

Performance Engineering in Data Grids

Erwin Laure*, Heinz Stockinger and Kurt Stockinger

CERN, European Organization for Nuclear Research, Geneva, Switzerland

SUMMARY

The vision of Grid computing is to facilitate world-wide resource sharing among distributed collaborations. With the help of numerous national and international Grid projects this vision is becoming reality and Grid systems are attracting an ever increasing user base. However, Grids are still quite complex software systems whose efficient use is a difficult and error prone task.

In this paper we present performance engineering techniques that aim to facilitate an efficient use of Grid systems, in particular systems that deal with the management of large scale data sets in the tera- and petabyte range (also referred to as *Data Grids*). These techniques are applicable at different layers of a Grid architecture and we discuss the tools required at each of these layers to implement them. Having discussed important performance engineering techniques we investigate how major Grid projects deal with performance issues particularly related to Data Grids and how they implement the techniques presented.

1. Introduction

Grid computing has emerged over the past few years as a viable technique to enable large-scale resource sharing among geographically distributed collaborations forming a so-called *Virtual Organization (VO)*. In particular, advances in network technologies that significantly increased throughput over wide area connections paved the way to the Grid vision of truly world-wide computing. Many communities already showed interest in leveraging Grid technologies for coping with the ever increasing requirements in their field of expertise. Examples include governmental organizations, biotechnology and health organizations, physicists, and economists, to name but a few. The diversity of the user communities is reflected in the way Grid technology is used. The envisaged usage patterns range from distributed supercomputing

*Correspondence to: Erwin.Laure@cern.ch

Contract/grant sponsor: This work was partially funded by the European Commission program IST-2000-25182 through the EU DataGrid Project.



over high-throughput and data intensive computing to on-demand computing and collaborative computing [15].

The existence of basic Grid middleware such as Globus [14], Legion [17], or UNICORE [35] allowed the construction of first Grid environments, and an ever increasing number of projects in Europe, the U.S., and the Asia-Pacific are looking into higher level Grid services and customized Grid solutions for specific application domains. The aim of the Global Grid Forum (GGF) is to promote and support the development, deployment, and implementation of Grid technologies and applications via the creation and documentation of “best practices” - technical specifications, user experiences, and implementation guidelines [16].

Despite all these efforts Grid computing is not yet in the mainstream of computing, mainly because of the complexity involved in wide area computing and the lack of high level programming environments and associated performance engineering tools and techniques.

In this paper we discuss major performance engineering techniques in the field of data intensive Grid computing (also referred to as *Data Grid*). These techniques aim to minimize costs incurred by accessing data in a Grid environment and thereby increase the performance of single applications and the throughput of the entire system. In order to systematically categorize these techniques we first identify critical performance issues in Data Grids and present a generic layered Grid architecture in which our techniques can be implemented. Subsequently, we examine a set of major Grid projects and discuss how they deal with the performance issues identified.

The remainder of this paper is organized as follows: in Section 2 we introduce the concept of *Data Grids* and the major performance issues to be taken into account in this context, we present some key application areas for Data Grids, and discuss a generic layered Grid architecture. Section 3 presents performance engineering techniques we believe to be most important to cope with the challenges intrinsic to Data Grids. The application of these techniques in major Grid projects is studied in Section 4 and we conclude the paper with some concluding remarks in Section 5.

2. Data Grids

An increasing number of scientific disciplines are using large (potentially distributed) collections of data reaching the tera- and even petabyte scale. These datasets need to be made seamlessly available to large scale distributed user communities who want to analyze this data, potentially using computationally expensive techniques. Grid architectures that deal with the management of such large scale data collections are typically referred to as *Data Grids* [11, 39].

In this section we first give an overview of Data Grid application domains and then present the main characteristics of Data Grids pointing out the major issues pertinent to performance engineering. Finally, we present a generic Data Grid architecture that allows us to deal efficiently with these characteristics.

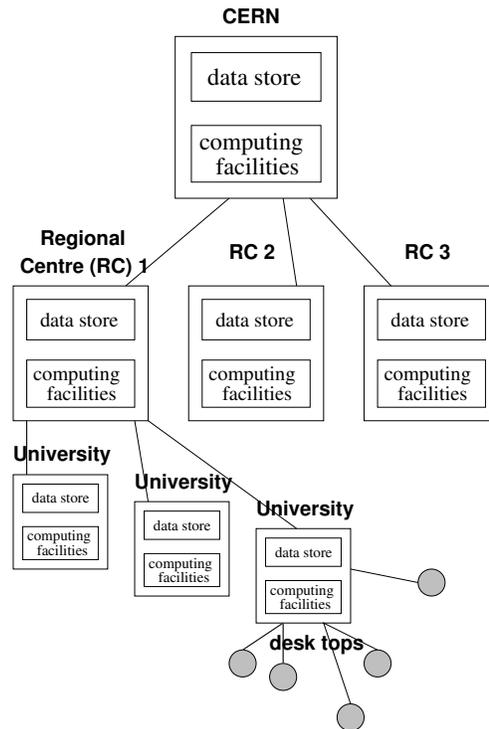


Figure 1. HEP Multi-Tier Architecture

2.1. Data Grid Application Domains

Data Grid technologies can be leveraged in a variety of application domains. As motivating examples we briefly present the highlights of three application areas that are tackled within the EU DataGrid Project [12, 13]: High Energy Physics, Bio Informatics, and Earth Observation.

