5 Remote Storage Resource Management in WS-PGRADE/gUSE

Ákos Hajnal, Zoltán Farkas, Péter Kacsuk, and Tamás Pintér

Abstract. State-of-the-art gateways are connected to several distributed computing infrastructures (DCIs) that are able to run jobs and workflows simultaneously in all those different DCIs. Data of e-Science applications might be stored on different storage resources, making it difficult for various user communities to browse, access, or update it. This chapter presents the Data Avenue concept and related tools that aim at hiding technical details of accessing different storage resources, and providing an easy-to-use file browsing interface to manage, upload and download, and even transfer data between different types of storage resources. Usage scenarios to exploit Data Avenue services as well as security considerations are discussed.

5.1 Introduction

Numerous storage solutions have been evolved during the past decades, which are actively used today in distributed computing environments. Besides the conventional HTTP/FTP servers, storage resources accessible over Secure File Transfer Protocol (SFTP), GridFTP (GSIFTP) [GSIFTP] protocols, Storage Resource Management (SRM) systems [SRM], cloud-related Simple Storage Services (S3) [S3], as well as logical file systems such as LCG File Catalogs [LFC], and integrated Rule-Oriented Data-management Systems (iRODS) [IRODS] can potentially be used from within distributed computing infrastructures such as grids or clusters.

Each storage solution has been developed to serve specific goals by fulfilling specific criteria. However, accessing these storage resources typically requires dedicated protocols and tools to allow users to organize, manage, upload or download data via a graphical or command line interface. Selecting a suitable storage resource for our requirements thus involves setting up the appropriate software environment, and learning how to use the related tools.

Various user communities wish to exploit the computational power of distributed infrastructures. It generally cannot be expected that these communities – including agronomists, biologists, and astronomers – have deep IT expertise, as technical details of the underlying infrastructure are out of the domain of their main interest. Providing user-friendly, graphical user interfaces is thus of high practical importance. Most storage resources and tools provided today, however,

only partly fulfill this criterion, which makes wider use of storage resources difficult or impossible.

As technology evolves or requirements change, it may become necessary to move our existing data from one storage to another. Although there exist tools that can connect our local machines to a particular storage resource (e.g., GridFTP GUI [GridFTP GUI], DragonDisk [DragonDisk], Cyberduck [Cyberduck]) as well as tools capable of transferring data between storage resources of the same type (Globus Online [GlobusOnline], Transmit [Transmit]), it is generally an unsolved issue to migrate data between storage resources of different types. Downloading data to our local machine, then uploading to the new storage is often not feasible (due to disk capacity or file size limits), therefore there is a practical need for a tool that enables transferring of data between different storage resources.

In a distributed computation infrastructure, computing elements require the appropriate application programming interfaces (APIs) to access remote storage resources, to fetch input data, and store computational outputs, respectively. Such an API for the selected storage resource may not be available on the computing elements in the given infrastructure by default. Writing code that is only required to access storage resources into application logic is generally avoidable; the requirements of preinstallation of the storage-access libraries limit the portability of the application. Therefore, providing a solution that allows uniform access to different storage resources in a way that they are available in any distributed infrastructure is of high importance.

This chapter proposes a solution called Data Avenue to address the problems above. Data Avenue provides a web-based, intuitive graphical user interface, which requires no software installation or particular learning to use. It completely hides the technical details of connecting to a particular storage resource, and provides a uniform rendering of the set of data (files and folders), which allows users to easily manage, organize, upload and download data. Data Avenue is capable of performing data migration from one storage to another without additional effort; and finally, using “HTTP tunneling” provided by Data Avenue, computing elements can access various storage resources simply using the HTTP protocol, which available in all infrastructures.

The chapter is organized as follows. Section 5.2 presents the Data Avenue concept and its main components. Section 5.3 describes how Data Avenue is used in WS-PGRADE/gUSE. Section 5.4 discusses security concerns. Finally, Sect. 5.5 concludes the chapter.

5.2 Data Avenue

Data Avenue was designed to separate the core functionalities from the graphical user interface. Storage resources are directly managed by a component called Data Avenue Web Services, referred to as the “Blacktop” for short, whose services are used by the Data Avenue User Interface component, called the “UI” for short. The Blacktop is a web application providing a web service interface

(SOAP/REST) to which clients – including the UI – can connect. The UI has been implemented as a portlet deployable in any portlet container, and it provides a web-based, graphical interface for the users. Therefore the designed architecture, depicted in Fig. 5.1, offers a number of use-cases to exploit the Data Avenue services.

