A Data Placement Service for Simplified Grid Job Management

Michael Höfling

May 2008
Bachelor's Thesis in Computing Science, 15 ECTS credits
Supervisor at CS-UmU: Johan Tordsson
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

While Grid job scheduling has received much attention in the last years, relatively few researchers have studied data placement issues. Although job management is important in Grid computing, data management and placement is likely to be among the most challenging issues for future Grid applications such as the Large Hadron Collider (LHC) experiments. The current job management frameworks by the Umeå Grid Infrastructure Research and Development (GIRD) group lack data management capabilities. In this thesis, we study the design and implementation of the Data Placement Service (DPS), an easy to use and integrate service for data placement for Grid environments. The incremental development of this service is described from the first design to a final reference implementation. Existing data management systems are surveyed and their capabilities are compared with that of the DPS. The thesis is concluded with a discussion of future directions of this work.

Contents

Chapter 1

Introduction to Grid Computing

The topic of this Bachelor's thesis is data services for simplified Grid job management. In order to set the context of this thesis we first have a look on what Grid computing itself is.

1.1 Grid Computing

The name Grid computing origins from the comparison of a computational Grid with the power grid. In the latter, every customer can plug in the power connector and use the distributed source of electricity. While the electricity producers are spread all over the country or even the continent, the customer always gets the desired power without having to worry about where and how it is produced. This infrastructure has many advantages. The power grid structure is well equipped to deal with power over and under capacities. Existing power plants can be used more efficiently. Power plants do not have to be built near the customers. The reliability of the power supply system is increased.

To compare Grid computing with the power grid a common vocabulary is needed. In 2002 Ian Foster[6], one of the pioneers in the field of Grid computing, created a three point checklist that defines a Grid as a system that

1. coordinates resources that are not subject to centralized control ...
2. ... using standard, open, general-purpose protocols and interfaces ...
3. ... to deliver nontrivial qualities of service[1].

Using this checklist on the power grid this means that we have to coordinate resources that are not subject to centralized control, e.g., power plants, transmission stations, transformers. These are using standard, open, general-purpose protocols and interfaces, e.g., alternating current and shockproof sockets. Furthermore, they deliver nontrivial qualities of service, e.g., constant power supply of 230 Volts during extreme system load,

[1]Quality of service means that it is guaranteed that a service is available under certain agreements, e.g., 99.999% availability of the public switched telephone network in Germany.

e.g., during the evening when everyone is coming home and wants to cook something or watch TV.

Talking about Grid computing this means that users want to get direct access to computers, software, data or other resources, e.g. devices such as sensors [7]. To coordinate the usage of resources in a controlled way without a centralized point of control, individuals and institutions have to make up sharing rules, e.g., who is allowed to share and under what conditions sharing is acceptable. A set of individuals and/or institutions defined by such sharing rules form a Virtual Organization (VO) [7]. Virtual here implies that the organizations can internally consist of different physical participating and contributing organizations and resources. VOs can also be imagined as an allocation of computing and storage resources. Thus said, the resources to coordinate are in general computing resources, e.g., PCs, workstations, servers, storage elements and many more. Inside the Grid and also to communicate with the Grid standard, open, general-purpose protocols and interfaces, e.g., HTTP and XML, are used. The nontrivial qualities of service that are provided by a Grid are obvious, i.e., the usage of a Grid only makes sense if the availability and reliability can be guaranteed.

Within Grid computing there exist (at least) two different points of view, the job centric view and the data centric view. In a job centric view we are using the Grid as a distributed supercomputer. A job manager is acting like a scheduler in an operating system, i.e., it selects the best computing resource according to the job requirements. In contrast, a data centric view means that we are using the Grid as a distributed data storage. A data manager is acting like the I/O subsystem in an operating system, i.e., depending on the need of space and access time the data manager selects the best available storage. While both views can exist on their own, they normally appear in a combined form as for example a computational job typically needs storage access to load input data and upon completion store output data.

1.2 A Real-World Example Of A Grid

To give a real-world example of a working global Grid the Large Hadron Collider (LHC) is presented. The LHC is a particle accelerator and collider with a circumference of 27 km[15] at the European Organization for Nuclear Research (CERN) in Geneva, Switzerland. It will produce around 10 to 15 Petabytes[2] of information a year. As this amount of data is difficult to comprehend, we remark that this is equivalent to a 20 km high stack of CDs[15]. As storing this amount of data produced at that speed is not possible at a single site even with todays storage technology, Grid technology is used. At the moment the CERN's own storage systems are capable to store a maximum of 1 Petabyte which is obviously not enough when the LHC is fully operable. To store and process this huge amount of information the Enabling Grids for E-Science in Europe Project (EGEE) [1] has been initiated to join the computational and storage resources from dozens of organizations all around Europe to meet the LHC's requirements [15].

1.3 Software for Grids

Grid computing is implemented as middleware to give users transparent access to the Grid. Most current Grid middlewares are based on Service Oriented Architecture (SOA)

[2]That is 10^7 Gigabytes, about 100.000 DVDs.

technology, e.g., SOAP, Web services. One commonly used middleware that is also used in this thesis is the Globus Toolkit 4 (GT4). The components and technologies of GT4 that are used in the thesis are briefly introduced here. Explaining GT4 in detail is however out of the scope of this thesis. There exist literature like [3, 15, 10] that can be consulted by the interested reader for a more detailed explanation of GT4.