High Energy Physics The High Energy Physics (HEP) community has the need of sharing information, very large databases (several petabytes) and large computational resources (thousands of fast PCs) throughout its centers distributed across Europe, and in several other countries all over the world. One of the main concerns of the HEP community is to improve the efficiency and speed of their data analysis by integrating the processing power and data storage systems available at distributed sites. The world's most powerful particle accelerator (the *Large Hadron Collider (LHC)*) is currently being constructed at CERN and is expected to produce several petabytes of data per year starting in 2007. Several thousand researchers all over the world will access this data for their analysis; a multi-tier architecture has been

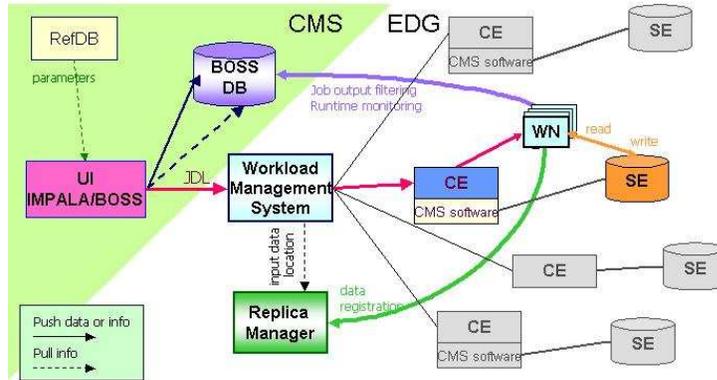


Figure 2. CMS Environment

devised for distributing the data world wide (cf. Figure 1). An overview of the HEP community requirements can be found in [40].

Practical experience, that the HEP community has gained in using Grid environments, is reported for example in [9] where Monte Carlo simulation inside the CMS experiment is described. Monte Carlo simulation is a necessary pre-requisite for physicists to validate their analysis algorithms which will eventually be applied to the real data produced by the LHC. The whole simulation consists of multiple steps involving different computational and storage requirements. The first step (*CMKIN*) takes about 0.5 seconds on a 1 GHz PIII machine and produces a 50 KByte file per physics event. The following (*CMSIM*) step is several orders of magnitudes more complex, producing about 1.8MB of data and taking about 6 minutes per physics event. Both steps are processed in a pipelined fashion, each stage of the pipeline processing 125 physics events. In total, some 500 million events need to be produced before 2007. The whole simulation is automated within the CMS production environment consisting of a database *RefDB* keeping track of the simulation requests and progress, an automatic submission system *IMPALA/BOSS* and the *BOSS* database keeping track of the progress of the individual jobs. Figure 2 illustrates how this environment is integrated with a Grid system (in this case the EDG system, cf. Section 4.1). In the first step, the *CMKIN* jobs run on computing resources available for CMS simulation, the output files are stored on associated storage resources and a replica manager (using the replica catalog system) keeps track of the file location. Subsequently, *CMSIM* jobs are directed by the workload management system to

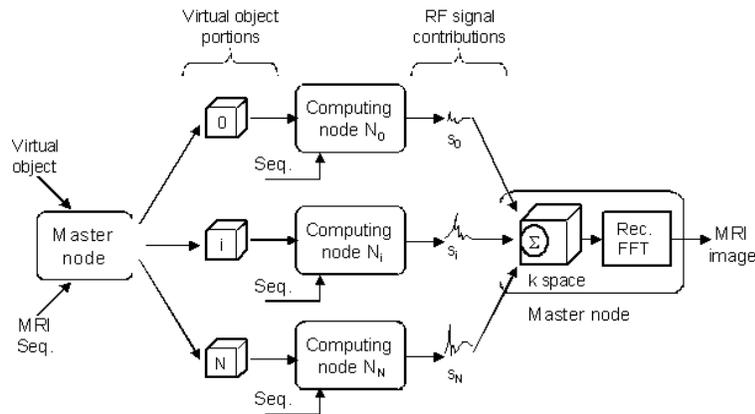


Figure 3. Parallel Biomedical Image Reconstruction

computing resources having access to the previously produced files. Progress is monitored in the CMS BOSS database.

Bio-Informatics The field of Bio-Informatics (also known as Computational Biology) has led to numerous important discoveries and practical applications and is considered as one of today's most promising and expanding scientific fields. The recent explosion of data, acquired by automated gene sequencers and other experimental techniques, requires vast amounts of computing power to analyze the biological functions of genes. There is also a main difference to the two other applications mentioned here since Bio-Informatics deploys many traditional parallel computing models (incl. the usage of MPI). Additionally, thousands of databases contain already collected molecular data, which could be correlated and exploited. Other examples of biomedical applications include mining of biomedical databases and image reconstruction and comparison.

Image reconstruction, for instance, is a computationally intensive task that requires the exploitation of massive parallelism. The reconstruction of high resolution (1024^2) 2D objects takes about 3 days; 3D images with the same resolution (1024^3) will take about 278 years on a standard sequential machine. Fortunately, massive parallelism can be exploited quite easily by running the reconstruction only on subsets of the original image (cf. Figure 3) yielding a parallel efficiency of more than 80% [38].

Content based queries of medical images, on the other hand, first require analyzing image metadata to reduce the search space. Subsequently analysis algorithms are run on the candidate images which may be distributed on many biomedical sites. This kind of search is typically performed through biomedical Grid portals as depicted in Figure 4.

Today, the Bio-Informatics community lacks the necessary infrastructure to process all this data. Therefore, the development of an international infrastructure, which will provide the

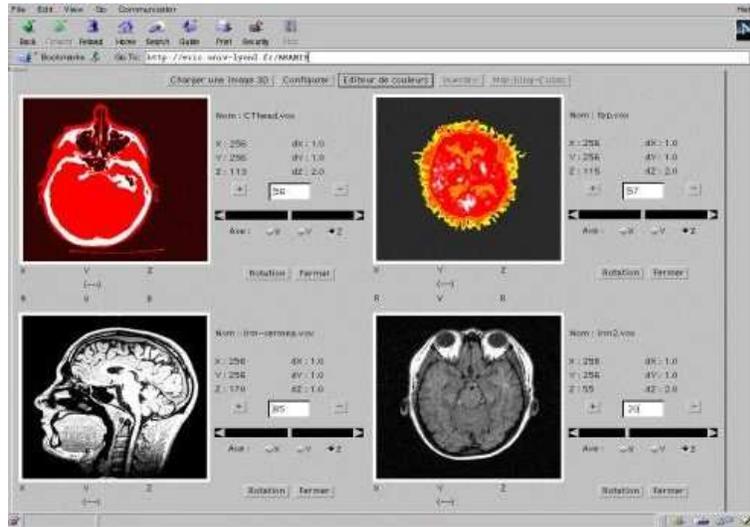


Figure 4. A Grid Portal to Biomedical Image Databases

means of correlating large amounts of data in a transparent way, is of great interest. More information on biomedical requirements can, for instance, be found in [37].