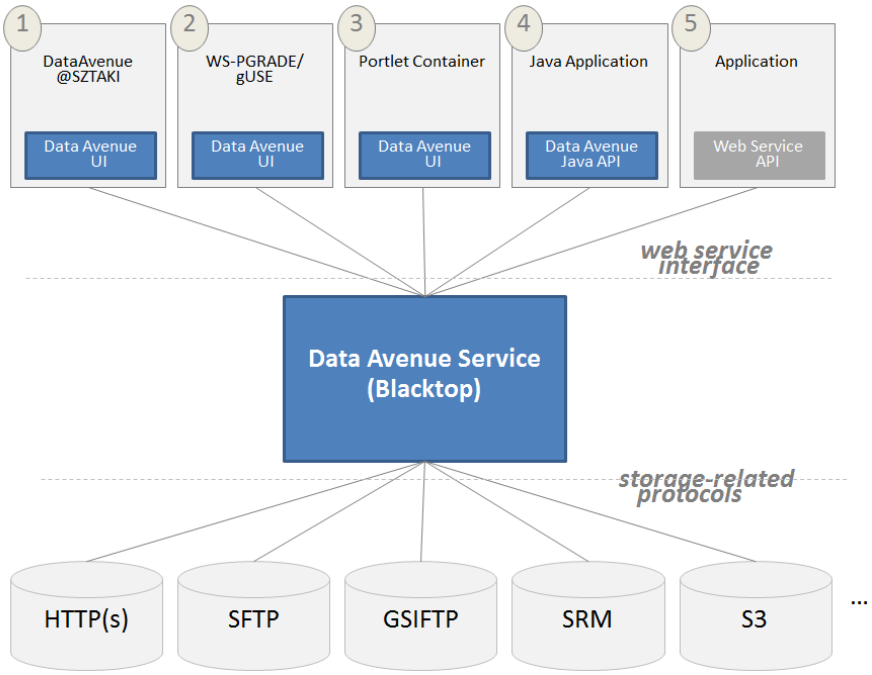

Fig. 5.1 Data Avenue overall architecture and use-cases

1. Data Avenue @SZTAKI is a public deployment of Data Avenue UI, hosted at MTA SZTAKI, available as a public web page, where users can freely try out Data Avenue services.
2. As of release 3.6.0, Data Avenue UI is part of the WS-PGRADE/gUSE portlet set; this way, remote resources can also be managed from within the WS-PGRADE/gUSE environment.
3. Data Avenue UI can be deployed in any portlet container and exploit Data Avenue services through the Blacktop.
4. Using the Java API library provided for Data Avenue, Java applications can easily use Data Avenue services.
5. Finally, applications written in programming languages other than Java can also access Data Avenue services via standard web service calls, using the appropriate web service API library available in that environment.

In the following sections, first the user interface is presented, and then design concepts of the Blacktop are described in more detail.

5.2.1 Data Avenue User Interface (UI)

The graphical user interface of Data Avenue is implemented as a portlet, which can be deployed in any portlet container (such as Liferay). It provides a web-based interface for the users; in this way, the usage of Data Avenue services requires no software installation other than a web-browser.

Data Avenue performs a uniform rendering of the files and directory structure regardless of the particular technology used by the remote storage resource we browse. The interface layout (Fig. 5.2) is divided into two panels.

