follows. Section 2 describes and summarizes the science gateway use cases. Section 3 describes how the XSEDE architectural components should be used to implement the use cases from section 2.

1.2 References

[UCDM] XSEDE Data Management Use Cases, Version 1.6, March 20, 2014. (http://hdl.handle.net/2142/48909)

[ARCH] XSEDE Architecture Overview 2.0. (http://hdl.handle.net/2142/50274)

[GFFS-1] Grimshaw, A., Morgan, M. and Kalyanaraman, A. GFFS: THE XSEDE GLOBAL FEDERATED FILE SYSTEM. Parallel Processing Letters, 23 (02).

[GROM] Genesis II Omnibus Reference Manual (http://genesis2.virginia.edu/wiki/uploads/Main/GenesisII_omnibus_reference_manual.pdf)

[L3D] XSEDE Architecture Level 3 Decomposition. (http://hdl.handle.net/2142/45114)

[RNS] Pereira, M., Tatebe, O., Luan, L., Anderson, T. and Xu, J. Resource Name Service Specification, Open Grid Forum, GFD-101, 2006.

[ByteIO] Morgan, M. ByteIO Specification, Global Grid Forum. GFD-86, 2006.

[UCCAN-1] XSEDE Canonical Use Case 1 - Level 3 Decomposition. (http://hdl.handle.net/2142/73149)

[UCCAN-2] XSEDE Canonical Use Case 2 Response: Managed File Transfer. (http://hdl.handle.net/2142/73209)

[UCCAN-3] XSEDE Canonical Use Case 3 Response: Remote File Access. (http://hdl.handle.net/2142/50327)

[UCCAN-4] XSEDE Canaonic Use Case 4 Response: Interactive Login. (http://hdl.handle.net/2142/73210)

[UCCAN-6] Canonical Use Case 6: Authenticate to one or more SP resources, SP services, and XSEDE central services: Architectural Response v1.1.2.

[UCCAN-9] Canonical Use Case 9: XSEDE User Identity and Access Management: Architectural Response v1.1. (http://hdl.handle.net/2142/73214)

[UCCAN-8-12] X-WAVE Architecture Realization, XSEDE Canonical Use Case 8 & 12:Search for Resource Information & Update Resource Information

[UCIDM] XSEDE Identity Management Use Cases: Architectural Response, Version 1.3. July 30, 2015. (https://drive.google.com/drive/u/0/folders/0B-TKED6wiDyISnlaWGpkM3NaclU)

[UCSGW] XSEDE Science Gateway Use Cases: Architectural Response, Version 1.0. April 20, 2015. (https://drive.google.com/drive/u/0/folders/0B_9f7nQcDIOAeU1OY00temhjb1U)

2. Data management use cases

This section describes and summarizes the data management use cases. This summary includes both the functional descriptions and the non-functional characteristics (aka, "quality attributes") for these use cases. The full descriptions of these use cases are documented in [UCDM].

Following familiar software engineering principles[1], the XSEDE Architecture and Design team engaged with stakeholders from XSEDE's "big data" and advanced user services groups to document these use cases, detailing the desired user experience and associated quality attributes. The resulting use cases and quality attributes are listed in Table 1. In Section 3, we describe how these use cases and quality attribute scenarios should be implemented in the context of the XSEDE system architecture.

Table 1: Data management use cases (shaded) and their associated quality attributes

DM-1	**Share and manage a common repository of data with a distributed user community**
QAS-DM-1.1	A single site-wide failure lasting less than 8-12 hours will not interrupt data availability.
QAS-DM-1.2	Parallel reads and writes are available at all times.
QAS-DM-1.3	No collection- or project-specific software or tools are required to store or retrieve data.
DM-2	**Coordinated computation and analysis involving a shared body of data**
QAS-DM-2.1	A single site-wide failure lasting less than 8-12 hours will not interrupt data availability.
QAS-DM-2.2	Parallel reads and writes are available at all times.
DM-3	**Shared use of large-scale/streaming sensor input data**
QAS-DM-3.1	95% success rate for storing data received by the input agent
QAS-DM-3.2	95% success rate for accessing stored sensor data
QAS-DM-3.3	Stored data can be retrieved at a continuous rate of 10MB/sec
DM-4	**Migration of data between resources**
QAS-DM-4.1	Only one initiation step is required per migration endpoint.
QAS-DM-4.2	100% of data must be transferred between resources (w/verification).
QAS-DM-4.3	Any data that isn't transferred or verified will automatically be retransferred.
DM-5	**Metadata Storage and Access**

[1] Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C. and Wood, W. Quality Attribute Workshops (QAWs), Third Edition, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-2003-TR-016, 2003.

QAS-DM-5.1	Metadata storage and retrieval commands must clearly indicate whether no metadata exists or no metadata was found that matched the parameters given.
QAS-DM-5.2	Metadata storage and search functions must be able to operate on large numbers of records within a reasonable time. (100 million per minute?)
QAS-DM-5.3	If multiple metadata servers/repositories are used to provide failover capability, there must be no more than 30 minutes differential in stored metadata.

3. Architectural responses

Many of the elements of the XSEDE system architecture that implement these use cases are already documented in XSEDE's canonical use cases.

- DM-1 is mainly accomplished via UCCAN-2 for data access and UCCAN-4 for remote login. UCCAN-3 provides an alternate data access interface to UCCAN-2.
- DM-2, DM-3, and DM-5 are mainly accomplished via either UCCAN-1 or UCCAN-4 for computation, with support from either UCCAN-3 or UCCAN-2 for data access. (Note: Significant parts of DM-5 are not implemented as part of the XSEDE system and must be supplied by the user.)
- DM-4 is accomplished via UCCAN-2 for data transfer.

The remainder of this document provides details of the implementation that are not covered explicitly in the canonical use cases above. In particular, this document provides some additional detail on the component-to-component interactions and significantly more information about how quality attributes should be satisfied.

