IST-2001-32133
GridLab - A Grid Application Toolkit and Testbed

Deliverable D8.6 - Data Management Integration

Author(s):	Felix Hupfeld, Andrei Hutanu, Andre Merzky, Thorsten Schütt, Brygg Ullmer
Document Filename:	
Work package:	WP 8 - Data Handling and Visualization
Partner(s):	
Lead Partner:	ZIB
Config ID:	GridLab-08-DATA-0006-M28
Document classification:	Internal

Abstract: This document describes the integration of the data management services with the GAT, Cactus and Triana.

Information Society
Technologies

Contents

1 Introduction **2**

2 GAT **2**

2.1 interface GATAdvertService extends GATMonitorable 2

2.2 interface GATFile extends GATMonitorable, GATAdvertisable 4

2.3 interface GATLogicalFile extends GATMonitorable, GATAdvertisable 5

3 Cactus **6**

4 Triana **7**

5 Visualization Clients **7**

6 Summary **9**

1 Introduction

This report is part of the deliverable D8.6. The main part of this deliverable, the source code, is available via the GridLab webpages. The functionality, configuration and installation of the software was already described in depth in the deliverable D8.9 Sections 2 - 6. In this deliverable we focus on the integration with other work packages and user applications. According to the technical Annex I,

> *this deliverable provides the integration of the data management techniques into the GAT, into the Cactus framework, and into Triana and visualization clients (as prototype implementation).*

The following sections describe the integration with the GAT, Cactus, Triana and visualization clients respectively.

2 GAT

The GAT consists of two major parts: (a) a set of generic interfaces for the most common Grid services and (b) a set of adaptors which implement these interfaces. The generic interfaces can be classified into 4 areas. WP8 is involved in the first two subsystems.

- GAT Advertisement Subsystem
- GAT File and Stream Subsystem
- GAT Event and Monitoring Subsystem
- GAT Resource Subsystem

The *GAT Advertisement Subsystem* provides means to advertise files, endpoints for communication and other entities. Jobs can use this service e.g. to publish an IP address and a port which then is used to get information about a job or to steer a job by changing parameters on-the-fly. Jobs can also announce the location of files which contain intermediate results. Users could use this info to track the progress of their jobs. The meta data directory can be either queried by attributes or browsing the hierarchical namespace.
WP8 was heavily involved in the discussions which led to the specification of the advertisement interface in the GAT. We provided our knowledge as a designer of advertisement services. And we also influenced the specification based on the discussions we had with users of our service.
Finally we wrote the standard GridLab adaptor for this interface which is based on the meta-data service. The adaptor is part of the standard GAT distribution and the installation and configuration is covered in the GAT installation documentation. See also `http://www.gridlab.org/WorkPackages/wp-1/adaptorreleases.html`. The interface is described as follows:

2.1 interface GATAdvertService extends GATMonitorable

Operations

Add Add an GATAdvertisable instance and related meta data to the GATAdvertService, at a given path (absolute or relative to the current working directory). If an GATAdvertService entry exists at the specified path, that entry gets overwritten, and a warning is issued.
Inputs:

GATAdvertisable — advert — Instance to be entered into the GATAdvertService.

GATTable — metaData — Meta data to be associated with the passed GATAdvertisable.

String — path — Path (either absolute or relative to the current working directory) of the new entry.

Delete Remove an GATAdvertisable instance and related meta data from the GATAdvertService, at a given path (absolute or relative to the current working directory).
Inputs:

String — path — Path (either absolute or relative to the current working directory) of the entry to be deleted.

GetAdvertisable Gets an advertized object specified by its path in the metadata directory.
Inputs:

String — path — Path (either absolute or relative to the current working directory) of the entry.

Outputs:

The GATAdvertisable which is referenced by the given path.

GetMetaData Gets the meta data of an advertized object specified by its path in the metadata directory.
Inputs:

String — path — Path (either absolute or relative to the current working directory) of the entry.

Outputs:

A GATTable containing the metadata.

Find Query the GATAdvertService for entries matching the specified set of meta data.
Inputs:

GATTable — metaData — Metadata describing the entries to be searched for.

Outputs:

A set of paths, each pointing to a matching entry.

setPWD Specify the element of the GATAdvertService namespace to be used as the current working directory. Relative paths are resolved relative to this directory.
Inputs:

String — path — New absolute or relative reference path.

getPWD Returns the current directory of the GATAdvertService namespace used as reference for relative paths.
Outputs:

The absolute path to the current working directory.

