Faculta de Informática

Universidad Complutense de Madrid

Máster en Investigación en Informática

Proyecto fin de Máster en Ingeniería de Computadores

Data Location-Aware Job Scheduling in the Grid

Application to the GridWay Metascheduler

Autor: **Antonio Delgado Peris**
Correspondencia: **antonio.delgadoperis@ciemat.es**

Director: **Eduardo Huedo Cuesta**

Fecha: **20 de junio de 2008**
Curso académico **2007 - 2008**

Para la realización de este trabajo se utilizaron recursos facilitados por el Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas. Con estas líneas quiero expresar mi sincero agradecimiento.

This work was carried out using resources provided by the Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas. I hereby would like to express my sincere gratitude.

Por la presente, autorizo a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y el prototipo desarrollado.

Antonio Delgado Peris.

Madrid, 20 de junio de 2008.

Data Location-Aware Job Scheduling in the Grid

Application to the GridWay Metascheduler

Antonio Delgado Peris

20 - 06 - 2008

Palabras clave: grid, computación distribuida, planificación de tareas, asignación de recursos, tareas intensivas en datos, GridWay, EGEE.

Keywords: grid, distributed computing, job scheduling, resource allocation, data-intensive tasks, GridWay, EGEE.

Director de proyecto: Eduardo Huedo Cuesta.

Máster en Investigación en Informática.
Proyecto fin de Máster en Ingeniería de Computadores.

Faculta de Informática. Universidad Complutense de Madrid.

Table of Contents

Visión general

Las tecnologías grid permiten integrar recursos heterogéneos distribuidos. Las aplicaciones del mundo científico y empresarial ganan así acceso a vastas infraestructuras que traspasan límites administrativos. En este entorno distribuido, los trabajos de computación pueden ser ejecutados en cualquier recurso y se hacen necesarias estrategias para seleccionar el destino óptimo para cada tarea, con el objetivo de maximizar la eficiencia de las aplicaciones. Los metaplanificadores como GridWay son los encargados de enviar los trabajos a los recursos y en los momentos apropiados.

Este proyecto se ocupa de la planificación de tareas que consumen grandes cantidades de datos ya existentes. Muestra cómo los metaplanificadores necesitan tener en cuenta dónde se hallan los datos para poder minimizar replicaciones innecesarias y para optimizar la eficiencia de los trabajos. También describe las modificaciones realizadas en GridWay para hacerle capaz de aceptar requisitos de datos de los trabajos y flexible para implantar diferentes algoritmos en función de ellos. Se repasan y prueban con GridWay algunos de estos algoritmos y finalmente se propone una aproximación al problema general de planificación de trabajos y colocación de datos en el grid.

Como ejemplo de motivación, se discuten las exigentes necesidades en datos del experimento CMS, participante del proyecto EGEE. Éstas servirán como referencia para evaluar los algoritmos propuestos y nuestra propia implementación.

La organización de este trabajo es la siguiente: el capítulo 1 introduce el concepto de grid, describe las principales tecnologías grid usadas hoy en día y el metaplanificador GridWay, y presenta la infraestructura EGEE. El capítulo 2 estudia los retos planteados por la planificación de tareas con importantes necesidades de datos, y repasa cómo los afronta EGEE y qué proponen otros autores para resolverlos. El capítulo 3 se apoya en las conclusiones del capítulo 2 para describir nuestra aproximación al problema, las modificaciones realizadas en GridWay y los experimentos llevados a cabo para probar nuestra solución. Finalmente, el capítulo 4 resume las conclusiones del proyecto y las posibles mejoras de trabajo futuro.

Overview

Grid technologies have brought the promise of seamless integration of distributed heterogeneous resources. Applications from both industry and research communities will gain access to vast infrastructures across administrative boundaries. In this distributed environment, computing tasks may run anywhere and strategies to select the best possible destination for each piece of work become fundamental to improve the applications' efficiency. It is the duty of metaschedulers such as GridWay to allocate computing jobs to the most appropriate resources at the proper time.

This work focuses on the scheduling of jobs that consume huge amounts of existing data. It shows why metaschedulers need to take into account the location of data in order to optimize job efficiency and avoid unnecessary data replication. It also describes how GridWay has been modified to accept data requirements in job requests and made flexible to implement different data-aware scheduling algorithms. Some of these algorithms are reviewed and tested with the new GridWay and a proposal for the solution of the general problem of data placement and job allocation in the grid is presented.

As a motivating example, the demanding data needs of the CMS experiment, within the EGEE project, are discussed. This serves to evaluate proposed algorithms and our own implementation.

This text is organized as follows. Chapter 1 introduces the concept of grid, describes the main grid technologies in use today and the GridWay metascheduler, and presents the EGEE infrastructure. Chapter 2 studies the challenges posed by the scheduling of jobs with important data requirements. It reviews how this is dealt with in EGEE and what other authors propose to address it. Chapter 3 builds on the lessons of Chapter 2 and describes our approach to the problem, the modifications performed in GridWay to make it data location-aware and the experiments that were carried out to test our solution. Finally, Chapter 4 summarizes the conclusions of the project and the improvements that future work may bring.

1. Introduction

Previous to the description of the problem under study, this first chapter introduces the reader to the environment in which it appears; namely, the grid. In this context, the grid middleware, the software relevant to the scheduling of jobs, and its reference implementation, the Globus Toolkit, are presented. Lastly, the middleware component we are working with, the GridWay metascheduler, and the reference target for our implementation, the EGEE infrastructure, will be described.

1.1. What is a Grid

Since the publication of [1] in 1998, the term g*rid* [1] has become increasingly popular. Expressions like g*rid computing* or g*rid infrastructure* have been used in a variety of contexts and the exact definition of the concepts they refer to have not always been completely clear. Subject of hype to certain degree, there are probably numerous cases in which the term has been used with a different meaning from the one that was originally intended. Within the scientific community, however, it is generally accepted that a *computing grid infrastructure* (or just a *grid*) denotes a dynamically changing set of computing resources distributed among different administrative domains that are accessed in a seamless and secure way ([2], [3], [4], [5]). Consumers of these resources are individuals and institutions grouped in what have been called *virtual organizations* (VOs) [3]. These VOs act according to well defined rules stating which grid resources are shared and who is allowed to access them and under which conditions.

According to the previous definitions, the term *grid computing* refers to the protocols and interfaces, tools and techniques that enable the distributed access to those resources in the grid for the different virtual organizations.

[1] In the literature, the term *grid* is often capitalized (*Grid*). In our view, this use is not appropriate; at least not yet. We can compare this with the capitalization of the word Internet. While there is only one worldwide Internet, there is not yet a global integrated grid, but a multiplicity of technologies and infrastructures. *The Grid* does not exist yet. We will therefore use the lowercase term throughout the text.

The software that vertebrates and makes grid computing possible is usually called *grid middleware* [3], which will be studied in more detail in Section 1.2. The reference implementation of such software is the one provided by the Globus project [6]. We will describe Globus in more detail in Section 1.3.

Although in theory one can envision a global unique grid where only defined authorization policies limit access to the world-wide set of computing resources, this is not yet the case today. There are, however, many examples of corporative grid infrastructures, as well as several research projects currently offering a grid for scientific collaboration. Most of these efforts are built on Globus protocols and software, either directly deploying the Globus Toolkit or using middleware which is based on Globus. Examples of such grid projects are the German Grid Initiative (D-Grid) [7], the TeraGrid project [8] or the European Union's initiative Enabling Grids for E-SciencE (EGEE) [5]. Since it is the target for our implementation and the framework for our tests, we will describe the EGEE project in more detail in Section 1.5.

1.2. Grid Middleware

The definition of grid that we have presented is general enough to leave space for virtually any type of computing resource to be shared; e.g.: processing power, storage capacity or remote harnessing of any kind of scientific instrumentation. But even for a single and well defined type of resource, several different implementations may exist, possibly offering incompatible interfaces. Such situations make it often impossible for an application to directly access all available resources in a seamless way. It is at this point where the grid middleware comes to help by *virtualizing* those resources so that a single interface is offered to applications. Through this uniform interface, any of the implementations can be accessed. But as important as the uniform access is the advertisement of these resources using a coherent description schema, so that consumers can discover the existing services and choose the one they want to access [3]. Thus, summarizing up to now, the middleware must provide the means to describe, discover and use available resources in the grid.

In order to enable sharing of any kind of resource, the middleware should make use of standard, open, general-purpose protocols and interfaces. This is opposite to application-specific solutions.

Other characteristics that are usually required to grid middleware ([2], [3]) are the following:

- The middleware provides the functionality above in a secure way, what at least means that users and services are authenticated and authorization policies are enforced. Otherwise, services are useless for most purposes.
- Coordinated resources are not centrally controlled. This is implied by our definition of grid, where resources are distributed across different administrative domains. If a centralized control existed, other solutions that do not make use of grid technologies would be sufficient.
- Nontrivial qualities of service are delivered, such that the coordination of resources brings out a combined utility of the grid that is greater than the sum of the parts. Otherwise, grid technologies would not be of too much interest.

Within the grid community, it is also common to find the distinction between *core grid middleware* and *higher-level grid middleware*. Although it can all be considered middleware, in the sense that it does not directly offer functionality to users but rather to applications, the former refers to the set of most basic services required in a grid, while the latter extends this functionality by building on it. As explained in Section 1.4, the GridWay metascheduler, whose scheduling algorithm is the subject of study of this project, can be considered higher-level middleware making use of Globus services, which constitute the core middleware it is building on.

1.3. The Globus Toolkit

The Globus project was introduced by Ian Foster and Carl Kesselman in [6]. In the terminology of that article, Globus addressed the problem of metacomputing; i.e.: managing execution environments in which distributed computing resources are connected through high speed networks. After those same authors started to use the

concept of grid in [1], the *Globus Toolkit* (first version released in late 1998), became the reference software for the development of grid systems and applications. Since that moment on, it has been considered by many as the *de facto* standard for grid computing.

The Globus project became later the current *Globus Alliance,* which, according to the statement in its web page[2], is *an international collaboration that conducts research and development to create fundamental grid technologies*. With the help of their contributors, they keep developing and releasing the Globus Toolkit as open source software. This toolkit includes libraries and services required to build core grid middleware, as described in section 1.2. The architecture they define follows the *hourglass model*, according to which a small set of core abstractions and protocols are at the neck, while many different resource technologies can be supported (bottom of the hourglass) and a lot of diverse applications can be built upon them (top of the hourglass).

The services offered by the Globus Toolkit include, amongst others, resource monitoring and discovery services (*MDS*), resource allocation and management (*GRAM*), a public key security infrastructure (*GSI*), and file transfer services: GridFTP for a secure and scalable data transfer and, building on this, the reliable file transfer service (*RFT*). These constitute the basics for secure and transparent access to distributed resources.

The Globus Toolkit is now at version 4 (GT4). This version, described in detail in [9], follows the principles of the Open Grid Services Architecture (OGSA) firstly proposed in [10]. Currently OGSA specifications are being developed by the Open Grid Forum (OGF)[3], and they aim to describe a service-oriented architecture for grid computing. In GT4, OGSA services are implemented as state-aware web services (WS), following the WS-Resource Framework (WSRF) specifications developed by the OASIS[4] standards organization, and introduced in [11]. Services in WSRF are similar to traditional web services, with the main difference that they support operations to

[2] http://www.globus.org

3 http:www.ogf.org

[4] http://www.oasis-open.org

remember *state* between different invocations (i.e., they are *stateful*, while traditional web services are *stateless*).

Previous versions of Globus (GT1 and GT2) did not use a web services-based approach. Several existing grid infrastructures still use middleware build upon pre-WS Globus Toolkit components. In particular, this is the case of the EGEE infrastructure, which has been used as target grid for this project. In any case, for what concerns resource selection in GridWay, the fact that web services are used or not is not really relevant, and all the work should apply to both WS or pre-WS grid infrastructures.

1.4. The GridWay Metascheduler

So far we have described the possibilities of grid technologies and in particular of the Globus Toolkit in a general and abstract way; we have been referring to distributed computing resources that can be accessed by applications leveraging grid services. What these *computing resources* may in reality represent is open to essentially anything that can be offered by a service and applications want to consume in a distributed computing environment. There are however a few paradigmatic examples, main targets of grid infrastructures in current operation, which are of most interest for our present work. These are *processing power*, *storage capacity* and *data*.

We will start with the second one. The grid may be used store sets of files, database records, or any other format of electronic data on any kind of storage technology. The storage systems available to hold users data are in this case the resources. Applications may want to discover the type and capacity of those storage systems before selecting them for their data. Once that is done, they access those resources; i.e.: they store (*write*) the data.

