Project no. FP6-004265

CoreGRID

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies

Network of Excellence

D.RMS.04 – Proposal of Multi-level Scheduling Models

Due date of deliverable: August 30, 2006
Actual submission date: October 26, 2006

Start date of project: 1 September 2004 Duration: 48 months

Organisation name of lead contractor for this deliverable: Universität Dortmund

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	PU

Keyword List:
Grid, Load Balancing, Resource Management, Resource Allocation, Job Scheduling, Workflow Management, Grid Middleware

Table of content

1 Executive Summary

Grids are geographically distributed platforms composed of heterogeneous resources that users can access via a single interface, providing common resource-access technology and operational services across widely distributed and dynamic virtual organizations, i.e., institutions or individuals that share resources.

Resources are generally meant as reusable entities employable to execute applications, and comprise processing units, secondary storage, network links, software, data, special-purpose devices, etc. Grid computing differs from conventional distributed computing as it focuses on large-scale coordinated resource sharing by independent providers in different administrative domains. Grids are conceived to support innovative applications in many fields, and they are today mainly used as effective infrastructures for distributed high-performance computing and data processing. However, their application areas are shifting from scientific computing towards industry and business-related applications.

Grid resource management and scheduling components are important for building Grids. They are responsible for the selection and allocation of Grid resources to current and future applications. Thus, they are the building blocks for making Grids available to user communities.

The objective of this deliverable is to provide a description and classification ?characteristics? of the solutions proposed by the researchers participating in the activity conducted within task 6.2 of WP6. This is an algorithmic extension to the proposed Grid scheduling architecture which had been proposed by the Institute in one if its earlier deliverables [64].

In this report we present a short introduction to the scheduling problem and to a Grid scheduling model. This is followed by a short overview on some selected implementations of Grid schedulers as available by CoreGRID partners. These solutions are currently available and are used in existing projects. To foster the exchange between CoreGRID partners and to highlight the background of available solutions a survey has been carried out among the partner in WP6 who work on Grid schedulers. The results of this survey are also presented in this document. They show that CoreGRID partners consider and use features like SLAs and co-allocation of resources. Many manage and schedule several resource types beyond pure CPU-power. These are typical features which have already been identified as crucial requirements for future Grid systems.

While the partners tackle similar problems, the survey shows that the technical approach is quite different in their solutions. This can be deduced from the historical evolution of the implementation, the environment and participation in projects. On a short to medium time-scale it is not expected that a single common solution will arise. Moreover, it seems reasonable to expect a consolidation of solutions by working towards interoperability. Most partner systems still work on small scale Grids.

The extension towards large scale Grid environments with common interfaces and scalable services is not yet available. This will be the upcoming challenge for future Grid systems. The work towards interoperability will have to consider the automatic management and scheduling of resources, as well as set of commonly used standards. It is expected that this is achieved by an additional scheduling level on top of the existing management layer. The presented approaches already provide first solutions in this area which can be starting points for further extensions.

2 General Grid Scheduling Model

The central purpose of distributed computing is to enable a community of users to perform work on a pool of shared resources. Since the number of jobs to be executed in most cases outnumbers the number of available resources, one has to decide how to allocate resources to jobs. Historically, this has been known as the Scheduling Problem. A large amount of research in scheduling was motivated by the availability of massively parallel machines in the early 1990s, and the desire to use these very expensive resources as efficiently as possible. Nowadays, Grid is becoming the emerging computing platform [31],[32]. Grids enable the sharing, selection, and aggregation of a wide variety of resources including supercomputers, storage systems, data sources, and specialized devices. These resources are potentially geographically distributed and owned by different organizations, and they are used for solving large-scale computational and data intensive problems in science, engineering, and commerce. This led to the concept of virtual organizations [33] and enterprises [34] as envisioned in [35], as a temporary alliance of enterprises or organizations, which come together to share resources, skills, or core competencies in order to better respond to business opportunities, and whose cooperation is supported by computer networks.

A Grid can be seen as a seamless, integrated computational and collaborative environment. One of the user's access points to a Grid can be a Grid Resource Broker (GRB). Users interact with it to submit and execute jobs. The GRB performs resource discovery, scheduling, and the managing of application jobs on distributed Grid resources. Grid computing makes the scheduling problem even more difficult since, due to its heterogeneous and highly distributed nature, a single centralized scheduler algorithm cannot serve it.

Some of the main requirements of Grid Resource Management Systems (RMS) are the adaptability to different application domains, scalability, reliability, and fault tolerance, as well as the adoption of dynamic information monitoring and storing schemes. The Open Grid Forum is currently working on the definition of a general service-based resource management architecture for Grid environments (OGSA), providing services for, e.g.: maintaining static and dynamic information about data and software components, network resources, and corresponding reservations; monitoring jobs during their execution; associating resource reservations with costs; and assigning tasks to resources. Each of these services may have a different internal organization (e.g. centralized, decentralized, hierarchical).

The management of processor time, memory, network, storage, and other components in a Grid is clearly very important. The overall aim is to efficiently and effectively schedule applications that need to utilize the available resources. From the user point of view, resource management should be transparent. The user has to interact with those resource management services being confined to a manipulating mechanism for submitting applications.
A Grid RMS consists of many vital components [36],[37]. The scheduler plays a major role when issues like acceptance, usability, or performance of a machine are considered. The objective of the scheduler is twofold: to maximize the overall resource utilization guarantying the required applications performance.

The scheduling problem has shown to be NP-complete in its general as well as in some restricted forms [38]. More in detail, according to [39], a valid schedule is the assignment of tasks to specific time intervals of resources, such that no two tasks use any resource at the same time, or such that the capacity of the resource is not exceeded by the tasks. The schedule of tasks is optimal if it minimizes a given optimality criterion (objective function). A scheduling problem is specified by a set of machines, a set of tasks, an optimality criterion, environmental specifications, and by other possible constraints. The solution of a scheduling problem is an optimal schedule in the environment that satisfies all constraints. Obviously, the functionality and performance of a Grid scheduler depends on the available features of the underlying local resource management systems. Therefore these schedulers have to be able to collect information describing the computational resources in Grid as well as information describing their current state and usage. The state estimation of Grid resources, and its relation with the scheduling policies adopted, is a hot topic of research.

2.1 Description of the Scheduling Model

Henceforward we use the following terminology for Grid RMS. Usually the basic building blocks of Grid applications are called *jobs*. A Job is in general a sequential or parallel user application with some associated constraints (e.g. QoS, minimum parallel degree). A Job identifies the atomic computation that a user requires to be computed by a Grid system. *Jobs* are composed of (possibly dependent) tasks. A Task is the minimal computation unity of a job. Like jobs, each task has some constraints that must be respected (e.g. turnaround time), but while we can consider job constraints as high level constraints, task constraints are more specific and involve also communication features (e.g. synchronization between tasks). Furthermore, there are some constraints that concern dependencies among different tasks of the same job (e.g. data relations, precedence). Tasks are coordinated by Grid RMS for their execution. This includes search and selection of suitable and available resources as well as the temporal coordination and facilitation of the execution itself. A task, when executed on a certain host, may generate a number of local processes (see Figure 1) on which such high-level Grid RMS may generally have limited (if any) control. The same control limitation exists over other resource types, such as locally-managed networks, secondary storage devices, etc.

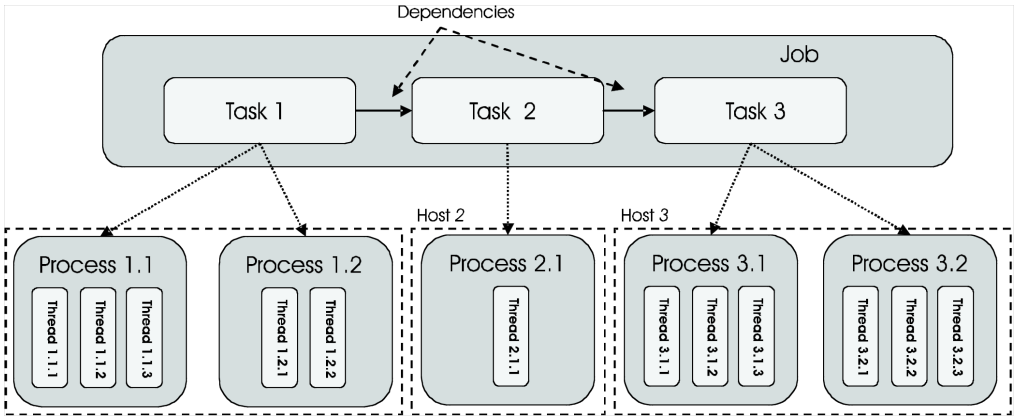


Figure 1 Jobs, tasks, and local processes

To properly describe scheduling scenarios, a suitable *scheduling model* must be adopted. In general, a scheduling model comprises (see Figure 1**Erreur ! Source du renvoi introuvable.**):

- A *resource model*, describing characteristics, performance, availability and cost of resources. Important requirements of this model are the timeframe-specificity of predictions, dynamicity, and capability to adapt to changes. This also includes the consideration of different level of dynamicity in the resource information.
- A *job model*, provided to users for describing jobs. This model may be based on dataflow or workflow formalisms, which mainly resort to task dependency graphs, or may use formalisms that are specifically targeted to particular domains.
- A *performance metric*, for measuring the performances to be improved.
- A *scheduling policy*, comprising (i) a scheduling process, i.e., the sequence of actions to be taken on relevant events (e.g. new job submission or resource availability), and (ii) a scheduling algorithm, defining the way in which tasks are assigned to resources. The main requirement of the scheduling policy is the practical applicability. That is, a good trade-off between effectiveness and efficiency, as the overhead of scheduling activity should be in reasonable balance with the execution of the task itself. It is well known that scheduling in Grids is very complex (see also the later discussion). Therefore the search of a mere optimal task assignment can be computational hard or might often be infeasible. Thus, a sensible trade-off is required to provide suitable scheduling solutions in an online operation in a fair amount of time in relation to the actual task execution.
- A *programming model*, provided to external components/environments for interacting with the scheduler and describing detailed features of an application programming.

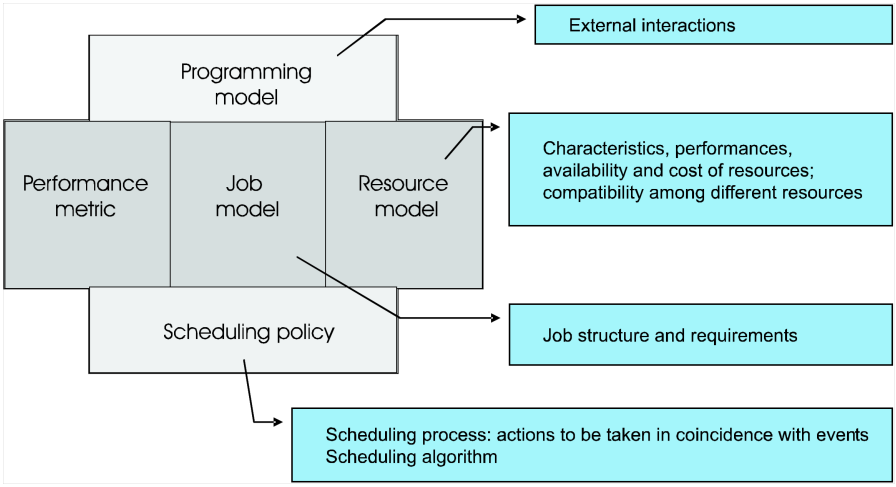


Figure 2: Model for Grid Scheduling

In order to take scheduling and assignment decisions, a variety of parameters, such as the following ones can be considered:

Job Priority. It is one of the most important parameters that influences scheduling decisions. If there is a running job that has a lower priority than another waiting job it is a candidate to stop its execution and to leave the resources to the higher priority job.

Estimation of Job Completion Time. It is important to consider how much time the job candidate to be stopped needs to finish its execution. In some cases, for example when the required service level is not a mandatory constraint, if such time is shorter than that required to stop a job, it should be useful to wait the end of the running job.

Estimation of Job Datasets Transfer. As in the previous case the cost related to move the datasets of a running lower priority job have to be consider before to stop its execution. In some cases it should be convenient to wait the end of the running job and to delay the execution of a new higher priority job.

Job Service Level. If the arriving job has a high service level, it will be executed in any case, despite costs associated to stopping and/or migration of a running job.

Job scheduling mechanisms can be supported by heuristics able to estimate costs such as those above mentioned. Moreover, policies to balance the workload among both the clusters and the cluster nodes plays an important role in order to efficiently exploit the available resources, to recover failures, and bad decisions taken on the basis of wrong estimates of job duration.