3.0 XSEDE's architecture for data management

This section provides a brief overview of the way in which the XSEDE architecture is intended to support data management tasks. We first refer to [ARCH §D], in which we introduce XSEDE's high-level architecture, copied below in Figure 1.

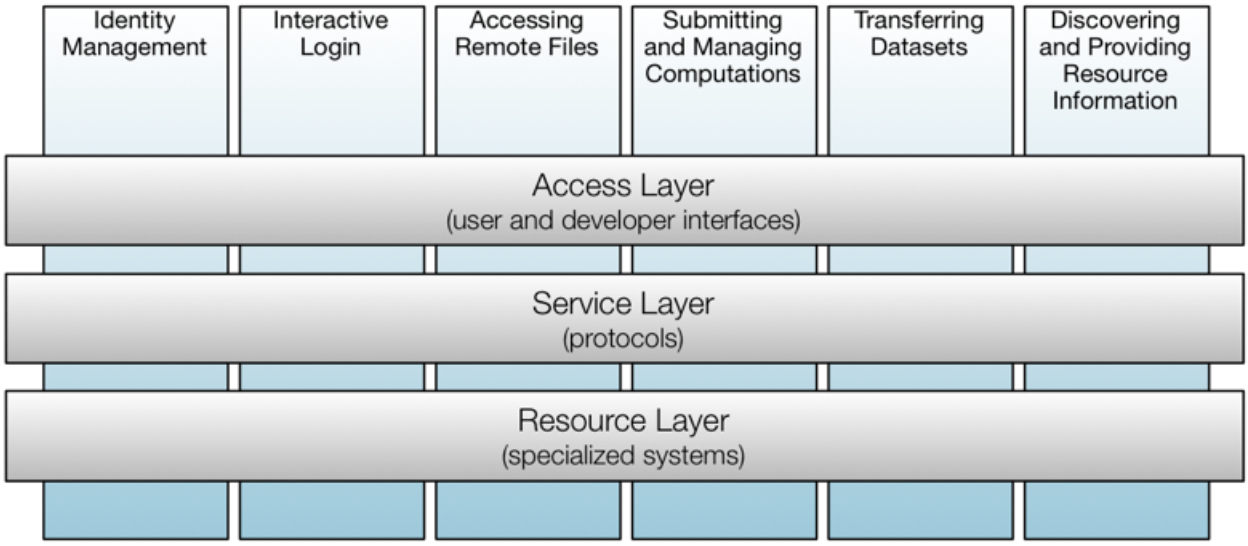


Figure 1. XSEDE architecture layers and system-wide functions

The top-most layer of the XSEDE architecture, the access layer, consists of three kinds of interfaces, described in [ARCH §D.1]: *graphical user interfaces* (GUIs), *command-line interfaces* (CLIs), and *application programmer interfaces* (APIs). Each of these interfaces may be available in one or both of the following forms: *thin-client interfaces*, accessible via software that is typically pre-installed on standard systems (Windows, Macintosh, Linux); and *thick-client interfaces*, which require software to be installed that probably isn't pre-installed on a standard system.

The service layer and resource layer are described in [ARCH §D.2] and [ARCH §D.3], respectively. For those who need to manage data using XSEDE services, the following points may be of particular interest.

1. XSEDE's foundational architecture provides several important data management services. The architecture is designed to provide a *foundation* for data management systems, but not to be a complete solution on its own. Primarily, XSEDE is designed to be a place to **create** data, a place to **analyze and process** data, and a place to **store** large collections of data. We know that user communities may need other services--in addition to XSEDE's--to provide all the pieces needed for a full data management solution. Elements of these more specialized services--that XSEDE does not provide--include: collection curation, metadata and cataloguing, and publication.

   The XSEDE architecture provides both access layer data management services as well as service layer interfaces. These are collectively called the XSEDE Global Federated File System (GFFS) The access layer interfaces include command line tools, a graphical user interface, and FUSE/SMB drivers that map the GFFS into a local client operating system file system (L3D 3.4.1 GFFS, 5.1.3 Data Management, 5.2.2 Data Management,

and the GORM E.3). The XSEDE architecture provides service layer interfaces for file storage, replication and consistency management of stored files, and directory services to keep track of files. File Create/Read/Update/Delete operations are provided via the OGSA-Byte IO interfaces in combination with the Resource Namespace Service (directory) interfaces. These are described in the L3D section 5.1.3 “Data Management”. Replication management is handled using the naming and binding infrastructure (L3D 4.1.3 Naming and Binding: The Naming of Endpoints) and the replicated byteIO implementation L3D (5.2.2.3).

2. *Data storage* is a service offered by independent XSEDE service providers (SPs) and others via the service layer elements described above and via the SP local file system interfaces. These services are integrated with the XSEDE system and XSEDE provides interfaces for accessing them, but they are ultimately designed, managed, and supported independently. Their direct interfaces are represented in the XSEDE architecture as specific elements in the resource layer. Technical support for the interfaces in each layer is provided in different ways.
    a. Access-layer interfaces are supported by the XSEDE organization (support@xsede.org).
    b. Support for service-layer interfaces is collaborative: end users may seek help with service-layer interfaces from the XSEDE organization or from the individual SP, and in either case, the appropriate parties will resolve issues.
    c. Each XSEDE SP supports its own resource-layer interfaces--usually directly and without XSEDE involvement.
3. The access layer is designed to provide a familiar easy to use interface, and mitigate faults and failures in the service layer. For example, when transferring a large dataset of 10,000 files from one resource to another, the service layer may fail to transfer all of the files, returning an error message for the others. Access-layer interfaces will often automatically catch these failures and retry the failed transfers or even notify XSEDE staff who can intervene manually on the user’s behalf. Some service-layer interfaces provide this kind of resiliency, but not all do.

For each of the Data Management use cases in this document, we will identify the access-layer, service-layer, and resource-layer interfaces that we believe are most beneficial to developers, and we will identify alternatives for special cases.

