Data Bridge: solving diverse data access in scientific applications

Zoltán Farkas and Péter Kacsuk and
Ákos Balasko and Krisztián Karóczkai
MTA SZTAKI
Hungary, 1518 Budapest, Pf. 63.
Email: zoltan.farkas@sztaki.mta.hu

Marc Santcroos and Silvia Olabarriaga
AMC
The Netherlands, 1100 DE Amsterdam, PO Box 22700
Email: m.a.santcroos@amc.uva.nl

***Abstract*—The nature of data for scientific computation is very diverse in the age of big data. First, it may be available at a number of locations, e.g. the scientist's machine, some institutional filesystem, a remote service, or some sort of database. Second, the size of the data may vary from a few kilobytes to many terabytes. In order to be available for computation, data has to be transferred to the location where the computation takes place. This requires a diverse set of middleware tools that are compatible both with the data and the compute resources. However, using this tools requires additional knowledge and makes running the experiments an inconvenient task. In this paper we present the Data Bridge, a high-level service that can be used easily in scientific computations to perform data transfer to and from a diverse set of storage services. The Data Bridge not only unifies access to different types of storage services, but it can also be used at different levels (e.g., single jobs, parameter sweeps, scientific workflows) in scientific computations.**

I. Introduction

There are many different distributed computing infrastructures (DCIs) that users would like to access from scientific workflows and science gateways. In order to hide the different APIs needed to access these very different DCIs such as grids, clouds, clusters, desktop grids, and supercomputers, we have developed the DCI Bridge service [1] and connected it to the WS-PGRADE/gUSE science gateway framework. As a result, application-oriented science gateways, which were developed by the customization of the WS-PGRADE/gUSE framework, can currently access all these types of DCIs transparently from the nodes of WS-PGRADE workflows. As a consequence workflows can be ported between DCIs by simply re-configuring them for another infrastructure.

Many of these scientific applications manipulate a large amount of data of different types, for example medical images or DNA sequences. The data is stored on different types of storage systems, like a local file system, a distributed or shared file system, some sort of service catalog, or in a database system. Access to these data might be difficult, as detailed knowledge and specific tools are needed to fetch or upload the data. Although easy-to-use (web) interfaces might be available for the individual systems, using them in combination is required in complex processing systems such as scientific workflows. In order to provide access to a diverse set of storage resources, the system needs to be aware and provide support for data access using the different APIs of each storage service.

Our approach for solving this problem is very similar to the concept of the successful DCI Bridge. In the scope of the SCI-BUS project [2], we designed a new service called the Data Bridge service, which provides a unified interface for accessing different storage services, e.g., HTTP, FTP, GridFTP [3], SRM [4] and Amazon S3 [5]. In the current paper we describe the main concepts and features of this new service.

The organization of the paper is as follows: we first present an overview of related work, and then we gather the requirements towards the system. Afterwards we present the Data Bridge in detail and show how a complex scenario can be supported. Finally, we discuss our preliminary results, and present our current and future work.

II. Related work

OGSA-DAI [6] provides a web service for accessing different data resources such as relational or XML databases, files or web services. The data can not only be queried and updated, but modified and transformed as well. The OGSA-DAI service was used in many different projects.

The Storage Resource Broker (SRB) [7] offers a middleware that provides clients a uniform access to a diverse set of storage resources. Two type of "drivers" are available: file-type drivers (for example UNIX filesystems) and DB-type drivers (for example IBM DB2 and Oracle databases).

The integrated Rule-Oriented Data-management System (iRODS) [8] is a scalable grid software solution for managing files in the order of hundred million and total size in the order of petabytes. It is capable of making use of a number of authentication mechanisms, supports a wide range of physical storages, and has support for metadata attributes.

jSAGA [9], the Java implementation of the Simple API for Grid Applications (SAGA) Open Grid Forum (OGF) specification [10] provides a Java API to access different grid services, including storage services as well. jSAGA provides an easily extensible platform for accessing FTP, GridFTP, iRODS, SRB, LFC service.

Globus Online [11] offers a service for managing data available on GridFTP endpoints. It has a web interface, it is also usable through command-line tools, and it provides a REST API as well. Users of Globus Online can monitor their data transfers, which have automatic error recovery.