1.3.1 Web Service Container

The Web service container of GT4 offers a runtime environment for Web services. By default GT4 Web services are registered in the Web service container. This container itself offers a basic catalog functionality for service discovery.

1.3.2 Monitoring and Discovery Service

To extend the basic catalog functionality of the Web service container even further, GT4 provides a service to monitor and discover all kinds of Grid resources, e.g., services or storages. The service is called Monitor and Discovery Service (MDS). As mentioned above, MDS can be thought of as a catalog service for Grid resources, i.e., it can be used to find out what resources are available on the Grid. Resources can register themselves to the MDS. MDS uses soft states to keep track of the registered resources, i.e., each resource entry has an expiration date to avoid stale entries. Furthermore, users have the possibility to retrieve a fine grained selection of informations of resources using XPath. The information can be obtained both synchronous and asynchronous. The MDS is accessible through a Web interface called WebMDS.

1.3.3 GridFTP

File transfers inside GT4-based Grids are normally done using the GridFTP transfer protocol. GridFTP is an extended version of the FTP protocol that is optimized for large parallel data transfers. It supports authentication through X.509 certificates. Furthermore, it supports partial and third-party file transfers[3]. GridFTP is the defacto standard file transfer protocol used in Grid computing environments. There exist GridFTP command-line tools and APIs for several programming languages, e.g., for C and Java.

1.3.4 Reliable File Transfer

As the GridFTP protocol itself does not offer reliable file transfer, GT4 offers a service for this purpose. The service is called Reliable File Transfer (RFT) and uses GridFTP as transfer protocol. RFT stores information about active transfers in a database and exposes this information through a Web service interface, i.e., it is able to for example recover failed or interrupted transfers. File transfers are handled like jobs by the RFT service.

1.3.5 Grid Security Infrastructure

Security is very important in Grid computing environments as a Grid is not necessarily limited by organizational boundaries. GT4 thus offers a flexible toolkit for authentication and authorization purposes that is supported in all components of the GT4

[3]Third-party file transfer means server-to-server file transfer.

framework. The system is called Grid Security Infrastructure (GSI). Based on public key encryption, GSI enables users to delegate access rights through *proxy certificates*. Using an X.509 certificate a user can create a proxy certificate that can be used as a credential. This mechanism gives an easy to use single sign-on in a Grid computing environment. Services like GridFTP, RFT and MDS make use of GSI. Furthermore, GT4 offers APIs to easily integrate GSI in new applications.

Chapter 2

Motivation

The Grid Infrastructure Research and Development (GIRD) [2] group's current research focus is on Grid job management. A job in the scope of this thesis means a program that runs on a computer and that uses data stored somewhere, e.g., on the Grid itself, as input. The GIRD group has experience in building services for resource brokering (selection of which resource should be used for a job) and in the construction of tools for automatic management of large numbers of jobs. The difference between management and placement is the kind of decisions that have to be made to fulfill the task. While management means to keep track of the managed resource, i.e., jobs or files, placement is responsible to determine where to place the resource effectively. Even though both management and placement work hand in hand they are different things. This said, also job management and data management cannot be compared with each other as both have different demands to the corresponding resources.

The current frameworks must be complemented with service(s) for data management and data placement. In this context, data placement means not transfer of data, but rather deciding on, and keeping track of the location of files and offering an easy to use interface. The data placement decisions are similar to job placement decisions. A job placement decision can be thought as a short-term resource reservation whereas a data placement decision is a long-term resource reservation. Furthermore, computational and storage resources cannot be compared with each other. There exist systems that try to unify job and data management like, e.g., the Stork system as mentioned in Section 4.3 but in general data and job management are two different tasks to face in a Grid environment. For data transfer itself, there are already established solutions, e.g., the commonly used GridFTP introduced in Section 1.3.3. This said, the problem is formulated as follows.

2.1 Problem Statement

At the moment, submitting a job that makes use of data not available on the Grid is a complicated and time consuming task. A user needs to know a lot about the computing environment before submitting the job, e.g.,

– How many storages are available in the Grid? (resource discovery)

– How can the storages be accessed? (resource discovery)

- How much space is left on the available storages? (resource allocation)
- Am I allowed to read and write files on the storages? (security/policy)
- How can I ensure that I do not overwrite existing files? (file naming collision)
- Was the file successfully uploaded? (reliable file transfer)

Currently, users have to deal with all these issues on their own before submitting a job with external data. The user has to find an appropriate storage, figure out how to upload the external data to the storage, determine the URL where the file can be accessed and include the URL in the job description. While this is a feasible task when there are only a few jobs to submit, it adds too much overhead for large sets of jobs.

2.2 Purpose

The purpose of this thesis is to design and implement a data placement service that:

1. Finds appropiate locations for data.
2. Generates unique identifiers for files, in order to avoid naming collisions.
3. Handles the file transfer.
4. Keeps track of multiple physical copies of each file.