3.0.1 Identity management

All of the data management use cases begin with the user “logging in to a resource.” Although each XSEDE service provider provides and manages an independent login mechanism for each of their resources, the XSEDE system also provides a consistent authentication mechanism that can be used with any of these resources or with other XSEDE system interfaces. It is often unnecessary for the end user to “log in to a resource” in order to access that resource’s capabilities or the data stored on it.

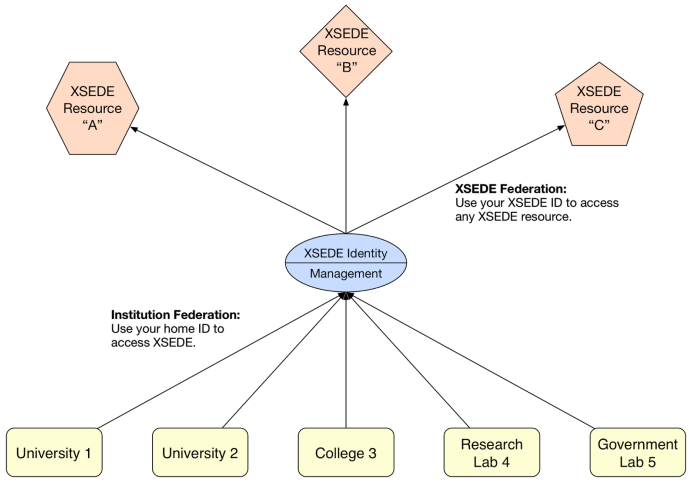


Figure 2. XSEDE's identity management provides two kinds of federation.

XSEDE's system-wide identity management mechanism is federated. This federation works in two distinct ways, as illustrated in Figure 2. The first form, *XSEDE federation,* means that many actions, including data management, can be performed by authenticating (once) to the XSEDE system and then making system requests. These requests are translated by the system into specific requests on specific XSEDE resources, with the user's XSEDE-wide identity automatically translated into resource-specific credentials. In this way, each user has a single XSEDE-wide identity that can serve as a "key" to their unique identities on each XSEDE resource. The second form of federation, *institution federation*, means that for many participating institutions, users can use their home institution's credentials to authenticate to XSEDE. (Most XSEDE users already have a username/password for their home university, college, or research laboratory. Users at non-participating institutions can create a new username and password to access XSEDE.) Combined with the first federation feature, this means that at participating institutions, a researcher can use his/her home ID/password to access the XSEDE resources to which he/she is authorized.

Anyone may establish an XSEDE identity, and anyone at a participating institution may link his/her home institution identity to his/her XSEDE identity. Neither of these are sufficient to allow the user to access an XSEDE resource. To access an XSEDE resource, the user also must be granted an *allocation* to use the resource. Once an allocation is granted, the user will have a unique identity on the resource, and this resource identity will automatically become associated with his/her XSEDE identity.

In each of the following use case responses, we will assume that the user has established an XSEDE identity, has linked his/her home institution ID to his/her XSEDE ID, and has been granted an allocation to use one or more XSEDE resources.

3.0.2 Data interfaces

The XSEDE system is designed to meet the needs of a wide range of research disciplines and methodologies. We consequently support two distinct access modes for data.

- The *managed file transfer* mode requires very little software installation or configuration and can be used with very little preparation. “Managed data transfer” means, in essence, moving data around the system (or into/out of the system) to get it where it can be used. The user (or the user’s application) issues a series of requests for files to be moved from one location to another. These locations include XSEDE systems, campus systems, the user’s local systems, or systems at other research facilities. Thus, the user’s work consists of orchestrating a series of data transfers and data transformations at various locations. Each transfer request can include any number of files and directories, and the system ensures that all files and directories are transferred correctly and completely, notifying the user (or application) when each transfer request is complete. In addition to complete transfers, several forms of one-time synchronization (mirroring) are also supported. This access mode is described in detail in [UCCAN-2].
- The XSEDE Global Federated File System (GFFS) was developed to address the challenge of securely sharing data and storage between universities and supercomputing centers, and directly between universities [GFFS-1]. In a nutshell the GFFS provides a secure global directory structure into which storage, compute resources, identity resources, and file system directory trees can be linked and then subsequently securely accessed using command line tools, a GUI, and by mapping the GFFS directory structure into the local file system using a GFFS-aware FUSE driver or SMB server. This is described more fully later in section 3.1.2, and in [UCCAN-3].

3.1 DM-1 Share and manage a common repository of data with a distributed user community

The description of this use case is: “A group of users, individually or collectively, build a collection of data and software code, which must be shared with and managed by a community of users using many XSEDE resources.” The use case description also defines the following two specific user types: a *collection manager* with elevated privileges to deposit and manage data in the collection, and a *collection user* with limited privileges to access data and code in the collection.

This use case appears to provide a “base case” for several others in the data management area (DM-2, DM-3, and DM-5), so we will make note of important relationships in the response to this use case.

The use case as described in DM-1 does not specify the mode or style in which the data should be available to the community of end users. XSEDE can provide two basic styles for this access: (1) an upload/download interface via which community members can access the data via file transfer tools, or (2) a virtual file system interface that allows community members to

"mount" the collection locally as if it were contained in a locally attached storage system (with notable performance differences). Each of the following subsections covers one of these styles.

3.1.1 Upload/download interface

The upload/download interface is implemented using XSEDE's managed file transfer use case [UCCAN-2]. This interface requires very little up-front setup, software installation, or configuration and can be initiated with little or no preparation. The main preparation is to ensure that the data collection can be physically housed on one or more systems that support the Globus sharing feature. Not all XSEDE storage services provide this feature, so if XSEDE storage is required, some effort must be spent consulting the XSEDE documentation to identify the right storage systems and to obtain allocations on those systems. (The storage system does not need to be an XSEDE service. Many storage providers, including major academic and research institutions, also offer Globus sharing. Establishing a private storage service using a cloud provider such as Amazon S3 or Microsoft Azure is quite straightforward, though of course these commercial services charge their own usage fees.)

