Grid Data Services for Production Grids
Peter Kunszt, Gavin McCance, Krzysztof Nienartowicz, Akos Frohner
CERN

Abstract.
Production Grids are deployments of Grids through a well-defined set of sites with a high level of commitment, adhering to a certain standard of support and with a clear process of accepting Virtual Organizations as their users. In this document we report our experience, lessons learned and initial recommendations for designing data management services for such an environment. We also discuss some ideas on integrating application-level metadata with core Grid data services.

1. Introduction
One of the largest Production Grids today is the LHC Computing Grid (LCG). The middleware that has been deployed on the first LCG system (LCG-1) is based on the Virtual Data Toolkit (VDT), the EU DataGrid (EDG) and other components. Our experience stems from deploying and supporting the EDG data management middleware for EDG, LCG-1 and LCG-2, and on feedback from the various user groups who have interfaced their applications to our middleware. In this article we describe what we think should be among the driving considerations for designing the next generation Grid Data Services. For the work of the GGF Data Area we hope to provide valuable insights that might help to define and prioritize the future directions of work and research.

2. Design Considerations
Production Grids distinguish themselves from experimental and research Grids primarily through their high availability, fail-safety and error reporting mechanisms. They are usually larger and better supported. In order to achieve a high level of stability and availability, the Grid components need to be very robust especially with respect to network outages. In addition to these rather obvious requirements, the following aspects of Production Grid environments influence interface design:

Security
In our experience, the security model is pervasive to the system. The middleware cannot be designed such that security can be ‘added on later’. It goes horizontally through all components. Requirements come not only from the users and applications but also from the site administrators (see next section). A system administrator has considerable interest in the security model of the system and this imposes requirements upon the design of the service. Things to consider are

- Delegation: Middleware components are modular. The interfaces are defined among others through GGF recommendations but implementations may be coming from different providers. In order to provide complex high-level functionality required by the user, the modules call upon each other’s services on behalf of the user. This necessitates a delegation mechanism by which the user can delegate very specific rights to the service, and by which the service can forward the delegated rights to other services.
- Delegation chains: The sites running the middleware components have sometimes very strict requirements based on site policy that they have to monitor and track the
connections made to services inside their administrative domain. It is a legal requirement of most services that the user be identifiable, and this is particularly the case for data services that may involve the consumption of a large amounts of resources. A site administrator needs to be able to track the chain of delegations of a request prior to arriving to a site. Certificate chains are a possible technology to provide such a tracking. Audit trails may be based on such information.

- **Applying local policy:** The middleware components need to be able to analyze the incoming delegation and chain and map the request into a local role/capability according to site policy. An example for a site policy might be: “Local users have more storage available than Grid users. If a Grid user happens to be local, apply the local policy.”

- **Authorization semantics:** Each Grid data middleware service needs to implement some semantically meaningful authorization mechanism specific to the service type. For example a metadata catalog service probably needs to distinguish between read-only, read-write and administrative users. A finer distinction could be applied by defining authorization mechanisms like “maximal returned query result set is 100”, “maximally allowed tablespace in catalog is 100M”, “user may query table A but not table B”.

An interface design should take into account all of the above, including mechanisms and options to deal with the possible authentication, authorization, auditing and of course all failure modes. Examples will be presented in the final paper.

**Change and evolution**

In a distributed environment when there are potentially dozens or hundreds of sites running some version of the Grid middleware it is impossible to impose the same version on all sites. The interface has to deal very cleanly with version mismatch and should have intrinsic mechanisms to provide backwards and to some extent also forward compatibility with itself through different versions. Clients should be able to connect to different versions and even newer service versions should at least produce a well understood and parseable error message if an operation is not possible due to version mismatch.

A well-designed interface is simple to adapt to new standards, take as an example the upcoming migration from OGSI to WSRF-based interfaces. Detailed examples will be put into the final paper.

**Heterogeneous environments**

The best way to deal with the heterogeneous environments in Grids is to abstract out the interfaces from the underlying technology. This is true for file systems, file catalogs, RDBMS, mass storage systems, etc. The Data Area covers most relevant topics (Grid File Systems, Grid Storage Management, Database Access and Integration, Replication). The success of these abstractions will ultimately define the success and usability of Production Grids.