2.3 Usage Scenario

The envisioned scenario is a user who, e.g., from a laptop, wants to run a few jobs. As the user does not want to stay connected to the Grid, e.g., through the Internet, for the duration of the jobs, some input files must be transferred to the Grid before the jobs are submitted. In order to do this, the user contacts the data placement service with the name and size of input file(s). The data placement service looks for appropriate storage and stores the file(s) there. The user receives an URL where the file(s) can be accessed. This URL can be included in the job description when submitting the jobs.

Chapter 3

Developing the Data Placement Service

In this chapter we describe the incremental development process used for the Data Placement Service (DPS) from the first design idea to the reference implementation. At the end of this chapter the reference implementation is presented. In the following the term client is used for both a user and a program using the DPS.

3.1 Initial Design Idea

The initial design idea is a simple data placement system. Figure 3.1 shows a sketch of the system's design. The system components are the StorageBrokerService and the Storage. The Storage can be an FTP or GridFTP server. The StorageBrokerService can be a Web service. It is responsible to allocate enough space on the Storage and return a valid transfer URL (TURL) to the user.

If users want to upload files to the DPS they first ask the StorageBrokerService to get a TURL for the upload, see Step 1 in Figure 3.1. Next, in Step 2, the StorageBrokerService generates the TURL and returns it to the client. The client can then use this TURL to upload the file(s) to the Storage in Step 3. In Step 4, the TURL is included

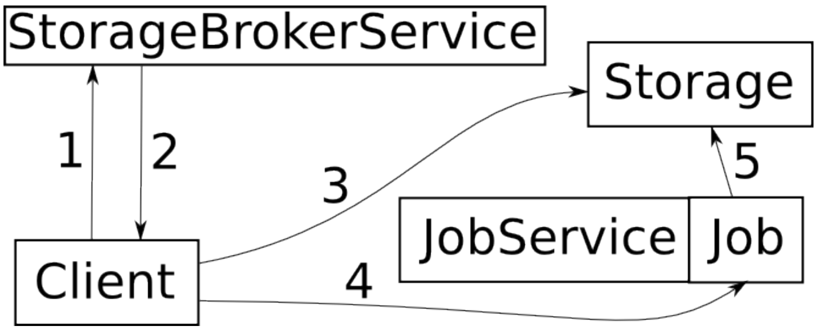

Figure 3.1: Sketch of the first design.

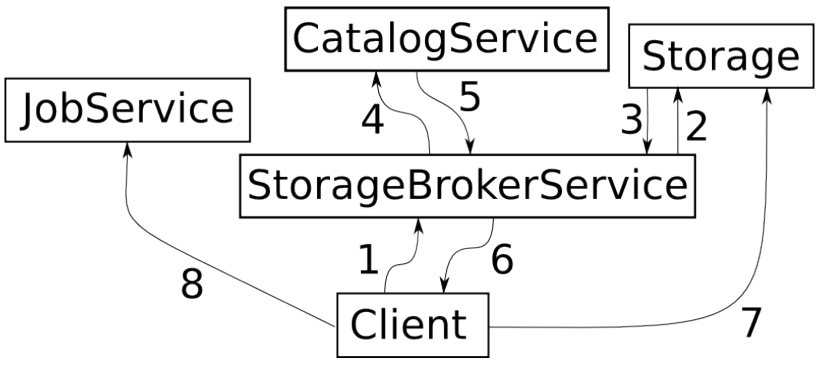

Figure 3.2: Sketch of the reworked design.

in the job description that is sent to the JobService. Finally the job itself can use the TURL to access the file(s) in the Storage in Step 5.

This early design of the system has several shortcomings. The system only supports one Storage at a time. Furthermore, the system is unable to manage any metadata of the stored files. To be more precise, the system acts only as a resource allocator. Due to the limited extensibility and scalability, e.g., the inability to add more Storages, this system design has been discarded.

3.2 Reworked Design

As the Data Placement Service (DPS) should complement the existing job management tools it should be as easy to integrate and extend as possible. Thus a more novel system design compared to the original design idea from Section 3.1 is needed.

Figure 3.2 shows the sketch of the reworked design. On the first sight the differences to the initial design are not that obvious. We do have a StorageBrokerService and a Storage again. What has been added from the component view is the CatalogService. The real differences to the initial design are more to be found on the functional side. The basic functionality of the StorageBrokerService is the same as in the initial design, i.e., it offers a method to get a TURL. The difference to the preceding design is that a file is no longer identified by its TURL but a unique identifier, the logical file name (LFN). The CatalogService is responsible to store and resolve the mapping between the LFN and the physical locations of the file that are from now on called physical file names (PFNs). Thus said, a file can exists on multiple storages but still be identified by one name, its LFN. A physical copy of the file can be obtained by accessing it via the PFN. Table 3.1 shows an example mapping between LFN and PFN. Using this kind of file name abstraction, replication is supported. Replication in the context of the DPS means to store a file on multiple, physically separated storages to improve reliability, performance and fault-tolerance. In this design the Storage is no longer a single file server but a set of file servers. At this point in the design process, there were no concrete ideas how to interface the physical storages.