Enabling the "login" function for the collection manager (the first step in the use case) is simply a matter of signing up to use XSEDE [UCCAN-9][IDM-1] and logging in to the XSEDE User Portal (XUP) [IDM-2]. (This will already be done if the user has applied for a storage allocation, because the application process itself requires registration and access to XUP.) As described in §3.0.1, the collection manager may link his/her home institution credentials to his/her XSEDE identity and use those home credentials to login to XUP, or he/she may establish a separate XSEDE username/password.

Moving/copying data into the shared collection (the second step in the use case) is accomplished using XSEDE's file transfer functions, which are made available via XUP and Globus. [UCCAN-2] Assuming both the source and destination have Globus Connect software[2], the collection manager can either use the XUP/Globus file transfer interface via a web browser, the Globus command-line interface (via SSH), or a RESTful web application to request any number of data transfers from the source to the destination. Once the request is made, the collection manager can disconnect from the interface and wait for an email notification signifying the completion of the transfer(s).

The collection manager can accomplish step 3 using the same interface as step 2: instead of requesting transfers, the collection manager connects to the data collection endpoint and adjusts directories and file locations. Alternatively, the collection manager could use a remote login connection [UCCAN-4] to login to the system on which the data collection is housed and

[2] The specific instructions depend on where the data is located initially and where the shared collection will be housed. If the data is initially on the collection manager's personal (non-shared) system, he/she should install and launch the Globus Connect Personal software (described in [UCCAN-2]) to make the data available to XSEDE. If the data is located or housed on a shared non-XSEDE system (a campus server or another research facility), that system must have Globus Connect Server installed and the user must be authorized to use the system.

make adjustments using the native system interfaces. (If the collection is housed on a non-XSEDE system, the remote login methods described in [UCCAN-4] may not be appropriate. Consult the specific system's documentation for instructions.)

Step 4 is where Globus sharing features become important. The system on which the data collection is housed must support these sharing features for this procedure to work. The collection manager connects to the system that houses the collection, navigates to the top-most directory to the shared with the community, and clicks the "Share" button to create a *shared endpoint*. This endpoint will be the address given to the community to access the data collection. Once the shared endpoint is created, the collection manager can set sharing permissions on the collection (or any subdirectory within the collection) to control access by specific community members. The collection manager can define groups of community members and assign permissions based on group identities to simplify the access controls.

Step 5 is exactly the same as Step 1, except that this time it is the collection user who is "logging in" (authenticating). Again, what is important here is simply that the collection user has established an XSEDE identity. Unlike the collection manager, it is unnecessary for the collection user to be associated with an XSEDE resource allocation.

Step 6, in which the collection user makes a local copy of the data, is accomplished via XSEDE's file transfer functions [UCCAN-2]. As in Step 2, the destination for the data must be running some form of Globus Connect, either Globus Connect Personal (for a personal system) or Globus Connect Server (for a shared system). The collection user may choose from the XUP/Globus web interface, the Globus command-line interface (via SSH), or any application that makes use of Globus's REST API.

Step 7 is entirely up to the collection user and his/her computing environment.

Step 8 is exactly the same as Step 6.

When using the upload/download interface, the three quality attributes for this use case are satisfied as follows.

- QAS-DM-1.1 (availability during a single-site failure) is satisfied with respect to the access layer: all of the interfaces described above will remain available in the event of a single-site failure. Both XUP and Globus use service replication techniques (see [L3D §9]) to keep these services available during single-site failures, either automatically (Globus) or with operator intervention (XUP). With respect to the service and resource layers, the XSEDE architecture allows service providers who house data collections to make use of multi-site replication to keep their services operational during single-site failure, but the XSEDE architecture *does not require* this to be done by the service provider. Thus, if the service provider builds the service with replication, and the endpoint provided to XSEDE is a replicated service, then single-site failure should not be

visible to collection users. If, however, the service provider does not build the service with replication, a failure at the service provider site will be visible to users as a "temporarily unavailable" notice when the collection user attempts to access the site in Steps 6 or 8, or when the collection manager attempts to access the site in Steps 2-4.

- QAS-DM-1.2 (multi-user access availability) is satisfied because all of the interfaces described above are multi-user and designed to support high numbers (hundreds to thousands) of simultaneous users. Again, this is predicated on the design of the service that hosts the data collection, which is beyond the scope of the XSEDE system itself. If the host service can not support parallel use by many users, the XSEDE system will not be able to provide it either, but the interfaces will fail gracefully and report temporary unavailability of the resource.
- QAS-DM-1.3 (no collection- or project-specific software) is satisfied because all of the interfaces described above are the same for all data collections and projects supported by XSEDE.

3.1.2 Global Federated File System

The Global Federated File System (GFFS) is a federated service oriented architecture that provides secure access to federated resources such as files and file systems, storage, computation clusters, and identity management services via single unified path-based name system [citations, including XSEDE architecture]. Users can place compute servers, storage servers, sub-trees of their file system, or other resources (including running jobs) into the GFFS namespace. Resources, storage, files, directories, are service endpoints of standard services that are implemented in Web Service containers, processes running on a computer. GFFS directories are implemented using Web Services resources that support the RNS [RNS] interface. GFFS files are implemented using Web Services resources that support OGSA ByteIO [ByteIO] interface. We use the Genesis II container (layered on Apache/Axis) in the GFFS.

Resources in the GFFS are virtualized and homogenized using standard interfaces that abstract the essential aspects of what it means to be a file, a directory, a compute service, a running job, or an authentication agent. The resources layer definition describes the individual interfaces, the inter process communication mechanisms, the authentication, authorization, and data integrity mechanism, the naming and binding mechanism, how the naming and binding mechanism can be used to realize the traditional distributed systems transparencies (location, migration, replication, and failure), and sequence diagrams that show how higher level use cases can be realized by composing services.

End users see only pathnames and are unaware of the location of resource instances, the number of replicas of the resource instance (or where they are located), whether a resource has migrated while they are interacting with it, or whether a particular resource instance has failed and the user has been redirected to a replica. Thus, the GFFS implements six of what are commonly referred to as the Golden Transparencies, access, location, migration, failure, replication, and implementation transparency.