Once the data has been stored somewhere in the grid, it can be considered a resource of its own. An application may want to retrieve some piece of data for the user, copy it to a different location from where it is residing, or consume it as input for some calculation. All these use cases require the ability to locate the existing data, comply with data access authorization policies, and effectively accessing the data (*reading* it).

Let us finally consider the last example of computing resource as listed above. Consuming *processing power*[5] is the ability to run applications in some node in the grid. Traditionally, processing power has been offered by supercomputers or *Distributed Resource Management Systems* (DRMS). These are often called *batch systems*. A batch system accepts users computing tasks (usually called *jobs*) and executes them in one of the local resources it controls. If all the local resources are busy, then the batch system is able to queue task requests until some resource is available. These systems usually offer additional capabilities like the support of different priorities (per user, per task length…), or the implementation of fair share mechanisms.

For what DRMS concerns, there are several implementations available today; e.g.: Torque[6], LSF[7], Sun's Grid Engine (SGE)[8] or Condor[9]. These systems typically offer incompatible user interfaces. The grid middleware provides the means to retrieve information about each existing batch systems and also a common interface to access any of them; i.e.: to pass a task (job) to them, so that they make sure it is executed. In Globus, the MDS is used to get information about processing resources and the GRAM service is used to access them.

As indicated, MDS and GRAM are core level services that make it possible to run jobs on the grid, but their use requires a high level of expertise and is probably too complex for end user applications. The duty of deciding when and where (among the known resources) jobs are run and controlling their execution is called *workload management*. This involves discovering what processing resources are available, selecting the best ones for a given task, preparing the remote systems (optionally copying necessary input files), dispatching the jobs, watching the evolution of the status of these jobs (making sure they are run OK) and recovering the output produced by them.

---

[5] The term *computing resources* is often used to mean *processing power resources*, as opposed to data resources. Up to know we have used this in a generic way, meaning either the first or the second.

[6] http://www.clusterresources.com/pages/products/torque-resource-manager.php

[7] http://www.platform.com/Products/platform-lsf

[8] http://gridengine.sunsource.net/

[9] http://www.cs.wisc.edu/condor/

In a batch system, the component in charge of the workload management is called *scheduler*. In the grid, the component that manages the job dispatching to available resources (each being a batch system) may be called *metascheduler*. GridWay [4] is a metascheduler that uses Globus core services to offer higher-level functionality to applications and users, thus simplifying the use of the grid.

The GridWay project [10] is being developed by the Distributed Systems Architecture[11] Group from the University Complutense of Madrid[12]. In January 2007, GridWay became a full Globus project [13] and, starting with Globus Toolkit 4.0.5, GridWay is now included by default with the Globus Toolkit distribution.

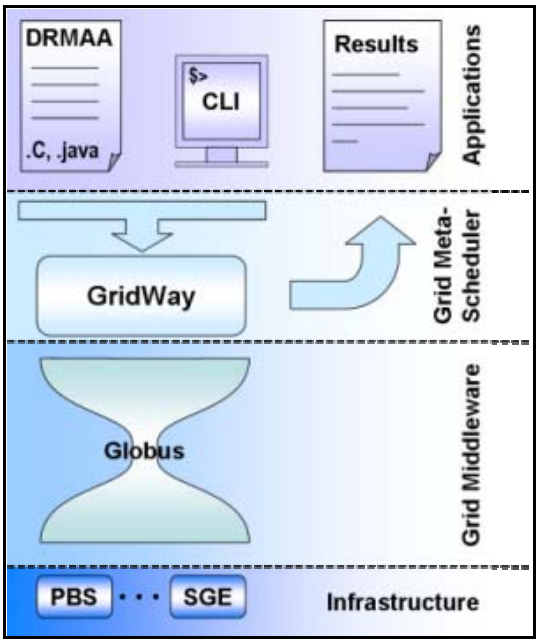

Figure 1: GridWay, high-level middleware for Globus-based grids

The GridWay metascheduler offers a DRMS-like command line interface for users to submit jobs and watch their evolution. It allows users to submit, kill, migrate, monitor and synchronize jobs, as well as to watch information about available resources. It also supports the *Distributed Resource Management Application API*

[10] http://www.gridway.org

[11] http://dsa-research.org

[12] http://www.ucm.es

[13] http://dev.globus.org/wiki/GridWay

(DRMAA) for job submission, which is an OGF standard [12]. The relation between GridWay, Globus and the user applications is illustrated in Figure 1.

GridWay was designed with a modular architecture (see Figure 2), in which several components may be loaded as plugins. This is the case of the *Transfer Manager* (used to stage input files), the *Information Manager* (used to retrieve resource information) and the *Execution Manager* (used to submit jobs). Thanks to this, GridWay is able to, for instance, submit jobs using WS and pre-WS GRAM.

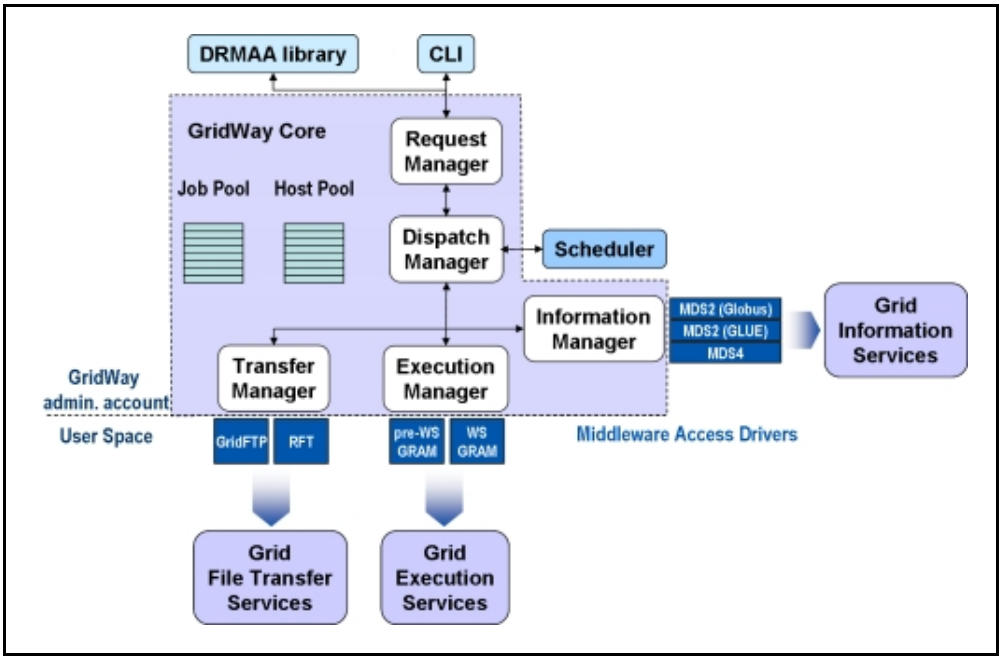

Figure 2: GridWay, modular architecture

GridWay's decision regarding where to send user jobs is based on configurable system policies as well as on per-job user requirements. We will refer to this process of job distribution as *job scheduling*, and we will focus our attention on it in Chapter 2.1.

1.5. The EGEE Project

The Enabling Grids for E-sciencE (EGEE) project[14], described in [5], is one of the largest (if not the largest) multi-disciplinary grid infrastructures in the world. It presently brings together more than 120 organizations to create a computing system

[14] http://www.eu-egee.org

composed of around 250 sites (centres of resources) in 48 countries and with more than 70 000 CPUs available to some 8000 users of the scientific community. Funded by the European Commission, the EGEE project was originally started on the 1st April 2004, and it has been renewed two times since then. The current project is in fact called EGEE-III.

From a technical point of view, the architecture of software services of EGEE is described in [13]. The middleware in which this architecture is based is called gLite [14]. gLite was built on Globus version 2 (GT2) with many additions and enhancements.

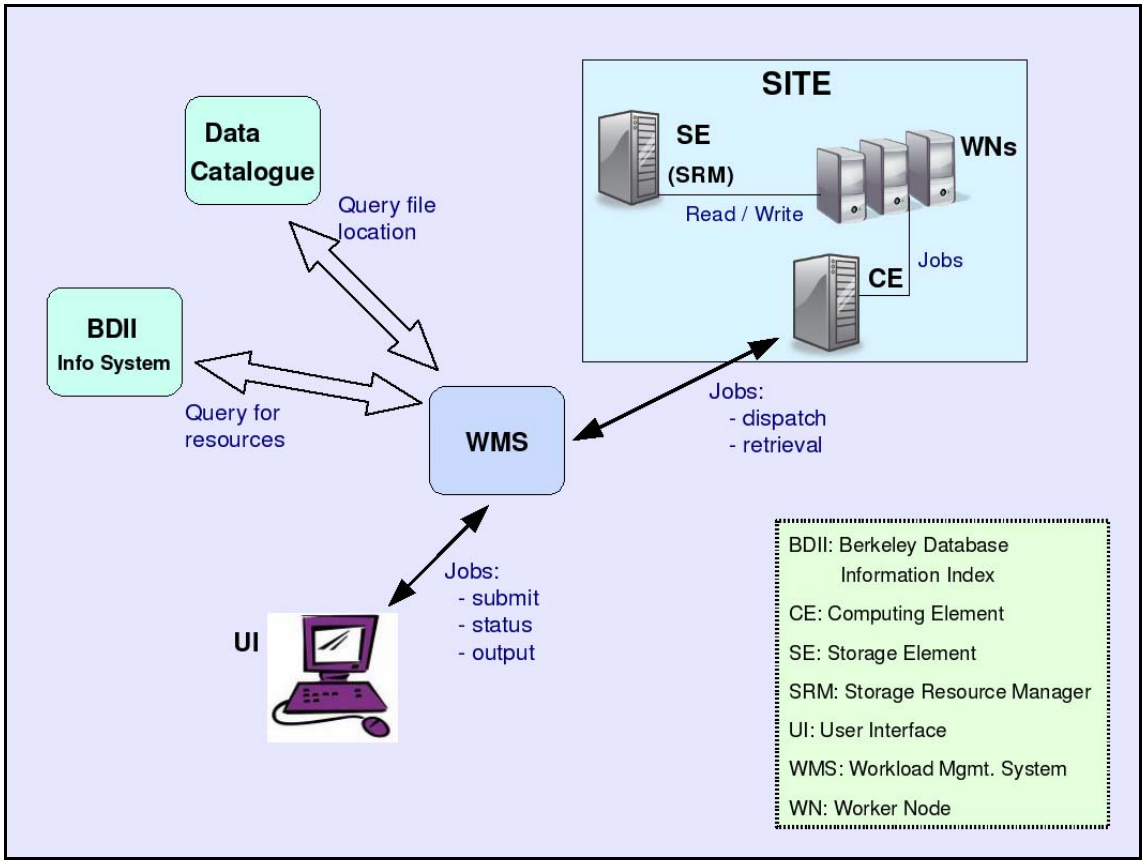

Figure 3: EGEE, main gLite services

Figure 3 illustrates some of the main elements composing the EGEE infrastructure (those of particular interest for our discussion). The u*ser interface* (UI) represents simply the client software from where the user interacts with the grid, by submitting jobs but also, not shown in the figure, by querying about and accessing data. Physical centres of resources are called *sites*. They hold processing and storage resources, which

are respectively grouped in the so called *computing elements* (CE) and *storage elements* (SE). The storage elements present a uniform interface to ease data access and management. It is called the *Storage Resource Manager* (SRM). User jobs arrive logically at the CE, but once there they run in any of the *worker nodes* (WN) sitting behind it. From a worker node, they read and write data from and to the SE. The *Berkeley Database Information Index* (BDII) is the implementation chosen for the information system (based on Globus MDS). The data catalogues contain information about what data there is in the grid and where it is located (in which SE). Finally, the *workload management system* (WMS), which was previously also called *resource broker* (RB), is the metascheduler of the system. These are all services for grid-wide or VO-wide use.

We will discuss scheduling in more detail in Chapter 2.1. A brief summary follows: the WMS accepts job requests from the users and dispatches them to appropriate CEs based on the user requirements, the CE characteristics (as informed by the BDII) and possibly data location information (if required). Currently, the WMS and the CEs in EGEE can still communicate using the Globus GRAM protocol (pre-WS). In fact, during its evolution, gLite has replaced some Globus components with its own implementations, but it is still largely compatible with Globus-based infrastructures. For what this work is concerned, it is sufficient to say that GridWay is capable of gathering information from and submit jobs to EGEE resources. If job submission ever changed in gLite, GridWay could always be adapted accordingly thanks to its modular design (i.e.: adding a new Execution Manager plugin).