If users want to upload files in this design they call the StorageBrokerService, see Step 1 in Figure 3.2. The StorageBrokerService queries the Storage to find appropriate storage locations for the client request in Step 2 and Step 3. In Step 4 and Step 5,

Table 3.1: Example mapping between LFN and PFN.

LFN	PFN
myfile	gsiftp://server1:2811/myfile1 ftp://server2:2811/myfile2

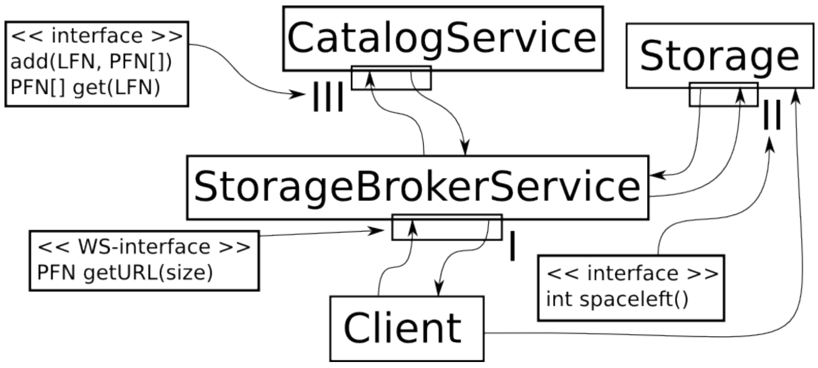

Figure 3.3: Design with interfaces.

the StorageBrokerService registers the PFN(s) by the Storage with the suggested LFN from the client in the CatalogService. Next, the PFN(s) are returned to the client in Step 6. The client uses the PFN(s) to upload the file(s) in Step 7 and finally includes the PFN(s) in the job description in Step 8.

3.2.1 Interface Declaration

The CatalogService, the Storage and the StorageBrokerService all communicate with each other through plain Java interfaces. A client accesses the DPS through a Web service interface. The client gets a valid PFN from the StorageBroker service that can be used for storing the file. This means that the actual file transfer is done by the client. Figure 3.3 illustrates the services and their interfaces. *I*, *II* and *III* enumerate the interfaces.

The CatalogService interface offers two methods for catalog interaction, the `add` and the `get` method. The `add` method gives users the possibility to register LFN-PFN mappings in the catalog. The `get` method is used to resolve LFNs to PFNs. The StorageBrokerService interface offers only one method, `getURL`. The `getURL` method returns a valid TURL. A user can use this TURL to upload files to the DPS. The TURL points to a location with at least `size` bytes allocated space. The Storage interface offers one method, `spaceleft`. The `spaceleft` method returns how much space is left on the storage itself.

As the interfaces also go through the incremental development process, more methods were added later in the development process that are not mentioned at this point as they are not required to understand the DPS.

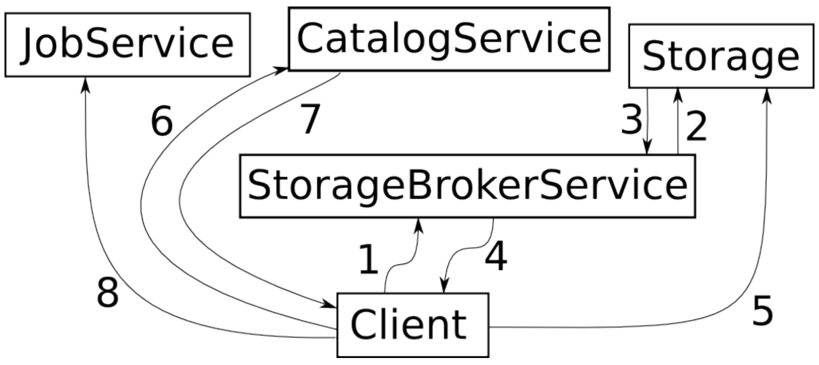

Figure 3.4: Design with detached CatalogService and StorageBrokerService.

3.2.2 CatalogService and StorageBrokerService

Once interfaces are defined a test implementation investigated the feasibility of the design. Although promising, the test implementation identified some new issues. The client is responsible for uploading the file using the given TURL from the StorageBroker. This means that until the client is done uploading the data, the information provided by the CatalogService is inconsistent as the file is not available yet. The reason for this is that the StorageBrokerService registers the LFN-PFN mapping in the CatalogService before returning the PFN to the client, see Steps 4 and 5 in Figure 3.2. Furthermore, there is no possibility for clients to access the CatalogService directly, e.g. to resolve a LFN into PFNs. If a file upload fails after several retries, it is not possible to purge the entry from the CatalogService, leading to inconsistency.

To solve this problem the CatalogService and the StorageBrokerService are detached from each other. A Web service interface is added to the CatalogService, as illustrated in Figure 3.4. From now on, clients are able to access the CatalogService directly, instead of only through the StorageBrokerService. Due to the changed system design the workflow to use the DPS changed as follows. If users want to upload files in this design they call the `getURL` method of the StorageBrokerService, see Step 1 in Figure 3.4. The StorageBrokerService asks the Storage in Step 2 and Step 3 whether there is appropriate storage available for the client request. It then returns the PFN(s) to the client in Step 4. Next, in Step 5, the client uses the PFN(s) to upload the file(s). After the transfer is completed the client registers the LFN-PFN mapping in the CatalogService in Steps 6 and 7 and finally includes the PFN(s) in the job description in Step 8.