The GFFS namespace and resources are accessed primarily via three mechanisms: a shell-command line tool (e.g., cp <pathname> <pathname>), a graphical user interface, and via operating specific file system mechanisms that effectively "mount" the GFFS into the local file system. For example, one can FUSE mount the GFFS in Linux, or use an SMB server to provide GFFS access in Windows. Once "mounted" GFFS files and directories can be used by users and applications without any knowledge of the GFFS or distributed computing. It (the GFFS), looks and feels like a file system. The figure below is an illustration of how the FUSE driver works in Linux.

In the GFFS data, i.e., files and directories, are realized as either ***exports*** or as objects stored in an object storage system. Whether a file or directory is in an export or an object store is transparent to the user.

A user exports a directory tree on a computer system, e.g., *d:\data\experiment1* on a Windows system or */home/grimshaw/experiment1* in Unix, and links the export to a path in the GFFS. For example, I have linked *c:\..My Documents\XSEDE\archteam* to the path */home/xsede.org/grimshaw/xcgandrew/XSEDE/ArchDocs*. Once exported, the data can be accessed by any authorized user from anywhere.

Storage containers (a.k.a. object stores) can exist on just about any machine that has a public IP address. When a new file or directory is created it is by default placed on the same storage container (or storage container group) as the parent directory. The result is that entire sub-trees will be stored together. The user may explicitly override the default behavior by specifying on which storage container they would like to place a new directory or file, e.g.,

mkidir --rns-service=/path-to-container newDirectory.

Files and directories in storage containers can be *K-replicated* for availability and performance. Eventual consistency is guaranteed. As of October 2015 replication for exports is not available. The same techniques used for non exports could be used the same way, but it has not been a priority project for XSEDE.

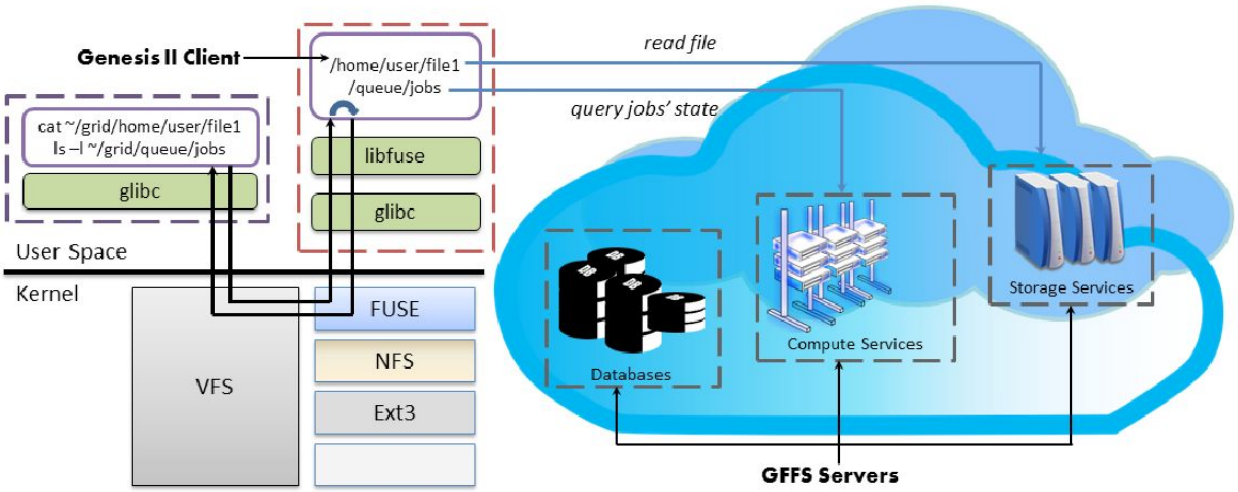


Figure 1. Access to the GFFS via a Linux FUSE driver in Linux. The GFFS is "mounted" and then accessed as a local file system.

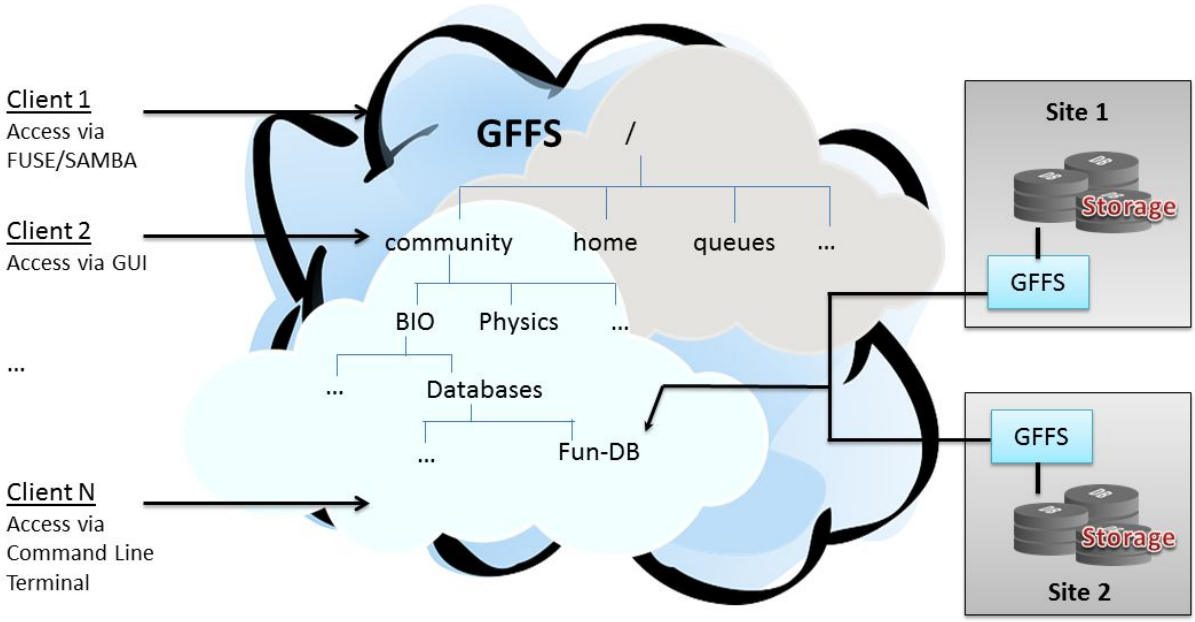
Supporting this use case (DM-1 Share and manage a common repository of data for a distributed user community) using the GFFS is quite simple. The storage location(s) for the collection is decided upon as well as a path in the GFFS. The storage location(s) should have sufficient capacity and external bandwidth to support access. In order to support the availability quality attribute the data will be stored in a storage container rather than as an export.