The presented tools and services offer varying features, but because of their different scopes, all of them have their weaknesses outside of their intented purposes, for example, they only support a limited variety of storage services (OGSA-DAI, SRB, Globus Online), do not provide an API (iRODS), are not available as a service (jSAGA), or are based on a too heavy software stack (OGSA-DAI, SRB). Our aim with the Data Bridge is to mix the advantages of these tools in a lightweight service.

III. Requirements towards the system

In this section we present some use cases for the Data Bridge service. The three main considerations are the following: easy-to-use user interface to browse and manage data stored on a storage resource; transfer of data between different types of storage resources, and finally data handling from compute resources.

Figure 1 covers the high-level needs of users concerning ease of use: a convenient user interface is necessary for end-users so that they can browse, download and upload data from and to the storage resource. Moreover, users should be enabled to select data for running experiments on the DCI. In case of web portals, such interfaces are typically implemented as a portlet, thus the user interface can be built as a viewing portlet (Storage View) that uses functions exposed by a Data Bridge service to access the various storage resources. The same interface can also be used to upload and download data from and to the user's own machine to and from the selected storage. The advantage of using the Data Bridge service clearly appears if storage resources of multiple types should be accessed: instead of interfacing with multiple storage APIs from the storage browsing graphical interface, the developer of the Storage View component should access only the Data Bridge service.

Figure 2 covers the case where large amount of data needs to be transferred from a given storage resource to another storage service, considering the generic case of Storage services located in different DCIs. This is particularly necessary in case the experiment, or even a single job, runs on an infrastructure that is different from the one where the data is stored on. In such case the data needs to be transferred from the original storage location to the storage location of the compute resource to ensure high data transfer rate during data processing time. In Figure 2, the user is requesting the Data Bridge to perform the copy operation, but this can also be initiated programmatically by the workflow management system, for example. The Data Bridge has access to both Storage 1 and Storage 2, and performs the copy. However, would the Data Bridge have no access to Storage 2, it could make use of an other Data Bridge in a hierarchical organization.

The third use case covers the needs of handling data in jobs: once a job has been started on a compute node, the job may need to access data on a storage service for input or output. The Data Bridge can be used in this case as a unified interface to get or put the data. As shown in Figure 3, a wrapper script fetches input files for the job before the job is actually started, and it uploads the output files of the job to target storage after the job has been executed. Such wrapping methods can be used to make storage-aware legacy applications without the need to modify the application itself. As a consequence, using the Data Bridge with this wrapped execution offers a convenient way to provide access to a diverse set of storage resources for any kind of application.