Some of the bigger virtual organizations making use of the EGEE infrastructure (in terms of requirements and number of users) are coming from the high energy physics world and in particular from the collaborations running experiments in the Large Hadron Collider (LHC)[15], located at the European Laboratory for Particle Physics (CERN)[16]. For our work, the most relevant characteristic of these experiments are the incredibly large amount of experimental data that they need to process and store. Taking

[15] http://lhc.web.cern.ch/lhc

[16] http://www.cern.ch

the example of CMS[17], the VO we will use for our test case, the experiment will produce tens of petabytes of data, which will be analyzed by physicists located in more than a hundred sites around the world. The challenge that the management of this amount of data imposes has led to the development of new middleware to locate, transfer or access data in both gLite [15] and within the VO software frameworks [16].

Finally, we must note that the development of the gLite middleware is only part of the work undertaken within the EGEE project. A production grid infrastructure of such size requires big efforts in side but essential tasks like user support, services monitoring and general operational activities. The project also supports a large testing infrastructure for pre-release testing, provides lots of documentation and performs dissemination activities like the organization of tutorials and training courses.

[17] http://cms.cern.ch

2. Resource Allocation and Data-Intensive Jobs

Grid metaschedulers aim to dispatch computing tasks to the most appropriate resource at the most convenient time. The goal is usually the completion of the tasks in the shortest possible time. As will be explained, when applying this to tasks consuming big amounts of data, certain particular aspects must be considered. In order to deeply understand these implications, we will study how the scheduling is performed in our reference example in EGEE, and review what other proposals addressing this problem exist in the literature.

2.1. Job scheduling

In the previous chapter, we described how grid technologies make it possible for users and applications to gather information about what resources are available in the grid, how they can be accessed, and what their most significant characteristics are. With this information, a user or program is able to choose one of these resources and access it. In what regards processing power, resources are batch systems (computing elements, in EGEE terminology), and accessing them means sending computing tasks, *jobs*, to them.

We have also seen that, in order to ease workload management, a service called metascheduler (also called WMS) may be used to accept job submission requests and, based on available resources information, allocate jobs to batch systems, i.e.: *schedule jobs*. This is illustrated schematically in Figure 4.

Other systems that do not follow that pattern have been also proposed for job scheduling. For instance, auction-like systems where each resource bids for a given job request have been described. Calana [17] is an example of this. Its authors argue that this system eliminates the dependence on a global information system that unavoidably holds some old information (gathered at some time in the past). As [18] indicates, however, auction-like systems have communication problems of their own, since job proposal have to be distributed to all resources and bids must be gathered and compared. We will not deal with this kind of systems since GridWay, the subject of our study, is a

metascheduler that follows the more traditional approach described before, and this approach is also the one used in the most important grids in operation today.

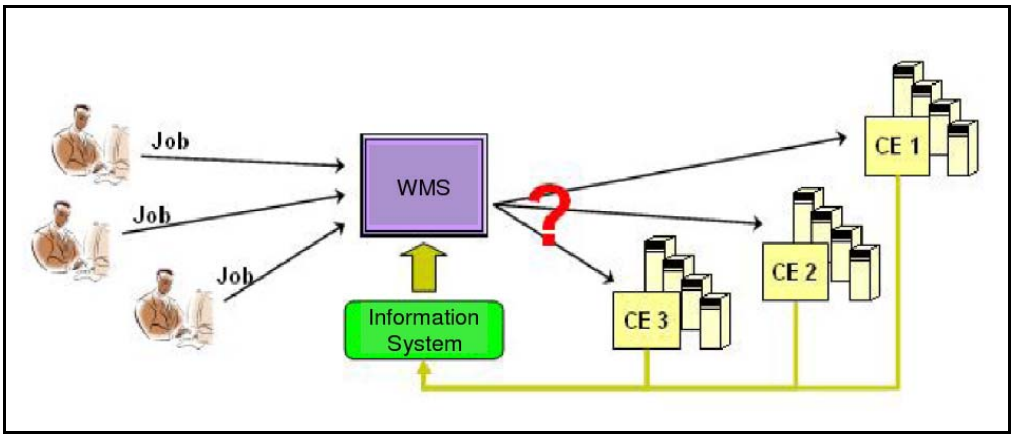

Figure 4: Job scheduling in the grid

The mechanism used to schedule jobs and the algorithm utilized to select the target of the submissions affect the efficiency of the job's execution in the grid. This *efficiency* refers to the time elapsed from job submission to retrieval of the output, usually referred to as *job turnaround time*. Metaschedulers should aim to minimize this time.

Notice that we have defined efficiency for the case of a single job, but we can also consider the efficiency of a bunch of jobs as referring to the average of their individual turnaround times. If we extend this definition to all the jobs of a VO or in the whole grid, we could talk about global efficiency. Typically, a user will be interested in the efficiency of her jobs, because she wants her results back as soon as possible. However, a VO should be concerned with improving the global efficiency of a VO and in general grid operators should pursue the increase of the global efficiency for the whole grid.

The parameters that the resource selection algorithm must consider when scheduling a job to the grid are typically indicated by the user herself. For the EGEE WMS and for GridWay, the job description (*job template* in GridWay's terminology) includes a *requirements expression* and a *rank expression*. The first one is a logical expression, built with logical and relational operators, variables and fixed values (integers or strings). The whole expression is evaluated to True or False, and only those hosts for which the result is True are eligible to run the job at hand. The rank expression is an arithmetic expression in which variables and integers are permitted. This expression is

numerically evaluated for each resource satisfying the requirements expression. This result is then used to order the resources from most to least desirable, and the job is submitted to the resource with highest rank.

The variables that can be used in the requirements and rank expressions are mostly those defining the resource characteristics and their values can be retrieved from the information system[18]. A user may ask for nodes with a certain operating system version or a particular software release installed. She may also want to rank resources according to the speed of their processors. This implies that a common scheme defining the attributes to be published in the information system and their format is shared between users and resources. In GT2 the MDS schema was used, while the Grid Laboratory Uniform Environment (GLUE) [19] is enforced in the EGEE framework and other related projects. In GridWay, there are information manager plugins capable of understanding both MDS schema and GLUE.

Apart from the resource characteristics retrieved through the information system, GridWay takes into consideration three additional aspects when ranking matching resources. These are the *current resource behaviour*, the *past grid usage*, and a *fixed priority rate*. The former refers to the statistics that GridWay collects regarding failure rates and execution times (for the input files transfer, for the queue waiting time and for the execution itself). The second deals basically with the same type of statistics but collected for some time in the past (querying an internal database that keeps the history of resource behaviour). The value for the latest criteria –fixed priority rate- is set by the user for specified resources or for resources discovered by specified information manager plugins. By means of the fixed priority rate, the GridWay administrator can statically assign an offset to the priority value of certain privileged resources.

The values obtained according to the three previous criteria are combined with the evaluation of the user supplied rank expression to compute the priority of a resource. Firstly, the values of the current and past usage statistics are added and normalized, and

[18] For GridWay, the complete list of usable variables can be found in its user guide: http://www.gridway.org/documentation/stable/userguide/

that result is added to the normalized values of fixed priority and rank expression. As shown in Equation 1, every priority value (P_i) is given a configurable weight (W_i).

$$(1) \quad P_{total} = W_{usage} (W_{hist} P_{hist} + (1 - W_{hist}) P_{current}) + W_{fixed} P_{fixed} + W_{rank} P_{rank}$$

Finally, GridWay also applies an exponential linear back-off strategy to ban resources for which submissions have failed a certain time ago. After a failure, and for a period of time that is incremented after each successive error, resources are simply not considered for submission.

2.2. Data-Intensive Jobs

In the previous sections, we have described data and processing power as though they were two completely independent resources. For some use cases, they can be considered as such. A simulation job may just require an executable and some input parameters to run for hours. Alternatively, a user may search some grid data to download to her workstation. In these cases, there is no relation between the data and the CPU resources. But in many occasions, in particular for scientific applications, user applications require data already present in the grid to perform some computation with it. Moreover, the amount of data required by a job is sometimes such that the time required to get it is not negligible in comparison with the time invested in its processing.

Let us consider the example of the CMS experiment (following [20]), which, as we described already, is represented by the CMS virtual organization in the EGEE framework. The CMS collaboration has built a very specialized detector to track certain type of particle collisions occurring inside in the LHC accelerator. For each of these collisions, also called *events*, a non-negligible amount of data is generated by the measurement of particle properties like trajectory or speed. The high rate at which these events occur results in the production a huge amount of data, which needs to be distributed over the grid for physicists' consumption. To this *raw* detector data, several versions of processed –refined- data and the result of event simulation (necessary for calibration and validation of the detector measurements) have to be added. In total,

several petabytes of data are produced each year. Of those, an important fraction needs to be replicated at several centres for redundancy and access performance reasons.

The demanding requirements that the management of all this data imposes have led CMS to adopt a centralized policy for VO-wide operations and to establish a hierarchy of resource centres: the T0 site, from where the detector data is distributed, the T1 centres, with large storage capabilities, and the T2 sites, smaller but with a considerable computing capacity if taken as a whole. The development of certain CMS-specific data management services has been also deemed necessary. Among them, the main examples are the catalogues to keep track and locate existing data –the *Dataset Bookkeeping System* (DBS) and the *Data Location Service* (DLS)-, and the system to move and replicate data between remote sites: the *Physics Experiment Data Export* (PhEDEx). The use of these systems and the application of the policies described above allow CMS to organize its workflows in function of the data it manages.

A CMS operations team organizes the production of simulated data and the processing of real raw data. It also defines where this data is to be stored; namely, in the T1 centres. But once all the real and simulated, raw and processed, data is stored (what is a quite complicated problem on its own right), the numerous physicists in the collaboration –spread all over the world- want to run their analysis applications upon them. In this case there is no central team in charge of the task since each physicist or analysis group selects the data that is interesting for them. The estimated numbers for a typical analysis task are as follows: they run over 500 000 physical events, each event representing around 2 MB of data, and the task divided in about 100 jobs. This means that such a task will consume around 1 TB of data, which is usually stored as a set of 2 GB files. At common processing speed in modern computers, the analysis of an event requires about 2 seconds of CPU, and thus 2.8 hours are required for each one of the 100 analysis jobs (280 CPU hours in total). One additional and important point is that all the data required for a given analysis is very likely to belong to the same physical dataset (i.e.: sharing certain physical conditions), and thus likely to be stored together in the same storage element.

The numbers presented above illustrate the problem we are facing. If a metascheduler like GridWay decides to send user jobs to one processing resource or

another based on the characteristics of this resource only (e.g.: processor speed), then the jobs will likely need to copy 1 TB of data around, which means that the computing nodes will be wasting a lot of CPU time (just copying input data before processing). In Chapter 3, we will make some measurements on how much this time may represent.

Another consequence of the jobs copying data to the site where they are running is that the number of replicas of the data increases. For such huge files, the storage space is a limited resource and so the replicas would have to be recycled with a certain frequency and probably retrieved from master copy more often than needed (this may imply a read from tape, which is a very costly operation). If no deletion policy is in place –which is currently the case, since it is not easy to say when jobs can force the deletion of someone else's old data- what will happen first is that jobs fill storage space and then fail to bring the data they need and abort. The users will then need to manually ask site administrators to empty space. In general, having more data transfers than needed implies fewer resources available –disks, tape drives, network bandwidth- for the data movements that are really required –e.g., distribution of fresh detector data.

There is a third possible problem. If data is copied to the computing node's file system (rather than to dedicated storage), it will consume 10 GB of disk space per job. Since a node can run 2 or 4 simultaneous jobs (one per core), it is certainly possible that the available disk space in the node is exhausted and the jobs crash.

For the reasons just explained, it seems obvious that a job scheduling mechanism that does not take into account the location of the data the jobs will require is likely to be quite inefficient in certain cases. If, on the contrary, resources are selected so that data transfers are minimized, the efficiency can be probably increased. In the next section, we study the approach adopted in EGEE and in particular by CMS. In section 2.4, other work found in the literature that addresses this problem is reviewed.

2.3. The EGEE Approach

The EGEE project and the grid infrastructure it provides –built on gLite middleware- was presented in Section 1.5. As indicated, several of the VOs in EGEE

have huge requirements in terms of data produced and consumed by their applications. Managing all this data is a complex task. It is no surprise that the EGEE infrastructure is sometimes referred to as a *data grid*, in opposition to other *computing grids*, where processing power is the most demanded resource and storage needs are modest.

It was also mentioned that specific data management middleware was developed for gLite (and within the VOs' frameworks). In particular, the concept of storage element is used to designate a resource for storage, and the SRM is a uniform interface for storage elements. Thus, storage and processing capacity are treated as different resources.