Earth Observation Earth Observation Science applications often require large computational power and access to large data files stored in geographically distributed archives [41]. One example of such applications is GOME (Global Ozone Monitoring Experiment) with the goal to study ozone distribution patterns in the Earth's atmosphere over a given period of time. In 1995 ESA launched the European Research Satellite 2 (ERS-2) including GOME.

Each day 14 data files of 15 MB are acquired. The complete data-set acquired by GOME since June 1995 amounting to 77 GB/year is contained in the ESA mass storage archive. More recent instruments produce up to 5GB/day. The total data volume stored in the ESA archives amounts to more than 800 TB. The data is processed and validated against ground based measurements (collected by LIDAR devices) to produce a global map of ozone concentration and distribution. Figure 5 illustrates how the raw satellite data is processed to so called *Level 2* products which are subsequently validated and visualized. Some indication of the data volumes involved can also be found in Figure 5. A critical issue for running this kind of applications on a Grid environment is the caching of *Level 2* data such that it does not have to be re-produced for every single query.

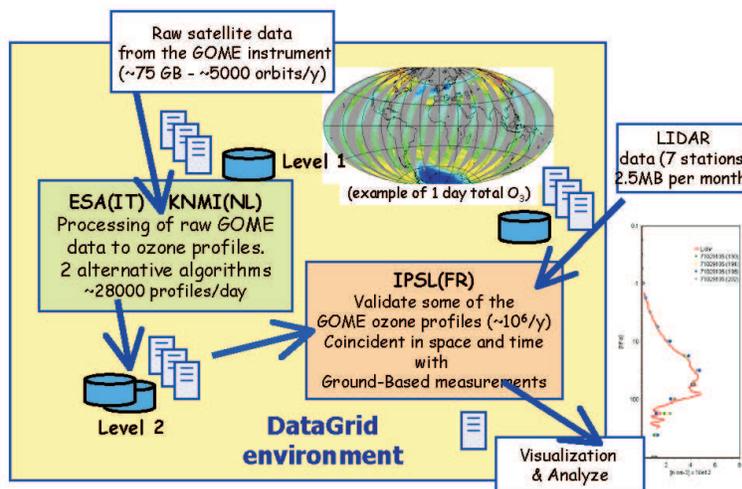


Figure 5. GOME Analysis Architecture

2.2. Data Grid Characteristics

Data Grids are a specialization in the general field of Grid computing dealing with the management of large amounts of data in the tera- and petabyte scale. Such large data volumes need to be stored on hierarchical storage devices, including tape based mass storage systems, which are, for the time being, quite heterogeneous in nature. Network performance is of paramount importance when dealing with large datasets over the wide area. Finally, Data Grid applications are typically high throughput applications rather than traditional high performance ones. In summary, the key characteristics of Data Grids include:

- Large datasets
- Heterogeneous storage systems
- Impact of network characteristics
- High throughput computing

We are now going to discuss each of these items in more detail in order to derive the requirements for the performance engineering techniques we present in the following section. For more definitions on Grid Data Management issues, we refer the reader to [31].

Large datasets Although the common denominator of Data Grid applications is that they deal with large amounts of data, there are significant differences with respect to data creation



and access patterns. For instance, data in High Energy Physics is produced at a single location (the detector) while biomedical data mining applications leverage a large number of already existing databases which are located virtually anywhere in the world. Access patterns range from sequential to chaotic schemes and may be read only or read/write access. Data Grid architectures are required to deal with all these different characteristics, however, better performance can be obtained if the usage patterns of certain applications are known a-priori and major gains can be made if data is used primarily in write once, read often mode.

Heterogeneous storage systems A distinguishing feature of Grid computing is that it is based on heterogeneous environments. This is true for computational (clusters, SMPs, DMMPs) as well as storage devices. Hierarchical storage devices, in particular, exhibit huge differences in access latencies, ranging from seconds to several hours, depending whether the data is on disk or tape and whether the tape is already mounted or not. It is of paramount importance to any Data Grid architecture that these latency variations are appropriately taken into account [33]. Moreover, most hierarchical storage systems are already well established and used for daily production outside the Grid. Grid middleware is therefore required to interface to these legacy systems with as little perturbations as possible.

Impact of network characteristics Dealing with large datasets over wide area networks requires the exploitation of high performance networks in order to avoid bottlenecks. With recent advances, network bandwidth significantly increased enabling fast and reliable data transfer; for instance, GridFTP [1] allows for parallel streams for increasing the throughput of data intensive applications. However, many high performance distributed applications use only a small fraction of the available bandwidth due to improperly tuned network settings. Even though the networking community has been working for a long time on TCP buffer size tuning and parallel streams, these findings are not yet widely used in the Grid community. In addition, network resources are mostly not entirely dedicated to Grid computing which introduces additional uncertainties.

High throughput computing Data Grid applications are typically more interested in achieving high throughput than on exploiting the highest possible performance within a single executable. Classical *High Performance Computing (HPC)* tries to minimize the execution time of a single program by distributing the computational workload among multiple CPUs, often on massively parallel architectures. This mode of computing is also referred to as *parallel computing*. Many techniques and programming languages have been developed that help the programmer in parallelizing applications [2, 20, 26, 29].