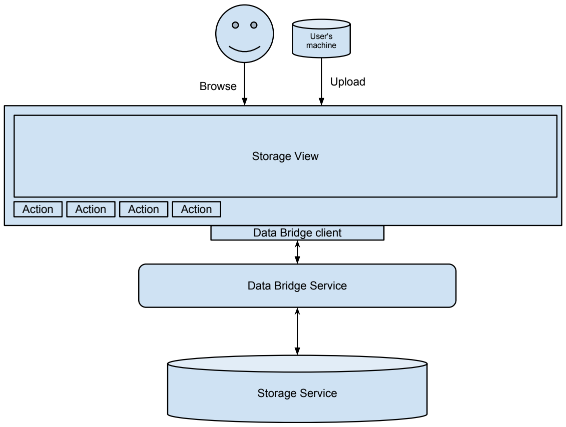

Fig. 1. Storage browsing and data upload

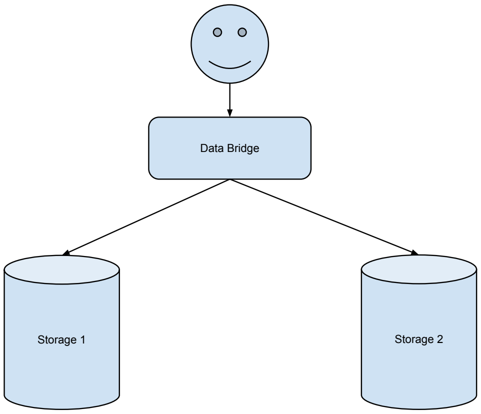

Fig. 2. Transfer between different storage types

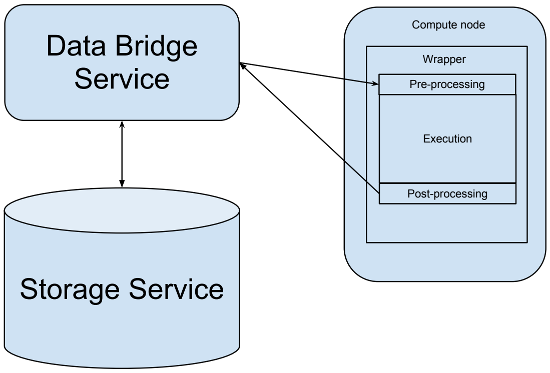

Fig. 3. Using the Data Bridge from a Compute node

IV. Data Bridge

In this section we present the Data Bridge design in detail. First we show the high-level architecture, and then we present the different components in more detail.

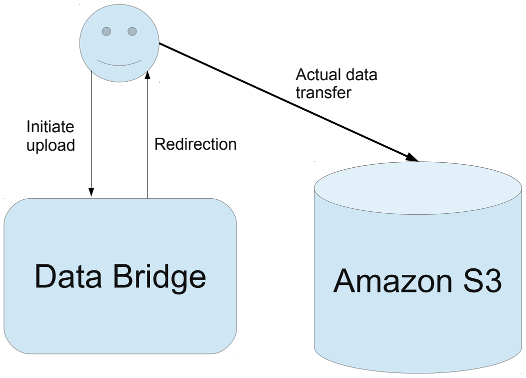

Fig. 4. Data transfer redirection

The Data Bridge itself is implemented as a web application exposing a number of operations for managing the data located on different storage. For the client, two main interfaces are available: one web service interface for initiating operations, and one servlet interface for performing any necessary data up- and download.

Whenever possible, the Data Bridge is making use of data redirection to minimize its network traffic: if the destination storage is able to accept data through a simple HTTP protocol (GET, PUT or POST methods), file up- and download requests sent to the Data Bridge are redirected to the storage. Thus, the data won't be transferred through the Data Bridge, but will be transferred directly between the client and the storage service. The outline of this redirection in case of a client initiating upload to an Amazon S3 storage is shown in Figure 4: first the client connects to the Data Bridge initiating data upload. Next, the Data Bridge responds with a redirection to URL where the targeted Amazon S3 can accept the data, and finally the client sends the data directly to the S3 storage.

Although we originally considered including support for metadata attributes of databases in the Data Bridge, the first version is limited to flat files on Storage services that organize their data in a hierarchical structure, such as files and directories. In the rest of this paper we refer to the actual data as file, and to the location of the data as directory.

A. *High-level architecture*

Figure 5 shows the high-level architecture of the Data Bridge. Five main components or component groups can be distinguished: the Public interface and HTTP Servlet accept requests from external components; the Adaptor Manager arranges the execution of the requested operations; the Adaptor Interface, along with the different Adaptors, communicate with the different Storage systems supported by the Data Bridge; and the Temporary URL queue, which temporarily stores incoming data put and get requests. Depending on the incoming requests, the different Threads perform the requested operation by making use of the relevant adaptor through the generic Adaptor interface. We describe the different interfaces and components in detail.

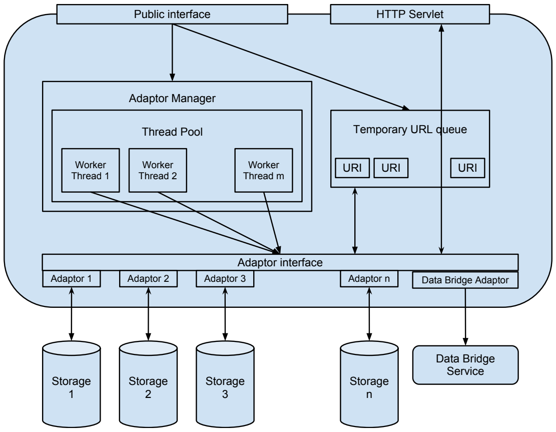

Fig. 5. Data Bridge architecture

B. *Interfaces*

There are two interfaces in the Data Bridge: the Public interface and the Adaptor interface. These interfaces expose the same set of operations, but through different technologies: the Public interface offers a Web service, whereas the Adaptor interface describes a Java API that the different adaptors should implement.