Following standard Globus practices, gLite users can ship a small quantity of input data with their job. This data is copied to the worker node where the job lands and it is available to it. The same strategy is followed by GridWay when users ask to stage input files to the executing host. This data is not treated as *grid data*, that is, it is not *managed* (not registered in catalogues or stored in SEs). However, due to the problems discussed earlier, and to potential scalability issues with the WMS services, only a few megabytes may be shipped with the job in this way. If a job requires larger amounts of input data, it must previously store it in a SE and read it from there. The SEs in EGEE offer read/write interfaces for direct access from *close* computing resources. Resources are arbitrarily defined as *close* by grid administrators, usually indicating that resources are located within the same site. The job is thus not forced to copy input files in the WN's file system (although it may be done at occasions).

Regarding job scheduling, the EGEE architecture allows users to tie processing and data requirements (normally files) in their job requests. The user is able to express some *input data* needs for her jobs. The WMS will try to locate the list of SEs holding the specified data and will then try to send the jobs to CEs defined as close to those SEs in the information system. Other job requirements are evaluated only on those CEs. The whole process is called *matchmaking*. If one input file is specified, then the job will only go to a site holding that file. If several files are indicated, then, according to [21], the WMS will send the job to the site holding the highest number of requested files. In other words, by means of the matchmaking process, the WMS will select only CEs satisfying the job requirements (and so *close* to the requested data). Within the list of

matching CEs, a user provided rank expression (or the default one, if none is given) is used to prioritize them.

We must note here that the information about grid data is different from other resource descriptions available through the common information system. This is due to the huge amount of files that the grid must keep track of. It is too much information and too dynamically changing to be contained in a general purpose information service. Like in other grids, dedicated data catalogues are used in EGEE. These catalogues associate logical file names to existing physical replicas of those files. They sometimes are VO-specific and may contain additional semantic information. In EGEE, the standard interface to access those catalogues is the Data Location Interface (DLI), which is described in [22]. The interface basically returns the list of replica locations for a given logical name. The same interface can be used for files or other entities like datasets. Therefore, the WMS needs to query a specified file catalogue (or the default one) using the DLI in order to obtain the necessary data location information. This system is illustrated with a job scheduling example in Figure 5.

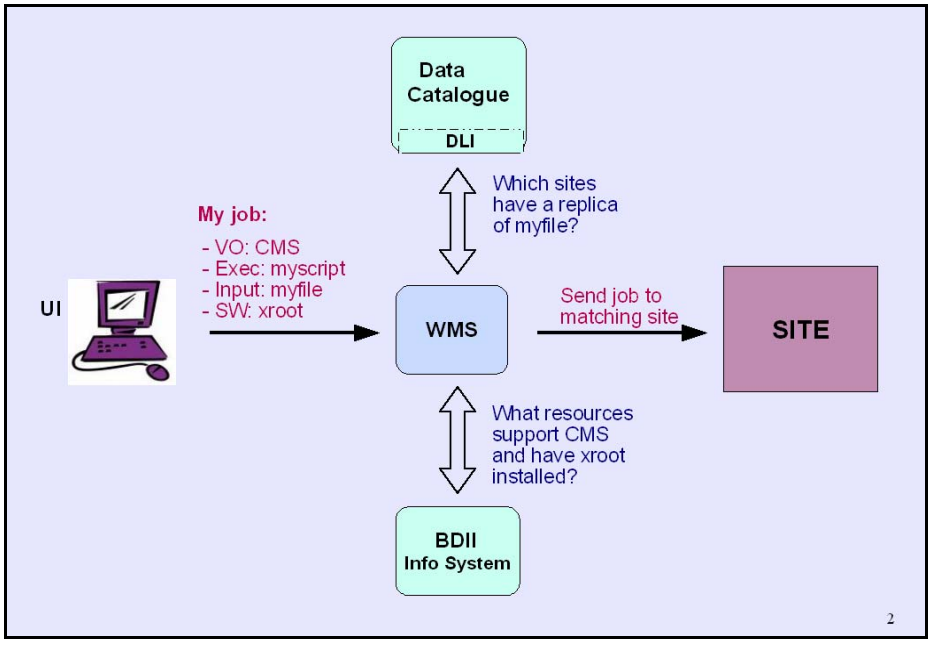

Figure 5: Considering data location for job scheduling in EGEE

With this approach, users are free to decide if they want to force jobs to go where their data is or not. If their requirements of data are modest, they will probably not use this feature, and so the WMS will choose the best suited CE (the one with fewer jobs

queuing or with fastest CPUs). If they do need a lot of data, they will choose the CEs holding the replicas, because practice teaches that the cost of transferring the data is usually highest than the penalty for not choosing the best CE.

We must note here that the WMS does not make it possible to combine both policies or to find an intermediate approach. Since input data considerations can only be expressed in the requirement but not in the rank expression of job requests, either WMS considers only the sites holding required data or it does not care about its location at all.

Within EGEE, the analysis jobs of the CMS VO are always very concerned about location of data. The ratio between transfer and processing time is too high not to be. For that reason, CMS always includes data requirements in their analysis job requests and thus they are always sent to the places where the data is. This avoids many problems as discussed previously but implies that data must be replicated in an organized manner before the analysis jobs are sent. In particular, since the master copy of the data is located at T1 sites where no analysis jobs are allowed, a replica of this data must be made to one or more T2 sites. Once replicated, jobs can run on T2 sites.

One problem with the described approach is that it ignores the status of the computation resources. For instance, if only one site is holding a replica of the data the user is interested in, and this site is down or has many jobs waiting in the queue, then the user cannot run her analysis (or it will take a really long time). In this case, though, a manual copy of the data can be made (or requested to the central operations team), which is considered a small prize to pay in order to avoid the problems described before.

2.4. Data Location-Aware Job Scheduling in the Literature

In the scientific community, it is commonly agreed that data location awareness is an important factor to consider when performing job scheduling in the grid. This is in general concluded because of the long time spent transferring data before jobs can actually compute something. In Section 2.2, we indicated two other problems that may appear if data is not considered when scheduling.

One of the first works considering this problem was [23]. Its authors have no doubt that schedulers need to consider where data is located. They propose an algorithm that takes care of replicating popular (most accessed) files in a way that is totally decoupled from the job scheduling itself. Their job allocation strategy consists then in just sending the jobs where the data resides. This approach tries to avoid the complexities related to a combined mechanism for data replication and job scheduling. By asynchronously replicating most demanded data, they expect to distribute workload among sites.

If we compare this approach to current CMS activities, we observe that they share the scheduling algorithm –jobs to data-, and rely on an independent data placement system (DPS) that takes care of file movement. While in [23] automatic replication of popular files is proposed, in CMS the PhEDEx system [24] is based on dataset subscriptions. Analysis groups subscribe a given site to certain dataset, and PhEDEx takes care of transferring all the files in the dataset in the most efficient way. This allows for a more reasoned and thus optimized usage of resources but it clearly requires more human effort. This method of operation can only work for highly structured and organized VOs.

There were also several articles produced as a result of the research conducted to develop the middleware that would later evolve to become gLite. In [25], the main conclusion that data must be considered when scheduling jobs is reached as well. They find that this is optimum when both the number of jobs queuing at a site and the penalty that transferring files implies are taken into account. For the later value they use information of file catalogues and information on the relative network speed of the links between sites. They also propose economic-based algorithms for selecting the best replica to transfer when the selected resource for a job is not holding the necessary data. Building on these results, the Replica Optimization Service (ROS), which was able to calculate the cost of transferring one of the existing files to a given computing resource, was presented in [26]. This was designed for the European Data Grid (EDG) project [27], which was the precursor of EGEE. However, the ROS relied on a complicated network monitoring infrastructure and was not maintained in EGEE.

A further iteration on these efforts is the more recent [28]. This presented the Data Intensive and Network Aware (DIANA) metascheduling system. DIANA's job allocation takes profit of a distributed network monitoring service that provides estimation of data transfer costs and of a data location service keeping track of file replicas. For each job to be scheduled a handful of computing nodes are selected according to their CPU power, number of queuing jobs and the costs associated with the different file transfers: submission of the job, retrieval of results and input data replication. As indicated, these costs are only computed for the few previously selected resources; it is acknowledged than calculating for the whole grid would be too costly and thus impractical. Once the job's destination has been chosen, DIANA selects the best replicas of the input files that need to be transferred to that node, if necessary.

Other authors have tried to incorporate both data location and processing resources characteristics into the job scheduling decision. As example of this, [29] introduces a grid broker (scheduler) that is capable of finding the location of the file replicas required by a job and then calculate the time needed to transfer that data to available computation nodes. This delay is added to the estimated completion time for the job in each of the nodes, considering queue of waiting jobs and processor speed, and the resource with shortest result is chosen. In [30], authors simulate the behaviour of several scheduling algorithms. They conclude that heuristics that minimize the sum of data transfer and job execution time perform better than one that just sends jobs to data locations, except for the case where the input data is huge, the job rate is high and the estimation of job completion times is not very reliable. They also consider automatic data replication policies that improve overall efficiency, but this is out of the scope of our study.

All these works show that only considering both data transfer and job waiting and execution times optimal scheduling is possible. However, balancing the computing nodes' characteristics and the delays of data transfers is not trivial. There are a few practical difficulties in the implementation of these algorithms. As noticed by [30], it is not always easy for users to indicate for how long their job will run, even if the number of available processors at the resource and their speed are known. Estimating delay caused by transfers is also a demanding task. It requires good, stable and up-to-date knowledge of the network topology, i.e., bandwidth between sites. There might be other aspects that need being taking into account. The type of storage of the data is an

example. Accessing data that resides on tape requires a previous staging to disk. This may take hours. In environments like EGEE, this case is really frequent and data placement systems and policies are designed to pre-stage data from tape before transfers are scheduled. Furthermore, it should not be forgotten that computing all transfer times for each possible pair of data source node and computation node may be a demanding task when the number of nodes in the grid is high. Therefore, algorithms considering a subset of the best nodes only –like the one described in [28]- seem appropriate.

In relation to these proposals, let us indicate that even if these approaches optimized job efficiency, they would probably not minimize data replication. Uncontrolled data movements may, as discussed earlier, create problems unrelated to pure job efficiency but rather with storage systems themselves and overall data transfers performance. This may not affect the efficiency of a particular job but it will probably affect the global efficiency for a VO, especially when storage capacity is scarce. We therefore believe that it might be desirable to penalize job destinations causing data replication (in addition to the temporal cost of input data transferring). The weight of this penalty would however very much depend on VO policies and available storage capacity.

A different approach is taken by [31]. Like in previously mentioned [23], its authors argue that in order to improve efficiency and also simplify the systems, it is best to separate the data placement and the scheduling machinery. But in contrast to the previous work, they now consider the case of complex workflows (with interrelated tasks and required data staging). Their experiments show that for large amounts of data, an independent data placement service, which knows about all data transfers, is more efficient and causes less overhead than an ad-hoc staging of files for each individual job. This data placement service must pre-stage the data according to instruction coming from the workflow management system, but it can group transfers and tune related parameters in order to optimize those.

2.5. Lessons Summary

Based on all that has been presented in this chapter, we have reached some conclusions regarding job scheduling for data-intensive applications. These lessons

have been already outlined throughout the chapter. Now we summarize them in schematic format as follows:

- Poor results are obtained when data location is not considered
  - Job efficiency is in general decreased
  - Data movement rate is higher
  - Storage element concept is required (not to overload computing nodes)

- *Send jobs to data* approach is the simplest one but is helpful and it is used
  - It avoids input data staging delay
  - It minimizes data transfers
  - It is suboptimal for data residing in inaccessible or congested sites
  - It requires an independent data placement system

- Algorithms combining data location and computing nodes status proposed
  - They optimize individual job efficiency
  - They are difficult to implement: network monitoring, data replication…
  - They do not minimize data movement rate (but should make it significantly lower than in the case where no data location is considered)
  - They couple job scheduling and data movements; what may be suboptimal compared to an independent data placement service

- In general, the optimal solution may depend on specific requirements of VOs
  - Some VOs are not data-intensive and are not concerned by all this
  - Some VOs independently arrange data location and movements and may not want job schedulers to interfere with that
  - The metrics to weigh data transfer versus processing resource status and characteristics are not obvious and may require tuning by the VO

3. Making GridWay Data Location-Aware

So far we have described the challenge that scheduling data-intensive jobs in the grid represents and we have reviewed different possible approaches to deal with it. Based on what we have learnt, we now propose and implement a solution to make the GridWay metascheduler take into account data location when scheduling. We have applied this to the EGEE infrastructure, where data-intensive jobs are common, and some test results are presented. Finally, since the problem we are dealing with may affect the whole workflow policy of a VO and the implications of this fall out of the scope of this project, we sketch a more general solution, which has not been implemented, but can be considered a proposal for future work.

3.1. Implemented Solution

GridWay is a general purpose metascheduler. Although it supports dependencies between jobs and so it can be used to schedule workflow tasks, it is not a complete workflow management system. Moreover, replacing a data placement system is not within the aspirations of GridWay. The implemented solution will therefore not deal with the complete problem of allocating jobs and data to maximize job efficiency. It will focus on job scheduling, which is the task for which GridWay is responsible. A more general approach to the problem is however sketched in Section 3.5.