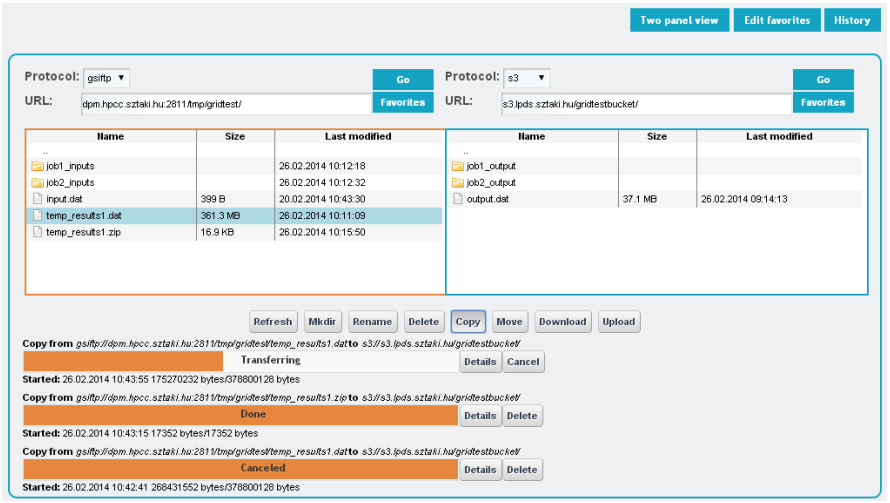

Fig. 5.2 Data Avenue graphical user interface

To access a storage, the user is only required to select the storage resource type (protocol), specify its host (URL), and if necessary, fill the authentication data, called *credentials*, e.g., username–password, proxy, access key–secret key. (Optionally, the host name may be followed by an initial working path.) At the time of this writing, storages accessible over HTTP, HTTPS, SFTP, GSIFTP, SRM, and S3 are supported.

Once authenticated, the list of files and subdirectory entries are listed on the active panel (indicated by a lighter border line) along with details showing size and last modification date information. The actual working path (URL) can be changed by navigating into a subdirectory (double clicking on it), or to parent directory (“..”), respectively. New subdirectories can be created (*Mkdir*), and any files or directories can be renamed (*Rename*) or deleted (*Delete*). Files can be uploaded from the local disk (*Upload*), or downloaded to it (*Download*), respectively. The list of directory contents can also be updated (*Refresh*).

To transfer files or directories from a *source* storage resource to a *target* storage resource, one panel should be connected to the source storage, and the other to the target one. After selecting a file or a directory on the source panel, it can be

copied (*Copy*) or moved (*Move*) to the target storage. Such transfers are performed simultaneously, and progress statuses are displayed below the panels (these tasks will keep running, even if the user leaves the web page). Active transfers can be aborted (*Cancel*), and status details of previous transfers can be queried (*Details*) back for a specific period of time (*History*). History entries not yet of interest can be removed (*Delete*).

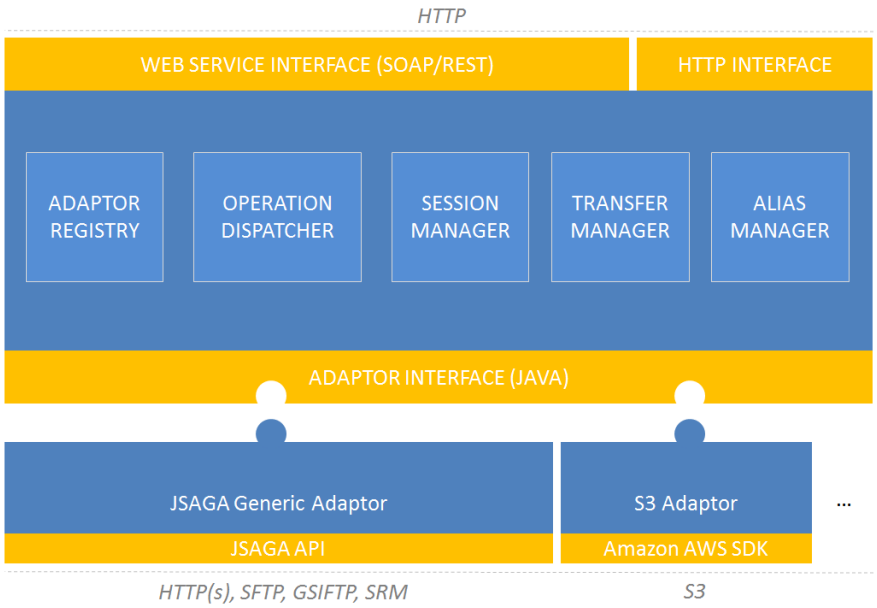

Fig. 5.3 Blacktop architecture

It is also possible to save preferred storage locations as "Favorites". Favorites store additional, not security-sensitive data (such as username, VO name, etc.), which help in accessing frequently visited storages more rapidly. Favorite details can also be modified later (*Edit favorites*).

As can be seen from the foregoing, the Data Avenue UI provides an intuitive interface for users to easily manage, organize, upload, or download data from remote storage resources. Use of the web-based graphical user interface requires no software installation or IT expertise.