3.2.3 StorageDiscovery and StorageSelection

Initially the StorageBrokerService internally holds a list of available storages. This approach is static and not extendable. Hence the internal structure of the Storage-BrokerService is redesigned and two new modules are added: StorageDiscovery and StorageSelection. Figure 3.5 shows the resulting system.

StorageDiscovery

The StorageDiscovery module is responsible to discover all available storages in the Grid. The chain-of-responsibility pattern[9] has been used here to enable the implementation

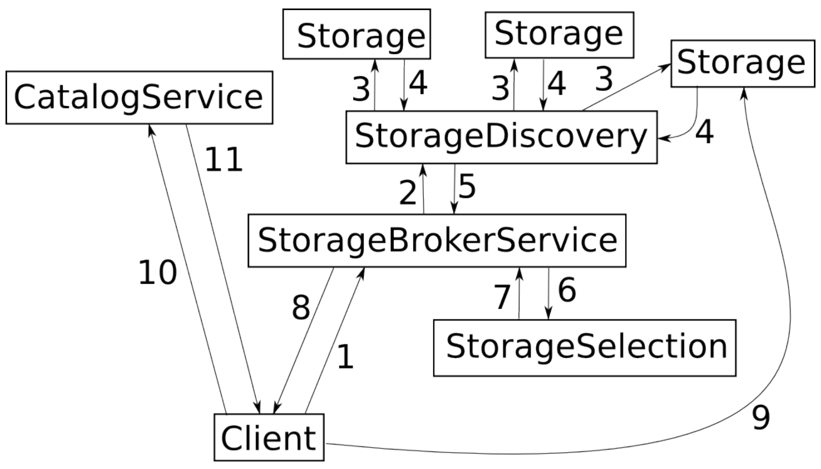

Figure 3.5: Design with StorageDiscovery and StorageSelection.

of dynamic storage discovery. The discovery process itself is not covered here as it is up to the classes implementing the StorageDiscovery interface. Possible implementations include a plain text file with a list of predefined storages or a dynamic discovery mechanism utilizing the GT4 MDS.

Storage

The Storage interface offers methods to check for available free space, allocate space and return a URL to the space, and to return the base URI of the storage. It however turned out that common data transfer mechanisms such as FTP and GridFTP lack support for the free space check operation. Integration with technology that would enable an implementation of this interface is discussed in Section 4.4. The adapter pattern[9] is used here to decouple the StorageBrokerService from the underlying storage technology.

StorageSelection

The StorageSelection module selects storages from a list of storages according to a given heuristic. At the moment two heuristics are supported by the StorageSelection service.

The first one is *random storage selection.* This means that the storages list is returned without any modifications and without any specific ordering. One of the benefits of this heuristic is, that it is very easy to implement and it gives a good distribution of data among available storages. One drawback is, that you cannot be sure that the best storage is selected every time. For the average case random storage selection might deliver best results.

The second heuristic is *worst fit storage selection.* This means that the given list is sorted in descending order by free storage space. While the heuristic itself is very easy to implement, the storages have to provide confident free storage space information. Worst fit is useful to reduce the risk of full disks as it always chooses the storage with most available storage space first.

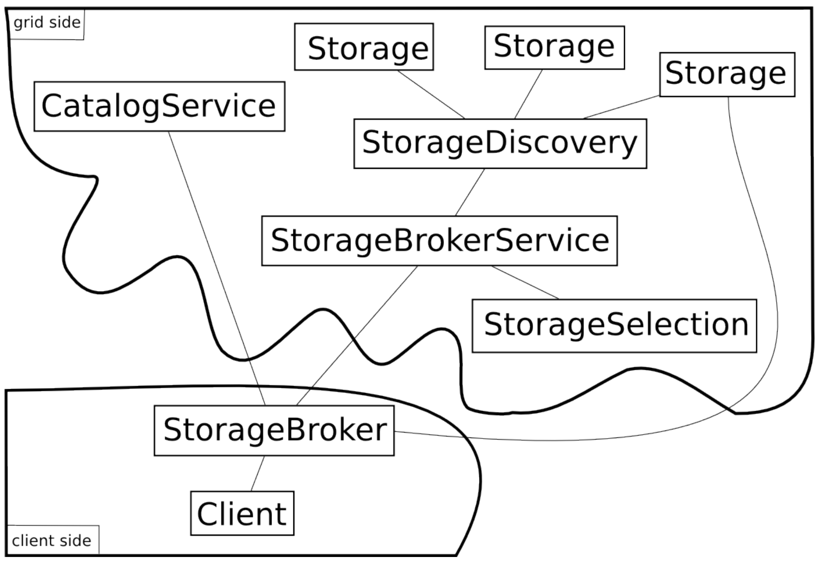

Figure 3.6: The design of the reference implementation.

CatalogService

In the reworked design the CatalogService is storing additional information. The logical file name (LFN), the physical file names (PFN), a hash value for the file and the name of the used hash algorithm are stored. With storing the hash value in the CatalogService users are able to check whether files are uploaded and downloaded correctly and hence detect any file inconsistency. To make the CatalogService more versatile the name of the hash algorithm is stored as well to enable the support for different algorithms.