Since there seems to be no general answer to the problem of balancing computing nodes capabilities and location of input data, our system lets the submitter set the required weights. Following GridWay's traditional approach, it is the user, via the specified *requirements* and *rank expressions*, who decides how to prioritize resources. Up to now, only values retrieved from the information system could be used in those expressions. With the modifications we propose, it is possible for the user to incorporate data location information as well. In particular, the rank expression may include a boolean value for the presence or absence of an input file in a site, and a numerical value for its size. This user interface is explained in some more detail in Section 3.2.

The modifications required in the GridWay daemon to take input data into account are described in Section 4.3.

By providing the desired requirement and rank expressions, a user (or in general a VO) may decide to completely ignore input data location, may on the contrary force jobs to go to sites hosting that input data, or may choose an intermediate approach and freely set weight to things like available number of processors, processor speed and data presence. In this way GridWay is made flexible to cope with different VO necessities or policies. The user or VO still needs to carefully choose the right requirement and rank expressions, but we believe that is the users or VOs who can best choose this and so it is wise to let them set them as needed.

We would like to note that GridWay's current scheduler imposes the requirement that a processing resource must have no waiting jobs (i.e.: must be publishing some available slots) to be eligible. This practice is being reconsidered by the GridWay team because there are occasions in which it is completely acceptable to submit a job to a site where some jobs are queuing. The new job will queue for some time, but it will eventually run. If GridWay does not send the job, other jobs may arrive and the queue might never empty completely. This behaviour of the scheduler is of importance in our case because a site that holds the data a job requires may indeed have a waiting queue and it would be very interesting to include this fact as a variable in the rank expression, rather than set is as an absolute requirement, as it is done currently. In an environment like EGEE, where resources are often busy, the number of jobs queuing at a site is taken as ranking expression more often than processor speeds. The different performance of a job once it is running in one site or another has usually smaller relative influence in its turnaround time than the delay the job may have suffered before being able to get a slot to run. Although GridWay does not offer the possibility to play with queuing times at the moment, all our considerations would apply to such a prioritization algorithm also (and this may well be possible in the near term future). For the experiments shown in Section 3.4, we have always used resources offering free slots and have only used CPU speed (and data presence, of course) to compare different sites.

Once a job has landed on a site, it is not clear how many of the files specified in the rank expression are available at the local storage. The files indicated in the requirements

expression are present for sure, but those in the rank expression may or may not be there. It depends on which resource was finally selected. For this reason, our implementation will keep a list of all requested files and check which ones are present in the final destination for the job. This information is made available in the job environment, so that the user's code can copy them to local storage as appropriate. It is arguable if the job wrapper submitted by GridWay should copy these files automatically, but this would not be trivial. The tools used to copy files depend on the middleware (in EGEE are not the same than in GT4-based grids), although the underlying protocol is usually GridFTP in every case (this also could change). We consider it more flexible to warn the user that several specified files are not there and let her copy or access them in the best way for her (this may mean performing a request to a data placement service, for example). This is already more than what the current EGEE scheduler –WMS- does. Again, details of the interface are given in Section 3.2.

There is an important piece that we are missing in our solution and this is the ability to estimate the time that the transfer of a given file between two nodes in the grid will take. Without that, it is difficult to balance processing and data requirements. Given a site that holds the required input data but has slow processors and another site with faster CPUs but with not replica of the data, at what size should the data weigh more than the CPUs in the priority calculation? We cannot know, because we ignore how long it will take to move the data. Given two sites with same processing capabilities, we would like to be able to tell which one is *closer* to the data in terms of transfer delay.

We have seen that the main factor usually considered when computing elapsed transfer time is the quality of the network link (bandwidth, congestion). For example, authors of [29] and [30] suggest the use of a generic network information service, like the Network Weather Service (NWS), described in [32]. This service would provide the scheduler with the conditions of a network link between two given sites. We have not considered such a service for our current implementation in GridWay because the grid infrastructure we are working with, EGEE, does not offer it, and we consider out of the scope of this work to provide a new such service. It would be possible, however, to extend our design to use that kind of information. This would imply querying the NWS for the links between the sites holding the required data and the other sites we are considering as possible job destinations. Since the number of possible sites might be

very high, we should look for ways to optimize this, such as using bulk queries or restrict these to a limited number of nodes; discarding those with lowest rank according to other criteria

There may exist however other factors affecting the time elapsed in data transfers. We have already indicated that a potentially important one is the type of storage of the original data. If the data is on tape, then access to this data will probably suffer from a long initial delay, because it must be first staged to disk. Certainly, this circumstance may be indicated with a metadata attribute in the data catalogue, or in the information system. However, these practices may differ from one VO to another and it would be difficult for a general purpose scheduler to consider all possibilities. Moreover, there may be other factors, like VO policies to avoid replication of certain data or to particular sites. Considerations like these complicate the design of the scheduler and also extend its responsibilities beyond its natural domain: the scheduling of jobs. In the line of the reasoning presented in [31], we believe that it would be best to decouple job scheduling from data replication tasks. It is out of the scope of the present work to develop a component that completely solve these challenges but we have outlined what we consider would be the best approach to deal with them in Section 3.5.

3.2. Interface with the User

As already indicated, the modified version of GridWay that we have implemented allows users to set requirements and rank expressions in their job templates that consider data location. Remember that the requirements are specified in GridWay by means of a logical expression, while the rank takes the form of an arithmetic expression. Both expressions admit variables whose values are obtained from the information system by GridWay.

A finite list of available variables is not enough for the case of data needs, since users must be able to specify which file (or dataset) they require. Therefore, the parser of job templates was extended to allow for the inclusion of functions[19]. Functions are treated in the same way as variables (their value will be replaced by GridWay when

[19] For specification of job requests, GridWay supports native job templates and the Job Submission Description Language (JSDL). For the moment, the new functions are available for job templates only.

evaluating the expression) with the difference that they accept arguments. For data requirements, the following functions have been defined:

- *CLOSE_DATA(logical_file_name)*: Usable in the requirements expression. For each CE, it is evaluated as True only if specified data is held by a close SE.
- *HAS_CLOSE_DATA(logical_file_name)*: For rank expression. For each CE, it is evaluated as 1 if specified data is held by a close SE, and 0 otherwise.
- *SIZE_CLOSE_DATA(logical_file_name)*: For rank expression. For each CE, evaluated as the size of the specified data if held by a close SE, 0 otherwise.

This has been considered the easiest and most convenient way for users to express their necessities –it is an extension of what existed already- while it provides the necessary flexibility. In other interfaces, such as the job requests used in EGEE, data needs are not indicated within the requirements or rank expressions, but with additional clauses in the job description. In practice, this means that they are evaluated as further requirements but cannot be used to affect the rank.

Additionally, a new general variable has been added to the job template. This is the *DATA_CATALOG* variable, used to indicate the endpoint where the catalogue service can be accessed in order to query for the specified data. This is necessary, since there may be several different catalogues for different VOs. Moreover, right now the implementation assumes the catalogue is offering the DLI interface, which is the one present in EGEE, but if other catalogue interfaces were to be supported, a new variable *DATA_CATALOG_TYPE* should be added, so that GridWay could select the appropriate interface to talk to it.

Apart from the specification of users' data needs, there is another modification in the way that GridWay interacts with the user. This is the addition of new variables in the user's job environment. These variables are:

- *GW_CLOSE_SE*: its value is a list of SEs registered as close to the CE where the job is running. The list is composed of SE hostnames separated by coloms.

- *GW_REMOTE_FILES_{i}*: Set of variables (for increasing values of *i*, starting at 0) whose values are the files that were specified in the job's template, but are not present in any of the close SEs (as indicated by the catalogue).

As explained in Section 3.1, these two variables may be used by the job to replicate the necessary files to one of the local SEs (most usual case is that there is only one) and access them from there. The files that are already in a local SE are not listed, so that the application knows it does not need to copy them.

3.3. GridWay Daemon

The GridWay daemon is the process that listens for job requests and acts on them. In the previous sections we have already described the functionality it offers. When a job request is received, its requirements and rank expressions are evaluated for each known resource, in order to select one as job destination. Modifications have been made on GridWay so that the parsed data functions (described in previous section) trigger a query to the indicated catalogue. This information is then used to evaluate the functions and produce a numerical or boolean value. To make this possible, the information manager plugin (for EGEE) has been modified as well, so that it passes the list of close SEs for each resource. This information is also available to users via the normal GridWay command line interface, which displays resources characteristics and status.

Given that GridWay evaluates the expressions once for each possible destination resource, a data function needs to be evaluated once for each known host. In order to avoid multiple catalogue queries requesting the same information –with the consequent extra delay that this would cause-, a catalogue data cache has been implemented. A per-job cache would have solved this problem, but since different jobs may well require the same data, a global cache, managed by the GridWay daemon is a better solution. With the global cache, each requested file produces just one query to the catalogue, and subsequent evaluations will just use the information from the cache. The cache has been implemented as a linked list with a limited size and with the last used entry always at the front of the list. When the maximum size is reached, entries are removed from the cache on a *least recently used* basis. In addition, each entry includes a timestamp.

Before trusting the information of an entry, GridWay checks that the information is not older than a configurable time limit. Otherwise, the entry is discarded and the location of the data is queried to the catalogue again.

We must note here that the functionality for the *SIZE_CLOSE_DATA* function (to retrieve file's size) is not completely ready to be incorporated into a distributable GridWay yet. The DLI interface offers no method to query for data size. The different catalogues present in EGEE do offer this information, but each catalogue's proprietary API must be used. Such API has been used for the experiments presented in this work, but it presents practical problems for a GridWay distribution, since it might require several catalogue clients being distributed with it. Of course, a user probably knows the size of her data before submitting, but still we think it is better if GridWay offers this, so we will look into how to better address this problem (e.g. requesting an extension of the DLI specification).

As a summary of what has been described up to now, the diagram in Figure 6 schematically represents the process triggered when GridWay's job template parser encounters a *HAS_CLOSE_DATA* function in the rank expression. This process is followed for each possible destination host. Firstly, the argument of the function is saved in the list of requested files for the job. Next, if the required information is already in the cache and it is not too old, it is used for the evaluation. Otherwise, the catalogue is queried and the response cached. If any of the close SEs associated to the candidate destination holds the specified data, then a True value is returned. Otherwise, False is given back.

The process for the case of the *SIZE_CLOSE_DATA* function is resolved in a very similar manner, with the difference that size information is asked to the catalogue. The *CLOSE_DATA* function of the requirements expression is almost the same than this, except that the argument is not added to the list of requested files, because a file in the requirements expression will be, by definition, present in the selected destination. This list is parsed by GridWay when preparing the environment for the job. Since this occurs after the match-making process, the destination node is already known and all the necessary information regarding files location has been already retrieved from the catalogue and stored in the cache. The code preparing the environment needs just to

check the cache and eliminate from the list the files that are located in the node selected as destination. The remaining files are passed to the job as *GW_REMOTE_FILE* environment variables.