5.2.2 Data Avenue Web Services (Blacktop)

The core functionality of Data Avenue services is realized by the Blacktop component. Blacktop services are publicly available through a standard web service interface over HTTP (via SOAP messages or REST), hosted at MTA SZTAKI.

Web service operations – whose functionalities are reflected in the user interface presented in the previous section – include directory contents listing (*list*), di-

rectory creation (*mkdir*), directory deletion (*rmdir*), file deletion (*delete*), and file or directory renaming (*rename*). In addition, meta-information retrieval operations are available to query what types of storage resources are supported by the Blacktop, what credentials are needed to authenticate to a given storage resource, and what sort of operations are available on the selected storage (e.g., read, write, directory creation, etc.).

The Blacktop was designed as an extendable plugin architecture, where plugins are aware of communicating with the particular storage resources using the appropriate storage-specific protocols. These adaptors are connected to the Blacktop through an Adaptor interface as shown in Fig.5.3.

At the time of this writing, two adaptors have been implemented and connected to the Blacktop: *JSAGA Generic Adaptor* and *S3 Adaptor*. JSAGA Generic Adaptor is based on JSAGA API ([JSAGA]), which is a Java implementation of the Simple API for Grid Applications (SAGA) specification from the Open Grid Forum. This adaptor is used by the Blacktop to connect to HTTP(s), SFTP, GSIFTP, and SRM storage resources. S3 Adaptor uses Amazon's AWS SDK to connect to Simple Storage Services (Amazon, Ceph Object Gateways) over the S3 protocol, by which Data Avenue can connect to cloud storage resources as well.

An adaptor implementation must be developed according to the Data Avenue Adaptor interface to be able to be attached by the Blacktop, which consists of a set of low-level storage operations such as *list*, *mkdir*, *delete*, *rename*, *read resource stream*, *write resource stream*, etc. It should be noted that not all operations can be performed on every storage, e.g., writing is not possible on HTTP storages; therefore adaptor implementations are limited to the relevant operations only. The Blacktop seeks for adaptor implementations on startup, and registers adaptors in the *Adaptor Registry*. Adaptors are associated with the protocols they can handle; one adaptor can support more than one protocol or storage resource, respectively.

Web service operation requests sent to the Blacktop use Uniform Resource Identifiers (URIs) to refer to storage resources (files or folders). These resources are of the form: `protocol://host/path`, where *protocol* specifies the protocol used to communicate with the storage resource, *host* specifies the location of the storage, and *path* specifies the location where the file or a folder resides on the storage. For example: `gsiftp://gridftp.host.net/home/user/file.dat` refers to a file named `file.dat`, in the subdirectory `/home/user/` on host `gridftp.host.net` accessible over GridFTP protocol. Based on the protocol (scheme) part of the URI, the Blacktop can choose the appropriate adaptor in the Adaptor registry, and dispatch the operation request. This is performed by the *Operation dispatcher* component.

Connecting to a storage resource for the first time can be expensive due to a number of required initial activities (e.g., creating clients, opening ports, creating VOMS proxies, etc.). This cost can be avoided in subsequent accesses using *sessions*. Blacktop creates a *session container* for each "client session" to be passed to the adaptor, which can store any objects related to the current session, to be reused in subsequent calls. The Blacktop ensures that the same session container is

passed to the adaptor on subsequent requests from the same client, and also ensures that objects within the container are properly closed on session timeout before being discarded. These tasks are performed by the *Session manager* component.

In contrast to *synchronous* operations (such as *list*, *mkdir*, *rmdir*, *delete*, *rename*), which are known to be completed when the operation response is returned, the execution of *copy* and *move* operations may take a longer time (even hours, which likely exceeds client sessions). Such *asynchronous* transfers are performed simultaneously, and for this reason, copy/move operations return a reference (*id*) of the related transfer just started. This id can be used to poll (*getState*) progress (bytes transferred) or state details information (transferring, done, failed) about the task, or abort it (*cancel*), respectively. The Blacktop maintains, creates, and passes a so-called *transfer monitor* object (*listener*) for each transfer to the corresponding adaptor, through which progress and status information can be updated by the related adaptors. Transfer monitors are managed by the *Transfer manager* component.