Returning to the envisioned use-case for the final design, the users initially invoke `getURL` in the StorageBrokerService, see Step 1 in Figure 3.5. The StorageBrokerService uses the StorageDiscovery in Step 2 to get all available Storages. The StorageDiscovery queries all Storages for availability in Steps 3 and 4. Afterwards, it returns the list of Storages to the StorageBrokerService in Step 5. Next, in Step 6, the StorageBrokerService gives the list to the StorageSelection. The StorageSelection orders the selected storages according to the preferred heuristics and returns the new list to the StorageBrokerService in Step 7. The StorageBrokerService returns this list to the client, see Step 8 in Figure 3.5. The client uses the PFN(s) to upload the file(s) in Step 9 and then registers the LFN-PFN mappings in the CatalogService in Step 10 and Step 11. Finally the client includes the PFN(s) in the job description, not shown in Figure 3.5.

This design is the final design and basis for the reference implementation and all further improvements.

3.3 Reference Implementation

In the final design the client is responsible for coordinating the steps required before the data is uploaded and registered in the DPS. To make the whole process more transparent and less error prone the StorageBroker is added. The StorageBroker is responsible for the coordination of all steps shown in Figure 3.5. It resides on the client side and calls the CatalogService and StorageBrokerService in the correct order. The StorageBroker acts like a client library interfacing and utilizing both services and the storages and hence acts as a facade[9]. Thus said, it was introduced to ease the use of the DPS for the user. The following two subsections outline the technical details of the reference implementation.

3.3.1 Client Side

The StorageBroker in the reference implementation is using the Apache Axis UUID mechanisms to generate the globally unique file names. The Java MessageDigest class is used to generate the hash value of the files. The StorageBroker uses the user's credentials to access the StorageBrokerService and CatalogService. Thanks to GSI this is done transparently without any changes in the interfaces. File uploading is performed by the StorageBroker using a generic, extensible Uploader class. The Uploader class currently supports GridFTP and FTP as transfer protocols. The Uploader uses the user's credentials to authenticate itself against the GridFTP server. The StorageBroker is performing simple replication on file upload. Each file that is to be uploaded is replicated on a specific number of storages. If one of the storages is unavailable, users can retrieve other copies of the requested files by asking the CatalogService for another PFN of the file. Currently the number of replicas is predefined in the StorageBroker class.

3.3.2 Grid Side

On the Grid side, the back end of the CatalogService is driven by a PostgreSQL database that is accessed using JDBC. Furthermore, the reference implementation is using a plain text file with predefined storage elements as back end for the StorageDiscovery. The GT4 Web service core is used to implement the services.

Chapter 4

Conclusions

In this chapter we take a look at the usage scenario fulfillment and discuss the limitations of the reference implementation. To broaden our horizon related work in the field of data management and data placement for Grid computing is surveyed. With respect to the current system's limitations ideas for the future work are presented. This Section is concluded with a brief summary of the thesis work.

4.1 Usage Scenario Fulfillment

The reference implementation as described in Section 3.3 is tested on the computing science department's Grid test environment. The test setup consists of four storage servers. On one of the servers a GT4 Java web service container is running to host both the CatalogService and the StorageBrokerService. In addition a GridFTP server is running on this machine. The other three servers are only serving as GridFTP servers. The DPS is invoked using an external computer. In the tests, a plain text file is transferred. Except for the absence of reliable file transfer, the usage scenario from Section 2.3 is fulfilled successfully.

4.2 Limitations

The current reference implementation has some limitations. FTP and GridFTP are the only supported storage and transfer protocols. Even though both protocols are used to transfer files to the DPS they cannot guarantee reliable file transfer. Moreover, as there is no standardized way to determine free space on FTP and GridFTP servers, the DPS is unable to determine the free space on the storages. Instead the service uses the FTP *ALLO* command for space allocation on the respective servers[14]. The implementation of this command seems to be optional for servers, thus it can lead to two possible worst-case scenarios. In the first one no space allocation is performed and thus the usage of the command is needless. In the second scenario space allocation with unknown reservation lifetime is performed. This can lead to a server that claims to be out of storage space but is actually physically unused. This scenario can happen if all file transfers fail or a client refuses to upload a file. Furthermore, the FTP and GridFTP servers have to be configured in such a way that the users have read and write privileges in their home directories. The user has to know the LFN to find the PFNs of the file as the CatalogService does not support directory listing. When a file is no longer needed, it has to be deleted manually. The CatalogService entries can be removed by users through a special method. However, removing files from Storages by hand leads to CatalogService inconsistency.

4.3 Related Work

There exist some projects that focus on data placement in Grid computing environments. This section briefly discusses related work in this area and compares the DPS reference implementation to existing solutions.