It provides functions for adding, deleting and finding advertisements. Each advertisements has a unique identifier which is interpreted as a path in a filesystem like tree-structure. Additionally metadata can be associated with an advertisement. The metadata is a set of key-value-pairs. This duo-fold naming scheme allows browsing the system by using the filename-based identifiers and by searching on the metadata.
This design was chosen to make the interface look like a filesystem which confronts the application developer with a known metaphor. Using known concepts we can reduce the overhead for the application developer when he starts to use our interfaces. And we can implement backends based on directory services like e.g. LDAP.
The *GAT File and Stream Subsystem* is further divided into:

- file movement
- tracking of file locations and location independent handling of files

Again for all 3 areas we provided our expertise in this area to create a specification of interfaces which are easy to use and reasonable powerful to fit a large range of applications.
The standard GridLab adaptor for the *file movement* was implemented by us and is based on the file browsing and file movement service (see D8.9). The adaptor itself can be downloaded via the GAT webpages (`http://www.gridlab.org/WorkPackages/wp-1/adaptorreleases.html`) whereas the service is provided via the WP8 webpage (`https://www.gridlab.org/WorkPackages/wp-8/file_service/gridlab_file_service-1.9.tar.gz`). Installation instructions as well as usage guides are also provided on the corresponding webpages and/or in the distributions.

2.2 interface GATFile extends GATMonitorable, GATAdvertisable

An instance of this class presents an abstract, system-independent view of a physical file. User interfaces and operating systems use system-dependent pathname strings to identify physical files. GAT, however, uses an operating system independent pathname string to identify a physical file. A physical file in GAT is identified by a URI.

Operations

ToURI This method returns the URI of this File
Outputs:

A String containing the URI of this file.

Copy This method copies the physical file represented by this File instance to a physical file identified by the passed URI.
Inputs:

URI — location — The new location.

Move This method moves the physical file represented by this File instance to a physical file identified by the passed URI.
Inputs:

URI — location — The new location.

Delete Deletes this physical file.
The interface basically contains 3 functions (copy, move and delete) which cover the most basic needs when dealing with files in a distributed environment (remote file access isn't yet covered in the GAT).
For the *tracking of file locations and location independent handling of files* we compared the interfaces of the GridLab replica catalog with other replica management systems. Based on this comparison we came up with a generic interface which can be mapped to all these services and which is still powerful enough to be usable. Finding the greatest common divisor and still having an interface which provides enough functionality to be usable became a challenging task. But we feel that in the end the specification which was developed in the discussion with WP1, Cactus and Triana is a good compromise:

2.3 interface GATLogicalFile extends GATMonitorable, GATAdvertisable

Operations

AddFile Adds the passed GATFile instance to the set of physical files represented by this GATLogicalFile instance.
Inputs:

GATFile — file — A GATFile instance to be added to the set of physical files prepresented by this GATLogicalFile instance.

AddURI Adds the physical file at the passed URI to the set of physical files represented by this LogicalFile instance.
Inputs:

URI — location — The URI of a physical file to add to the set of physical files represented by this LogicalFile instance.

RemoveFile Removes the passed File instance from the set of physical files represented by this LogicalFile instance.
Inputs:

GATFile — file — A File instance to remove from the set of physical files represented by this LogicalFile instance.

RemoveURI Removes the physical file at the passed URI from the set of physical files represented by this LogicalFile instance.
Inputs:

URI — location — The URI of a physical file to remove from the set of physical files represented by this LogicalFile instance.

GetURIs Returns a list of URI instances each of which is the URI of a physical file represented by this instance.
Outputs:

The list of URIs represented by this GATLogicalFile instance.

GetFiles Returns a list of GATFile instances each of which is a GATFile corresponding to a physical file represented by this instance.
Outputs:

The list of GATFiles represented by this GATLogicalFile instance.

Replicate Replicates the logical file represented by this instance to the physical file specified by the passed URI.
Inputs:

URI — location — The URI of the new physical file.

Logical files are identified by a path in a virtual filesystem. For each logical file the replica catalog stores a set of locations of files – the physical files. The interface provides functions for managing these locations. The basic functions can be used to add or delete new locations or to get a list of all locations for a given logical file. For moving physical files around the interface provides the replicate function. When called on a logical file it will choose one of its physical files, move this file to the given location and update the replica catalog accordingly.
Further information about the adaptor as well as the source is provided on the WP1 webpages (http://www.gridlab.org/WorkPackages/wp-1/adaptorreleases.html). The replica catalog was already described in the last deliverables, especially in D8.9. The replica catalog can be found at https://www.gridlab.org/WorkPackages/wp-8/software/zibdms-0.3.0-pre1.tar.bz2.

3 Cactus

Cactus is parallel code for numerical simulations used in numerical relativity, astrophysics, bioinformatics and chemical engineering.