An adaptor may declare to support copy operations itself between the handled storage resources (e.g., an adaptor can exploit third-party transfer between GridFTP servers or S3 regions). In this case, such a copy task is entirely delegated by the Blacktop to the adaptor for optimum performance. In other cases, however, Blacktop performs copy or move operations via streaming, by reading and writing resource input and output streams (e.g., copy between storage resources managed by different adaptors).

File uploads and downloads are performed via so-called *aliases*. Aliases are unique, temporary HTTP URLs created on the Blacktop host, which can be read (HTTP GET) or written (HTTP PUT or POST) over HTTP protocol. This interface is represented by the HTTP INTERFACE box in Fig. 5.3. On upload or download, the Blacktop is first requested to create an alias (*createAlias*) for a remote file residing on some remote storage; later on, client's read/write operations from/to the HTTP alias URL are translated automatically to actual storage read/write operations by the Blacktop. This technique is also known as HTTP tunneling. Aliases can be created for reading or writing. An alias URL thus indeed behaves as an "alias" of the remote resource.

Aliases can be created with an arbitrary *lifetime*. In this way, storage resources can also be shared over HTTP for a longer time (days, weeks, or months), or can be made accessible for other parties who are not aware of the particular storage resource protocol (e.g., computing elements having HTTP capability only). Aliases are managed and made persistent by the *Alias manager* component.

Different storages use different authentication mechanisms. The most frequent authentication types are: username-password authentication (SFTP), access key–secret key (S3), x509 proxy-based authentication (with or without VOMS extension, used by GridFTP or SRM). Such authentication data are called credentials that need to be passed to the Blacktop service to be able to connect to the related storage resources. Due to the session management of the Blacktop, these creden-

tials have to be passed in the very first operation and are used as long as the session does not expire.

When creating aliases, credentials that are needed to access a remote file or a folder must be given to the Blacktop. These credentials will be used throughout the whole lifecycle of the alias each time a request comes over the HTTP alias. Note that clients using an alias (potentially different from the client who had created them) are not required to be aware of the credentials needed to access the storage resource.

5.3 Data Avenue in WS-PGRADE/gUSE

Data Avenue can be used in two levels in WS-PGRADE/gUSE. As mentioned earlier, starting from the WS-PGRADE/gUSE release 3.6.0, Data Avenue UI is part of the WS-PGRADE/gUSE portlet set. The Data Avenue UI portlet is available through the menu item “Data Avenue”; portal users can thus manage remote storage resources, upload or download files within the WS-PGRADE/gUSE environment in the same way as described in Sect. 5.2.1, and as shown in Fig. 5.4.

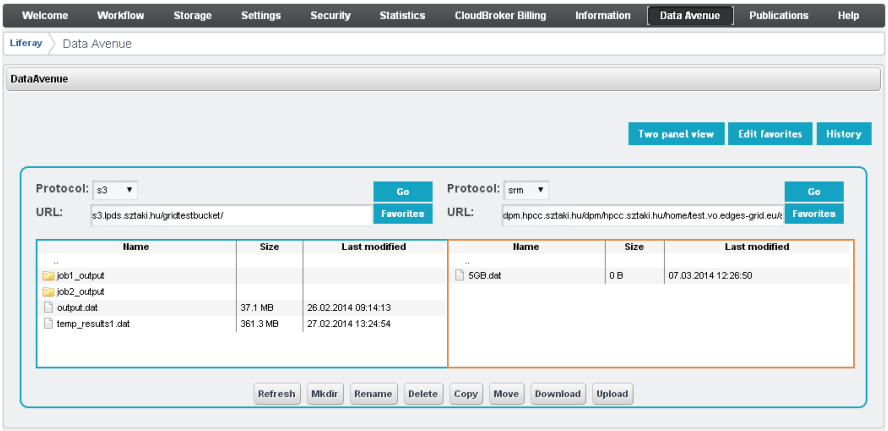

Figure 5.4 Data Avenue in WS-PGRADE

Beyond this feature, WS-PGRADE/gUSE also makes it possible to exploit remote storage resources during workflow execution. This feature is described in more detail in the following paragraphs.