The Smart Storage Element (SSE) is developed by the NorduGrid project as part of the Advanced Resource Connector (ARC) software[12]. The SSE is part of ARC's Data Storage Infrastructure (DSI). The DSI consists of SSEs, Data Index Services (Data IS) and the Indexing Services Infrastructure (ISI). The Data IS is interfacing the ISI. At the moment the Globus Replication Catalog (RC) and Replica Location System (RLS) are used as indexing service back ends. The SSE itself is integrated in the HTTPSD server[11]. It is associated with at least one IS. Thus the SSE handles registration of the uploaded files transparently to the user. If a user requests a file from an SSE and the file is not located on the SSE the SSE contacts neighboring SSEs to get a copy of the file. The same mechanism is used to replicate files inside the DSI. HTTPS and HTTPG are used as protocols for data transfer. The SSE is also accessible through the Storage Resource Manager (SRM) interface. The SRM is an open standard for Grid middleware to communicate with site specific storage fabrics[4].

The DPS developed as part of this thesis and the ARC SSE share some similarities. The basic concept of a metadata catalog exists in both systems. In our design the metadata catalog resides in the CatalogService. In ARC's Data Storage Infrastructure the metadata catalog resides in the Data IS and ISI. While the SSE is working almost without human interaction, more manual work is needed when using the DPS.

The dCache data management system is a joint venture between the Deutsches Elektronen-Synchrotron (DESY) and the Fermi National Accelerator Laboratory (FNAL)[8]. One of the key design features of dCache is that the location and multiplicity of the data is autonomously determined by the system based on configuration, CPU load and disk space. The name space is uniquely represented within a single file system tree. In dCache file servers are called pools. These pools can be added and removed at any time. Data availability is guaranteed by replication over several disc pools. Thus dCache is tolerant against failures of pools. The pools are presented as one federated hard drive to the user. The system is automatically load balancing the data between the pools to decrease access times to the files. File access is supported through a native protocol called dCap that can be used via command line tools or a C client API. It is also possible to access dCache through GridFTP. Other protocols can be implemented as needed through a provided interface. Furthermore, dCache supports the SRM interface enabling it to be used in generic data Grid environments like the LHC Grid.

Compared to the DPS the dCache takes a different approach. The level of abstraction that dCache uses for storages is higher than the one in DPS. The dCache system is designed to work with tertiary storage managers and intelligent dynamic data staging. Furthermore, dCache is taking into account the configuration, CPU load and disk space of the storages whereas DPS is only aware of the storage disk space.

The Storage Resource Broker (SRB) is a data management system developed at the San Diego Supercomputer Center (SDSC)[5]. It provides global persistent identifiers for naming files across several storages. The network spread storages are presented as one global file system. A large variety of storage types are supported. SRB supports replication of files between sites and caching of copies of files into local storage systems. To enforce consistency constraints SRB makes use of transactions.

Similar to the DPS the central part of the SRB system design is the metadata catalog. Unlike DPS, SRB supports the creation of collections and thus organizes data in a different way in the catalog. SRB supports federated servers which is not possible with DPS.

The university of Wisconsin-Madison developed a data placement scheduler called Stork[13]. The Stork system handles data placement in analogy with how computational jobs are managed, i.e., data placement jobs are queued, scheduled, monitored, managed and even checkpointed. The system is designed to achieve a maximum of reliability with as little human intervention as possible. A specialty of the Stork system is the support of protocol translation, i.e., the system is able to translate between a variety of transfer protocols, e.g., FTP, GridFTP, SRB, SRM and UniTree. Furthermore, it supports run-time protocol auto-tuning to achieve optimal bandwidth usage. Each storage element in the Stork system can offer different protocols and Stork selects the best fitting protocol taking into account dynamic environmental changes like network congestion, link failures, and even file system errors.

As with DPS, the main focus of Stork is on data placement. Scheduling data placement like computational jobs separates Stork both from DPS and all the other presented projects here.

4.4 Future Work

Even though the reference implementation is working well in the scope of what was intended for the thesis there are ideas on how to improve it.

Currently only GridFTP and FTP storages can be used as data storage facilities in the back end of the DPS. It would be interesting to investigate the interfacing with existing storage broker systems, e.g., SDSC SRB.

As the currently used GIRD Grid job management software is only aware of GridFTP URLs the DPS uses early binding. Early binding in this context means that a PFN is included in the job description before the job is submitted. As the relation between the LFN and PFN is resolved before the job is submitted this binding is called early binding. Conversely, late binding in this context means that the LFN is included in the job description before submitting the job. The resolving from LFN to PFN is done either on the first access or on each access to the file. The motivation to integrate support for late binding in the job management tool is the loose coupling between a job and its physically stored data that improves robustness and fault tolerance, e.g., against failing storage servers. The job only has to keep track of the LFN(s). On demand, the job gets a valid copy of the file. Support for both late and early binding is already implemented in the DPS. The drawback of late binding is that job management tools have to be modified to be catalog-aware in order to resolve LFNs.

The CatalogService stores the hash value of all managed files, i.e., using this information for data consistency checks is straightforward. A data consistency tool could periodically check whether the file replicas on the various storages are consistent. In case of data inconsistency, the erroneous file can be replaced with a correct replica. If there only exists one copy of the corrupted file it is removed from the storage and the corresponding entry is removed from the CatalogService. Data consistency can thus be guaranteed to a certain extent. These kind of consistency checks only make sense if files are read-only, but this is often the case for scientific data, e.g., gathered from instrument output.