> The name Cactus comes from the design of a central core (or "flesh") which connects to application modules (or "thorns") through an extensible interface. Thorns can implement custom developed scientific or engineering applications, such as computational fluid dynamics. Other thorns from a standard computational toolkit provide a range of computational capabilities, such as parallel I/O, data distribution, or checkpointing. http://www.cactuscode.org/

The integration with Cactus was done on two levels. On the one hand we exposed the functionality of our services through the GAT to Cactus. On the other hand we helped them to fulfill their requirements by giving support on the data management functionalities in the GAT. The integration of the GAT into Cactus was performed by adding a new Thorn – the so called CGAT Thorn.
Currently Cactus is publishing output data and checkpoint files in the replica catalog and moving the checkpoint files between the machines using GAT file movement. These functionalities are provided by the replica catalogue and the file service.

4 Triana

The integration with Triana was similar to the integration with Cactus. The basic functionalities are provided by the GAT. But in the case of Triana we also provided an efficient method for accessing files in a special format on remote machines. This file format is only used in one community therefore this code was integrated directly in Triana and not via the GAT.
The Geo600 project stores its files in the Frame file format and Triana is one of the applications which are used to analyze those data. Typically these files are stored on a small number of storage elements in the Grid. In WP8 we provide an efficient solution to access these files. In this particular case Triana reads only a small part of the Frame file. We built a system which can efficiently access regular subsets of remote files by using compact pattern descriptions – the pitfalls library [4].
For this prototype we used the already existing Frame-file access library and replaced the I/O layer with our own. We are translating the I/O calls with calls to our pitfalls library which we presented in D 8.5. Now Triana can transparently and efficiently access remote files using the same interface as before.
For the final release we plan to make this process adaptive to the environment. There are basically two extremes: (a) you have a small file and want to read all the data and (b) you have a huge file and have sparse file access. For the former case we will copy one of the replicas to the local machine and use local file access afterwards. For the latter case it is more efficient to use remote file access as described above. We will build a model together with the Adaptive work package to decide at runtime which method is more efficient.
The model has to take several parameters into account, like network bandwidth, network latency, ratio between file size and accessed data size.

5 Visualization Clients

The visualization part of the work package is closely connected with the data management efforts. As visualization is a complex form of data analysis, it is clear that in a grid context, it will use a number of data management clients, and it will even drive the development of new and specialized data management services [4, 1].
In our case, the grid visualization efforts are twofold. We are prototyping a special visualization service for Cactus simulations (see Fig. 5) in close cooperation with WP-2 and in parallel we are developing generic distributed visualization techniques using the scientific visualization software Amira [2, 3, 5].
The prototype visualization service uses the replica catalog to retrieve the physical location of a directory registered with a logical file name by the running Cactus simulation using the GAT. The file management services are used from the visualization service to retrieve a list of files of interest and then to copy them (all at once) in a temporary directory. The temporary directory is located on the same machine where the visualization service is running.
While replicating the files locally is acceptable for small files as the ones we are targeting right now in our visualization service, for large output files like for example the HDF5 output files generated by Cactus or the ones used in the Bone3D project (http://bone3d.zib.de/), the visualization software needs special efficient remote data access techniques. Such techniques were developed in the GridLab project for both HDF5 [1] (`http://www.zib.de/visual/projects/gridlab/hdf5/`) and generic binary files [4], and even if they are in some ways too specific to a certain way of doing remote data access to be integrated in the GAT, they are used by the visualization clients (see Fig. 2) as they provide the means to achieve interactive visualization.