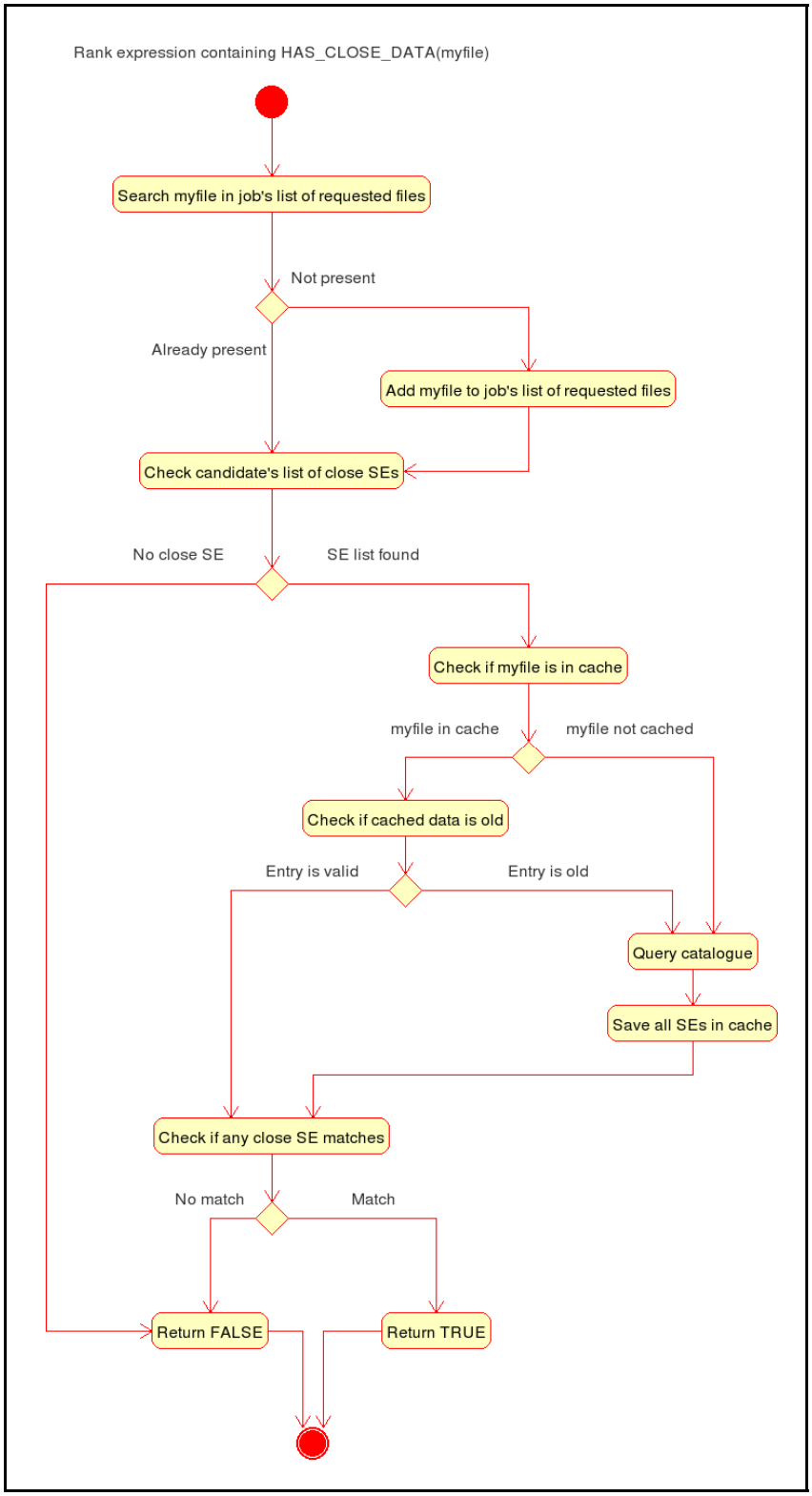

Figure 6: Activity diagram: resolution of a *HAS_CLOSE_DATA* function in the rank expression

3.4. Experiences with the System

Several experiments have been conducted in order to evaluate the functionality and performance of our implementation. All the tests were done using resources of the EGEE infrastructure and standard gLite middleware. The only additional element utilized was our modified version of GridWay.

3.4.1. Delay in the Resource Selection

We have measured the delay that the queries to the catalogue introduce in the match-making process. Our aim is to evaluate if the new capabilities that make GridWay aware of data location are too costly from the point of view of the time it takes to select destinations for each job request. Table 1 summarizes the results for consecutive jobs submitted with different requirement and rank expressions. The first row in each of the columns corresponds to a job submitted with an empty data location cache. Subsequent jobs could use previously cached information.

The number of sites discovered by GridWay as possible job destinations for these tests was 12. The use of the data location cache implies that the number of queries to the catalogue does not increase with the number of sites, but with the number of requested input files. The different requirements and rank expressions used in the job templates are the following:

- *Nothing.tmpl*: no input data specified in requirements or rank expressions.
- *Reqs.tmpl*: Two input files in the requirements expression.
- *Rank.tmpl*: The same two files in the requirements and three additional files (one of which is not located in the catalogue) in the rank expression.

Type of job	Match-making time	Type of job	Match-making time
Rank.tmpl	2.691830	Reqs.tmpl	2.406040
Reqs.tmpl	0.004601	Nothing.tmpl	0.002329
Nothing.tmpl	0.000137	Nothing.tmpl	0.001485
Nothing.tmpl	0.000125	Rank.tmpl	2.504032
Rank.tmpl	0.005583	Reqs.tmpl	0.004422

Reqs.tmpl	0.004928	Reqs.tmpl	0.004294
Reqs.tmpl	0.004608	Rank.tmpl	0.005533
Nothing.tmpl	0.000110	Nothing.tmpl	0.001537
Rank.tmpl	0.005103	Reqs.tmpl	0.004865
Nothing.tmpl	0.000150	Rank.tmpl	0.005797

Table 1: Match-making time in GridWay with data requirements

As we can see, the match-making time is considerable higher for the first time that a file is requested; i.e., the first *Reqs.tmpl* submission triggers the catalogue query for two files, and a *Rank.tmpl* submission causes the query for all of the files (or the three left if it comes after a *Reqs.tmpl* request). This confirms that the global cache of data location reduces match-making time as expected. The delay for the worst cases lasts anyhow only of a few seconds, while the whole submission time is much bigger as can be seen in Table 2, where a file is specified in the requirement expressions for all the submissions.

Match-making time	Submission to start delay
0.003049	56.300435
0.003124	43.859633
0.003442	66.742958
0.003127	49.240011
0.003454	52.357754
0.003117	39.614477
0.003757	56.235452
0.003498	44.173567

Table 2: Comparison of match-making time and job start delay

These measurements were made in submissions to a resource with no queue of waiting jobs. Namely, the job started to run immediately once arrived at the site. The delay that the whole submission chain introduces combined with the fact that GridWay only schedules jobs on a periodical basis (each 30 seconds by default) make the time lost in match-making negligible. We should not forget either that job turnaround time is

usually much longer than this, and it is dominated by running time (or maybe queue waiting time, in some occasions).

3.4.2. Scheduling Algorithms Review

We will try now to use our scheduler to confirm some of our previous conclusions regarding different scheduling algorithms for data-intensive jobs. Namely, we argued that data transfer times may be comparable to processing times and thus not to take them into account may result in worst job turnaround times. But we will also see that sending jobs to data is not always the optimum approach, especially if we have good knowledge of the time our jobs will spend running. In the process, we show how the requirements and rank expressions may be used to put each one of these different approaches into practice.

For our test we used several sites with different processing speeds. As discussed before, GridWay will not send jobs to a site without free slots, so we will not consider queuing times in this experiment. However, if this was included, it would just add to the total job time and it would be reflected in final performance in just the same manner as processing speed is currently. The utilized job templates use rank expressions that favour sites publishing a higher value for the CPU_MHZ attribute. Some of them combine this with data location functions. The requirements expression is only used for the *send jobs to data* algorithm.

Let us summarize the characteristics of our experiment:

- There are three sites: A, B, C.
    - A and B have a CPU_MHZ value of 2800.
    - C has a CPU_MHZ value of 1001.
- Once the data has been transferred to a node, the jobs will take:
    - around 500 seconds to run in A or B.
    - around 675 seconds to run in C.
- We will use four different algorithms (different rank expressions)

(a) Not considering data location.

- `RANK = CPU_MHZ + QUEUE_FREENODECOUNT`

(b) Sending jobs to data.

- `REQUIREMENTS = CLOSE_DATA("myfile")`
- `RANK = CPU_MHZ + QUEUE_FREENODECOUNT`

(c) Balancing data and CPU equally (as proven by practical experience).

- `RANK = CPU_MHZ + QUEUE_FREENODECOUNT + SIZE_CLOSE_DATA("myfile") / 135`

(d) Modification of (c) to additionally penalize data transfers.

- `RANK = CPU_MHZ + QUEUE_FREENODECOUNT + SIZE_CLOSE_DATA("myfile") / 68`

- For each algorithm, ten jobs are sent, each one requiring one of ten files located at C. The size of these files ranges from 3 to 405 MB.

Table 3 shows the results for all forty jobs. For each one, the table shows the used algorithm, the size of the input data required by the job, the CPU_MHZ value of the site where it run, the time the input data transfer took, the time the processing of that data took, the complete time the job run (including transfer time) and the value of the rank expression for the selected site.

Algorithm	Data Size	CPU	Exec Site	Transfer Time	Proc. time	Job Time	Rank
A	03 MB	2800	Site A	8	509	517	2815
A	45 MB	2400	Site B	43	524	567	2407
A	90 MB	2800	Site B	187	509	696	2816
A	135 MB	2800	Site B	389	509	898	2816
A	180 MB	2800	Site A	150	509	659	2821
A	225 MB	2800	Site A	347	510	857	2809
A	270 MB	2800	Site A	626	509	1135	2814
A	315 MB	2800	Site B	243	510	753	2833
A	360 MB	2800	Site A	301	509	810	2808

A	405 MB	2800	Site B	593	509	1102	2817
B	03 MB	1001	Site C	8	669	677	1054
B	45 MB	1001	Site C	8	670	678	1049
B	90 MB	1001	Site C	8	669	677	1049
B	135 MB	1001	Site C	25	670	695	1048
B	180 MB	1001	Site C	25	669	694	1045
B	225 MB	1001	Site C	15	677	692	1038
B	270 MB	1001	Site C	14	669	683	1039
B	315 MB	1001	Site C	16	669	685	1035
B	360 MB	1001	Site C	22	670	692	1035
B	405 MB	1001	Site C	29	672	701	1036
C	03 MB	2800	Site A	9	509	518	2823
C	45 MB	2800	Site B	94	509	603	2821
C	90 MB	2800	Site B	92	509	601	2819
C	135 MB	2800	Site B	176	509	685	2822
C	180 MB	2800	Site A	108	509	617	2816
C	225 MB	1001	Site C	15	670	685	2867
C	270 MB	1001	Site C	14	669	683	3001
C	315 MB	1001	Site C	20	669	689	3334
C	360 MB	1001	Site C	35	674	709	3667
C	405 MB	1001	Site C	25	669	694	4001
D	03 MB	2800	Site A	9	509	518	2844
D	45 MB	2800	Site B	176	509	685	2835
D	90 MB	2800	Site B	115	509	624	2829
D	135 MB	1001	Site C	33	669	702	2986
D	180 MB	1001	Site C	74	669	743	3648
D	225 MB	1001	Site C	19	673	692	4309
D	270 MB	1001	Site C	21	670	691	4971
D	315 MB	1001	Site C	26	669	695	5633
D	360 MB	1001	Site C	24	672	696	6295
D	405 MB	1001	Site C	19	675	694	6956

Table 3: Comparison of different scheduling algorithms for data-intensive jobs

For a given job, the optimum destination is the one that minimizes the value in the job time column. Extending this to a whole group of ten jobs (for all data sizes), we might say that the best algorithm is the one minimizing the average time for the ten jobs. However, we might also use the number of transfers as the metric for the comparison. For some VOs, minimizing this value might be as important as optimizing the job turnaround time.

Attending on how the algorithms behave, we can say that:

(a) All jobs sent to site A or B.

(b) All jobs sent to site C.

(c) For small files jobs are sent to site A or B, for large files jobs are sent to site C.

(d) Same results as (c), but the threshold is lower (most jobs go to site C).

These decisions produce the following results for each algorithm:

(a) Good job times for small files, worse times for large files. 10 transfers.

(b) Bad job times for small files, better times for large files. 0 transfers.

(c) Good job times for all cases. 5 transfers.

(d) Fairly good times for all, a bit worst than (c) for a few cases. 3 transfers.

The graph in Figure 7 compares the different policies in terms of job running time as a function of input data size. There are some oscillations in the transfer times –which was expectable-, but in general we see that algorithm (a) behaves OK only for small input files and it is worse when large files have to be transferred. All the other algorithms select the site holding the input data when this is of big size, but they differ for small files. Policy (b) chooses site C for every case, but (c) and (d) prefer to choose faster sites accepting the transfer of some files, for the cases where these are small. By doing this, they get better numbers.