2.2 Grid Scheduling Architecture

The architecture of a scheduler may depend on the number of resources managed, and the domain in which resources are located. In general, we can distinguish three different models for structuring schedulers: Centralized, Decentralized, Hierarchical [36],[39].
The first one can be used for managing single or multiple resources located either in a single or multiple domains. It can only support a uniform scheduling policy and suits well for cluster management (or batch queuing) systems such as Codine [40], LFS [41], and Condor [42]. It is not suitable for Grid as they are expected to deal with local policies imposed by resource owners.
The Decentralized model seems to fit for Grid environments. In this model the schedulers interact among themselves in order to decide which resources should be allocated to the jobs to be executed. There is no central component responsible for scheduling; hence this model appears to be highly scalable and fault-tolerant. The resource owners can define their policies that the scheduler has to enforce. However, because the status of remote jobs and resources is not available at single location, the possibility to design highly efficient schedules is questionable.
The Hierarchical model seems to better fit for Grid environments as it allows remote resource owners to enforce their own policy on external users, and removes single centralization points. This model looks like a hybrid model (combination of central and decentralized model). The scheduler at the top of the hierarchy is called Super-Scheduler or Resource-Broker. It interacts with the local schedulers in order to decide schedules. This

scheduler model constitutes the first step to build multilevel schedulers in which scheduling functionalities are distributed over some layers [43], of which some are closer to the user (higher-level scheduling instance) while others are closer to the resource (lower-level scheduling instance). A Grid scheduling layer must exploit the capabilities of the local scheduling systems to make efficient use of the corresponding Grid resources. However, those capabilities are not the same for all local scheduling systems due to system heterogeneity. Therefore, a negotiation process is required to resolve the features of lower-level scheduling instances that can be used by a higher-level scheduling instance.

In [9], the general architecture of a Grid scheduler is delineated. According to it, Grid scheduling involves three main phases: *Resource Discovery*; *Resource/System Selection*; and *Job Execution* (see Figure 3).

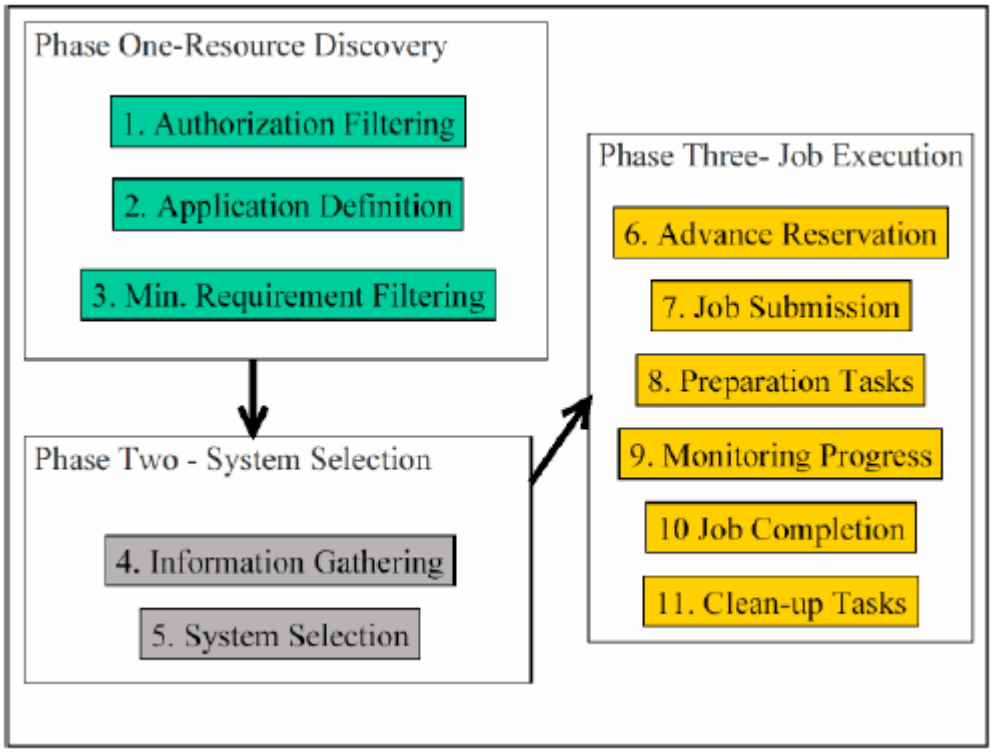


Figure 3: Steps of a general Grid scheduler. (Source: [20])

Resource Discovery is an important task within Grid RMS. In fact, in realistic Grid applications, it may generally be infeasible for users to manually find and specify all the needed resources during the composition of jobs. Therefore, Resource Discovery requires a standard way to express application requirements with respect to the resource information stored in the Grid Information System (GIS). It is thus needed an agreed-upon schema to describe the attributes of the systems, in order to understand what the values mean for different systems. This is an area of on-going work and research, with considerable debate about how to represent a schema (using LDAP, XML, SQL, etc.) and what structure should be inherent to the descriptions. Another important problem regards authorization filtering, which also requires a secure and scalable user accounting system. The availability of secure GIS publishing mechanisms should allow publishing user account information directly in the GIS, from which this information should be automatically retrieved by the scheduler [50].

Resource Selection usually occurs after the Resource Discover phase. While the first phase filters out unsuitable resources, this phase should determine from the list of suitable resources the best (subset of) resource(s) to map the application. This selection requires gathering detailed dynamic information from resources, e.g. by accessing local resource description repository or querying performance systems such as Network Weather System. This information should be used to rank resources, and to allow the scheduler to choose the ones that should ensure high performance in the execution of applications. While resource selection can be quite simple for sequential jobs, this task could become particularly complex for parallel applications. The problem of selecting and co-allocating sets of resources for parallel application has been discussed in [44].

The *Job Execution* phase can be very complex since the preparation of a job run can require various intermediary steps, like staging of files, advance reservation, etc. In addition, due to the dynamic nature of Grids, in which resource availability can change constantly, there is need of support of an automatic way to assign tasks to

resources. One of the main activities of this phase regards the monitoring of the progress of an application execution. In case a job execution is not making sufficient progress or is not meeting the required service level, the scheduler may stop the job execution and reschedule it by returning to Step 4 (see Fig. 1). Such rescheduling is significantly harder for parallel job executing on multiple sites. Dynamic scheduling algorithms are needed to perform this kind of operations.

Regarding scheduling strategies adopted within a RMS, we can distinguish the following main ones: system oriented, resource oriented and application oriented.

- *system-oriented* strategies (also known as high-throughput schedulers) adopt performance functions related to the overall system performance in executing the workload.
- *resource-oriented* schedulers aim at optimizing resource utilization or at imposing fairness criteria; here, the focus relies on the resource-oriented performance criteria;
- *application-oriented* (also known as *job-oriented*) scheduling strategies concentrate on the improvement of the performance of individual jobs.

As usual, there is a trade-off between the different approaches. Several authors believe that schedulers should provide interfaces whereby external agents can change the scheduling policy if needed. A similar approach is being adopted by the GrADS project, where the metascheduler service is an attempt to find a trade-off between the need to provide high performance to the individual applications, and to increase the throughput of the system, for example by preempting an executing application to improve the performance contract of a new application when the system appears to be overloaded.

Besides the scheduling strategies employed by the RMS, schedulers may be classified on the basis of the level at which they work:

- *process-level* schedulers are OS-level components that coordinate the execution of local processes and deal, e.g., with efficient local CPU usage (such schedulers are generally not considered part of GRM systems);
- *task-level* schedulers coordinate the execution of single tasks;
- *resource-level* schedulers deal with resource usage;
- *meta-schedulers* coordinate among other, usually local schedulers, i.e., they examine a set of potential schedules for jobs to be run on different resources and compute an overall schedule.

In Grid environments, all of these kinds of schedulers must co-exist, and they could in general pursue conflicting goals. For instance, improving the performances of a single job can worsen those of other jobs and of the overall system (and vice versa). Similarly schedulers operating at different levels might fail to adequately take into account the activities of another scheduler. That is, planning for future task allocations on resources can fail as resources are also considered by other schedulers. Thus, there is need for implicit or explicit interaction between the different schedulers in order to execute the tasks.

Figure 4 shows an example of two level Grid scheduler architecture, in which the first level (Super scheduler), the highest abstraction, schedules the jobs and their tasks among the available resources. The lower level (local scheduler) implements scheduling algorithms needed to schedule the job's tasks (or only job if it is internally sequential) on the selected resources.

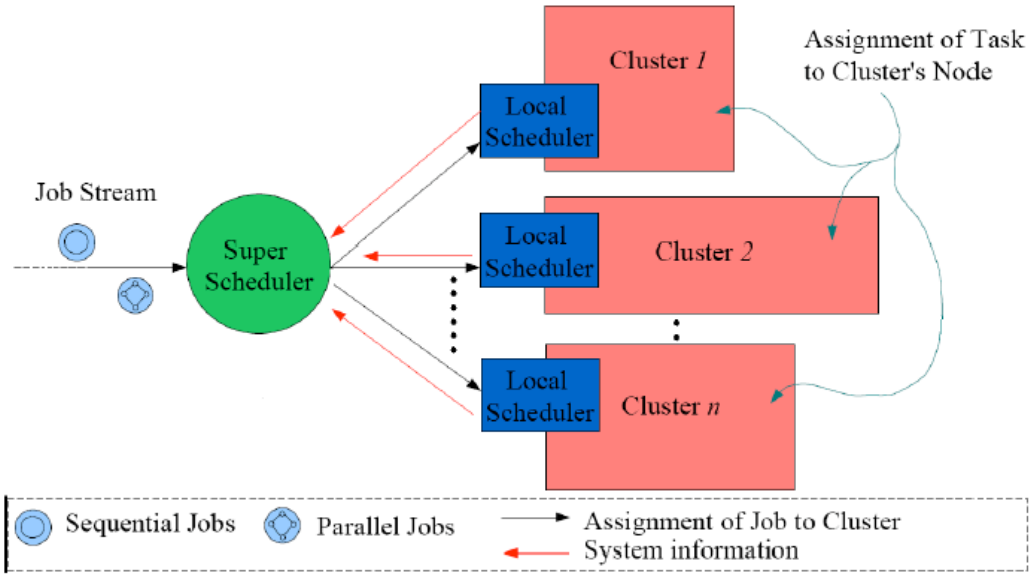


Figure 4: Example of two levels scheduling architecture

Typically, the high scheduling level has to handle a stream of submitted jobs, while the low level handles the scheduling and assignment of tasks on each cluster. The high level should coordinate all scheduling activities performed at low level, and it should act as a broker in routing each job (or batch of jobs) to the low level according to a given policy.

2.3 Scheduling Algorithms

On the basis of the amount and consistency of the knowledge available at each level of the scheduling architecture (e.g. application structure and computational requirements, service level provided by each cluster), several policies could be exploited. In the past many job-scheduling algorithms have been proposed [51][52][53][48][50][49] and classified according to several taxonomies [54][49]. One important classification is made between *on-line* and *offline* algorithms [49]. The notion of an on-line scheduling is intended to formalize a scenario, where the algorithm does not have a-priori information about input jobs, unlike the offline algorithms. It learns the input piece by piece, and has to react to the new requests with only a partial knowledge of the input. According to this classification, examples of on-line algorithms are: First-Come-First-Serve (FCFS) [52], Backfilling [53], List Scheduling [48], and Sufferage [39][63]. In the offline scheduling scenario, the sequence of jobs is known in advance, scheduling is based on information about all jobs in the sequence. Some examples of off-line scheduling algorithms are: SMART [62][60], Preemptive-Smith-Ratio-Scheduling (PSRS) [59], Longest Reference First (LRF)[58].

Another important classification distinguishes such algorithms as *static* or *dynamic* according to the nature of the scheduler. In static scheduling, jobs are assigned to the appropriate resources before their execution begins. Once started, they run on the same resources without interruption. Opposed to static, dynamic algorithms allow the re-evaluation of already determined assignment decisions during job execution. They can trigger job migration or interruption based on dynamic information about both the status of the system and the application workload. Several parameters such as: Job Priority, Estimation of Job Completion Time, Estimation of Job Datasets Transfer, or Job Service Level have to be considered in this phase. In Figure 5 the main goals and features of a dynamic scheduler are summarized.

	Goals	Mechanisms
Dynamic Scheduler	**System Throughput:** - Maximize the resource usage, - Balancing of the resource workload, - Respect of the QoS contract, - Respect of the timing slicing, assigned to the users **Application Performance:** - Maximize throughput of applications, - Respect of the QoS contract	**Monitoring System:** - Resource status, - Application Service Level Agreement (SLA) **Rescheduling Mechanisms:** - Migration (Checkpointing, Restart) - Reconfiguration System

Figure 5 Dynamic scheduler: goals and mechanisms.