On the other hand, *High Throughput Computing (HTC)* is more concerned with environments that can deliver large amounts of processing power over large periods of time [25]. It is more concerned with the overall performance of a set of applications over time rather than the performance of a single applications. Of course, an adequate performance of the individual applications is ultimately also important for HTC, but the description of related techniques is beyond the scope of this paper and can be found in the traditional HPC literature [2]. Grid environments (although potentially being comprised of HPC components [23, 24]) are typically used in a HTC style since they encompass many heterogeneous resources from

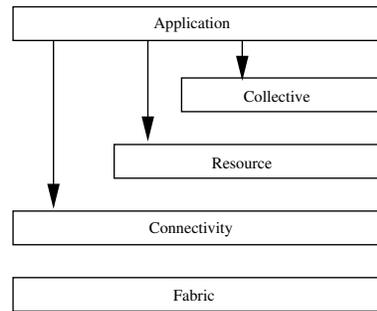


Figure 6. The Layered Grid Architecture

different administrative domains which are used by large heterogeneous user groups. However, as pointed out Bio-Informatics applications often follow also the traditional High Performance computing approach.

2.3. A Data Grid Architecture

A Data Grid architecture is a specialization of a general Grid architecture. It contains specific components to efficiently deal with the problems pertinent to data intensive computing, such as data storage and transport mechanisms, high performance networks, and higher level tools for data and workload management, which are introduced below.

A generic Grid architecture is defined in [21] and consists of multiple layers as depicted in Figure 6. Data Grid requirements are in particular dealt with by a set of components belonging to each of these layers. In the following paragraph we briefly list these components and their associated layers before we discuss their implications to Data Grid performance engineering in Section 3 in greater detail. There are certainly other components that have a large impact on the overall performance of a Data Grid. However, we focus deliberately on the ones concerned with data handling to give the discussion an appropriate focus.

- Components on the *Fabric* layer implement local, resource-specific operations. In Data Grids we are particularly concerned with local **data storage systems** and their interaction with local compute resources.
- The *Connectivity* layer interconnects Grid resources and facilitates data and information exchange among them. Data Grids usually require efficient **network protocols**.
- In the *Resource* layer, which is providing information and management protocols, a Data Grid architecture is concerned with efficient **data transport protocols**.
- Multiple Grid resources are coordinated by components of the *Collective* layer. Here we find global **data management systems** that provide a global view of the data stored in a Data Grid as well as **workload management systems** that perform global scheduling.



Layer	Component
Fabric	Data Store
Connectivity	Networking
Resource	Data Transport
Collective	Data Management Workload Management

Table I. Data Grid Layers and Selected Components

- Finally, *applications* employ the services provided in these layers, mainly from the collective layer, but potentially also access lower level services, either directly or through application domain specific portals.

Table I summarizes the components important to Data Grids and their mapping in the layers of a general Grid architecture.

3. Performance Engineering Techniques

Given the characteristics of Data Grid applications discussed above, the question that will guide us throughout the remainder of this paper is as follows: Which techniques can be leveraged to optimize the usage of Grid resources and what performance information is a pre-requisite to these techniques?

We deliberately focus on techniques related to data handling; performance engineering techniques related to CPU usage are out of the scope of this paper.

The following areas of performance engineering techniques can be identified as the ones having major impact on the overall performance of a Data Grid:

- Data Access,
- Networking,
- Replica Management,
- Replica Optimization, and
- Scheduling.

In discussing these techniques we also identify components that are required in the individual layers of a Data Grid architecture to facilitate them.

3.1. Data Access

For a Data Grid, access to large amounts of data is vital, where data is usually distributed and sometimes replicated. Before we go into detail with *global data access* and *replication*, we need to discuss *local data access* and where data is actually stored.

Several kinds of storage technologies can be considered ranging from simple files systems to complex (distributed) database management systems. Due to the heterogeneity of existing



storage technologies, Grids aim for a *unique interface* to data and storage systems. In addition, a local data store at a given site can consist of secondary and tertiary storage devices with different characteristics as regards access latency, availability and lifetime.

Local data access needs to take care of storing and retrieving data in an optimal way, i.e. performance engineering techniques are required, especially if hierarchical storage devices are used where caching techniques are applied in order to deal with fast data access. For example, a file needs to be opened for read access; another file needs to be stored on a fast cache and then written to a permanent, tertiary storage system. Apart from allowing access to data it is also important to guarantee the required space to store the data produced by an application. This can be achieved by *space reservation* on the target storage resource. Standards like the Storage Resource Manager (SRM) [8] take care of such functionalities and their optimization (see later in this paper).

Other important points are the *data access granularity* as well as *remote data access* (also called *remote I/O*) over a wide area network. Typically, the data access granularity is file based. However, also object based data access granularities are available [31]. This influences the way remote data access is done. Storage appliances like SRM are implemented in the *fabric* layer. In the remainder of this paper, we will mainly focus on the file level granularity and thus mainly talk about file access and file replication.

3.2. Networking

Dealing with large datasets over wide area networks requires the exploitation of high performance networks in order to avoid bottlenecks. Traditional techniques are multi-casting or allowing for parallel streams for file transfers. One goal of a typical Data Grid application is to identify the sites with the best network connections with respect to specific sites. For instance, output data should be stored at those sites that have the highest network bandwidth to the production site. In addition, files should be retrieved from those sites that have the highest network bandwidth to a particular destination site. Note that network performance only is not an optimal criterion for selecting sites since the performance and availability of data servers need to be taken into account, too [32]. However, for a first approximation, network performance is important for deciding where to send or from where to retrieve files.

In order to allow for optimal network usage, tools must be deployed that monitor the network traffic between various Grid sites. These tools typically reside in the *connectivity* layer.

3.3. Replica Management

In a Data Grid, data is typically stored in files which can be spread among geographically distributed Grid sites. The execution time for a job may vary considerably, depending on the computing resource chosen for job execution, the location of data files to which the job requires access, and the data access patterns.

Data replication is considered to be an important technique to reduce data access latency (for reading data) and to increase the robustness of Grid applications. This results in the reduction of job execution time. In more detail, replication involves the creation of identical copies of data files and their distribution over various Grid sites.