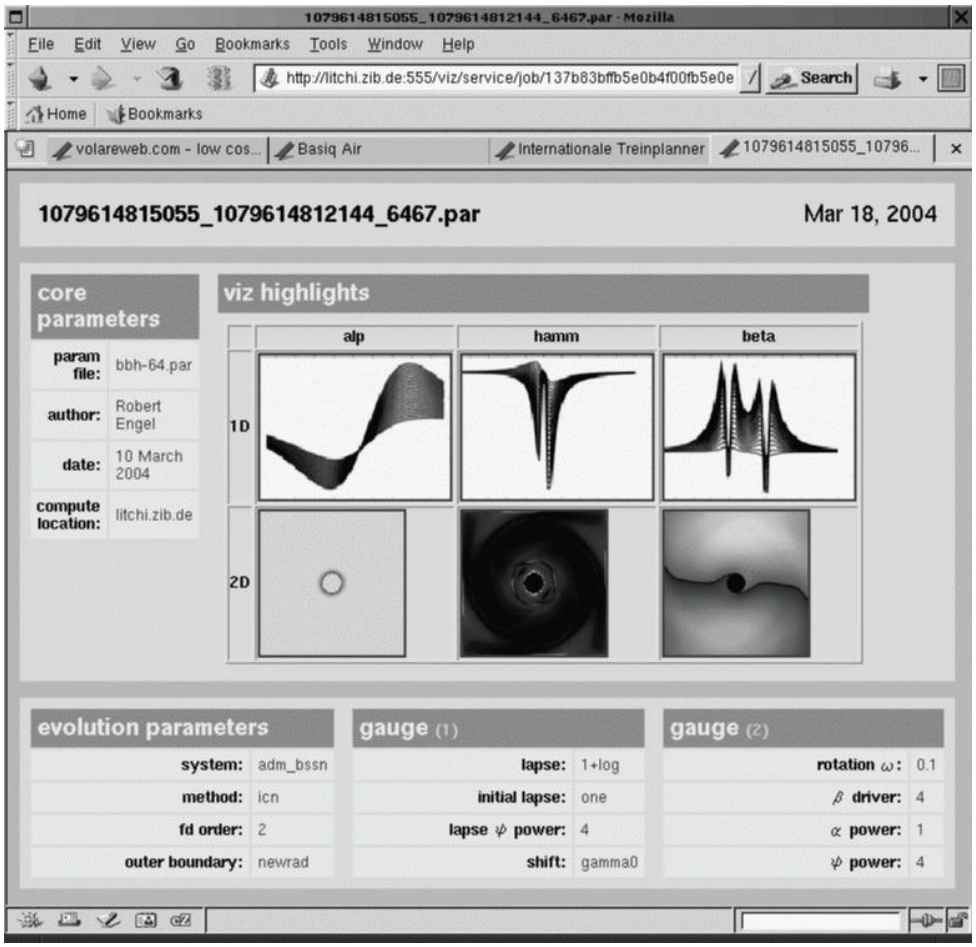

Figure 1: Screenshot of Visualization Service for Cactus

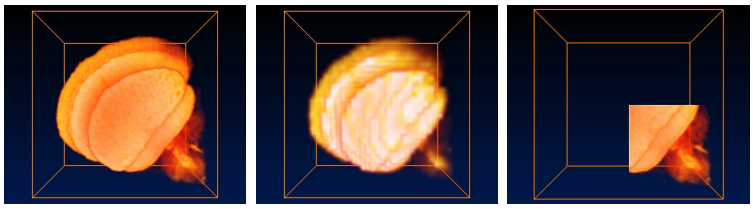

Figure 2: Example: (1) full-resolution 3D image of the brain of a honey bee, (2) same data at 1/4 resolution, and (3) lower-right corner

Fig. 2 shows the technique which was integrated into Amira for interactive remote visualization. The partial remote file access can be used to extract parts or subsampled versions of remote files for display. The HDF5 support is used for Cactus whereas for Triana we developed a remote partial file access for generic binary files.

6 Summary

In contrast to existing solutions GAT provides a generic interface to applications, that provides easy access to several independent implementations of Grid-Datamanagement services, including the services developed by GridLab. This makes application developers more independent from changes on the Grid-Infrastructure and services.

References

[1] H.-C. Hege, A. Hutanu, R. Kähler, A. Merzky, T. Radk e, E. Seidel, and B. Ullmer. Progressive retrieval and hierarchical visualization of large remote data. *Proceedings of the Workshop on Adaptive Grid Middleware*, September 2003.

[2] Hans-Christian Hege, Tino Weinkauf, Steffen Prohaska, and Andrei Hutanu. Distributed visualization and analysis of fluid dynamics data. In *Proc. Fourth International Symposium on Advanced Fluid Information and Transdisciplinary Fluid Integration*, Sendai, Japan, November 11-12 2004.

[3] Steffen Prohaska, Andrei Hutanu, Ralf Kähler, and Hans-Christian Hege. Interactive exploration of large remote micro-CT scans. In H. Rushmeier, J. J. van Wijk, and G. Turk, editors, *Proc. IEEE Visualization 2004*, pages 345–352, Austin, Texas, 2004.

[4] Thorstan Schütt, Andre Merzky, Andrei Hutanu, and Florian Schintke. Remote Partial File Access Using Compact Pattern Descriptions. In *Proceedings of the 4th International Symposium on Cluster Computing and the Grid (CCgrid)*, 2004.

[5] Brygg Ullmer, Andrei Hutanu, Werner Benger, and Hans-Christian Hege. Emerging tangible interfaces for facilitating collaborative immersive visualizations. In Oliver Staadt, editor, *to appear in Proc. of NSF Lake Tahoe Workshop on Collaborative Virtual Reality and Visualization*, Oct. 2003.