**Connectivity**

A major architectural constraint is that of connectivity. Often the nodes of the farm have no external connectivity and distributed data access services must be designed with this in mind for example by routing the external requests through a proxy service. Generally, this should be
entirely transparent to the users of the service; this means that the interface that is exposed to the user is the same regardless of whether the request for the data is being serviced locally or being serviced remotely via a proxy.

Another aspect of connectivity is managing the data flow. Iterating through large datasets or seeking through a data stream should be possible through the data access interface. We have found that failures have to be defined very carefully and propagated in detail to the application or user accessing the data in a distributed environment. Data services will need to deal with reliability to a much higher degree. Take as an example GridFTP as opposed to the Reliable FTP service. To ‘add on’ reliability to GridFTP resulted in a whole new project. If reliability were built in intrinsically into the FTP protocol, this would not have been necessary. Of course that was not an option, so this example is only to illustrate how the concept of reliability should drive the design of future Grid data services in order to reduce the need of ‘add on’ services later.

**Administration and Management**
Our experience has shown that the primary design requirements coming from the system administrators are for easy installation and configuration of the service and for easy control and management of the service without impacting users of other services that happen to be running on the same box. Ease of management and control means for example:

- Easy to move and scale. For scalability and fault tolerance reasons, a service should be able to be moved to a different machine or environment quickly with minimal effort and minimal impact upon the users. It should be easy to increase the capacity of a given service. A well-designed interface allows this without long service interruptions.
- Services should be properly partitioned, for example a service can be restarted without affecting any other service.

**Performance and Quality of Service**
Centralized services have the advantage of being always up-to-date. In a distributed system however, they can exhibit unacceptably long access times and ‘downtimes’ from a remote site due to network outages. Distributing data services is the only workable solution to this problem. Such services may build upon existing technologies like distributed RDBMS (database replication option), through reliable messaging queues, etc. Optimizing for the service semantics and usage patterns inside the distribution may further increase the performance.

By Quality of Service (QoS) we understand different levels of failure semantics and sophistication of error recovery mechanisms. Different levels of QoS will necessitate that Data services consider stale data, transactional state, rollback mechanisms, etc. Automatic recovery policies and management thereof also improves the QoS. Again the Reliable File Transfer service can be used as an illustration: different levels of QoS policy may be applied by configuration: e.g. upon failure retry once every hour, failing after 6 retries.
Higher levels of QoS and performance usually conflict. Grid Data services need to be designed with the ability to provide different levels of QoS to optimize either for performance or reliability of the service.

**Technology Considerations**
The software must be able to be run using existing commercial solutions, since open source solutions are unlikely to have the required robustness to run highly available clustered services. The final paper will contain a list of evaluated technologies with advantages and disadvantages.

3. **Metadata Considerations**

One of the most heavily debated set of requirements from our user communities deal with metadata. Metadata has very different semantics for each application, even to each user. To draw the line between generic metadata services that can be abstracted to core Grid Data Services and application specific metadata services is very difficult. This is difficult because the application metadata catalogs are usually pre-existing, and can have arbitrarily complex schema.

We have concluded that the most workable approach is to define a very generic, RDBMS-like or XML-like interface to a metadata catalog, which any application-specific metadata catalog can easily implement. The lower level grid services can then use this interface to pull the information that they need out of the metadata catalog.

An example is that of a dataset collection where the granularity of the primary data storage is at the file level. Users typically want to define their datasets in terms of some arbitrarily complex query over the metadata, rather than referring to a long list of logical files. In order to obtain a handle for a dataset, the user will supply the Grid systems with a query over their metadata. Since the replication systems of this Grid require accessing the data as files, the Grid uses the standard interface of the application metadata catalog to resolve the query into a long list of logical files which it then knows how to manipulate.

Similar mechanisms will permit the virtual file-system structures being requested by some of our users, whereby, the logical files in a given virtual directory are defined by a query over the application specific metadata catalog. The final paper will contain a more detailed explanation with examples and conclusions.