Data replication is widely used in database management as well as in distributed systems for performance and fault tolerance reasons. However, the Grid community tackles the problem of replication in a slightly different way [30]. Let us briefly discuss the differences with respect to performance optimization.

Typical transaction-based database management systems are designed to provide consistent states of data for concurrent updates. One of the most important replication features is to provide *update synchronization* among replicated data items. Hence, database replication is often a trade-off between supporting high performance read access by increasing the *replication factor* [32] (i.e. the number of identical replicas for a given file) and thus reducing the write performance due to higher update requirements. Moreover, traditional database management systems are deployed in the local area with moderate amounts of data.

On the other hand, in a Data Grid, data volumes reach up to several petabytes. What is more, data produced by typical large scale scientific experiments is often read-only and thus no update synchronization like that for databases is required. This allows increasing the replication factor for both speeding up read performance and increasing fault tolerance without the trade-off of keeping multiple replicas in synchronization.

Replica management involves a number of low level steps: the data needs to be physically copied using appropriate transport mechanisms, the correctness of the copy needs to be checked, the location of replicas needs to be stored in replica catalogs, and finally, consistency among replicas needs to be preserved both in case of spurious inconsistencies (such as in case of hardware failure or attacks) and in case of writable replicas. Components in the *fabric* layer such as homogeneous data access mechanisms, efficient data transport protocols in the *resource* layer, and replica catalogs in the *collective* layer are required to fulfil these tasks. However, higher level replica management services should be provided by the Data Grid middleware that hide most of the complexity of the underlying system and provide the user with a uniform interface. These services are found in the *collective* layer.

3.4. Replica Optimization

One of the goals of replica optimization is to minimize file access times by pointing access requests to appropriate replicas and pro-actively replicating frequently used files based on access statistics gathered. Replica optimization techniques can thus be divided into:

- replica selection
- replica initiation (automatic creation/deletion)

Replica optimization is tightly coupled with replica management. Often, optimizers are an integrated part of higher level replica management systems and thus found in the *collective* layer.

Replica Selection The replica selection aspect of replica optimization aims to select the best replica with respect to network and storage access latencies. In other words, if for a given file several replicas exist, the optimization algorithm determines the replica that should be accessed from a given location. Similarly, the algorithm may also be used to determine the best location for new replicas, i.e. where to store additional replicas of an existing file.



The performance improvement achieved by using a replica optimizer (a Grid service that takes care of replica selection and initiation) critically depends on the design of an algorithm that selects one of the replicas of the requested resource. This topic was intensively discussed in the context of Internet services [19] and distributed database management systems. Each of the replica selection algorithms may be designed with different goals, different metrics and different mechanisms for measuring the metrics that are used. For instance, the replica selection algorithm may aim at maximizing network throughput, reducing the network traffic on “expensive” links, or reducing the response time of jobs.

Most replica selection algorithms aim at selection of “close” replicas to either reduce response time or the load on network links. Closeness can be defined by various metrics such as response time, latency, round-trip time, network bandwidth, number of hops, or geographic proximity.

Replica Initiation The goal of replica initiation is to trigger replication and thus the creation of new replicas dynamically. The decision about when to create new replicas can be based on the file access history in order to optimize data locality for frequently requested files. This assumes that based on historical events we can partially predict future file access. By increasing the replication factor of a file, one can achieve better load balancing of file requests to less loaded sites but also fault tolerance in case some sites become unavailable.

In addition to creating new replicas, a replica optimizer can also decide to delete existing replicas due to several reasons: a file is infrequently requested, access space is required etc. In this way, the replica optimizer can play a role in optimizing storage space based on access patterns of a file.

3.5. Scheduling

Scheduling in Grid environments (i.e. *global scheduling*) is a much more complex task than scheduling on local clusters (i.e. *local scheduling*) which is accomplished e.g. by local batch schedulers like LSF, PBS, or the Maui scheduler. This is due to a number of inherent features of Grid environments not present in typical HPC, cluster, or local systems:

- resources belong to different administrative domains imposing different local policies;
- some resources may be restricted to accept only a certain subset of jobs managed by a Grid scheduler; similarly, not all resources accessible may qualify to run a Grid job due to certain job requirements;
- resources may have different performance characteristics, in particular with respect to CPU performance, storage system access, and network connectivity;
- Grid schedulers do not have full control over the resources, they do not *belong* to the Grid scheduler;
- Grid schedulers do not have full information on all jobs in the system: there will be multiple Grid schedulers as well as local schedulers concurrently scheduling the same resources.

As a consequence, a Grid scheduler typically performs the following steps [28]:

1. *Resource Discovery*: The set of accessible resources is filtered according to authorization constraints and application requirements.



2. *System Selection*: More detailed information on the state of the systems selected by the previous step is sought (e.g. current system load or status of queues) based on which a final decision on where to run the jobs is taken.
3. *Job Execution*: Eventually, the job is to be executed at the chosen site. This step may involve advanced reservation, job preparation, and job monitoring. Advanced Grid schedulers may re-consider their decision dynamically and re-schedule jobs if certain circumstances like high system load prevent the job from being executed efficiently.

In a Data Grid the task of the Grid scheduler is even more difficult since job execution efficiency not only depends on the computational characteristics of the execution site but also on the data access characteristics. Hence, a Grid scheduler has to take into account not only the computational requirements of a job but also its data requirements when making the scheduling decision. The additional steps the scheduler has to perform can be clustered to the main traditional steps as follows:

1. *Resource Discovery*: explore the location of all the input data the job requires as well as storage resources that are capable of storing the output data.
2. *System Selection*: the access time (mainly affected by latency and bandwidth) to input data and output data location needs to be taken into account when making the scheduling decision. The replication systems discussed above, in particular the replica optimization systems, may assist the Grid scheduler in this task and replicate data to places from where it can be accessed more efficiently.
3. *Job Execution*: jobs requiring frequent access to data are particularly sensitive against fluctuations in network performance. It is therefore important to dynamically re-consider the scheduling decisions taken as well as the decision from where to access data. This re-considering may result in job re-scheduling and/or replication of data to new places on the fly.