When designing workflows, so-called *ports* can be created for jobs that allow one to specify the inputs for individual computations, and to give name or location for the outputs, respectively. There are numerous options for determining the *type* of such ports. The *source* of an input port can be a local file or a remote file accessible over HTTP, cloud, grid, etc., protocols; similarly, there are several options for specifying the way of storing the results at the output ports of the workflow nodes. In order to use remote storage resources in workflows a so-called Data Avenue port type is available, which makes it possible to set remote input and output

files for jobs. Figure 5.5 shows such a workflow, where the workflow nodes run in GT5 and gLite grids and communicate via a S3 cloud storage. All the ports are Data Avenue port types enabling access to an SFTP, an S3 and a GridFTP storage at runtime.

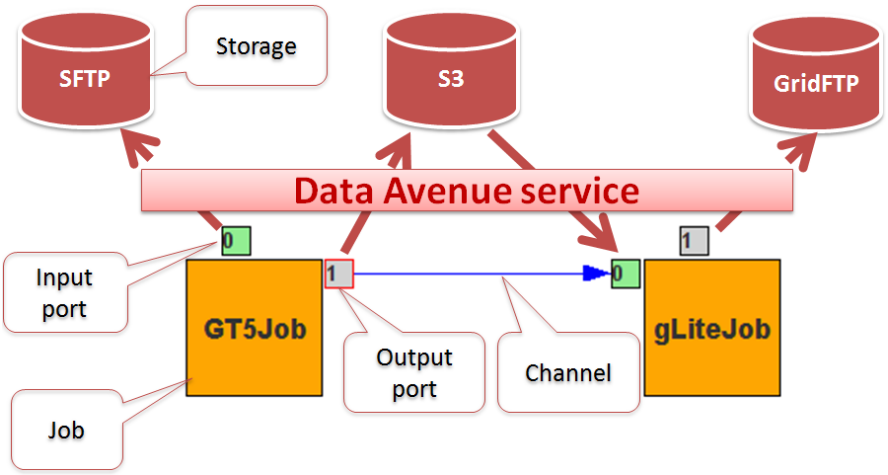

Fig. 5.5 Use of various storages from WS-PGRADE workflows by applying Data Avenue service

Figure 5.6 shows how such a Data Avenue port type can be defined. Inside the Job I/O window users can select Data Avenue as a port type (see bold arrow in the figure), and then a remote file can be chosen in a file browser window, which is a one-panel variant of the Data Avenue UI (without function buttons). Accessing and browsing remote storage resources can be performed in the same way as described in Sect. 5.2.1, which requires selecting storage type and host, and entering authentication data. When the configuration of a job has been completed and all remote files are assigned to the relevant ports, the URI of the selected remote files along with references of the necessary credential information (used to access the storage) are recorded in the workflow.

WS-PGRADE/gUSE automatically resolves remote file references during workflow execution and provides the relevant remote files for jobs transparently. Depending on whether the job is executed via a so-called *wrapper script*, one of following two mechanisms ensures fetching inputs and storing outputs:

1. If the job cannot be executed by a wrapper (e.g., on Portable Batch System clusters), the DCI Bridge downloads the necessary remote files first using the Data Avenue API. These are then passed to the computing element before job execution. Once the job completes and the results are passed back to the DCI Bridge, outputs are uploaded to the specified location by the DCI Bridge using Blacktop services.

Fig. 5.6 Workflow input selection using Data Avenue

2. If it is possible to use a wrapper for job execution (e.g., on clouds), DCI Bridge merely creates HTTP aliases for the appropriate remote storage resources, and these alias URLs are delegated along with the wrapper script to the computing element. The wrapper script first downloads the necessary input files via the provided URLs, executes the job, and then uploads the results to the output URLs. The advantage of this solution over (1) is that it lowers

the CPU load of the DCI Bridge, and also the network traffic, as remote files are transferred once, directly to the location of use.

Credentials required to access remote storage resources are not stored directly in the workflows. Instead, only references to credentials are recorded allowing safe publication and sharing of workflows − without compromising confidential authentication data. On workflow export, only credential references are saved, and these references need to be specified. The credential references are then resolved when the workflow is imported by another user, likely owning different authentication data.

It is also possible to use robot certificates for Data Avenue port types. These certificates are credentials defined with a global-scope within a portal instance. Robot certificates are set by workflow developers in advance, and portal users, in this way, are not required to have individual credentials to access storage or computing resources (authorization is granted to all portal users automatically).