The following operations are supported by the interfaces: list, mkdir, delete, get, put, and copy. The operations use URIs as their argument where applicable. The URI is an abstract Java object that all the adaptors should extend in order to contain all the necessary information to reference and handle the data to be managed on that particular type of Storage service. For example, in case of an HTTP adaptor the URI object should contain an URL, or in case of a GridFTP adaptor the URI object should contain a GridFTP URL and all the necessary credentials (X.509 proxy certificate) to access the data. An URI may represent either a file or a directory. All the operations return either a result or an error message depending on the success of the operation.

The list operation can be used to list the contents of a directory entry, represented as an URI, and returns a list of URIs found under the given entry. If the URI argument of the operation references a file, the operation returns a single entry with the URI of that file. Otherwise, the list of entries found in the given directory is returned.

The mkdir operation can be used to create a directory entry. The URI should represent a non-existing entry.

The delete operation can be used to recursively delete a directory or a file represented by the URI. That is, if the URI refers to a file, the file is removed. If the URI refers to a directory, the entire contents of the directory is removed. Upon termination, the function returns the list of URIs that were tried, either with a success or failure indicator (in the latter case, the appropriate error message is appended).

The get operation can be used to register a file download request. The only argument to the function is the URI, which should represent a file. The result of the operation is a temporary unique Data Bridge URL (using UUID), that can be accessed by the HTTP GET method to get the data

belonging to the file referenced by the URI. So if one wants to fetch a file from a storage service using the Data Bridge, first the get operation should be called to get a temporary HTTP GET URL from the Data Bridge, and if the registration has been successful, HTTP GET can be used to actually fetch the data. Figure 5 shows how these temporary URLs are organized in the data bridge.
The detailed usage of the get operation is as follows: first the client uses the get operation of the Data Bridge's Public interface with the URI to the data to download. This will result in putting the URI into the Temporary URL queue of the Data Bridge, with a unique ID. This ID prefixed with the HTTP servlet's URL is returned as the result of the get operation. Next, the client invokes an HTTP/GET method to the Data Bridge's HTTP servlet by using the temporary URL to actually have the data downloaded: this will make the HTTP servlet look up the URI belonging to the temporary URL, and will make use of the relevant adaptor to actually stream the data as a response to the HTTP/GET method. If the data on the given storage can be fetched using HTTP GET, the HTTP servlet instead of streaming the data through the Data Bridge, will redirect the client to the HTTP URL where the data can be fetched from the storage directly. Thus, whenever possible, the data should be streamed directly from its source, and not through the Data Bridge.
We have decided to implement the get operation following this two-phase approach due to the way how data is accessed from scientific workflows' jobs: before the jobs of the workflow are submitted, a component (for example the workflow system or job submission component) can register the file up- and download requests to the data bridge. Once this is done, the component can submit the actual jobs encapsulated in simple wrappers that are capable of fetching and uploading the data using simple HTTP methods (for example wget of cURL can be used to perform these tasks). Thus, the somewhat complex operation of invoking the web service interface is detached from the actual data transfer, so this latter task can be performed really simply.

The put operation is very similar to the get operation. The only difference is that in order to actually have the data uploaded, the client has to invoke an HTTP/POST method for the temporary URL returned by the put operation of the Public interface, and stream the data to be uploaded to the given temporary URL, or to the redirected storage URL is HTTP PUT is supported by the storage.

The get and put operations in a simple scenario are shown in Figure 6. In this case, before the submission of a job's that uses a data storage, the workflow system's data management registers the job's down- and upload request through the Data Bridge's Public Interface. After this is done, the temporary URLs associated with the input and output files are sent to the job sumbission component that is responsible for preparing an appropriate wrapper, that is capable of fetching input files before running the actual executable, and uploading the produced output files after the actual executable has terminated. The wrapper can be really light-weight, as it simply has to use HTTP GET and POST methods to fetch and upload input and output files, respectively.

The Temporary URL queue is periodically invalidated by the Data Bridge, entries older that specified in a configuration file are removed, resulting in an HTTP/404 (not found) status code.