Grid schedulers are typically part of *Workload Management Systems* residing in the *collective* layer. These systems not only perform the scheduling task but also monitor the status of jobs and provide transactional mechanisms to cope with problems during job execution.

4. Prototype Systems

Many projects are currently building Grid infrastructures including middleware solutions suitable for performance engineering in Grids. In this section we review some of the major Grid middleware projects with respect to the issues discussed above. We selected projects that explicitly deal with Data Grids. Note that our selection is by no means comprehensive and that the four projects do not necessarily have the same aims. For example, Globus and Condor typically provide more lower level middleware whereas EDG and SRB are more higher level tools. After the discussion of the individual projects we provide a summary of our findings.



4.1. European DataGrid (EDG)

<http://www.edg.org>

Data Access EDG has a hierarchical storage model where data can reside on three different levels, namely on disk, on the disk cache of a mass storage system, and on tape. Data is managed via a unique interface that is called *Storage Element (SE)*. The SE has similar functionality to a *Storage Resource Manager (SRM)* [8], a proposal for a unified interface to hierarchical storage systems. The main purpose of the SE is data and space management.

In addition, the SE provides performance information about the access latencies of files with various sizes. The access latency is calculated based on the location of the file, file access history, and job queues of the mass storage system.

Networking The network traffic on the EDG testbed is monitored at certain intervals using iperf [34]. This information is used for estimating the file transfer time between various sites on the wide area. The calculation of the total file access latency consists of the estimated transfer time and the access latency of the underlying storage system.

Replica Management The Replica Management Service called Reptor [18, 22] provides access to fast and secure transfer mechanisms based on GridFTP. Replica information is kept consistent in the distributed replica catalog referred to as the Replica Location Service [10].

Reptor manages data and associated meta-data by taking into account information provided by several Grid monitoring tools such as the Information Service and the Network Monitor. The implementation of Reptor is based on the web service paradigm in accordance with the emerging Open Grid Service Architecture.

Replica Optimization Within the replication framework, optimized replica selection is achieved by calculating the access latencies of the replicas and choosing the one with minimal access costs. Both network transfer times and storage access latencies are taken into account.

Scheduling The goal of EDG's Workload Management System (WMS) is to manage Grid resources conveniently, efficiently and effectively [36] based on the following components: the User Interface, the Resource Broker, the Job Submission Service, and the Logging and Bookkeeping Service.

The user interacts with the WMS via a User Interface that allows, among others, to submit jobs, control the execution of a job, and retrieve a job's output. A job is represented by a Job Description that is expressed via a Job Description Language (JDL). Jobs may either be sequential or parallel applications using MPI [26].

The task of the Resource Broker is to find the best match between the requirements of the jobs and the available Grid resources. In order to perform this optimization, the Resource Broker consults the Replica Management Service to retrieve information about the location of required input files and the Information Service about the current load of the computing and storage resources. The result of the optimization is a matching computing resource where the executing job has access to all resources specified in the Job Description Language, such as



CPU requirements or storage space. It is worth noting that parallel applications are scheduled to run on a single Grid site (which may be a large cluster or HPC system) and are not distributed among multiple sites.

4.2. Globus

<http://www.globus.org>

Data Access Globus provides a number of tools for data management in Grid systems. GridFTP [1] is a high-performance, secure protocol based tool for parallel data transfer, partial file transfer, and third-party (server-to-server) data transfer. Globus does not support mass storage system access. In addition, a reliable file transfer service (RFT) is included in the latest Globus version.

Networking Globus does not support networking optimization.

Replica Management Globus also includes tools for managing data replicas, namely the Replica Catalog and the Replica Manager. Currently, Globus supports two versions of catalogs, an LDAP based and an SQL-based flavor. The latter is called RLS - (Replica Location Service) [10] and was jointly designed and developed between Globus and the “Data Management Workpackage” of the EU Data Grid Project to maintain distributed information of replicas. The Replica Manager is based on GridFTP and the LDAP based replica catalog to provide basic replica management features.

Replica Optimization Globus does not support replica optimization.

Scheduling The Globus Resource Allocation Manager (GRAM) processes the requests for resources for remote application execution, allocates the required resources, and manages the active jobs. In addition, GRAM provides updated information about the capabilities and availability of the computing resources to the Metacomputing Directory Service (MDS), i.e. the Information Service.

GRAM provides an API for submitting and canceling a job request, as well as checking the status of a submitted job. The specifications are written by the user in the Resource Specification Language (RSL), and is processed by GRAM as part of the job request.

However, data locality issues are not considered in the scheduling phase. What is more, automatic job scheduling is not supported.

4.3. Condor

<http://www.cs.wisc.edu/condor/>

Data Access Storage requirements of Condor jobs are managed by NeST [7], which is a flexible, software-only storage appliance. NeST provides a generic data transfer architecture that supports multiple data transfer protocols such as GridFTP and NFS. Due to its dynamic,



self-adapting behavior, it runs efficiently on a wide range of hardware and software platforms. Moreover, NeST allows for management of storage space, mechanisms for resource and data discovery, user authentication and quality of service.

NeST does not support access to mass storage systems.

Networking Condor does not support networking optimization.

Replica Management Condor does not support replica management.

Replica Optimization Condor does not support replica optimization.

Scheduling Condor-G [4] is a task broker to schedule and manage thousands of jobs in a typical distributed Grid computing environment. It provides job monitoring, logging, notification, policy enforcement, fault tolerance, credential management.

Dependencies between jobs are managed by DAGMan (Directed Acyclic Graph Manager) which is a meta-scheduler for Condor. A directed acyclic graph (DAG) can be used to represent a set of programs where the input, output, or execution of one or more programs is dependent on one or more other programs. DAGMan is responsible for scheduling, recovery, and reporting for the set of programs submitted to Condor.