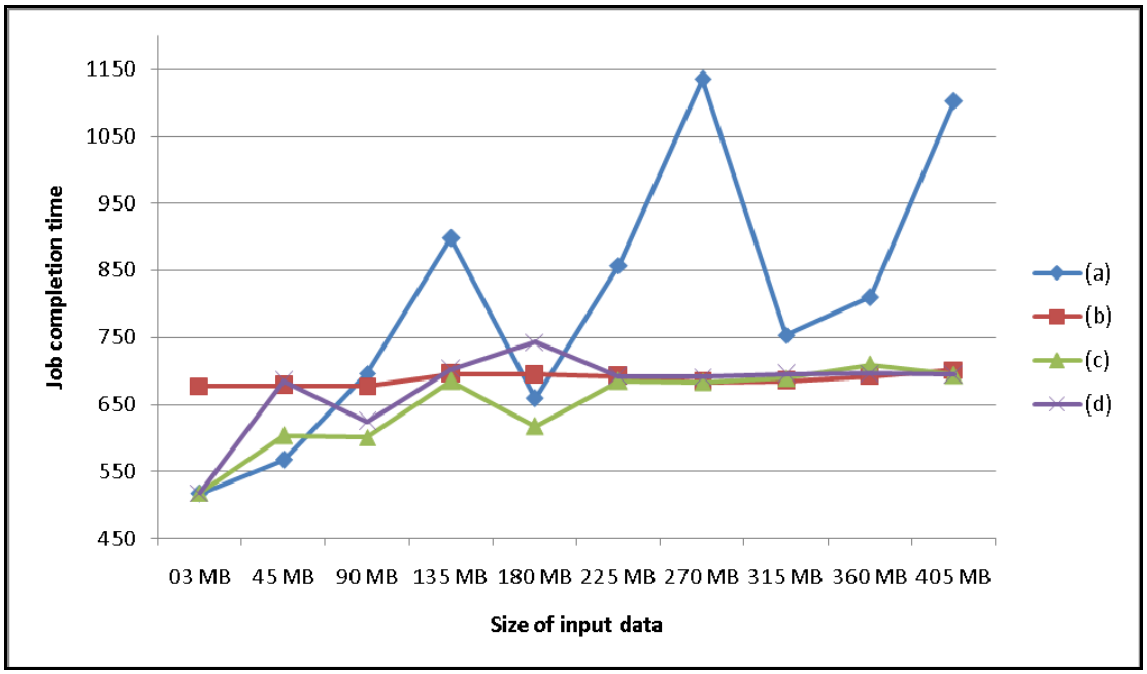

Figure 7: Job completion time versus size of input data for different algorithms

If we consider the number of transferred files, we will see that (b) obtains the absolute minimum, since it never moves files, and (a) is again the worst option. Between (c) and (d), a slightly worst average job time is traded for a reduction in the number of files transferred (transfer penalty). Table 4 summarizes average job times and number of transfers for each of the algorithms.

Algorithm	Average job time	Number of transfers
(a)	799.4	10
(b)	687.4	0
(c)	648.4	5
(d)	674.0	3

Table 4: Average job completion times for different scheduling algorithms

3.4.3. Applicability to the CMS Use Case

In Section 2.2, the numbers of typical CMS analysis task were presented: around 100 jobs, 1 TB of total input data consumed by the jobs, nominal file of 2 GB and around 2.8 hours of processing time per job. In order to assess the relative impact that the transfer of all that data would have on the total job turnaround time, we have performed some transfer measurements between two sites, a nearby one and a distant

one, to the third –local- one. We use the adjective *nearby* and *distant*, in the sense that the perceived data throughput is higher in the first case than in the second one. This is typically due to a better network link connecting the sites or more appropriate tuning of local parameters for the characteristics of one of the sites (TCP buffers as a function of round trip time, e.g.). Transfers from the local SE to the WN were also conducted. In every case the replicated files had a size of 2.5 GB. Results are summarized in Figure 8.

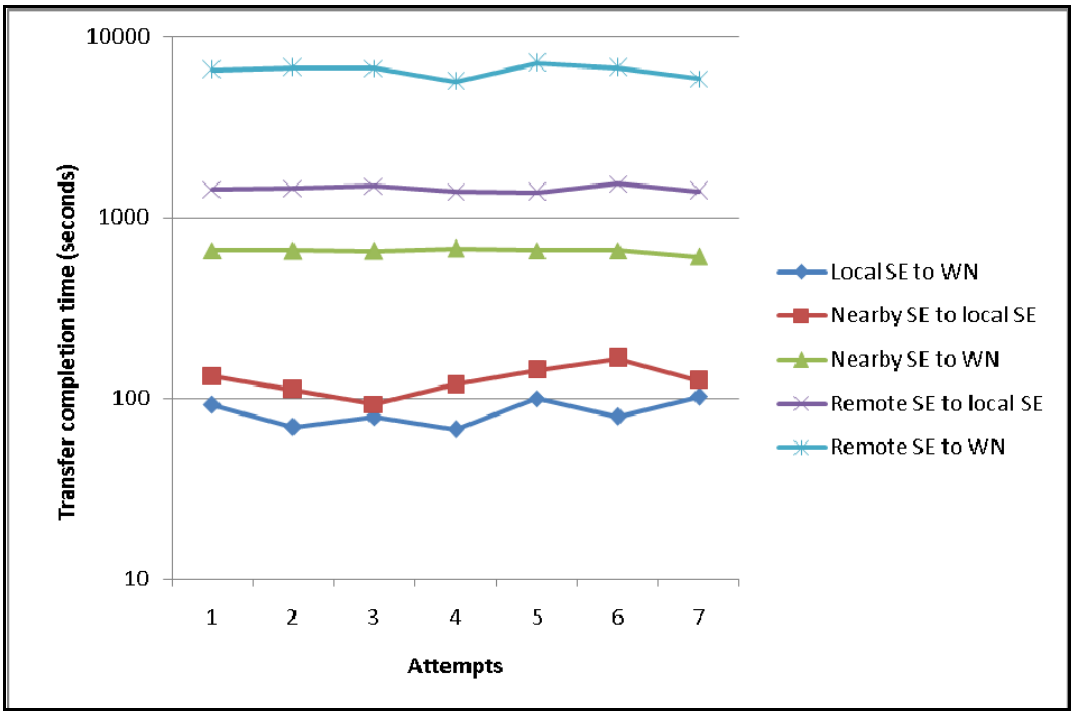

Figure 8: Transfer times for a 2.5 GB file to local storage/worker node

What we see is, firstly, that transfer to the WN take much longer that transfers to the local SE (except for the copy from local SE which sits within the same local network). This is to be expected, since storage elements are tuned to perform well when transferring data, while WNs are not optimized for that. Previously, we had already concluded (due to disk space reasons also) that staging huge input data files to the computing node is clearly suboptimal and it is better to replicate them to the local SE and read the files from there directly.

The times in Figure 8 were measured for the transfer of a 2.5 GB file. If we scale these times to the case of 10 GB per job, we will see that bringing the input data for a job will take 8.49 minutes for the nearby site and as much as 1.59 hours for the distant site. Certainly the transfer parameters could be improved for the latter case, but this is a

real world example, so it must be taken into account. Considering that a job might run for 2.8 hours once the job is there, this transfer delay is clearly unacceptable. For 100 jobs and these sites, 1 TB of data would be transferred and the whole process would take between 14 and 150 hours to complete.

Finally, we would like to compare the numbers we obtained in Section 3.4.2 for the different algorithms and these for a CMS typical analysis. For practical reasons, short length jobs and files of small size were used, but if we scale both by a factor of 20, we have results that are comparable to those of CMS case. Regarding processing time, we had before times ranging from 8.48 minutes to 11.28 minutes. Multiplying by a factor of 20, we have a range from 2.82 to 3.76 hours, which is more or less what a CMS analysis job would last, just considering that in EGEE jobs would probably have to queue for a while before getting a slot to run. As for what transfer times concerns, Figure 7 tell us that already with less than 250 seconds spent in the data replication, it is already wiser to send jobs to the site holding the files than to move them to the fastest CPU. If we scale these 250 seconds by a factor of 20, we get 1.38 hours, which is less than what it takes to move the 10 GB required by a CMS job from the remote SE to the local one.

3.5. General Proposal for the Job Scheduling and Data Placement Problems

The modifications performed in the GridWay metascheduler allow submitters to arbitrarily set rank expressions that prioritize computing resources in function of their characteristics at the moment but also the location of the required input data. Users and VOs are free to combine these parameters as they wish. But two problems remain: how to estimate how long a data transfer will take and, further, how to perform those data movements efficiently and respecting VO policies. Solving the second of these challenges is out of the aspirations of GridWay, but it conditions the first issue and certainly affects the general issue we have at hands.

3.5.1. Use of a Data Placement System

Our first consideration is that, since some data placement systems (DPS) already exist that schedule data movements for VOs, it is not our purpose to replace them. Those systems are the ones knowing more about data location, network topologies and VO policies regarding data movements.

Referring to the first problem, instead of asking a service like NWS and build a matrix of transfer times ourselves, we judge preferable to query a data service placement, which can take into account other factors like free space on sites or the type of storage of a given replica. A system like the one used by the CMS VO, PhEDEx, already knows about all that. In our view, the whole task of selecting the best sites to find a file replica or to transfer one to it could be accomplished by a service like PhEDEx.

So, as a first approach to the general problem, GridWay would replace the query to the catalogue by a query to the DPS. The arguments of this query would be just the requested files, and the response would be a list of prioritized sites, each with an assigned cost for the transfer of the files. This is illustrated in Figure 9. If for example a single file is requested (*best_sites* query) and this file is located on disk at sites A and B, the cost would be zero for them. If the file is also at site C, but it is only on tape, a higher cost –based on history of previous requests- would be given for this site. The cost for other sites not holding the data would depend on their network links to A and B and how much free space they possess. Of course, VO policies could be also imposed (e.g. set an infinite cost to site D, since no data can be copied to it). The DPS would return the list of the best sites where to access input data files (up to a configurable number of sites). GridWay would then perform the match-making of job requirements and calculate the rank for each resource in the usual way, but considering only the sites returned by the DPS and subtracting the returned transfer cost from each resource's rank. The job would be sent to the best remaining resource (E). Optionally, GridWay could return the chosen destination to the DPS, so that it performed the data movement on behalf of the user.

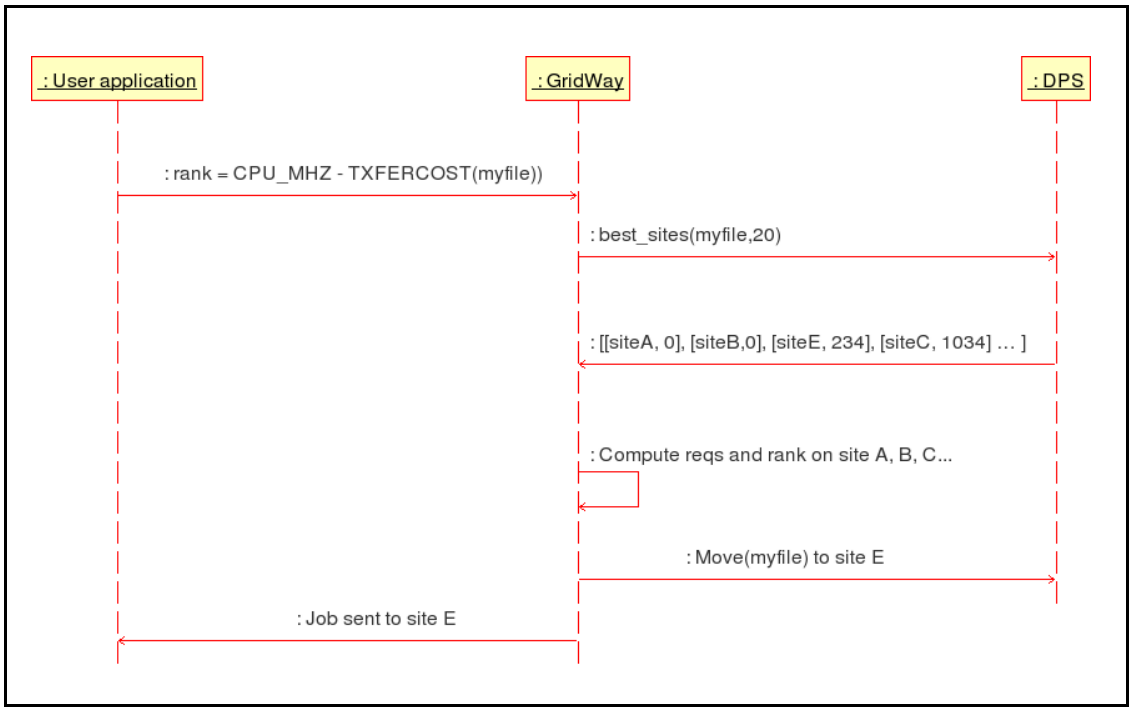

Figure 9: Use of a Data Placement System to find best sites for data

The *best_site* query could be made available as a normal grid service, so that it was accessible to GridWay or other clients by just using the appropriate endpoint (and user's credentials). It would be used in the same way that the DLI service is used currently. If other types of services existed (like there are different catalogues), plugins could be used to access each one of them.

3.5.2. Use of a Complete Workflow Management System

The approach sketched in the previous section addresses the problem of selecting the best resource to run a given job. It also tries to delegate the decisions on where and how to move data to the data placement system, which will most probably do it better than a metascheduler. This is probably an optimum solution for the scheduling of a single job. In some occasions, however, a stricter decoupling of job scheduling and data placement tasks could bring better global efficiency for a VO, or for the whole grid. As we indicated before, this was already suggested in [31].