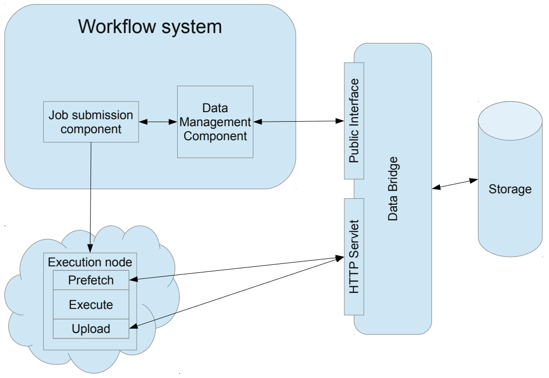

Fig. 6. Two-phase get and put operations

Finally, the copy operation can be used to copy data from one location to an other. It accepts three arguments: the source URI, the destination URI, and the optional URL of a destination Data Bridge service. If the optional URL of the destination Data Bridge service is not specified, the Data Bridge service serving the copy operation should be capable of handling both the source and destination URIs, that is it should have all the relevant adaptors enabled. If the optional URL of the destination Data Bridge service is specified, the Data Bridge service serving the copy operation will make use of its Data Bridge adaptor to issue a put operation to the destination Data Bridge service using the destination URI and the data served by the source URI's adaptor to perform the copy operation. This way, the Bridge services can be used in a master-slave scenario, where master Data Bridge can use slaves to perform the actual transfers to the selected target storage. Of course, third party transfer should be used whenever possible to minimize the amount of data transferred through the Data Bridge service. For example, the GridFTP protocol enables third party transfers.

C. Components

The remaining components of the Data Bridge that haven't been explained are the Adaptor Manager, the Thread Pool, the Worker Threads, the Adaptors, the Temporary URL queue and the HTTP Servlet.

The task of the Adaptors is to implement the Adaptor interface for a given type of storage service, and are responsible for actually communicating with storage service of the given type. In order to minimize the necessary implementation work, we have choosen to use jSAGA wherever possible to implement the different Adaptors. The advantage of using jSAGA is that is already provides a unified API to access files located on different types of storages, like FTP, GridFTP or SRM. It is important to note that the Adaptor Interface is not the same as the interface offered by jSAGA, thus in order to implement support for a storage type not handled by jSAGA, one does not have to implement that support in jSAGA itself.

The Worker Threads are responsible for performing the operations requested through the Public interface with the different Adaptors through using the Adaptor interface. Actually, the web service framework used (JAX-WS) starts different threads for the different web service requests, thus the Worker Threads simply start processing the requested operation once a call to the web service interface comes in. If follows from this, that the Thread Pool is a simple pool for the Worker Threads, also managed by the web service framework. The Adaptor Manager is responsible for managing the execution of requested Public interface operations through the Worker Threads.

Finally, the Temporary URL queue and the HTTP Servlet are responsible for actually serving and storing data belonging to previously registered get and put requests as described earlier. That is, if a get or put method is requested through the Public Interface, the affected URIs are registered in the Temporary URL queue, and the temporary URL is returned as the response to the get or put methods. As described earlier, after this cliens can invoke the HTTP Servlet through simple HTTP GET and POST methods to actually down- or upload the data, using the temporary URLs registered earlier.

V. Supporting scientific workflows with the Data Bridge

In this section we describe a complex usage scenario of the Data Bridge, in which end users of a science gateway are running experiments on a local PBS cluster that processes data residing on an Amazon S3 storage. Figure 7 shows the outlines of the infrastructure. As shown in the figure, WS-PGRADE/gUSE is used as the science gateway framework, where users are accessing two user interfaces in the form of portlets: the Storage Browsing portlet for browsing the storage and selecting input and output data for the scentific experiment, and the Workflow management portlet, that can be used to parametrize and run the workflows belonging to the experiments. As it is shown in the figure, the Storage Browsing portlet is communicating with the Data Bridge to expose the browsing and data selection functionality. Once the workflow has been configured and submitted, its nodes are processed by the WFI (WorkFlow Interpreter) component of gUSE. This component is responsible for scheduling nodes of the workflow for execution. Once a job is about to be submitted, it is sent to the DCI Bridge job submission service, that uses the Data Bridge to register the jobs' data down- and upload requests. In this scenario, the DCI Bridge is running wrappers to handle the data, with an optional redirection to the storage to minimize the necessary amount of network traffic through the Data Bridge.