Due to the dynamic nature of Grids and the lack of information on resources and jobs, the reliability of job execution is a crucial topic for Grid RMS. Here, often information on resource availability and performance is required prior to the decision process. Service Level Agreements (*SLAs*) are typically considered as suitable means to support the necessary reliability. Thus, resource reservations and Quality-of-Service guarantees are being pointed out as a major requirement of modern Grid RMS. In addition, more approaches have been proposed to improve reliability and performance of resources including e.g. performance prediction, job replication, and dynamic adaptation to performance changes.

Furthermore, scheduling and assignment decisions must be taken as fast as possible in order to minimize the aging effect. Moreover, policies to balance the workload among both, the Grid site and the site computational nodes, play an important role in order to efficiently exploit the available resources, to recover failures, and possible bad decisions taken on the basis of wrong estimates of job duration.

The objective of this deliverable is to provide a description and characteristics of the solutions proposed by the researchers participating to the activity conducted within the task 6.2 of WP6. The CoreGRID Institute RMS proposed a Grid scheduling architecture in [64] for which suitable algorithmic approaches are sought.

3 Grid Scheduling Approaches

Several partners in CoreGRID work on the implementation of Grid scheduling solutions. In the following, we will present an overview on the available approaches. Some of these solutions appear later following the survey results. Thus, a short introduction to the different systems is provided to show the similarities as well as the differences. This can be a starting point to discuss and investigate the interoperability of some solutions which have a similar structural approach.

The following Grid schedulers are considered:

- The Grid scheduler for the YML framework
- The OAR batch scheduler
- The Koala Grid scheduler
- The VIOLA Meta Scheduling Service
- The KnowledgeGrid scheduler
- The GRMS system
- The Intelligent Grid Scheduling System ISS

3.1 A multilevel scheduler for the Grid computing YML framework

Introduction

The multilevel scheduling model we propose is currently integrated in the YML framework [1] which will be described in the next section.

The context we focus on is characterized by the following points:

- the objective of the Grid is High Performance Computing;
- the applications are workflow based;
- the applications are mostly compute intensive rather than data intensive;
- the resources are owned by different providers and part of different VOs;
- the resources are managed by different middleware systems within the Grid infrastructure;
- the number of resource providers is about tens to hundreds.

Our approach aims at being decentralized and can be used in totally different scenarios like cooperative or commercial Grids. The goal is to schedule YML components issued from workflow applications submitted by one or more users. Guaranteed completion time has to be provided to the users which eventually pay for the resource usage depending on the negotiated access policy.

YML framework

YML is a framework providing tools to parallelize applications which has been developed at CNRS/PRiSM in collaboration with InriaFuturs/LIFL[2][3]. It focuses on two major aspects: the development of parallel applications and their execution in a Grid environment. YML makes this development independent of the Grid middleware used underneath and hides differences between them.
On the YML point of view, an application is divided into different computing sections, each of them containing some tasks executed sequentially or concurrently. A task, called a component, is a piece of work that can be mapped to one node in a parallel environment. It has some input and output parameters and is generally reusable in different parts of the application as well as in different applications. YML provides a special type of components, called graph component, which consists of the description of sub graphs. YML divides the development of a parallel application in three major steps:

A definition of new components. This definition consists in an Abstract and Implementation component description.
A description of the parallel application. This description is independent of any underlying middleware and makes use of the components as functional units. It specifies parallel and sequential parts of the application using the YvetteML graph description language and provides notifications to synchronize the execution of dependent components. This description is directly deduced from the graph representation of the application.
The compilation of the application. This step analyses and transforms the application graph into a list of parallel tasks with respect of the precedence constraints.

The three steps above are all middleware independent and ensure that no Grid relevant knowledge is needed to develop parallel applications.
After the compilation of the application, the execution can be started using the Workflow Scheduler which will interact with the underlying middleware and submit tasks through the dedicated backend. Each task is launched by a YML worker which will contact the Data Repository Server to obtain the component binary and input parameters to start the computation. The use of Data Repository Servers hides the data migrations to the developer and ensures that the necessary data is always available to all components of the application.

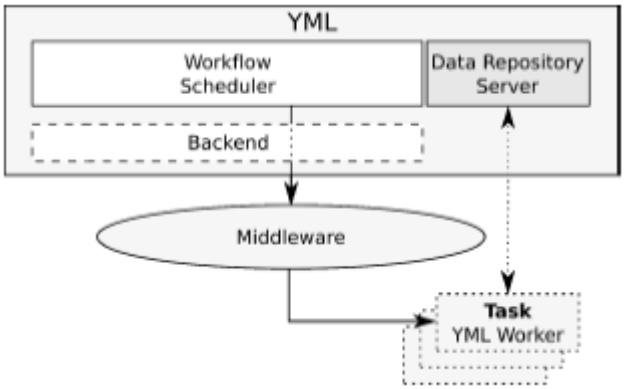


Figure 6: YML Workflow Scheduler interaction with the middleware

Multilevel scheduling model

The Workflow Scheduler presented in Figure 6 is currently adapted to propose multilevel scheduling features by integrating a new model described in this section.

This model is based on an economic approach of resources and defines different entities, which will interact in the Grid infrastructure. An entity can be a resource provider or consumer or both. Consumers require resources of the Grid which are owned by providers. When a provider receives a request from a consumer, he will answer by proposing a set of suitable schedules and associated cost for parts of the application regarding access policy of the consumer and availability of local resources. He can possibly subcontract parts or the whole application to other resource providers without mentioning anything to the consumer.

This model can be used in different scenarios: either cooperation or competition between sites in the Grid infrastructure. Moreover, a hierarchy with different layers of scheduling instances can be built with this model.

A new joining instance in the Grid infrastructure has to negotiate access policies with one or more resource providers. Each access policy will provide a set of static and dynamic parameters which will determine the usage conditions of the provided resources. These parameters will be used when an application is submitted, to obtain the list of suitable resources: static parameters represent permanent access conditions and do not require any interaction with other scheduling instances. In opposition, dynamic parameters will need to request each resource providers in order to filter the set of suitable resources.

As presented in Figure 7, the main idea is to provide a YML server to each Local Resource Manager (LRM). This YML server has 3 main objectives:

- to communicate with other YML servers and therefore, connect the different clusters in a common Grid;
- to interact with the underlying LRM using a specialized backend;
- to provide needed features missing in the LRM (as cluster (b) in Figure 7).

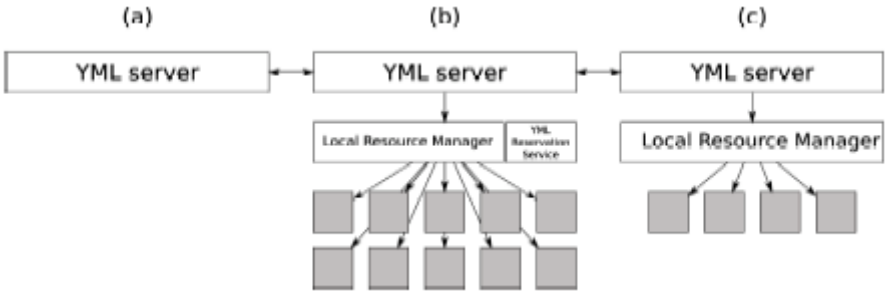


Figure 7: YML server cooperation

When a user wants to execute an application on the Grid infrastructure, he contacts the local YML server which analyses the application and decides whether it can provide needed resources or not (eventually, it may have no local resources like cluster (a) in Figure 7). It may forward the whole or parts of the request to other resource providers. This step will ensure collaboration between sites and a distribution of the global work in the whole infrastructure. Then, suitable schedules are sent back in return of each request and the user local YML server finally gathers the information and proposes different bids.

3.2 OAR Batch Scheduler

Architecture

From the start, OAR has been designed as an opened platform for research and experiments. The main contribution of OAR is its design, based on two high-level components: a SQL database (a central relational database engine) and the executive part. The database is used as the only mean to exchange information between modules, thus ensuring a complete opening of the system. The database engine is used to match resources (using the rich expressive power of SQL queries) and to store and exploit logging and accounting information. Additionally, a general purpose database is the best choice to ensure friendly and powerful data analysis and extraction. The most important benefit of this approach is that the powerful SQL language can be used for data analysis and extraction as well as for internal system management. Another advantage of using a standard database engine is that we should benefit of its robustness and efficiency. Although making SQL queries might induce some overhead compared to a specific implementation, the engine we use has good behavior under high workload. The database engine has few chances of being a bottleneck for system scalability as it can handle efficiently thousands of queries simultaneously (far more than we currently need). Furthermore, robustness only depends on modules that just have to let the system in a coherent state and might otherwise fail without much harm.
The actual executive part is completely written in Perl, which is perfectly suited for system tasks. It is made of several modules, including one for launching and controlling the execution of jobs and another for scheduling jobs. The monitoring tasks are handled by a separate tool (Taktuk) that is called from OAR and interfaced with the database. One of the goals of OAR is to make a research platform suited for scheduling experiments and resulting analysis. To help developers modifying the system, we made it very modular and the implementation in a high-level language makes the system rather small and extensible.

Job Submission

The submission of jobs in OAR works like in PBS: the interface is made of independent commands for submission (oarsub), cancellation (oardel) or the monitoring (oarstat). These commands are as separated as possible from the rest of the system, they send or retrieve information using directly the database and they interact with OAR modules by sending notifications to the central module. Job submission starts by a connection to the database to get the appropriate admission rules. These rules are used to set the value of parameters that are not provided by the user and to check the validity of the submission. Possible parameters include a queue name, a limit on the execution time, the number of needed nodes, and so on. The rules are stored as Perl code in the database and might be used to call an intermediate program so the admission can be as elaborate and general as needed.

Scheduling

One of the objectives of OAR is to be a simple and opened platform for experimentations and research. So, although the scheduler implemented in OAR is rich in functionalities, the algorithms its uses are still rather simple. All the most important functionalities such as priorities on jobs, reservations, resources matching and backfilling are implemented. The priorities are managed through submission queues. All the jobs are submitted to a particular queue which has its own admission rules, scheduling policy and priority. Reservations are a special case in which the user asks for a specific time slot. In this case, as long as the job meets the admission rules and the resources are available during the requested time slot, the schedule date of the job is definitively set. In our scheduler, resources required by jobs are matched with available ones as a user might need nodes with special properties (like single switch interconnection, or a mandatory quantity of RAM). Our scheduler also performs backfilling (use of idle time slots when large parallel jobs are waiting for execution) and handles Best Effort jobs (jobs that can be cancelled before the end of their allowed time).
The scheduling of all the jobs in the system is computed by a module we called ”metascheduler” which manages reservations and schedule each queue using its own scheduler. This module maintains an internal representation of the available resources similar to a Gantt diagram and updates this diagram by removing time slots already reserved. Initially, the only occupied time slots are the ones on which some job is executing and the ones that have been reserved. The whole algorithm schedules each queue in turn by decreasing priority using its associated scheduler.

Best-effort jobs

More and more systems are being constituted using idle machines of local private networks, accesses to clusters or even machines on the Internet. The use of such systems is termed either Global or Desktop computing (Condor, XtremWeb). This is usually implemented using some mechanism that detect idleness of a resource, get some task to be executed and perform the work. When the host resource is claimed back for its normal use, it is immediately restored (possibly aborting a Desktop computing task also termed Best effort task). Most of the time, these systems are designed to be as transparent as possible to the user. However, when using idle nodes of a cluster for Global computing, this transparency is not necessarily desired. The main problem is that regular jobs submitted to the batch scheduler are parallel which is not compatible with best effort jobs managed by the node itself (because all the nodes have to be freed in parallel). Furthermore, only a batch scheduler can manage parallel best effort jobs (because when one participating node is claimed by the user all the other should stop as well). So, to handle correctly best effort jobs, the batch scheduler has to manage them itself.
For a given job, best effort property is set by the module that validates incoming jobs. It is currently done when submitting a job to a waiting queue dedicated to best effort tasks. The scheduler should also have the possibility to cancel these jobs when their resources are required for the execution of some other task. In OAR this is made in two steps: first by setting flags on jobs from the scheduler (request for cancellation), which is then handled by a generic module in charge of all cancellations in the system, then by scheduling the waiting job when coming back to the scheduler.
Implementing Global computing this way requires that information for best effort jobs are propagated from the resources management function, through the scheduler, up to the central module, in order to be there after a job was transmitted to the cancellation module. Although several layers of the system are changed, this approach is still compatible with our initial layout of modules organization. It demonstrates the extensibility of OAR: because the system is small, it is still possible to modify several modules from all layers of the system. Further extensions could include choice policies for the job to cancel (for instance by startup date order, so that the youngest job is cancelled first in an attempt to let the oldest progress, or by the number of used nodes, so that the number of canceled jobs is minimized). Once again these modifications to the system are quite small and simple to perform. This is the consequence of the good modularity, the open internal state (the database) and the high level design (Perl) in OAR.