For example, suppose we wanted to create a new biology collection and place it in */community/Bio/DataBases/Fun-DB*. We assume below that the collection is to physically stored on containers at *path-to-site1* and *path-to-site2*. (See figure below). Assuming the storage containers are ready, a path to the collection is created on the primary container, e.g.,
mkdir --rns-service /path-to-site1 /community/Bio/DataBases/Fun-DB
Similarly, we would replicate

resolver -p */community/Bio/DataBases/Fun-DB* path-to-site2

replicate -p */community/Bio/DataBases/Fun-DB* path-to-site2

At this point the collection can be simply copied into */community/Bio/DataBases/Fun-DB*. See the canonical use case 3 description or the GORM for more information. Once copied in authorized users can access the data via FSUE/SAMBA drivers, via the GFFS GUI, or via command line tools.



Another important part of the set up is to determine the access controls on the collection. Each file and directory can have RWX permissions set, e.g.,

chmod -R */community/Bio/DataBases/Fun-DB* +r --everyone

chmod -R */community/Bio/DataBases/Fun-DB* +w /groups/xsede.org/Community

The above gives everyone read access, and members of the group “Community” write permission.

An alternative GFFS implementation is using GFFS *exports* [L3D, GORM]. An export leaves the data in file system at some site, and links the source directory tree into the GFFS namespace. For example a user could export the directory */home/user1/myData* on their workstation at University-A into */community/Bio/DataBases/DataSetA*. Replication of exports is not currently supported, but could be developed using the same architectural mechanisms used in the current file and directory replication.

The steps for the use case implementation assume that the set up steps described above. There are 8 steps in the use case.

Step 1. Authenticate
There are several mechanisms available to authenticate. The example below uses xsedeLogin. Note there are two group credentials. Group credentials can be used to denote users who have permission to read or modify the newly created collection.

grimshaw@cicero:~$ **grid xsedeLogin** --username=grimshaw --password=**********
Replacing client tool identity with MyProxy credentials for "CN=Andrew Grimshaw, O=National Center for Supercomputing Applications, C=US".
grimshaw@cicero:~$ **grid whoami**
Client Tool Identity:
(CONNECTION) "Andrew Grimshaw"

Additional Credentials:
(USER) "grimshaw" -> (CONNECTION) "Andrew Grimshaw"
(GROUP) "gffs-tutorial-group" -> (CONNECTION) "Andrew Grimshaw"
(GROUP) "gffs-users" -> (CONNECTION) "Andrew Grimshaw"
The security context has now been established for subsequent operations.

Steps 2 and 3 (collection manager moves data from a local system to the data collection and manipulates the contents of the collection) are accomplished via the Genesis II GFFS client command line tools, the GUI, or the FUSE file system driver. The collection manager should mount the GFFS on his/her local system [UCCAN-3] and then use local file management tools to access the collection in the GFFS namespace. The collection manager may copy any files accessible on his/her system into the data collection or manipulate the data collections' contents in GFFS.

Step 4 (collection manager sets permissions on components of the collection) is accomplished using GFFS's access control mechanisms. These mechanisms are described in [L3D §5.2.2] and illustrated below.

chmod -R */community/Bio/DataBases/Fun-DB* +r --everyone

chmod -R */community/Bio/DataBases/Fun-DB* +w /groups/xsede.org/Community

The above gives everyone read access, and members of the group "Community" write permission.

Steps 5-8 are all accomplished in exactly the same way as Steps 1-3, with the exception that it is the collection user, not the collection manager, who is performing the steps. This assumes that the collection manager has granted access authorization to the collection user by assigning specific permissions to the collection user's XSEDE identity or to any group that has the collection user as a member. This also assumes that the collection user has been giving the path to the data collection in GFFS.

When using the virtual file system interface, the three quality attributes for this use case are satisfied as follows.

- QAS-DM-1.1 (availability during a single-site failure) is satisfied with using the replication capabilities of the GFFS (L3D 5.2.2 Data Management, 5.3.7-8 Write to Replicated Files & Directories). We assume the access layer (user interfaces) are installed on the user's (collection manager's or collection user's) local system, and the authentication services and GFFS services the client relies on are operated using replication techniques [L3D §9] that allow for continued use during single-site failure. With respect to the site(s) hosting the data collection, the collection manager has the option of replicating the GFFS data, in which case a failure of one host site should not cause data to become unavailable. (The replication feature is described in [L3D §5.2.2] and [L3D 5.3.12] both

of which reference a paper by Valente and Grimshaw.[3]) If the collection manager does not choose to replicate the virtual file system export, then a failure at the host site will result in a fault when any user attempts to read from or write to the data collection.

- QAS-DM-1.2 (multi-user access availability) is satisfied because all of the interfaces described above are multi-user and designed to support high numbers (hundreds to thousands) of simultaneous users. Again, this is predicated on the design of the service that hosts the data collection, which is beyond the scope of the XSEDE system itself. If the host service can not support parallel use by many users, the XSEDE system will not be able to provide it either, but the interfaces will fail gracefully and report temporary unavailability of the resource.
- QAS-DM-1.3 (no collection- or project-specific software) is satisfied because all of the interfaces described above are the same for all data collections and projects supported by XSEDE.