In Condor, data transfers for copying, replicating and staging are called Data Placement (DaP) activities. Currently, a DaP scheduler named *Stork* is being developed to intelligently schedule both computational and data jobs to increase disk usage and throughput while decreasing I/O latencies.

Note that Condor does not support automatic scheduling as it is done in EDG.

4.4. SRB

<http://www.npaci.edu/DICE/SRB>

Data Access The Storage Resource Broker (SRB) [3] is a client-server based middleware tool which provides distributed clients with uniform access to different types of storage devices, diverse storage resources, and replicated data sets in a heterogeneous computing environment.

SRB provides its own Mass Storage System (MSS) that enables users to economically build their own mass storage system in which data migrate automatically between cache and tape. In addition, it is integrated with the High Performance Storage System (HPSS) for archival storage.

Networking By incorporating automatic parallel data transfers the SRB optimizes and matches the transfer to the network and server export rates, resulting in robust and fast transfers.

Replica Management The SRB client-server system solves many problems associated with traditional file systems. The SRB supports virtual collections consisting of digital entities scattered across distributed, heterogeneous storage resources, including file systems, archives,



and databases. These differences are transparent to users, negotiating all protocols, access permissions, etc. across multiple sites.

Metadata about the files is handled by the MCAT metadata catalog which allows for searching, accessing, and managing collections of data. MCAT is implemented with relational database technology and has been ported to work with Oracle, SQLServer, DB2, Sybase and Postgres.

The SRB synchronizes replicated data to ensure accurate mirroring and reliable file transfers using a persistent transfer mode that automatically retries transfers as needed.

Replica Optimization A so-called SRB Grid Brick provides a cost-effective, uniform data management environment using a Linux PC running only SRB to manage 1 to 2 terabytes of local disk. One can group multiple Grid Bricks into a single logical resource in order to guarantee a single file space. Grid bricks can be distributed over a network but will still appear as a single resource.

Replica selection is based on storage latency, however, network based replica selection is not supported.

Scheduling Using distributed SRB Grid Bricks, users can employ seamless, round-robin, random placement load-sharing. However, automatic wide area scheduling is not supported.

4.5. Summary of Observations

Project	Data Access	Networking	Replica Mgmt.	Replica Opt.	Scheduling
EDG	yes	yes	yes	yes	yes
Globus	partially	no	yes	no	partially
Condor	partially	no	no	no	partially
SRB	yes	partially	yes	partially	partially

Table II. Performance Engineering Methods of Selected Prototype Systems.

On having discussed the main Grid middleware implementations, we will now summarize the main performance engineering approaches of these projects. Table II provides an overview of the performance methods that are implemented in the selected prototype systems we discussed before. In particular, we identified the following levels of performance optimization:

- Access estimation and optimization of storage devices: especially for data intensive applications access latencies to tertiary storage systems are the main bottleneck. Access estimates have to take into consideration disk and tape latencies and turn around times of the batch systems that queue data intensive jobs. Important optimization techniques include intelligent caching of frequently used files on those storage systems that have, on the one hand, the lowest access latencies, but are, on the other hand, close to the CPU that request this data.



- Access estimation and optimization of network devices: Typical approaches include monitoring the network traffic and supporting parallel streams for file transfers.
- Replica management: One of the goals is to replicate files to those places where files are accessed.
- Replica optimization: Based on access latencies and network bandwidth estimates, replicas are selected.
- Job scheduling: Jobs are scheduled to those places that have the best CPU capacities taking into account data localities. Typical goals are to optimize the throughput of the whole system and balance the load in an intelligent way over the available Grid resources.

Grid middleware is very complex software that has to deal with all the uncertainties and changing characteristics of a Grid environment. To better analyze the impact of certain middleware components, Grid simulators have been developed. For instance, inside EDG a Grid simulator called *OptorSim* [5,6] has been developed in order to understand the performance issues of a complex Data Grid environment. The simulator is based on the topology of a typical Data Grid and comprises several scheduling and data replication algorithms to study the performance of Grid jobs with various access pattern characteristics. In addition, one can also study the impact of various network configurations and storage systems. One of the goals is to provide a set of Grid benchmarks similar to well established benchmarks from the database community to evaluate the performance of query optimizers.

Within U.S. Grid projects, a simulator with similar goals was implemented which is called *ChicagoSim* [27].

5. Conclusions

In this paper we provided an overview of Grid performance engineering mechanisms for optimizing the performance of tasks (jobs) within a typical Data Grid. We categorized the techniques into five aspects: data access, networking, replica management, replica optimization and scheduling. Next, we evaluated four selected Grid prototype implementations and discussed the currently implemented performance engineering approaches.

The goal of this paper was to provide a characterization and a set of mechanisms for improving the performance of currently deployed Data Grids. In addition, these performance engineering tools can serve as the basis for establishing a framework to compare the performance of different Grid environments. Our vision is to create typical Grid benchmarks similar to what is known in the database community for comparing the performance of query optimizers of different software vendors. The categorization work in this paper is a vital prerequisite for achieving this goal.

ACKNOWLEDGEMENTS

We would like to thank our colleagues from the EU DataGrid project for their support throughout this work and many stimulating discussions as well as Ian Willers (CERN) for proofreading this paper.