Final remarks

We have validated our design objectives for OAR by implementing from the initial system a policy for Global Computing jobs and by showing the good level of performance of our system in comparison with other systems. Another convincing validation is that OAR is currently used for the exploitation of a lightweight Grid of 700 processors in the project CiGri and has shown very good robustness up to now. OAR is also used as a cluster level scheduler in Grid5000, a 5000 CPUs nationwide infrastructure for research in Grid computing. Ultimately, OAR demonstrates that it is possible to build a complete functional and efficient batch scheduler from just a database engine and a scripting language, which was not obvious from start. Future works on OAR will aim the exploitation of this platform for research purposes including the implementation of theoretical advances in scheduling and clusters management (such as malleable jobs, heterogeneous platforms, unreliable network, Grid Computing, and so on).

3.3 The KOALA Grid Scheduler

In multi-cluster systems, and more generally, in grids, jobs may require co-allocation, i.e., the simultaneous allocation of resources such as processors and input files in multiple clusters. While such jobs may have reduced runtimes because they have access to more resources, waiting for processors in multiple clusters and for the input files to become available in the right locations, may introduce inefficiencies. Moreover, as single jobs now have to rely on multiple resource managers, co-allocation introduces reliability problems. In this section, we describe the design and implementation of a co-allocating grid scheduler named KOALA1, 1which was developed at Delft University of Technology as a two-level scheduling strategy in grids, and which has been in operation in the Distributed ASCI Supercomputer (DAS) in the Netherlands since September 2005.

Co-allocation in Grids: The Model

In this section we present the version of the co-allocation problem that we address with the design and implementation of our KOALA scheduler. Below, we first describe the homogeneous testbed that we use for the design, implementation, and testing of KOALA. Then, we present our model of co-allocation in multi-clusters and in grids.

The Distributed ASCI Supercomputer

The Distributed ASCI Supercomputer (DAS) [15] is an experimental computer testbed in the Netherlands that is exclusively used for research on parallel, distributed, and grid computing. This research includes work on the efficient execution of parallel applications in wide-area systems [17][18], on communication libraries optimized for wide-area use [12][14], on programming environments [12][13], and on resource management and scheduling [6][7][8]. The current, second-generation DAS system consist of five clusters of identical processors (in total 400), which are interconnected by the Dutch University backbone for wide-area communications. For local communication within the single clusters, Myrinet LAN is used. Each of the DAS clusters is an autonomous, independent system. Until mid 2005, all the DAS clusters used OpenPBS as the local resource manager. However, due to unreliability problems after the latest upgrade of OpenPBS, the decision was made to change the local resource manager of the clusters to the Sun N1 Grid Engine (SGE). All DAS clusters have their own file system. Therefore, in principle files (including executables) have to be moved explicitly between users' working spaces in different clusters.

The Job Model

By a job we mean a parallel application requiring files and processors that can be split up into several job components which can be scheduled to execute on multiple execution sites simultaneously (co-allocation) [5][9][10]. This allows the execution of large parallel applications requiring more processors than available on a single site [10][11]. Jobs that require co-allocation in grids may or may not specify the number and sizes of their components. Based on this, we consider three cases for the structure of the job requests:

Fixed request: The job request specifies the numbers of processors it needs in all the clusters from which processors must be allocated for its components.

Non-fixed request: The job request only specifies the numbers of processors it needs in the separate clusters, allowing the scheduler to choose the execution sites. The scheduler can either place the job components on the same or on different sites depending on the availability of processors.

Flexible request: The job request only specifies the total number of processors it requires. It is left to the scheduler to decide on the number of components, the number of processors for each component, and the execution sites for the components.

The Design of KOALA

In this section we describe the design of the KOALA grid scheduler, which is fully functional in the DAS system. Below, we describe the components of KOALA and how they work together to achieve co-allocation.

The KOALA components

The KOALA scheduler consists of the following five components: the Co-allocator (CO), the Information Service (IS), the Data Manager (DM), the Processor Claimer (PC), and the Runners. The structure of KOALA is depicted in Figure 8. The CO is responsible for placing jobs, i.e., for finding the execution sites with enough idle

[1] The name KOALA was solely chosen for its similarity in sound with the word co-allocation.

processors for their components. The CO chooses jobs to place based on their priorities from one of the KOALA placement queues. If the components require input files, the CO also selects the file sites for the components such that the estimated file transfer times to the execution sites are minimal. To decide on the execution sites and file sites for the job components, the CO uses one of our placement policies. Finding execution sites for the job components is only done for non-fixed job requests.
The IS is comprised of the Globus Toolkit's Metacomputing Directory Service and Replica Location Service (RLS), and Iperf, a tool to measure network bandwidth. A repository containing the bandwidths measured with Iperf is maintained and updated periodically. The MDS provides on request the information about the numbers of processors currently used and the RLS provides the mapping information from the logical names of files to their physical locations. Requests to the MDS and the bandwidth repository impose delays on placing jobs. Therefore, the IS caches information obtained from the MDS and the bandwidth repository with a fixed cache expiry time (a parameter of KOALA). Furthermore, the IS can be configured to do periodic cache updates from frequently used clusters before their cache expiry time.
The DM is used to manage file transfers, for which it uses both Globus GridFTP [4] and Globus Global Access to Secondary Storage (GASS) [16]. The DM is responsible for ensuring that input files arrive at their destinations before the job starts to run.
After a job has been placed, it is the task of the PC to ensure that processors will still be available when the job starts to run. If processor reservation is supported by local resource managers, the PC can reserve processors immediately after the placement of the components. Otherwise, the PC uses KOALA's claiming policy to postpone claiming of processors to a time close to the estimated job start time.
The Runners are used to submit job components to their respective execution sites; they allow us to extend the support for different application models in KOALA.

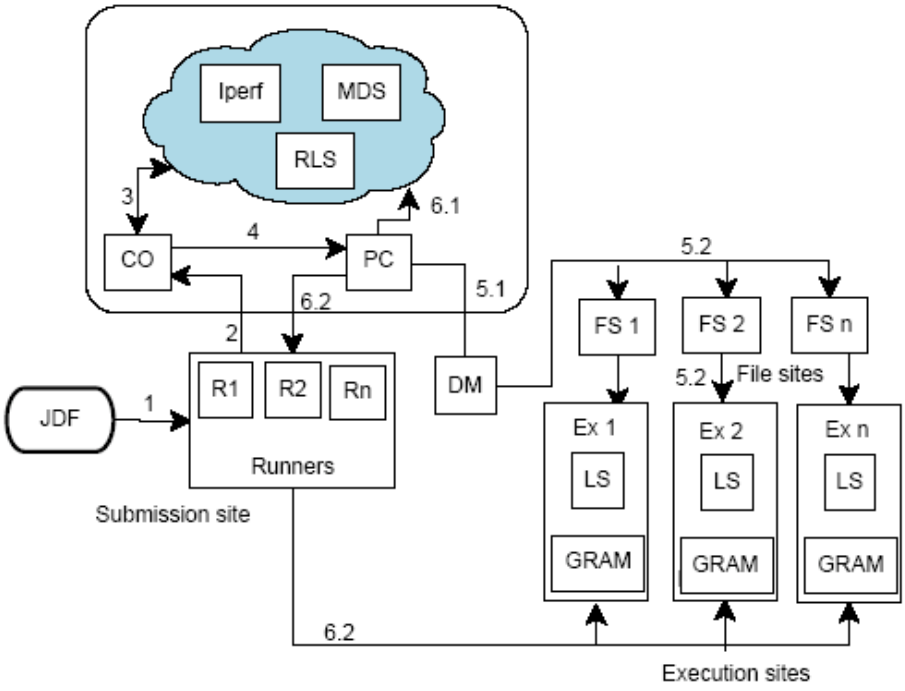


Figure 8: The interaction between the KOALA components.

KOALA Phases

A job submitted to KOALA goes through four phases, which are placing its components, transferring its input files, claiming processors for its components, and starting and monitoring its execution (see Figure 9).
In phase 1, a new job request arrives at one of the Runners (arrow 1 in Figure 8) in the form of a Job Description File (JDF). We use the Globus Resource Specification Language (RSL) [16] for JDFs, with the RSL "+"-construct to aggregate the components' requests into a single multi-request. After authenticating the user, the Runner submits the JDF to the CO (arrow 2), which in turn will append the job to the tail of one of the KOALA placement queues. The CO then retrieves the job from this queue and tries to place the job components based on information obtained from the IS (arrow 3). If the job placement fails, the job is returned to its respective placement queue. The placement procedure will be tried for the jobs in the placement queues at fixed intervals for a fixed number of times.

Phase 2 starts by the CO forwarding the successfully placed job to the PC (arrow 4). On receipt of the job, the PC instructs the DM (arrow 5.1) to initiate the third-party file transfers from the file sites to the execution sites of the job components (arrows 5.2).
In phase 3 the PC estimates the Job Start Time and the appropriate time that the processors required by a job can be claimed. At this time and if processor reservation is not supported by the local resource managers, the PC uses a claiming policy to determine the components that can be started based on the information from the IS (arrow 6.1). It is possible at the Job Claiming Time for processors not to be available anymore, e.g., they can then be in use by local jobs. If this occurs, the claiming procedure fails, the job is put into the claiming queue, and the claiming is tried again at a later time.
In phase 4, the Runner used to submit the job for scheduling in phase 1 receives the list of components that can be started (arrow 6.2) and forwards those components to their respective execution sites. At the execution sites, the job components are received by the Globus Resource Allocation Manager (GRAM), which is responsible for authenticating the owner of the job and sending the job to the local resource manager for execution.

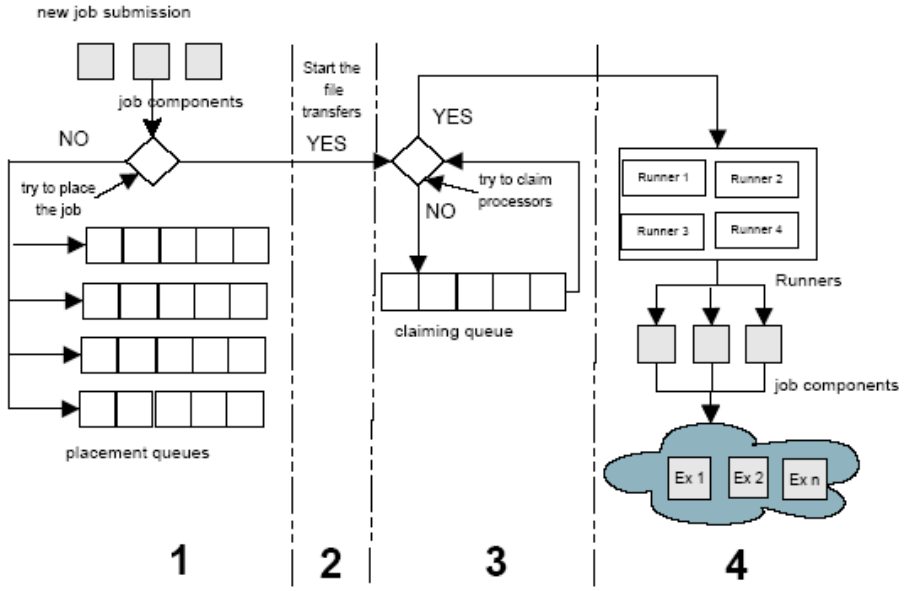


Figure 9: The four phases of job scheduling in KOALA.

3.4 The MetaScheduling-Service MSS

Introduction

To successfully execute distributed applications or workflows, usually different resources like compute nodes, visualization devices, storage devices, or network connectivity with a defined QoS are required at the same time or in a certain sequence. Orchestrating such resources locally within one organization represents only a minor task, whereas the orchestration of resources on a Grid level requires a service that is able to solve the same problems in an environment that may stretch across several administrative domains. Additional conditions have to be taken into account, like the compliance with site-specific policies or the protection of a site's autonomy. In the VIOLA project [24] we have developed such a meta-scheduling service (MSS), which is capable to co-allocate different types of resources (currently compute resources and network resources) in multiple administrative domains.

Required Functionality of the Meta Scheduling Service

To achieve co-allocation of resources managed by multiple, usually different scheduling systems, the minimal requirement these systems have to fulfill is to provide functions to

- schedule a single reservation some time in the future (e.g. "from 5:00 pm to 8:00 pm tomorrow") and to
- give an aggregated overview of the usage of the managed resources between now and a defined time in future.