[3] Valente, S. and Grimshaw, A., Replicated Grid Resources. 12th IEEE/ACM International Conference on Grid Computing, 2011, 198-206.

3.2 DM-2 Coordinated computation and analysis involving a shared body of data

Use case DM-2 is a specialized application of DM-1, in which the collection users perform steps 5-8 of DM-1 using XSEDE computation services as the “local system” where local copies are made, where data analysis and/or manipulation is performed, and from where data is eventually moved to the shared collection. In fact, the use case presents two iterations of users doing exactly the same thing: accessing the collection data, performing some analysis on a portion of the data, and adding some new (derived) data to the collection.

The critical “new” requirement for DM-2 is that all of the “local system” interfaces and components described in DM-1 must also be available on XSEDE compute resources.

- For the “upload/download”-styled interface, this means that XSEDE compute resources must provide a data transfer endpoint that can be used to get data to/from the compute resource. As long as there is a data transfer endpoint defined for any filesystem accessible from the compute service (e.g., a “scratch” system), the collection user can follow the procedure described in DM-1 to move data from the data collection host to the compute system he/she plans to use, then execute the compute jobs on the compute system, then again use the procedure in DM-1 to move the resulting data to the data collection host.
- For the virtual file system interface, this means that the XSEDE compute resources must provide the Genesis II client software so that the collection user can login to a “head’ or “interactive” node on the compute system [UCCAN-4], mount GFFS on the head/interactive node, move data from the collection to the compute system’s local filesystem (e.g., scratch space), execute the necessary compute jobs, then move the resulting data back to the data collection once the computation is complete.

The two quality attributes for this use case are the same as QAS-DM1-1.1 and QAS-DM-1.2, so they are satisfied in the same ways described in §3.1.1 and §3.1.2.

Note that another interpretation of this use case is that Execution Management Services, i.e., remote execution services, can access the collection data. This use case is described in both Canonical Use Case 1 and in Campus Bridging Use Case 4. Note that the EMS services permit either or both of staging data in/out using protocols that include http(s), GridFTP, scp, ftp, and the GFFS.

3.2.1 Variation 1 - User portal and science gateway integration

If the collection users access the data collection and XSEDE compute services via a science gateway, the use case becomes identical to SGW-2, in which the gateway transfers data to/from an XSEDE compute resource on behalf of a gateway user. The architectural response for SGW-2 is provided in [UCSGW], and that response covers both the upload/download and the virtual file system interface styles.

3.2.2 Variation 2 – Open (unrestricted) access

The second variation to DM-2 specifies that some of the data collection may be made available without any access restriction at all: that is, unrestricted. (It does not, however, say that access must be unauthenticated or that the user doesn't need to provide his/her identity.) Thus, the key change to the use case is that regardless of which user identity the end user authenticates to, access must be allowed. The collection manager should set access control on the data collection to allow any authenticated user to read the data. This will allow any user who has registered with XSEDE to read the data.

More concretely, there are three GFFS use cases: first where "everyone" should be allowed to read the data, one in which XSEDE users (every XSEDE user is a member of the *gffs-users* group), and the case where the community has its own group, "Community" can access the data.

```
chmod -R  /community/Bio/DataBases/Fun-DB  +r --everyone
chmod -R  /community/Bio/DataBases/Fun-DB  +r /groups/xsede.org/gffs-users.
chmod -R /community/Bio/DataBases/Fun-DB +w /groups/xsede.org/Community
```

3.3 DM-3 Shared use of large-scale/streaming sensor input data

This use case is very similar to DM-2, and, like it, builds on DM-1. The first unique addition in DM-3 is that new data is periodically added to the collection from a sensor input source via a new actor: the “input agent,” which may be automated rather than human-operated. (This occurs in Step 5.) A second addition is that analysis and/or processing jobs, as well as data, are shared by the collection users. (This occurs in Steps 4 and 6.)

Our prior experiences with streaming sensor data in a scientific context lead us to expect the following caveats to the scenario described in this use case.

1. It seems unlikely to us that a live experimental laboratory setup would allow for or require a direct connection between the experiment’s sensors and a (remote) XSEDE data service. Instead, we expect that the sensor data will be captured locally within the laboratory, and then--only as a secondary stage--used to feed data to external systems such as XSEDE. In fact, the use case description alludes to this in the “actors” section by defining separate actors for the sensors, the input gateway, and the input agent. This implies that the laboratory has the ability to buffer data locally before it is fed to the data stream that feeds the data collection.
2. It also seems unlikely to us that the raw data from the sensors will be needed instantaneously in the shared data collection. In our experience, human operators who are empowered to make decisions about the validity of the experiment and its data are nearly always in the loop, and are expected to “certify” or “validate” the data at least minimally prior to it being fed into the community data collection. This is not mentioned explicitly in the use case description, so it is an assumption that we bring to this response. This implies both a time delay between data generation and data streaming, and a “human in the loop” who has executive power to stop, start, and perhaps even edit the data stream.
3. We also note that simulation scenarios (where the “experiment” is simulated by code executing on a computer rather than actually happening in live space) typically have the same or analogous caveats as the two noted above.

Given the second implication--that there is a “human in the loop” who controls the data stream--we envision that the XSEDE user identity used for authentication (see DM-1) will be the identity belonging to the human who operates the “input agent.” The input agent should be an application running on the input gateway system, where the sensor data is buffered and stored locally prior to being added to the data stream. This input agent should use the same architecture and procedures described in DM-1 and DM-2 to authenticate to XSEDE and add data to the community data collection, under supervision by one or more human operators.

To be explicit, we do not expect XSEDE to have to support unattended, continuous data feeds over long periods of time (weeks or months) with no human involvement. We will address use cases with that requirement separately, should they be prioritized by XSEDE’s management.

With regard to sharing the code for data processing and analysis throughout the community, we propose that this is essentially the same as the science gateway use cases and that we use the architectural response to those use cases to cover this scenario.