Imagine that a user submits 500 jobs, each requiring the same set of 10 input files, all of which reside in the same site. Even if this site offers very few CPUs and another much bigger site would happily accept a replica of the files, the best possible scheduling

for a single job might still be to send the job to the first site because transferring all the files would cause too much overhead. Nevertheless, if we consider the 500 jobs, the time required to move the ten files would be negligible compared to the time that the 500 jobs will take to run, or queue, at the first site if no replica is made available. It is even possible that while the files are being transferred other jobs are scheduled first, so that job slots are not occupied with jobs that are just waiting for some data. A more intelligent system could try to optimize situations like this.

Certainly, providing a complete solution to this is out of the scope of GridWay's aspirations. It would be the duty of a higher-level component, perhaps a workflow management system, to take longer term decisions regarding data replication and planning of jobs destination. This component would get job requests as input, data and jobs information as feedback and VO or grid rules as policies. It would then schedule data replications that the DPS would execute and, by setting the requirements and rank expressions, instruct GridWay to allocate jobs as appropriate. The organization of such a system is illustrated by Figure 10.

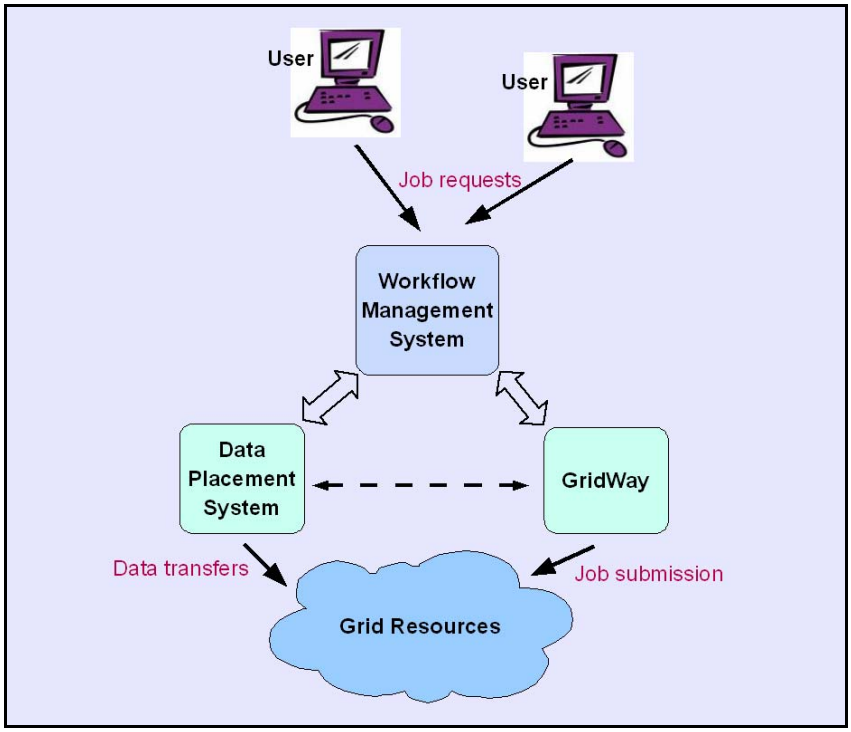

Figure 10: A workflow management system to schedule data and jobs

A system like this seems necessary only for VOs with a high job submission rate and with large necessities in terms of data. If a particular user needs to send a few jobs, a direct request to GridWay would be probably enough. If a DPS offers a query service so that data transfer times can be appropriately estimated, all the better. Finally, if resources are scarce for the volume of data and jobs that a VO manages, then a more intelligent system like this may be needed.

Notice that the VO of our reference example, CMS, actually uses an approach similar to that in Figure 10, where job submission and data placement system are completely independent, and both are subject to the decisions of a higher, more intelligent entity. The difference is that this entity is not a planner or a workflow manager, but a central operations team, a group of humans, who take the appropriate decisions.

4. Conclusions

4.1. Conclusions

This work describes the particularities affecting the scheduling of data-intensive jobs in a grid. It shows why location of input data must be taken into account when allocating jobs, not only to optimize jobs efficiency, but also to avoid excessive data replication and the problems that this entails. The fact that different users and virtual organizations may present heterogeneous needs and policies motivates us to argue in favour of a flexible system where different allocation algorithms that make use of data location information can be supported. This idea has led the process of enhancement of the existing GridWay metascheduler and has been demonstrated in several experiments with it.

Throughout the work, the relation between data placement decisions and job scheduling has been stressed. We have indicated that a metascheduler can only achieve optimum scheduling decisions for individual jobs with advice from a data placement service and that a system that maximizes global job efficiency for a VO requires further coordination between both parties. Although a complete solution to this problem is out of the aspirations of GridWay, we have addressed it with a proposal for a system that coordinates the planning of data and jobs allocation but keeps the execution of these allocation tasks decoupled.

4.2. Possible Improvements and Future Work

Possible improvements of our GridWay implementation have been indicated throughout the text. A consistent method for the retrieval of data size information needs to be integrated into GridWay's distribution. New catalogue interfaces –others than the standard in EGEE- should be supported and for that a new variable to indicate the type of catalogue to be queried should be added. Finally, the new functionality should be supported for JSDL requests also and not only for job templates.

There is certainly room for further work on the topics discussed by this project. The system might be extended to make use of information from a network weather service or from a data placement system. From a more general point of view, further research is needed in the area of coordinated planning of data placement and jobs allocation in grid environments.

Table of Abbreviations

BDII	Berkeley Database Information Index
CE	Computing Element
CERN	European Laboratory for Particle Physics
CMS	Compact Muon Solenoid
CPU	Central Processing Unit
DBS	Dataset Bookkeeping System
DIANA	Data Intensive And Network Aware
DLI	Data Location Interface
DLS	Data Location Service
DPS	Data Placement System
DRMAA	Distributed Resource Management Application API
DRMS	Distributed Resource Management Systems
EDG	European Data Grid
EGEEE	Enabling Grids for E-SciencE
GB	Gigabyte
GLUE	Grid Laboratory Uniform Environment
GRAM	Grid Resource Allocation and Management
GridFTP	Grid File Transfer Protocol
GSI	Grid Security Infrastructure
GTx	Globus Toolkit version x
JSDL	Job Submission Description Language
LHC	Large Hadron Collider
LSF	Load Sharing Facility
MB	Megabyte
MDS	Monitoring and Discovery Service
MHZ	Megahertz
NWS	Network Weather Service
OASIS	Organization for the Advancement of Structured Information Standards
OGF	Open Grid Forum
OGSA	Open Grid Services Architecture

PhEDEx	Physics Experiment Data Export
RB	Resource Broker
RFT	Reliable File Transfer
ROS	Replica Optimization Service
SE	Storage Element
SGE	Sun Grid Engine
SRM	Storage Resource Manager
TB	Terabyte
UI	User Interface
VO	Virtual Organization
WMS	Workload Management System
WN	Worker Node
WSRF	Web Services Resource Framework
WS	Web Service

List of Tables

List of Figures

References

[1] Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufman, 1999.

[2] I. Foster, *What is the Grid? A three point checklist*. GRIDtoday 1 (6), Available from http://www.gridtoday.com/02/0722/100136.html.

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

[4] E. Huedo, R.S. Montero, I.M. Llorente, *A framework for adaptive execution on Grids*. Software—Practice and experience 34 (7) (2004) 631–651.

[5] F. Gagliardi et al., *Building an infrastructure for scientific Grid computing: Status and goals of the EGEE project*. Philos. Trans.: Math. Phys. Eng. Sci., vol. 363, pp. 1729–1742, 2005.

[6] Foster I, Kesselman C., *Globus: A metacomputing infrastructure toolkit*. International Journal of Supercomputer Applications 1997; 11(2):115–128.

[7] W. Gentzsch, *D-grid, an e-science framework for German scientists*. Proc. 5th Int. Symp. Parallel Distrib. Comput. (ISPDC 2006). pp. 12–13.

[8] Catlett, C. et al., *TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications*. HPC and Grids in Action, Ed. Lucio Grandinetti, IOS Press 'Advances in Parallel Computing' series, Amsterdam, 2007.

[9] I. Foster, *Globus Toolkit Version 4: Software for Service-Oriented Systems*. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.

[10] I. Foster, C. Kesselman, J. Nick, S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[11] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, *The WS-Resource Framework*. http://www.globus.org/wsrf/specs/ws-wsrf.pdf. March 5, 2004.

[12] Peter Tröger, Hrabri Rajic, Andreas Haas and Piotr Domagalski, *Standardization of an API for Distributed Resource Management Systems*. In Proceedings of the Seventh

IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007), pages 619–626, Rio de Janeiro, Brazil, May 14-17 2007.

[13] EGEE Middleware Design Team, *EGEE Deliverable 1.4: EGEE Middleware Architecture*. https://edms.cern.ch/document/594698/.

[14] E. Laure et al., *Programming the Grid with gLite*. Computational Methods in Science and Technology, 12(1):33--45, 2006.

[15] Graeme A Stewart, David Cameron, Greig A Cowan and Gavin McCance, *Storage and Data Management in EGEE*. 5th Australasian Symposium on Grid Computing and e-Research (AusGrid 2007), Ballarat, January 2007.

[16] A. Delgado Peris, A. Fanfani, F. Farina et al., *Data Location, Transfer and Bookkeeping in CMS*. Nuclear Physics B - Proceedings Supplements, Volumes 177-178, Proceedings of the Hadron Collider Physics Symposium 2007, March 2008, Pages 279-280.

[17] Mathias Dalheimer, Franz-Josef Pfreundt, Peter Merz, *Agent-Based Grid Scheduling with Calana*. Parallel Processing and Applied Mathematics 2005: 741-750

[18] Marcos Dias de Assuncao and Rajkumar Buyya, *An Evaluation of Communication Demand of Auction Protocols in Grid Environments*. Proceedings of the 3rd International Workshop on Grid Economics & Business (GECON 2006), World Scientific Press, May 16, 2006, Singapore.

[19] S. Andreozzi, M. Sgaravatto, C. Vistoli, *Sharing a conceptual model of grid resources and services*. Proceedings of the Conference on Computing in High Energy and Nuclear Physics (CHEP 2003), La Jolla, CA, USA, March 2003.

[20] CMS Collaboration, *The Computing Project TDR*. CERN/LHCC, 2005-2005.

[21] F. Pacini, *Job Description Language Attributes Specification*. EGEE-JRA1-TEC-590869-JDL-Attributes-v0-8, https://edms.cern.ch/document/590869/1.

[22] H. Stockinger, F. Donno, G. Eulisse, M. Mazzucato, C. Steenberg, *Matchmaking, Datasets and Physics Analysis*. Workshop on Web and Grid Services for Scientific Data Analysis (WAGSSDA), IEEE Computer Society Press, Olso, Norway, June 14, 2005.

[23] K. Ranganathan and I. Foster, *Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications*. In Int. Symposium of High Performance Distributed Computing, Edinburgh, Scotland, July 2002.

[24] J. Rehn, et al, *PhEDEx high-throughput data transfer management system*. Proceedings of CHEP06, Mumbai, India. 2006.

[25] D.G. Cameron, R. Carvajal-Schiaffino, P. Millar, C. Nicholson, K. Stockinger and F. Zini, *Evaluating Scheduling and Replica Optimisation Strategies in OptorSim*. In 4th International Workshop on Grid Computing (Grid2003), Phoenix, Arizona, USA, 2003.

[26] W. Bell, D. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. *Design of a Replica Optimisation Framework*. Technical Report, DataGrid-02-TED-021215, Geneva, Switzerland, December 2002.

[27] P. Andreetto, et.al, *Practical Approaches to Grid Workload & Resource Management in the EGEE Project.* CHEP04, Interlaken, Switzerland, 2004.

[28] Richard McClatchey, Ashiq Anjum, Heinz Stockinger, Arshad Ali, Ian Willers, Michael Thomas, *Data Intensive and Network Aware (DIANA) Grid Scheduling*. Journal of Grid Computing, Springer-Verlag, 2007.

[29] Srikumar Venugopal, Rajkumar Buyya and Lyle Winton, *A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids*. Technical Report, GRIDS-TR-2004- I, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, February 2004.

[30] M. Tang, B.-S. Lee, X. Tang, C.-K. Yeo, *The impact of data replication on job scheduling performance in the Data Grid.* Future Generation Computer Systems 22 (3) (2006) 254–268.

[31] Chervenak, A., Deelman, E., Livny, M., Su, M.-H., Schuler, R., Bharathi, S., Mehta, G. and Vahi, K, *Data Placement for Scientific Applications in Distributed Environments*. 8th IEEE/ACM International Conference on Grid Computing (Grid 2007), Austin, TX, 2007.

[32] Wolski R, Spring N, Hayes J., *The Network Weather Service.* A distributed resource performance forecasting service for metacomputing. Journal of Future Generation Computing Systems 1999; 15(5–6):757–768.