REFERENCES

1. W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke. GridFTP Protocol Specification. GGF GridFTP Working Group Document, September 2002.
2. G. Andrews. *Concurrent Programming: Principles and Practice*. Benjamin/Cummings, 1991.
3. C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *CASCON'98 Conference*, Toronto, Canada, November 1998.
4. J. Basney, M. Livny, and T. Tannenbaum. High Throughput Computing with Condor. *HPCU news, Volume 1(2)*, 1(2), June 1997.
5. W. Bell, D. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. Simulation of Dynamic Grid Replication Strategies in OptorSim. *International Journal of High Performance Computing Applications*, 17(4), in print.
6. W. Bell, D. Cameron, R. Carvajal-Schiaffino, P. Millar, K. Stockinger, and Floriano Zini. Evaluation of an Economy-Based File Replication Strategy for a Data Grid. In *International Workshop on Agent based Cluster and Grid Computing at CCGrid 2003*, Tokyo, Japan, May 2003.
7. J. Bent, V. Venkataramani, N. LeRoy, A. Roy, J. Stanley, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Flexibility, Manageability, and Performance in a Grid Storage Appliance. In *Symposium on High Performance Distributed Computing*, Edinburgh, Scotland, July 2002.
8. I. Bird, B. Hess, A. Kowalski, Don P., R. Wellner, J. Gu, E. Otoo, A. Romosan, A. Sim, A. Shoshani, W. Hoschek, P. Kunszt, H. Stockinger, K. Stockinger, B. Tierney, and JP. Baud. Srm joint functional design. Global Grid Forum Document, GGF4, Toronto, February, 2002.
9. D. Bonacorsi, P. Capiluppi, A. Fanfani, et al. Running CMS Software on GRID Testbeds. In *Computing in High Energy and Nuclear Physics, CHEP03*, La Jolla, CA, USA, March 2003.
10. A. Chervenak, E. Deelman, I. Foster, L. Guy, A. Iamnitchi, C. Kesselman, W. Hoschek, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Giggie: A Framework for Constructing Scalable Replica Location Services. In *SC'2002*, Baltimore, USA, November 2002.
11. A. Chervenak, I. Foster, C. Kesselman, and C. Salisbury and S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 23:187–200, 2001.
12. European DataGrid Project. <http://www.eu-datagrid.org>.
13. EU DataGrid Project. Description of Work. EU DataGrid Project. Technical Annex V5, January 2002.
14. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl. J. Supercomputer Applications*, 11(2):115–128, 1997.
15. I. Foster and C. Kesselman, editors. *The Grid*. Morgan Kaufmann, 1999.
16. The Global Grid Forum. <http://www.ggf.org>.
17. A. Grimshaw, W. Wulf, et al. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1):39–45, January 1997.
18. L. Guy, P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. Replica Management in Data Grids. Technical report, GGF5 Working Draft, July 2002.
19. Guyton and M. Schwartz. Locating Nearby Copies of Replicated Internet Servers. In *Proceeding of ACM SIGCOMM'95*, 1995.
20. High Performance Fortran Language Specification Version 2.0, January 1997.
21. I. Foster and C. Kesselman and S. Tuecke. The Anatomy of the Grid. *The International Journal of High Performance Computing Applications*, 15(3):200–222, Fall 2001.
22. P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. Advanced Replica Management with Reptor. In *International Conference on Parallel Processing and Applied Mathematics*, Czestochowa, Poland, September 2003. Springer-Verlag.
23. E. Laure. *High Level Support for Distributed High Performance Computing*. PhD thesis, Institute for Software Science, University of Vienna, Austria, February 2001.
24. E. Laure. OpusJava: A Java Framework for Distributed High Performance Computing. *Future Generation Computer Systems*, 18(2):235–251, October 2001.
25. M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for High Throughput Computing. *SPEEDUP Journal*, 11(1), 1997.
26. Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 1.1*, June 1995.
27. K. Ranganathan and I. Foster. Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing*, 1(1):53–62, 2003.
28. J. Schopf. Ten Actions When SuperScheduling. GGF Scheduling Request for Comments:8.5, July 2001.



29. D. Skillicorn and D. Talia. Models and Languages for Parallel Computation. *ACM Computing Surveys*, 30(2):123–169, June 1998.
30. H. Stockinger. Distributed Database Management Systems and the Data Grid. In *18th IEEE Symposium on Mass Storage Systems and 9th NASA Goddard Conference on Mass Storage Systems and Technologies*, San Diego, California, April 17-20 2001.
31. H. Stockinger, O. Rana, R. Moore, and A. Merzky. Data Management in a Grid Environment. In *European High Performance Computing and Networking Conference (HPCN2001)*, Amsterdam, The Netherlands, June 25 - 27 2001.
32. H. Stockinger, K. Stockinger, E. Schikuta, and I. Willers. Towards a Cost Model for Distributed and Replicated Data Stores. In *9th Euromicro Workshop on Parallel and Distributed Processing PDP 2001, IEEE Computer Society Press*, Mantova, Italy, February 7-9 2001.
33. K. Stockinger, H. Stockinger, L. Dutka, R. Slota, D. Nikolow, and J. Kitowski. Access Cost Estimation for Unified Grid Storage Systems. In *4th International Workshop on Grid Computing (Grid2003)*, Phoenix, Arizona, November 17 2003.
34. A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. *Iperf Version 1.7.0*. <http://dast.nlanr.net/Projects/Iperf/>.
35. Unicore. <http://www.unicore.org>.
36. WP1—Workload Management. Definition of Architecture, Technical Plan and Evaluation Criteria for Scheduling, Resource Management, Security and Job Description. EU DataGrid Project. Deliverable D1.2, September 2001.
37. WP10—Grid-aware Biomedical Applications. Requirements for Grid-Aware Biology Applications. EU DataGrid Project. Deliverable D10.1, September 2001.
38. WP10—Grid-aware Biomedical Applications. Report on the 1st Bio-Testbed Release. EU DataGrid Project. Deliverable D10.3, April 2003.
39. WP12—Project Management. The DataGrid Architecture. EU DataGrid Project. Deliverable D12.4, February 2002.
40. WP8—HEP Technical Working Group. Long Term Specifications of LHC Experiments. EU DataGrid Project. Deliverable D8.1b, September 2001.
41. WP9—Earth Observation Applications. Requirements Specification—EO Application Requirements for GRID. EU DataGrid Project. Deliverable D9.1, August 2001.