Parameter-sweep applications are performed in slightly different way for performance reasons. Generator jobs generate a number of files that will serve as input for consecutive job(s). As there may be possibly thousands of files created by generators, these files are zipped first, resulting in an archive which is uploaded as a single tar.gz file via Data Avenue. Such uploads are performed using a special "archive" attribute; this mode instructs Data Avenue to extract the uploaded file on-the-fly, i.e., the extracted content will be stored on the remote storage resource. On such uploads Data Avenue returns the number of entries contained by the archive; this number is used later by the workflow interpreter to determine the number of jobs to be launched. The workflow interpreter then starts the required number of parallel jobs, providing the relevant entry (file) available expanded on the remote storage resource.

Notice that combining this technology with the coarse-grained interoperability solution of the SHIWA and ER-Flow projects, users can transfer remote data among different workflows like ASKALON, Kepler, MOTEUR, Taverna, etc., no matter in which DCI they are actually executed.

5.4 Security Considerations

As with using any mediation service, a number of security concerns may arise. This section describes the measures implemented in Data Avenue to preserve confidentiality of users' security-sensitive data.

In order to access Data Avenue services, clients must possess and pass a valid "ticket" in each request to be sent to the Blacktop. A ticket is a simple access code (similar to API key used by Google), valid for a specific period of time, which can be requested on the Data Avenue web site by filling out a simple form (asking for some basic information about the user such as name and e-mail address). Note that tickets, which "merely" allow of using Blacktop services, differ from credentials that are needed to access a particular storage resource. Tickets basically serve to

prevent anonymous overloading of the Blacktop, but also make it possible to trace back the history of user operations, if needed.

Data Avenue services are available through HTTPS connection, which ensures confidentiality of data passed through this channel, and is protected against eavesdropping. Besides that, the Blacktop itself is hosted in a secured infrastructure (behind firewalls, etc.). On the Blacktop side, credentials are stored in the system memory (not written to disk), and kept only for the time of the client session (after a specific period of time of inactivity, they are discarded).

Credentials related to aliases, which require authentication data to persist for their whole lifetimes, must be stored in a database. In addition to ordinary measures of protecting databases, Data Avenue stores credentials in the database in encrypted form. The encryption key is set by the system administrator on Blacktop startup, and using this key, the Blacktop can automatically encode/decode credentials when writing to/reading from the database. The security of the communication between the Blacktop and different storage resources is ensured by the protocol used to access the storage.

As with any clients, Data Avenue UI also requires a ticket to connect to the Blacktop. When the UI is used within portals – having possibly hundreds of users – portal users are not required to request tickets individually. Instead, a single "portal ticket" is requested by the system administrator for the portal, who sets this ticket in the portlet settings (preferences) once. To allow fine-grained, user-level access to the Data Avenue services, for each user registered within the portal wishing to use the UI, a so called "portal user ticket" is obtained automatically, which is used afterwards in every interaction between the portal user and the Blacktop. This solution avoids individual user ticket administration, and still ensures detailed access control.

Use of the Data Avenue Web Services through direct web service calls, or the Java API provided, respectively, also requires a ticket, called an "API ticket". This API ticket can also be requested on Data Avenue's web site.

5.5 Conclusions

Numerous storage solutions have been evolved during the past decades; however, accessing and managing data typically still requires IT expertise, installation, and learning the related tools. These kinds of skills often cannot be expected from every user community wishing to exploit the computation power of distributed computing infrastructures. Also, the requirements of dedicated APIs that make it possible to access different storage resources often prohibit the deployment of user applications, or, at least, limit their portability.

Data Avenue offers a flexible and extensible solution for these problems. It provides an intuitive, web-based graphical user interface, through which users can easily perform their most common operations on storage resources, and a Blacktop server, which provides a web service interface to perform operations on remote storage resources via simple web services. In this way, Data Avenue services are

accessible in various ways from portlet containers to custom applications written in any programming language.

Data Avenue has also been integrated into the WS-PGRADE/gUSE portal framework. On one hand, Data Avenue's user interface is available within the portal, and on the other hand, workflow execution is extended in a way that jobs can access remote input files, or upload their outputs. All operations required to manage remote storage resources are performed by the portal automatically, and are transparent for the users. Combining this technology with the coarse-grained interoperability solution of the SHIWA and ER-Flow projects, users can transfer remote data among different kinds of workflows (Kepler, Taverna, etc.), no matter in which DCI they are actually executed. Finally, extra care is taken on security issues that arise in any mediation service when using Data Avenue.