The steps for running an experiment from the user's point of view are as follows: uploading input data to the storage (Amazon S3) or search for it, configuring the experiment's workflow to use the selected data, run the workflow, and upon termination, get the results. For this, the user simply has to use the Storage Browsing portlet for data management and selection, and the Workflow management portlet for workflow configuration and experiment execution.

What is hidden from the user in this case is the complex interaction of the other components presented in Figure 7 that arrange the execution. The Storage Browsing component in the background makes use of the Data Bridge's Public interface and HTTP servlet to browse and manage data stored on the Amazon S3 storage. Once the input data for the experiment is selected, its location is saved to the workflow's configuration, and the workflow is submitted. At this point, the WFI (WorkFlow Interpreter) is responsible for arranging the workflow's execution in the form of jobs.

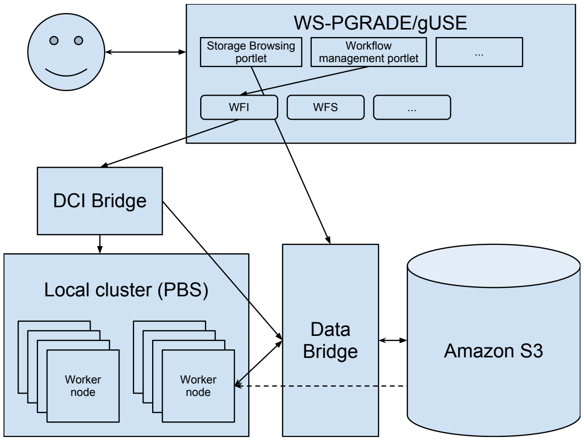

Fig. 7. Scientific workflow execution scenario

Once a job is about to be submitted, the DCI Bridge checks if the job is using any input or output files that can be managed only with the help of the Data Bridge (for example, because they reside on the Amazon S3 storage). For such files, the DCI Bridge makes use of the Data Bridge's get and put operations to get the temporary URLs that can be used by simple tools (like wget or cURL) to download and upload the data. That is, the DCI Bridge doesn't do any data transfer, it simply invokes the get and put operations to have the temporary URLs registered. For all these temporary URLs, appropriate handling sections are created in the job's wrapper script, that will actually perform the input files' download, and the output files' upload from and to the Amazon S3 storage through making use of the Data Bridge's HTTP servlet on the Worker node. Finally, the job is submitted to the cluster.

Once a submitted job starts on a Worker node, the wrapper script created by the DCI Bridge will fetch any input files from the Amazon S3 storage by using the appropriate temporary URLs (if the data is directly available from the S3 service, the Data Bridge's HTTP servlet will redirect the client to S3 to get the data directly from there). Next, as all the input files for the job are available locally, the real executable is started. And finally, once the real executable has finished, any output files destinated to the S3 storage are uploaded using the appropriate temporary URLs.

Thus, as it can be seen, data is handled in two phases: the DCI Bridge only registers data put and get requests to have temporary URLs available for data up- and download), and the actual data transfer always happens where the job is actually running, always making directly use of the storage service whenever possible.

Finally, if the workflow has been processed with some result, the user may use once again the Storage Browsing portlet to check the produced results, given that the user has configured the workflow to store the results on the S3 storage.

VI. Conclusions

In the paper we have presented a service-oriented approach to provide a unified, easy-to-use way to manage data stored on different types of storage services in the form of the Data Bridge. We have presented a number of data management scenarios (data browsing, data fetching/uploading, and data migration). Based on the needs of these scenarios, we have presented the architecture of the Data Bridge so it satisfies these needs.

The Data Bridge operates as a stand-alone web service that is able to perform basic operations on storages, namely listing, creating, removing, downloading, uploading and copying data files. All these operations are available as simple web service operations supported by a very simple servlet in case of the upload and download operations. The detached composition of the web service (for initiating operations) and the HTTP servlet (for performing data up- and download) enables the easy usage of the service from scientific workflows, where data transfer is initiated by the workflow management system, and the actual data transfer happens where the job is actually running, with the help of simple HTTP clients (like wget or cURL).