Specifically, SGW-3: Community Execution Management [UCSGW §3.3] describes two ways to use XSEDE's architecture to share computational capabilities (including code and specific analysis processes) with a community of users. Combined with DM-2 (§3.2), the community should be able to share both a common base of code and a common data collection on XSEDE systems. Note that SGW-3 assumes that the community will be running these processing/analysis jobs on XSEDE computation resources. If the community needs to (or chooses to) use non-XSEDE resources for these tasks, then we would refer back to DM-1 for the basic procedure for enabling access to the data collection on users' local systems and leave it to the community to solve the shared code issue.

3.4 DM-4 Migration of data between resources

This use case is quite different from DM-1 through DM-3, and considerably simpler. It deals with the scenario where one or more users has created data on an XSEDE resource, and now the data must be moved to a different XSEDE resource. The use case description is quite clear that this is to be an automated action with minimal user involvement. Once the user identifies the data to be moved and the destination, the action should be "fire and forget." The XSEDE system is expected to ensure that all of the data is successfully and accurately moved.

With the exception of the quality attributes and variations, this is precisely the same use case as UCCAN-2, Managed File Transfer [UCCAN-2]. For the basic use case, we refer to the architectural response to UCCAN-2.

QAS-DM-4.1, which specifies that the user should only have to initiate a transfer once per file system (or equivalent storage resource) is satisfied by the design of XSEDE's managed file transfer functions, described in UCCAN-2. The transfer service actively monitors all transfers and handles transient failures automatically, retrying individual files until success is achieved. Non-transient failures such as permission issues, endpoint issues, etc. are generally detected as soon as the transfer begins and reported to the user and administrators before anything happens.

QAS-DM-4.2 states that 100% of the data must be transferred successfully, with verification. UCCAN-2 also requires this, and the verification techniques used are described in [UCCAN-2].

QAS-DM-4.3 is essentially a combined restatement of 4.1 and 4.2.

The GFFS provides the ability for users to replicate directory trees onto resource containers. Once the replication is complete the original replicant can be eliminated. Users of the replicated resource will not be aware that the replication is in process or that the original replicant has

been removed. At most they will see a slight pause during data transfer or directory lookup operations.

For example: suppose *newfile1* is a replicated file

resolver -p */community/Bio/DataBases/Fun-DB* path-to-site2

replicate -p */community/Bio/DataBases/Fun-DB* path-to-site2

replicate -l newfile

will list all of the replicas, their ID number, the DNS address of the container where the replica is located, and the container ID.

```
Genesis II:\$> replicate -l newfile1
Replica 1: https://10.0.2.15:18080: container-id=581BEB33-A83B-325E-6C40-44055AAB212E
Replica 3: https://10.0.2.15:18080: container-id=581BEB33-A83B-325E-6C40-44055AAB212E
Replica 4: https://10.0.2.15:18080: container-id=581BEB33-A83B-325E-6C40-44055AAB212E
```

All three of the above replicas are in the same container in this example, but in general they will be in different locations.

replicate -d newfile 3
will destroy the replica with the replica ID of 3.

3.4.1 Variation 1 - Simultaneous use at source and destination

Variation 1, in which the data is used in both the old and the new location for a period of time, with synchronization required, is not covered by UCCAN-2, it is covered by the GFFS replication techniques described in UCCAN-3 and as described above in DM-1 and DM-3.

As described earlier the GFFS supports replication as well as simultaneous update by multiple clients in multiple locations. If the same file is being updated simultaneously by multiple clients all replica's will eventually become consistent.

3.4.2 Variation 2 - Administrator-initiated migration

Variation 2 changes the actor from an end user to a system administrator or an automated administrative process. With regard to this variation, it is important to point out that the XSEDE architecture is designed around the needs of end users, not system administrators. Consequently, the XSEDE architecture doesn't provide a straightforward solution to this scenario. For cases where the source and destination system are in the same data center (which we expect is the most frequent case), system administrators are unlikely to look to XSEDE to provide the solution, preferring instead to use their own local tools and techniques. Cases where the source and destination are at different data centers seem to happen rarely (once a year or less), and usually they happen in the context of a new system construction project. In these cases, it's likely that extra funds are explicitly included in the budget for data migration.

3.5 DM-5 Metadata storage and access

This use case depends a bit on the base data management use case, DM-1, but it mostly introduces an entirely new set of functionality related to metadata, or "data the describes the data."

The XSEDE architecture provides two features that support the addition of metadata to data collections. The first is the XSEDE architecture's **identity management services,** which provide a sound basis on which to build a service in which the identity of the end user is important (as it is in data collection curation activities).

The second feature is the WSRF-RP interface, available in X-WAVE components such as GFFS, UNICORE, and Genesis II. As described in the L3D 4.1.7 all GFFS resources (files and directories) support the *WSRF Resource Properties* interface. The WSRF-RP interface supports *getResourceProperties* and *setResourceProperties* functions as well as the *queryResourceProperties* functions. All of these functions operate on arbitrary XML data data. For example, a file could have resource properties set that describe the software and version that was used to generate the data file, the parameters to the software, and the data sets that were used.

The resource properties primitives can be used directly, e.g., the grid shell getResourceProperties call, or the grid shell "find" command that recursively descends a GFFS directory structure looking for objects with resource properties that match the query. One can also use mechanism described in [UCCAN-8-12]. Specifically one can construct repositories that can be queried directly by users and applications. The repositories can keep up-to-date data via either a push or pull model. In a pull model the repository regularly queries objects about their meta-data. In a push model objects push updates (publish) to the repository.

Another example of a data curation system built on XSEDE's architecture is the Globus data publication service.[4] Globus data publication is a subscription service that any XSEDE service provider or academic/research institution may subscribe to, and is tightly integrated with the identity management and data management services used by XSEDE. Of course, XSEDE's open architecture allows others to develop similar curation services.

[4] https://www.globus.org/data-publication.