Once a reservation is scheduled the starting time for it is fixed, i.e. it may not change except for the reservation being cancelled. This feature is called advance reservation. There are at least two possibilities to realize such advance reservation. The first possibility is to schedule a reservation for a requested time, called fixed time scheduling. The second possibility is to schedule a reservation not before a given time, which means a scheduling system tries to place the reservation at the requested time, otherwise it will be scheduled for the earliest possible time after the one requested. This results in a first fit reservation. The implementation of the MetaScheduling Service described here interacts with, but is not limited to, scheduling systems that implement the first fit behavior. The main function of the MetaScheduling Service is to negotiate the reservation of network-accessible resources that are managed by their respective local scheduling systems. The goal of the negotiation is to determine a common time slot where all required resources are available for the requested starting time of the job. The major challenges for a meta-scheduler are

- to find Grid resources suitable for the user's request,
- to take security issues like user authentication and authorization into account,
- to respect the autonomy of the sites offering the resources, and
- to cope with the heterogeneity of the local scheduling systems.

We do not address the problem of finding suitable resources here; this task is usually delegated to a Grid information system or a resource broker. Security issues are only considered here with respect to user authentication and authorization as the MetaScheduling Service has to reserve on behalf of the user's respective identity at the sites that he wants to use resources from. The implementation described below addresses both site autonomy and heterogeneity.

Implementation of the MetaScheduling Service

Figure 10 gives an overview on the multi-level scheduling architecture [30]. To interact with different types of scheduling systems we decided to use the adapter pattern approach. The role of such adapters is to provide a single interface to the MetaScheduling Service by encapsulating the specific interfaces of the different local scheduling systems. Thus the MetaScheduling Service can negotiate resource usage using a single interface. The adapters are connected to both the MetaScheduling service and the scheduling systems and may therefore be installed either at the site hosting the MetaScheduling Service, the (remote) sites where the scheduling systems are operated, or at any other system accessible through a network. Currently adapters are available for the following scheduling systems: EASY, the Portable Batch System Professional (PBS Pro), the Open Portable Batch System (OpenPBS) with the MAUI Scheduler, and one for ARGON, the resource management system for network resources developed in the VIOLA project [24].

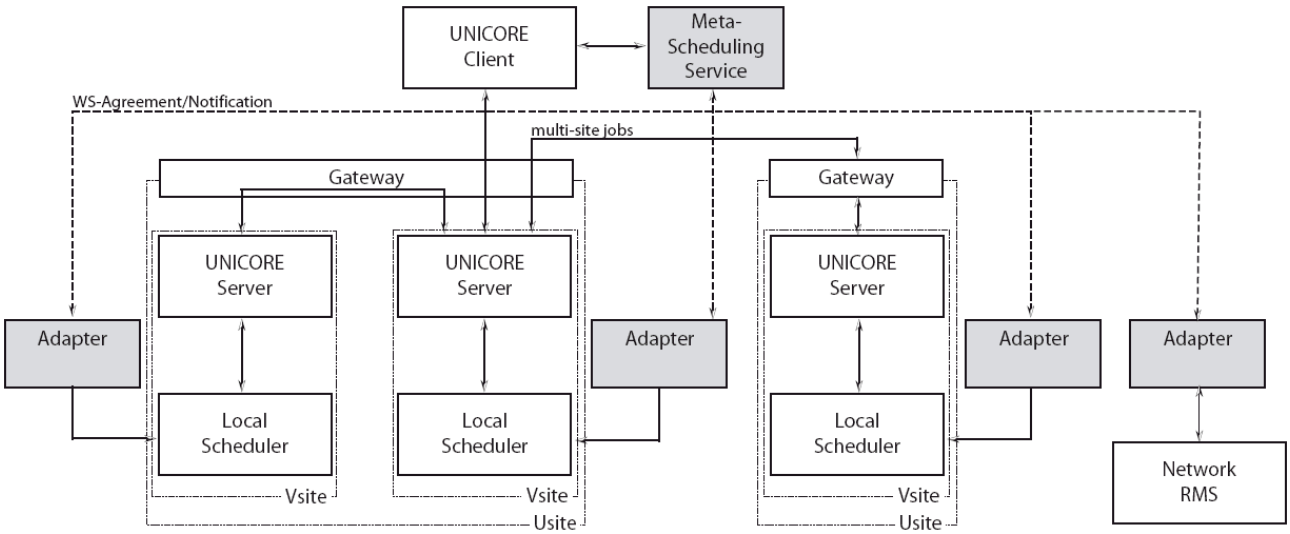


Figure 10: Multi-level Scheduling Architecture

Negotiation protocol

In this section we describe the protocol the MetaScheduling Service uses to negotiate the allocation of resources with the local scheduling systems. This protocol is illustrated in Figure 11. The user specifies the duration of the meta-job and additionally -for each subsystem -reservation characteristics like the number of nodes of a cluster or the bandwidth of the connections between nodes of different clusters. In the UNICORE based VIOLA testbed the UNICORE client is used to describe the request of the user. The client sends the job description to the MetaScheduling Service using the Web Services Agreement (WS-Agreement) protocol [23]. Based on the information in the agreement the MetaScheduling Service starts the resource negotiation process:

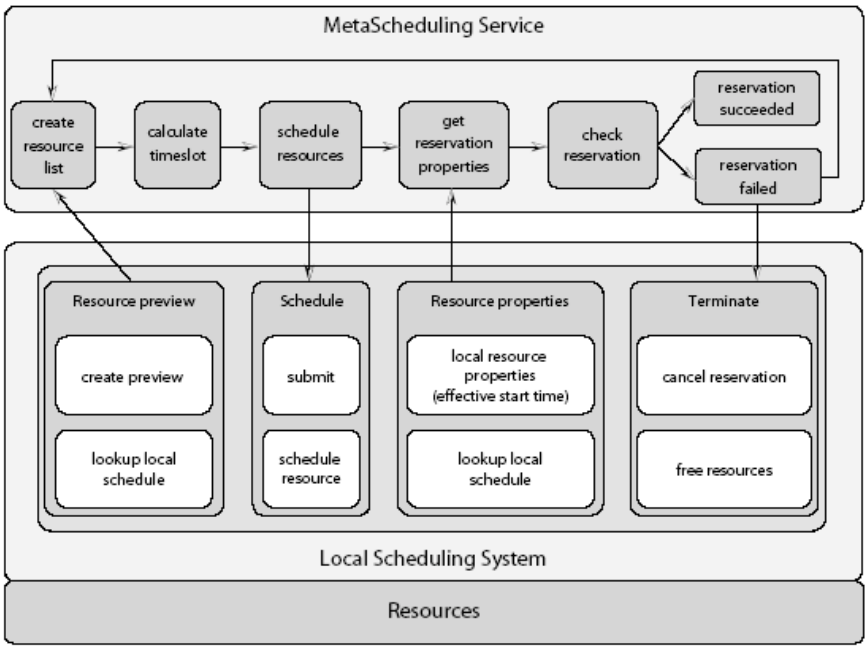


Figure 11: The negotiation process

1. The MetaScheduling Service queries the adapters of the selected local systems to get the earliest time the re-quested resources will be available. This time possibly has to be after an offset specified by the user.
2. The adapters acquire previews of the resource availability from the individual scheduling systems. Such a preview comprises a list of time frames during which the requested QoS (e.g. a fixed number of nodes) can be provided. It is possible that the preview contains only one entry or even zero entries if the resource is fully booked within the preview's time frame. Based on the preview the adapter calculates the possible start-time.
3. The possible start times are sent back to the MetaScheduling Service.
4. If the individual start times allow the co-allocation of the resources, the respective resources are reserved via the Adapters of the local schedulers. If the individual start times do not allow the co-allocation of the resources, the MetaScheduling Service uses the latest possible start time as the earliest start time for the next scheduling iteration. The process is repeated from step 1. until a common time frame is found or the end of the preview period for all the local systems is reached. The latter case generates an error condition.
5. In case the individual start times match, the MetaScheduling Service checks the scheduled start times for each reservation by asking the local schedulers for the properties of the reservation. This step is necessary because in the meantime new reservations may have been submitted by other users or processes to the local schedulers, preventing the scheduling of the reservation at the requested time.
6. If the MetaScheduling Service detects one or more reservations that are not scheduled at the requested time, all reservations will be cancelled. The latest effective start time of all reservations will be used as the earliest start time to repeat the process beginning with step 1.
7. If all reservations are scheduled for the appropriate time the co-allocation of the resources has been completed.
8. The IDs of the MetaScheduling Service and the local reservations are added to the agreement and a reference to the agreement is sent back to the client which initially submitted the meta-job (in case of the VIOLA testbed it is the UNICORE client).

After negotiation of all reservations the MSS continuously monitors the partial reservations. When all reservations entered the state running (active), the MSS queries the compute RMS in order to determine which nodes (IP addresses of the nodes) were finally assigned to a reservation. This information is collected and aggregated by the MSS, and is then published as runtime configuration to all subsystems.
Applications can use this information to obtain information on their runtime. Furthermore this information can be used by the network RMS to complete the configuration of network resources in order to provide the QoS defined.

Implementation of the Agreement

The development of the MetaScheduling Service includes, as already mentioned in Section 3.1, an implementation of the WS-Agreement specification [23]. The client submits a request using an agreement offer (a template provided by the MetaScheduling Service and filled in with the user's requirements by the client) to the MetaScheduling Service. It then negotiates the reservation of the resources in compliance with the QoS defined in the agreement template. As a result of the successfully completed negotiation process a valid agreement is returned to the client, containing the scheduled end times of the reservation, the reservation identifier, and further information.

Interface between MetaScheduling Service and Adapter

The interface/protocol between the MetaScheduling Service and the adapters is implemented using Web Services and it basically provides the five functions couldRunAt(), submit(), submit(), cancel(), state(), and bind(). These functions cover the whole negotiation process. The Web Services interface/protocol as implemented now is not standardized. It is planned to replace the current implementation with a WS-Agreement based approach [23] till the end of 2006.

Future work

The meta-scheduling service today is already capable to co-allocate different types of resources and to generate SLA's as a result of such a co-allocation process. Anyway, co-allocation of resources is a very specific task in grid resource management. Therefore we plan to extend the meta-scheduling service to support more complex SLA's. As an initial step we plan to support SLA's down to the resource level by adding WS-Agreement support to the adapters. These resource SLA's will be used by the meta-scheduler for orchestrating resources by adopting existing functionality and adding new functionality, e.g. for scheduling dependencies. The following work will be performed:

- implementing the WS-Agreement protocol for the adapters to support Service Level Agreements at the resource level,
- implementing adapters for additional resources (e.g. licenses and storage),
- adding support for complex SLA's to the meta-scheduling service,
- integrating the Intelligent Scheduling System (ISS) for selection of the candidates best fitted to execute an specific application [29], and
- integrating the Grid Scheduling Ontology [28] to generate SLA templates for a domain, based on the capabilities of the local resource management systems and resources.

3.5 Application-oriented Scheduler within the Knowledge Grid

In this section we show an actual implementation example of how our general model and architecture have been specialized to create a task-level job-oriented scheduler. The implementation has been done in the context of the KNOWLEDGE GRID (K-Grid). The KNOWLEDGE GRID is an architecture built atop basic Grid middleware services that defines more specific knowledge discovery services. The services are organized in two hierarchic levels: the Core K-Grid layer and the High-level K-Grid layer. In the following, the term KNOWLEDGE GRID node will denote a node implementing the KNOWLEDGE GRID services.
The Core K-Grid layer offers the basic services for the definition, composition and execution of a distributed knowledge discovery computation over the Grid. The Core K-Grid layer comprises two main services: the Knowledge Directory Service (KDS) and the Resource Allocation and Execution Management Service (RAEMS). The KDS is responsible for maintaining metadata describing KNOWLEDGE GRID resources. The metadata information, represented in XML, is stored in a Knowledge Metadata Repository (KMR).
The RAEMS comprises three main modules: a Scheduler, an Execution Manager and a Task Monitor. The scheduler is used to find a suitable mapping between an abstract execution plan (job) and available resources, with the goals of (i) satisfying the constraints (computing power, storage, memory, network performance) imposed by the execution plan, and (ii) minimizing its completion time. The instantiated execution plan is then managed by the Execution Manager, that translates it into requests to basic Grid services, submits these requests and, after their execution, stores knowledge results in the Knowledge Base Repository (KBR). Finally, the Task Monitor follows the execution of submitted tasks, and notifies the scheduler about significant events occurred.
The High-level K-Grid layer includes services used to compose, validate, and execute a parallel and distributed knowledge discovery computation. Moreover, this layer offers services to store and analyze the discovered knowledge.
The Data Access Service (DAS) is responsible for the search, selection, extraction, transformation and delivery of data to be mined. The Tools and Algorithms Access Service (TAAS) are responsible for searching, selecting and downloading data mining tools and algorithms. The Execution Plan Management Service (EPMS) manages execution plan represented by graphs describing interactions and data flows between data sources, extraction tools, DM tools and visualization tools. The EPMS allows defining the structure of an job by building the corresponding graph and adding a set of constraints about resources. Generated execution plans are stored, through the RAEMS, in the Knowledge Execution Plan Repository (KEPR). The Results Presentation Service (RPS) offers facilities for presenting and visualizing the knowledge models extracted (e.g., association rules, clustering models, classifications), and offers an API to store them in different formats in the Knowledge Base Repository. As seen above, the KNOWLEDGE GRID scheduler is part of the Resource Allocation and Execution Management Service. Thus, each KNOWLEDGE GRID node has its own scheduler which is responsible for responding to scheduling requests coming from the same or other nodes. Obviously, different scheduler instances can also communicate with each other to exchange the information they need.