We have presented the architecture of the Data Bridge in detail: we have shown all the interfaces, and highlighted internal components as well. The different interfaces (the Public Interface and the Adaptor Interface) operate at different levels: the Public Interface is exposed to the Data Bridge's clients as a web service, whereas the Adaptor Interface is a Java interface the different Adaptors responsible for actually communicating with the storages of different types should implement. This organizations offers a pluggable architecture, where implementation for new storage Adaptors becomes a relatively easy task. As already mentioned, the different Adaptors should interface with the different storage resources, and here we're making use of the unified jSAGA API where possible to minimize necessary development. Finally, in the architecture the Temporary URL queue serves as a volatile storage for registered file up- and download requests, thus helps to implement decoupling the actual data transfer from the invocation of the web service calls.

We have also presented a complex scenario which covers two of the cases presented in the Requirements section. This complex scenario is about configuring and running scientific experiments in the form of WS-PGRADE/gUSE science gateway framework workflows, that operate on data available in a cloud storage (Amazon S3). In this complex scenario users are making implicitly use of the Data Bridge service: the Storage Browsing portlet is used to browse, or up- and download the data, the Workflow management portlet is used to specify which data the experiment should be run on, finally, the DCI Bridge and its job wrappers are making use of the Data Bridge service to actually handle the data (download the data before the job's real executable is started, and upload produced data after the job's real executable has finished).

The complex scenario presented the usage of the Data Bridge from WS-PGRADE/gUSE, but it is not tied directly to any workflow system, meaning it can be used as a standalone service, and can easily be integrated into an existing workflow system or user interface to satisfy users' needs.

Although the Data Bridge is a usable service in its current form, the restriction to support storage services that organize data into a hierarchical structure (that is, files in some sort of directory structure) is rather strict. It follows from this restriction, that for example metadata services or databases are currently not supported by the Data Bridge. Fortunately the abstract URI data reference object of the Data Bridge can later be extended easily to support such data services as well.

Acknowledgment

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 283481 (SCI-BUS) and 312579 (ER-Flow).

References

[1] Kacsuk, Peter and Farkas, Zoltan and Kozlovszky, Miklos and Hermann, Gabor and Balasko, Akos and Karoczkai, Krisztian and Marton, Istvan: WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities, Journal of Grid Computing, vol 10, no 4, 2012

[2] SCI-BUS Project, https://www.sci-bus.eu/

[3] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster: The Globus Striped GridFTP Framework and Server, Proceedings of Super Computing 2005 (SC05), November 2005

[4] Arie Shoshani, Alex Sim, Junmin Gu: Storage Resource Managers: Middleware Components for Grid Storage, Tenth Goddard Conference on Mass Storage Systems and Technologies. 2002. p. 209.

[5] Amazon Simple Storage Service (Amazon S3), http://aws.amazon.com/s3/

[6] Constantinos Karasavvas, Mario Antonioletti, Malcolm Atkinson, Neil Chue Hong, Tom Sugden, Alastair Hume, Mike Jackson, Amrey Krause, Charaka Palansuriya: Introduction to OGSA-DAI Services, Scientific Applications of Grid Computing, Lecture Notes in Computer Science Volume 3458, 2005, pp 1-12

[7] Chaitanya Baru, Reagan Moore, Arcot Rajasekar, Michael Wan: The SDSC Storage Resource Broker, Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research, CASCON '98, p. 5, 1998

[8] Arcot Rajasekar, Reagan Moore, Chien-Yi Hou, Christopher A. Lee, Richard Marciano, Antoine de Torcy, Michael Wan, Wayne Schroeder, Sheau-Yen Chen, Lucas Gilbert, Paul Tooby, and Bing Zhu: iRODS Primer: Integrated Rule-Oriented Data System, Synthesis Lectures on Information Concepts, Retrieval, and Services, 2010, Vol. 2, No. 1 , Pages 1-143

[9] Sylvain Reynaud: Uniform Access to Heterogeneous Grid Infrastructures with JSAGA, Production Grids in Asia, pp 185-196, 2010

[10] A Simple API for Grid Applications (SAGA), http://www.ggf.org/documents/GFD.90.pdf

[11] Globus Online, https://www.globusonline.org/