The reference implementation offers a basic method of replication. A file is stored on a certain number of storages when it is uploaded using the DPS. To integrate a more dynamic, fault tolerant way of replication, an additional Data Replication Service (DRS) is needed. The DRS can periodically or on invocation check whether the files listed in the CatalogService have enough replicas on the storages according to heuristics.

The reference implementation uses a plain text file to store the list of storages that are available. Storage discovery is thus static, whereas the Grid has a dynamic nature when it comes to availability of resources. Storages may appear or become unavailable at any given time. One future task is to implement storage discovery using the GT4 MDS. In such a scenario, the storages have to register themselves in the MDS and the StorageDiscovery can look up available storages. Elaborating on this idea, a DRS can be invoked if a new storage is registering itself. Thus, an autonomous self-replicating DPS would be feasible.

The reference implementation offers no reliable file transfer. As mentioned in Section 4.4 FTP and GridFTP are the only supported transfer protocols so far. If the DPS would be restricted to use only GridFTP the usage of the GT4 RFT for fault tolerant transfer would be possible. On the other hand the Stork system, as discussed further in Section 4.3, seems to be an interesting approach to this problem as well[13].

Currently the DPS places data across the storages. While this is a good way to utilize all storage space available in the Grid it is not taking into account the requirement of close-to-computation placement. It would be better to introduce a possibility for intelligent data placement (IDP). IDP means that the data is placed as close to the place where it is needed as possible. To add dynamic application-aware IDP an additional module or service is needed that is invoked on LFN to PFN resolution. Furthermore, it is necessary to integrate data placement and job brokering as the StorageSelection needs information about where the job is to be executed and thus where the files have to be stored.

4.5 Summary

This Bachelor's Thesis describes a data placement solution for Grid computing environments. Chapter 1 briefly introduces the field of Grid computing and the used components of the Globus Toolkit 4. Then the motivation for the DPS and the current data placement issues of the GIRD Grid job management tools are discussed in Chapter 2. The several steps that led to the final design and the reference implementation of the DPS are described in Chapter 3. Finally related work and possible future improvements to the DPS are outlined and discussed in Chapter 4.

Chapter 5

Acknowledgements

I would like to thank the following persons who all helped me with this thesis project.

To Johan Tordsson, my supervisor at the Department of Computing Science, thank you for the help with report writing, inspiring discussions and help with the Globus Toolkit 4 internals. I would also like to thank P-O Östberg for the help when I ran into technical problems with the Globus Toolkit 4. To Henrik Thostrup Jensen, thank you for all the insightful suggestions during the requirement analysis, architecture design and the investigation of related work. I also acknowledge Tomas Ögren for the technical support.

Furthermore, I would like to thank Ulrika Back, Anja Brünig, Christoph Lutz and Sebastian Bardt for the moral support in the final stage of report writing.

References

[1] Enabling Grids for E-Science in Europe. `http://www.eu-egee.org/` (last visited March 2008).

[2] Grid Infrastructure Research and Development. `http://www.gird.se/` (last visited March 2008).

[3] The Globus Alliance. `http://www.globus.org/` (last visited March 2008).

[4] A. Shoshani and A. Sim and J. Gu. *Storage Resource Managers: Essential Components for the Grid.* Lawrence Berkeley National Laboratory, Berkeley, 2003.

[5] C. Baru, R. Moore, A. Rajasekar, and M. Wan, editors. *The SDSC Storage Resource Broker*, Toronto, 1998. CASCON'98 Conference.

[6] I. Foster. What is the Grid? A Three Point Checklist. *Grid Today*, July 2002. `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf` (last visited March 2008).

[7] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Intl. J. Supercomputer Applications*, 15(3):200–222, 2001.

[8] P. Fuhrmann. dCache, the Overview. *Deutsches Elektronen Synchrotron*, 2007. `http://www.dcache.org/manuals/dcache-whitepaper-light.pdf` (last visited March 2008).

[9] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman, Amsterdam, 1995.

[10] B. Jacob, M. Brown, K. Fukui, and N. Trivedi. *Introduction to Grid Computing.* IBM Corp., New York, 2005.

[11] A. Konstantinov. The HTTP(S,G) and SOAP Server/Framework. Technical Report NORDUGRID-TECH-9, The NorduGrid Collaboration, 2006. `http://www.nordugrid.org/documents/HTTP_SOAP.pdf` (last visited March 2008).

[12] A. Konstantinov. The NorduGrid "Smart" Storage Element. Technical Report NORDUGRID-TECH-10, The NorduGrid Collaboration, 2006. `http://www.nordugrid.org/documents/SE.pdf` (last visited March 2008).

[13] T. Kosar and M. Livny. A Framework for Reliable and Efficient Data Placement in Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 65(10):1146–1157, 2005.

[14] J. Postel and J. Reynolds. RFC 959: File Transfer Protocol (FTP), October 1985. `http://tools.ietf.org/html/rfc959` (last visited March 2008).

[15] B. Sotomayor and L. Childers. *Globus Toolkit 4 Programming Java Services*. Morgan Kaufmann Publishers, San Francisco, 2006.