The scheduler currently provides the following built-in functionalities:

- Scheduling algorithm. Several algorithms are provided, all potentially producing partial schedules.
- Scheduling process. Events that fire the re-scheduling activity are: (i) the completion of all tasks preceding a pending task in the current schedule; (ii) important performance variations; (iii) task/host failures.
- Data gathering. For the evaluation of current and future CPU availability, in the absence of service level agreements, the scheduler adopts as its information source the Network Weather Service (NWS), if present, or directly the hosts' descriptions. For evaluating the processing requirements of software components, the scheduler makes use of user-provided descriptions, if present, or a sampling method. Finally, for assessing network performances (bandwidth and latency) the scheduler adopts the NWS as well, if present, or a simple probing technique.
- Cost estimation. For estimating computational costs, the scheduler employs the formula

$$\varepsilon(ds;s;p;h;t) = \frac{req(ds;s;p)}{perf(h)\cdot avail(h,t)}$$

where *req(ds; s; p)* represents the processing requirements (with respect to a reference host) of software *s* run with parameters *p* on datasets *ds*, *perf(h)* is the no-load performance of host *h*, and *avail(h; t)* is the fraction of processing cycles available on host *h* at time *t* (this process is based on task profiling and analytical benchmarking. Finally, the scheduler makes use of user-provided descriptions of the relationships between input and output sizes of software components.

3.6 GRMS

Introduction

The Gridge Resource Management System (GRMS) [66] is an open source meta-scheduling system, which allows developers to build and deploy resource management systems for large scale distributed computing infrastructures. GRMS, based on dynamic resource selection, mapping and scheduling, combined with control and monitoring functionality, deals with dynamic Grid environment and resource management challenges including remote job submission and control, file staging, selection of the best resources for end-users, load-balancing among clusters, managing workflows and more. Therefore, the main goal of GRMS is to manage the whole process of remote job submission to various resources such as batch queuing systems or single computational nodes. Finally, GRMS can be considered as a robust system which provides abstraction of the complex Grid infrastructure as well as a toolbox which helps to form and adapts to distributing computing environments.

GRMS is a part of the Gridge toolkit consisting of a set of components useful for efficient Grid management and use. It can take an advantage of various low-level core Grid services, such as e.g. GRAM, GridFTP and Gridge Monitoring System, as well as various Grid middleware services, e.g. Gridge Authorization Service, Gridge Data Management Service and more. All these services working together provide a consistent, adaptive and robust Grid middleware layer which fits dynamically to many different distributing computing infrastructures. The current GRMS implementation requires Globus software to be installed on Grid resources, and uses Globus Core Services deployed on resources: GRAM, GridFtp, MDS (optional). GRMS supports Grid Security Infrastructure by providing the GSI-enabled web service interface for all clients, e.g. portals or applications, and thus can be integrated with any other middleware Grid environment. With the GAS, GRMS is able to manage both, job grouping and jobs within collaborative environments according to predefined VO security rules and policies. With the Data Management services from Gridge, GRMS can create and move logical files/catalogs and deal with data intensive experiments. Gridge Monitoring Service can be used by GRMS as an additional information system. Finally, Mobile service can be used to send notifications via SMS/emails about events related to users. jobs and as a gateway for GRMS mobile clients. GRMS is able to store all operations in a database. Based on this information a set of very useful statistics for both end users and administrators can be produced. All the data is also a source for further, more advanced analysis and reporting tools. GRMS architecture along with interactions with other components are presented in Figure 12.

GRMS Components

GRMS is composed of the following main modules (see Figure 12):

- **Workflow Module.** Provides GSI enabled web service interface for GRMS. This module is responsible for job description validation and putting a job to the queue during job submission process. For workflow jobs it creates a graph representation of tasks and check its correctness.
- **Job Queue.** Stores jobs which are ready for execution. It is prepared for implementation of any queue management strategy and scheduling algorithms.
- **Broker Module.** This is the main component of GRMS. It controls the whole process of job submission: gets jobs from queue, calls the Resource Discovery Module to find appropriate resources, evaluates resources to find the best one (according to a configuration), creates environment for job execution by transferring input data, calls the Job Manager Module to monitor status changes of a job, after a job is finished transfers output data to location specified by a user. It also provides information about jobs and their history in the system.
- **Resource Discovery Module.** Finds resources that fulfill requirements described in Job Description. Resources are described in an XML document which contains parameters important from the job scheduling point of view. The Resource Discovery Module can be configured to use many information sources. It can use, e.g. Globus MDS, iGrid service, Mercury Monitoring Service, and others.
- **Job Manager Module.** Is responsible for monitoring of status changes of a job, and for job control: job canceling, suspending and resuming.
- **Job Registry Module.** Is responsible for storing all information about jobs, and provides this data for other modules.

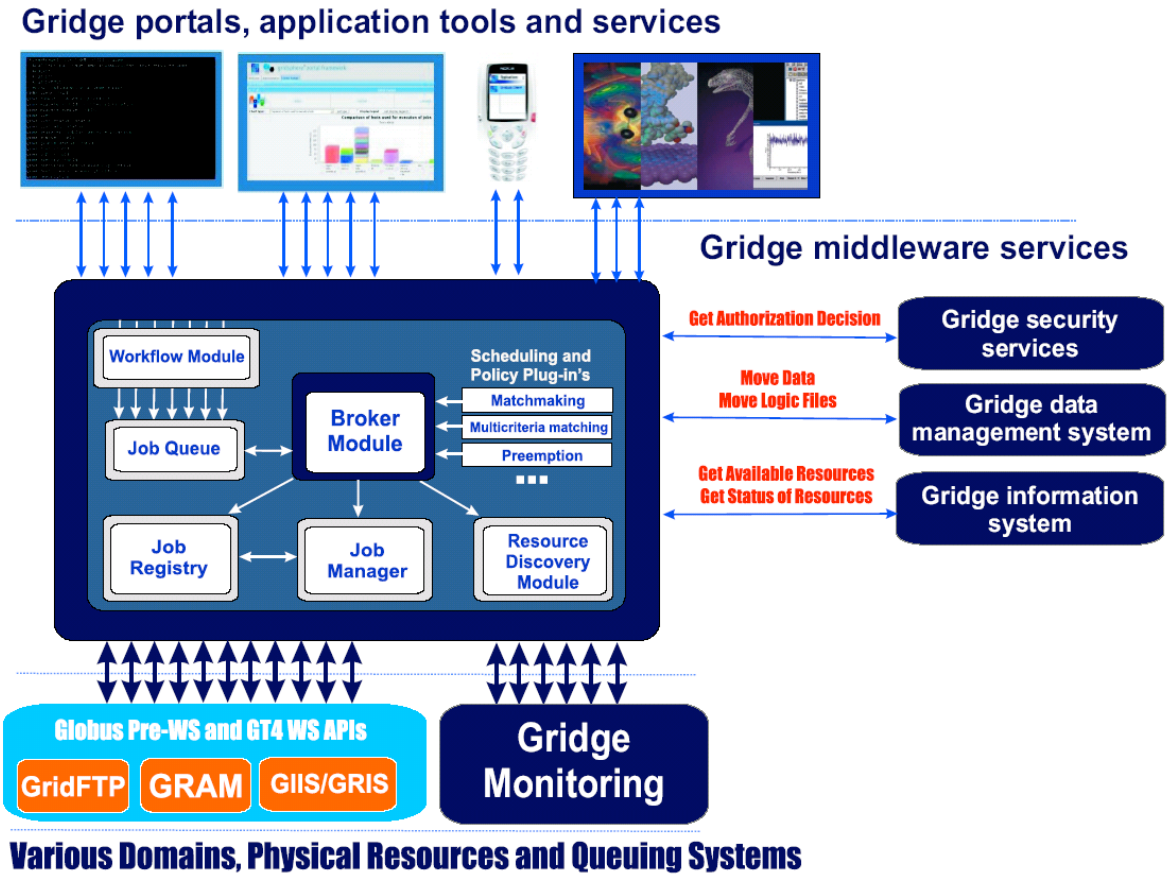


Figure 12: GRMS Architecture

Job Description

GRMS Job Description (GJD) is an XML based language that allows users to define the computing jobs and resource requirements. For each task there is a section in a job description document describing resource requirements and user preferences used for dynamic resource discovery and selection of the best resources. Another section defines the application: executable, input and output files required, arguments, environment, etc. One of the most interesting features of GRMS is its ability to deal with jobs defined as a set of tasks with precedence relationships (workflows). A user can submit the whole computational experiment that consists of many independent application executions. Two ways of expressing the dependencies between tasks are possible. The first one is a direct way, based on the parent-child relationship among tasks. In his case the execution of a child depends on the status change of its parents. The second way of expressing dependencies is associated with a data flow between the tasks. Here a user can specify that an output of one task becomes the input for the other one. Thus a user does not have to specify the exact file locations. However, in such a case it is user's responsibility to define file dependencies correctly. GRMS will reject execution of the job if data dependencies contradict the parent-child relationship. As it was already mentioned the basic way of introducing dependencies between tasks is by defining the parent-child dependencies. A very interesting and novel feature of GRMS which distinguishes it from the other systems is that execution of a child task can be triggered by any status change of a parent task. So, not only a task termination can trigger following executions. This feature is very useful in many scenarios. For instance we can imagine that a user would like to execute some application as soon as the other one starts running . e.g. for client server communication. The other example could be the flow of computation that depends on the failure of execution of one of the tasks (failover mechanisms). GJD also contains a section allowing users to specify their time constraints (e.g. deadlines, earliest start time etc.). Within the scope of the HPC-Europa project [67] a translation from JSDL1.0 to GJD have been developed and tested.

Scheduling

One of the main assumptions for GRMS is to perform remote jobs control and management in the way that satisfies users and their applications requirements as well as constraints and policies imposed by other stakeholders, i.e. resource owners and Grid or Virtual Organization administrators. Simultaneously, resource

administrators (resource owners) have full control over resources on which all jobs and operations will be performed by appropriate GRMS setup and installation.
GRMS uses a multi-criteria model for scheduling [68][71]. As it was mentioned above, during a scheduling process, first, a job is taken from a queue. After filtering resources according to resource requirements (hard constraints) the subset of resources is passed to the scheduling plugin. The multi-criteria resource selection plugin selects the best resource for a given job based on user preferences including importance of parameters specified in the job description (soft constraints). Of course, other algorithms can be used as well. For instance, plugins for scheduling parallel applications, rescheduling using migration mechanism [69], and use of performance prediction [70] have been developed.

In addition to the best effort approach presented above GRMS enables users to specify time constraints such as deadline, earliest job start time, required time window, days of week, and dates. Incoming jobs containing these requirements are, first, scheduled and then put into the queue. When the scheduled time is approaching a job is taken from the queue and executed. Currently GRMS manages time slots allocated to jobs centrally which means that it is assumed that GRMS is the only resource consumer in the environment. Reservation enforcement through agreements with local resource management systems (resource providers) belongs to the next steps in GRMS development.

3.7 The Intelligent Grid Scheduling System ISS

The main objective of the Intelligent Grid Scheduling System (ISS) [72] project is to provide a middleware infrastructure allowing optimal positioning and scheduling of real life applications over the GRID infrastructure provided by Swiss HPCN Grid programme. One of the main issues of ISS is to detect well-suited computational resources according to collected data on availability of machines, on the characteristics of applications with respect to the different machines in a Grid, and on specific needs of the user. The characteristics of an application are expressed in terms of a set of parameters that are measured during execution. All those data are kept in a dataWarehouse. These measured parameters are then reused for prediction at the next job submission. An original model has been developed, the Gamma model [74], allowing using these parameters to predict the behavior of a given application on different resources.

The decision on the choice of the resources on which the application is executed is based on the evaluation of cost function. In the cost function all terms are expressed as costs. The user can choose some free parameters and constraints such as a time limit, or a maximum cost. With all those data, the cost function model delivers a deterministic solution; we call it best-suited machine.

In fact, there are applications that can run on a Network of Workstations, some need NUMA or SPMD machines, others can be executed on a cluster of PCs, another category need MPP machines having high performance communication network and finally some category could need several of the above mentioned architecture. All these types of resources have their prices; the cost differences can achieve a factor of 10. A simulator is in construction that enables simulating a machine park with the lowest costs. Such a result can then be used to decide on future resource investment.

Thanks to the CoreGRID NoE, the resulting ISS middleware will first embedded in the VIOLA/UNICORE/Meta-Scheduler environment. For this purpose a CoreGRID fellow is being hired to do the implementation. The first testbed will be the productive clusters at EPFL, including a workstation park and clusters of 120, 160, 320 and 8000 processors with very different communication networks. The final goal is to validate this new environment with HPC resources in Switzerland.

Figure 13 presents the general architecture of the system. It is constituted of the following main elements: the Meta-Scheduling Service (MSS), the Resource Broker (RB), the Data Warehouse (DW), the System Information (SI) and finally the Monitoring Module (MM).
Figure 13 also shows the processes which are executed after a request for execution is submitted to the proposed Grid system. The ISS scenario consists of a pre-execution, the execution, and a post-execution phase. In the pre-execution, the resources are discovered, pre-selected according to application constraints (amount of memory, access rights, libraries, executable, etc), data is collected (availabilities of machines, application relevant parameters computed with the Gamma model [74], etc) and used to evaluate a cost function model [75] by means of which the best suited machine is selected. At the end of the execution, application-relevant data coming from the MM (Ganglia, the accounting system, etc.) are collected by the SI (currently implemented using VAMOS middleware service) and stored by the SI in the dataWarehouse.

In the cost function all terms are expressed in EUR. It includes terms related to the hardware and software costs (CPU, licenses, data transfers, etc), and costs due to the waiting time. There are a few free parameters that can either be given by the user, or are determined by the simulator. Specifically, if one is concerned about the best usage of machines in a Grid, all the free parameters have to be computed through a simulation process.

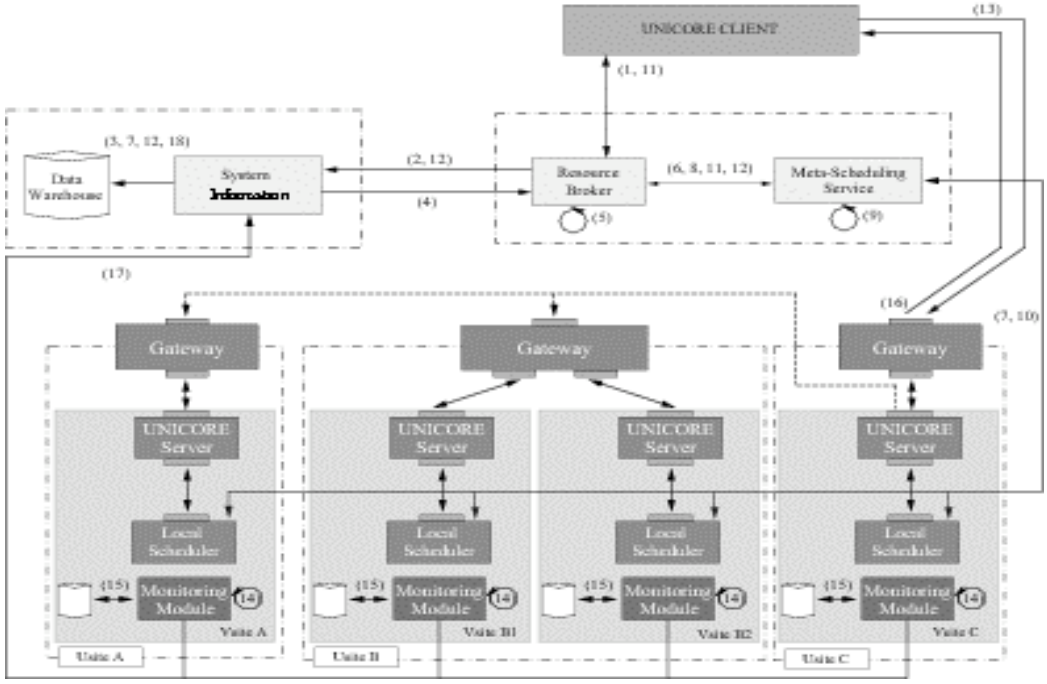


Figure 13: The overall architecture of the VIOLA/UNICORE/Meta-Scheduler/ISS environment. The numbers follow a logical time sequence in the execution of the scheduling process.

4 Survey Results

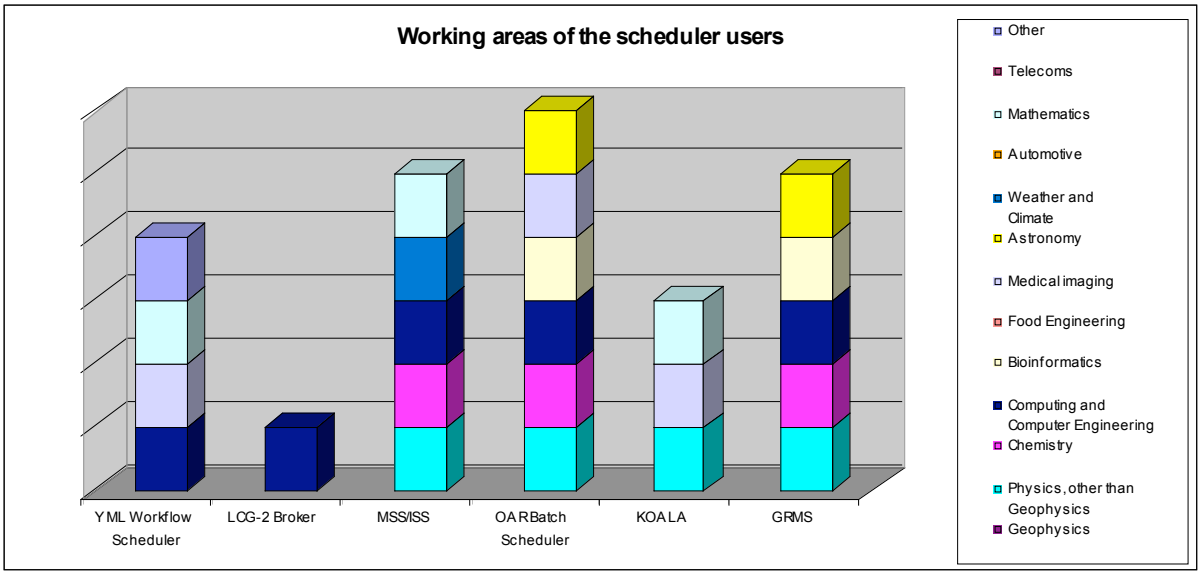
It should be noted that the following results, only reflect the scheduling models used by CoreGRID partners in WP6 who work on scheduling models (i.e. partners may use other scheduling models and systems on other systems in other projects or production). This cannot be extrapolated for general Grid use on a broader scale. Here, it should be considered that CoreGRID partners collaborate and that the use of certain approaches and tools may partially be influenced by this collaboration.
A survey has been made to show the current models in use by the partners to further explore the cooperation in research activity as well as in practical use of common tools among the partners in the future. The details of the questionnaire can be found in Appendix A.
As it can be seen from the questionnaire, we do not discuss all questions and only highlight some of the results. Some of the results have only been used to further understand the background of the different solutions.

4.1 Scheduling solutions

Institution	CoreGRID partner
YML Workflow Scheduler	Polytech-Mons, Belgium CETIC, Belgium
LCG-2 Broker	MTA-SZTAKI, Hungary
MSS/ISS	Fraunhofer Institute, Germany Ecole Polytechnique Federale de Lausanne (EPFL), Switzerland Research Center Juelich (FZJ), Germany
OAR Batch Scheduler	CNRS / ID-IMAG, Grenoble, France
KOALA	Delft University of Technology, Netherlands
GRMS	Poznan Supercomputing and Networking Center (PSNC), Poland

The survey results present an overview of the grid resource management systems that are employed by the CoreGRID partners. As one can see there are several solutions that are used by multiple partners, even on international level. Additionally, there are currently a lot of research activities within the institute, in order to archive a certain level of interoperability between the different grid scheduling systems.

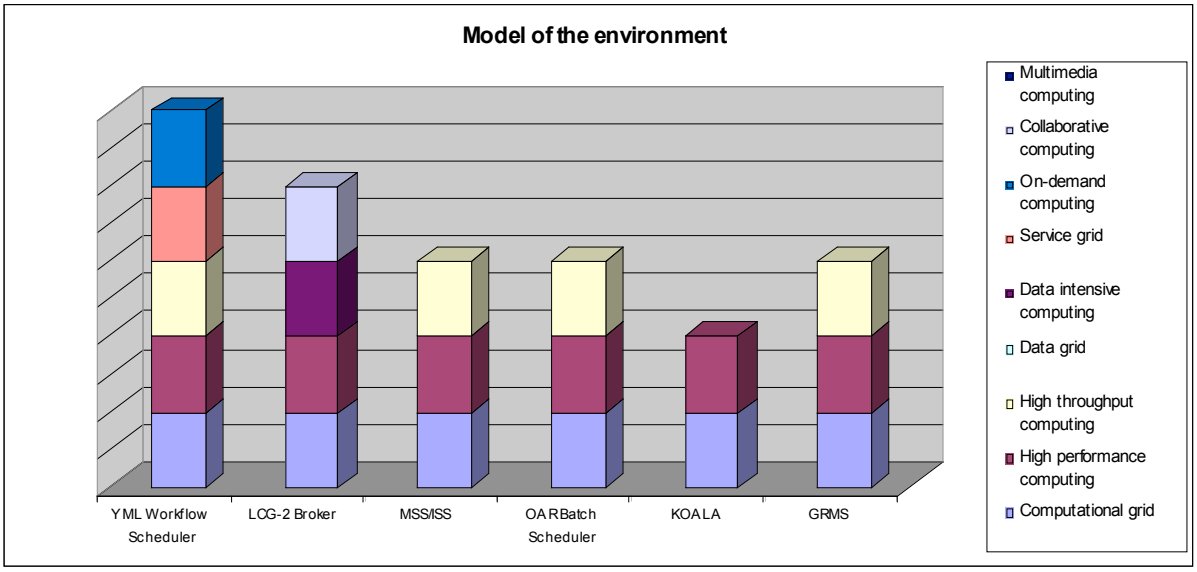
4.2 Working area of the Scheduler users

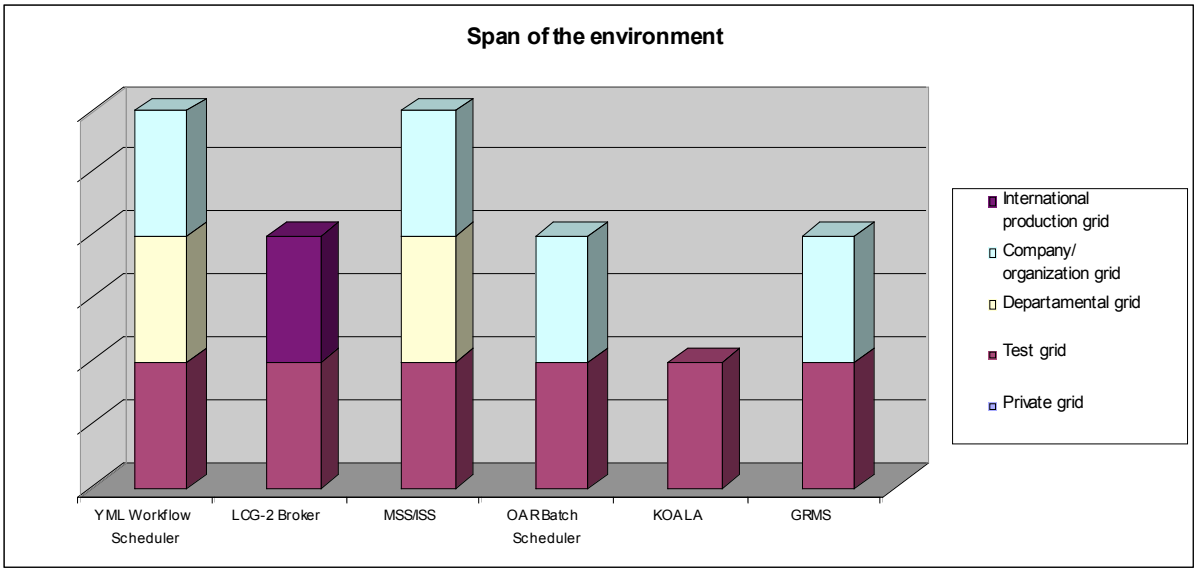


As one can see from the application environment, there is a distribution to many different areas with special consideration of physics and computing/computer engineering in general. Moreover, it can be seen that most sites do not focus on a particular application field but are used for several research disciplines. This is understandable as most CoreGRID partners come from research institutes whose resources are used for multi-disciplinary research.

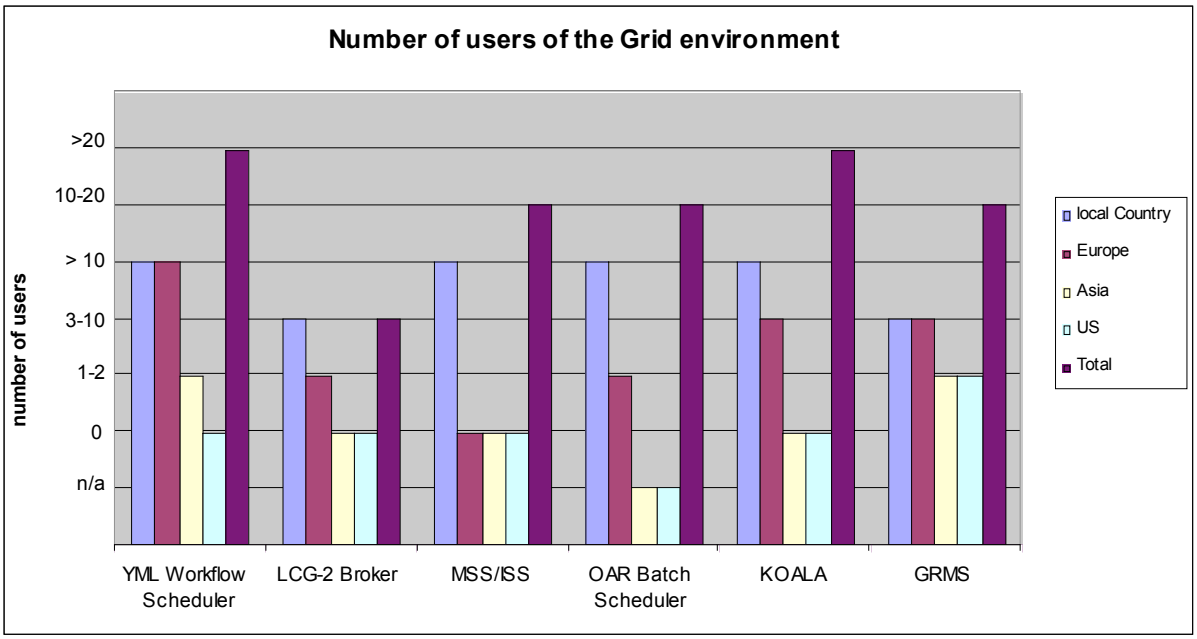
4.3 Grid environment

The following charts give an overview of the Grid environment the different organizations participating in CoreGRID are dealing with.





As it can be seen from these result figures, many Grids are still used as test Grids. This is probably caused by the fact that WP6 the partners are active in Grid research and not solely focusing on production use. Most of the Grids are not used on a global international scale. That means, that the foreseen usage profile of future Grids is not yet applied by the partners. A reason for this observation is probably the lack of large scale, interoperable Grid solutions, as well as missing standards for Global Grids and a suitable business/usage model for resource exchange.

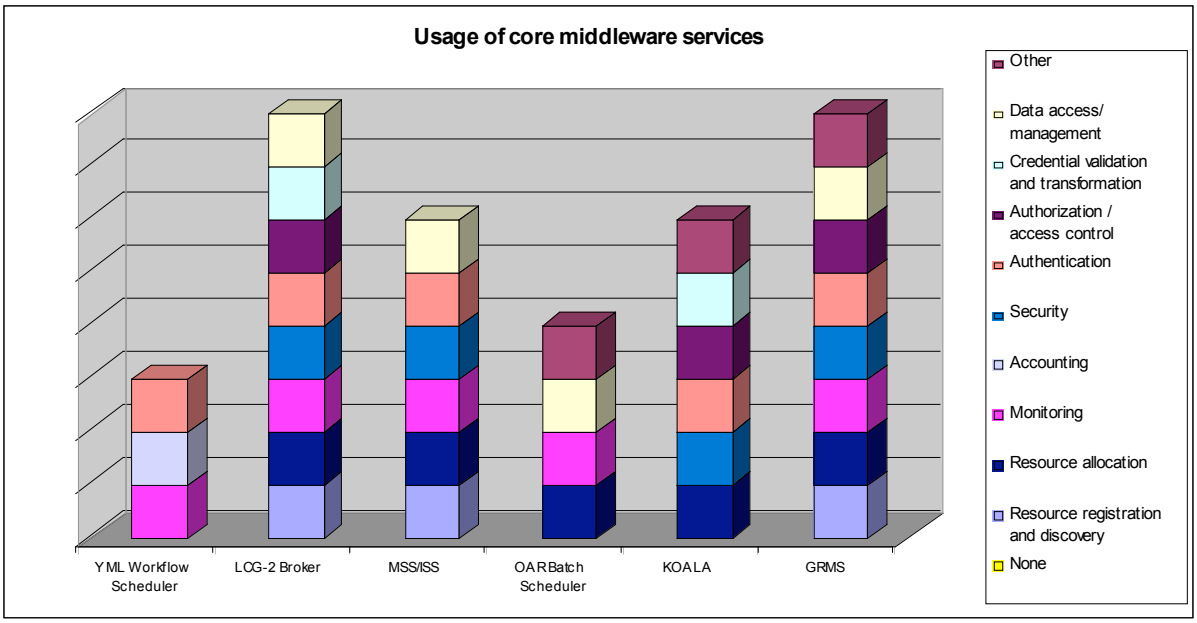


It is interesting to note, that most of the systems work on a very limited user base. Most of the users come from the local or national level.

4.4 Core Middleware Services

Institution	None	Resource registration and discovery	Resource allocation	Monitoring	Accounting
YML Workflow Scheduler				ARCo	ARCo
LCG-2 Broker		BDII, MDS	GRAM, Condor, DAGMan, PBS, LSF	Mercury	
MSS/ISS		UNICORE	UNICORE	VAMOS (new Application oriented monitoring tool) UNICORE	
OAR Batch Scheduler			OAR batch scheduler provides mechanisms for advance reservation	on-line Gantt charts for resource utilization	
KOALA			GRAM		
GRMS		Globus MDS	Globus GRAM, DRMAA Service Provider (DSP)	Mercury	

The overview indicates that different middleware systems are in use. There is no clear preference for a particular platform. Besides Globus (with GRAM) and UNICORE, the solutions as presented in the model overview in Section 3 occur.



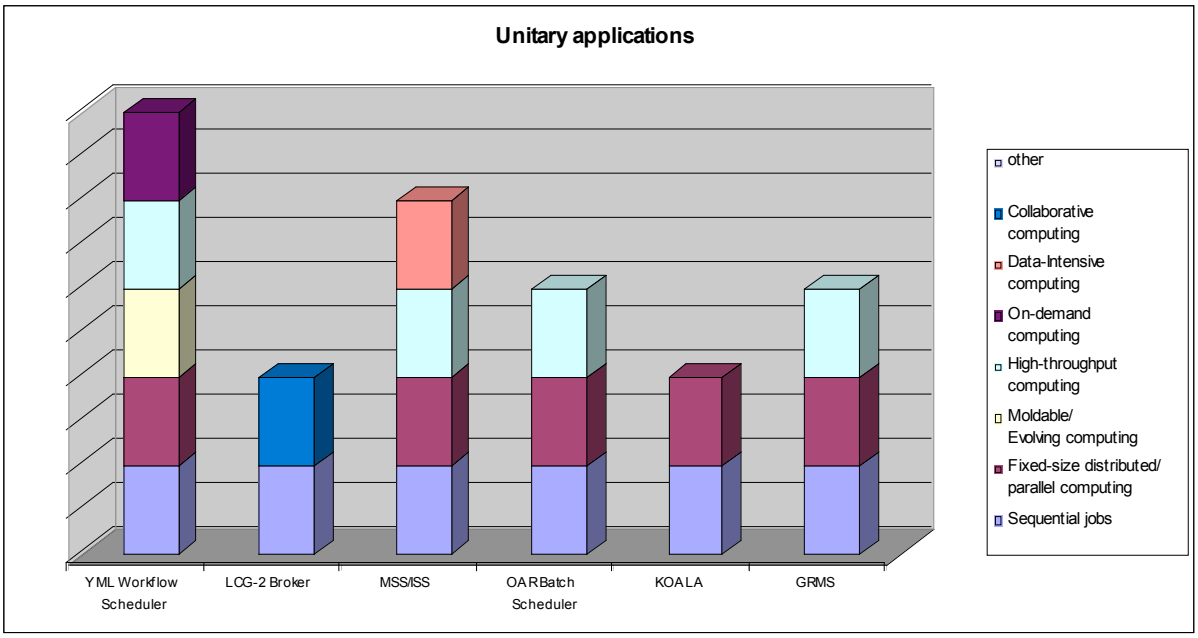
Most installations feature a complete set of middleware services for most functionalities. However, as shown before the actual middleware platform may differ between these installations.

4.5 Environment size and structure

Institution	Clusters / Units	User interface and Gateway	Computing nodes	Storage devices	Network
YML Workflow Scheduler	1 x cluster	1 x Sparc 64bit	33 nodes, 32 Bi-opteron + 1 Quadri-opteron	1 SAN + 1 tape	2x Gigabit Ethernet per each of the 33 nodes (control + data) + Infiniband between 8 nodes
LCG-2 Broker	12 x clusters with homogeneous nodes		12 clusters (746 computers)	n/a	n/a
ISS	3 x clusters, homoge-neous nodes, heteroge-neous soft, different gateways	Intel Xeon 2.8 GHz, Pentium 4 HT 2.8 GHz, AMD Opteron 2.8 GHz, Itanium2	120 Intel Xeon 2.8, 132 Intel P4HT, 32 Intel P4HT, 224 bi-AMD Opteron 2.8 GHz, 16 Itanium SGI NUMA Machine (Altix350)	Some TeraBytes	School internal network (10Mb/s), FastEthernet (100 Mb/s), GigaBit Ethernet (1000 Mb/s), Myrinet
MSS	5 x clusters, heterogeneous nodes, heteroge-neous software	multiple	84 nodes (max. usage), different hardware		10GB/Gigabit Ethernet inter-cluster, Myrinet and 10GB/Gigabit Ethernet intra-cluster
OAR Batch Scheduler	14 x clusters, homogeneous nodes in each cluster, homoge-neous software		currently about 2000 nodes, the goal is 5000 nodes, full list under: https://www.grid5000.fr/mediawiki/index.php/Grid5000:Hardware	each node 1 x HDD, mostly 80GB	intra-cluster: Gigabit Ethernet and Myrinet (on 5 clusters), inter-cluster 2,5GBit migrating to 10Gbit
KOALA	5 clusters, homoge-neous nodes, homoge-neous software	5 head nodes (1/cluster), 2 x Intel Pentium III per headnode	200 nodes, each node 2 x Intel Pentium III	Storage is done at the headnodes, 4 x HDD 92GB + 1 x HDD 136GB	Myrinet LANs, 1 Gb WAN

GRMS	project dependent (e.g. 12 clusters with homoge-neous nodes or 10 x heteroge-neous super-computers)	project dependent	project dependent, e.g. 100 homogeneous nodes, each node 2 x Itanium CPUs or 20 heterogeneous nodes	project dependent	project dependent, e.g. Myrinet inside clusters, 10GB WAN inter-cluster

4.6 Type of applications



Composite applications

Institution	***None***	***Chains-of-tools (composed of the unitary application types specified in the previous section)***	***Chains-of-tools (composed of other types of applications)***	***Parameter sweeps (composed of the unitary application types specified in the previous section)***	***Parameter sweeps (composed of other types of applications)***
YML Workflow Scheduler					
LCG-2 Broker					
MSS/ISS					
OAR Batch Scheduler				X	
KOALA					
GRMS					

Institution	Simple Workflows (composed of the unitary application types specified in the previous section)	Simple Workflows (composed of other types of applications)	Complex Workflows (composed of the unitary application types specified in the previous section)	Complex Workflows (composed of other types of applications)
YML Workflow Scheduler			X	
LCG-2 Broker		DAGMan		
MSS/ISS		X	X	
OAR Batch Scheduler	planned	planned		
KOALA				
GRMS	X			

Application specific libraries

Institution	Grid-related	Communication
YML Workflow Scheduler	Globus CoG UNICORE POP-C++, YML	MPI - version 1/2 - provider MPICH, LAM Java RMI PVM
LCG-2 Broker		MPI - provider MPICH PVM
MSS/ISS	UNICORE	MPI - version 1 - provider MPICH PVM OpenMP
OAR Batch Scheduler		MPI Java RMI OpenMP
KOALA	Globus CoG	MPI - version 1 - provider MPICH Java RMI Ibis
GRMS	GAT	MPI - version 1 - provider MPICH

4.7 Resource requirements of the applications

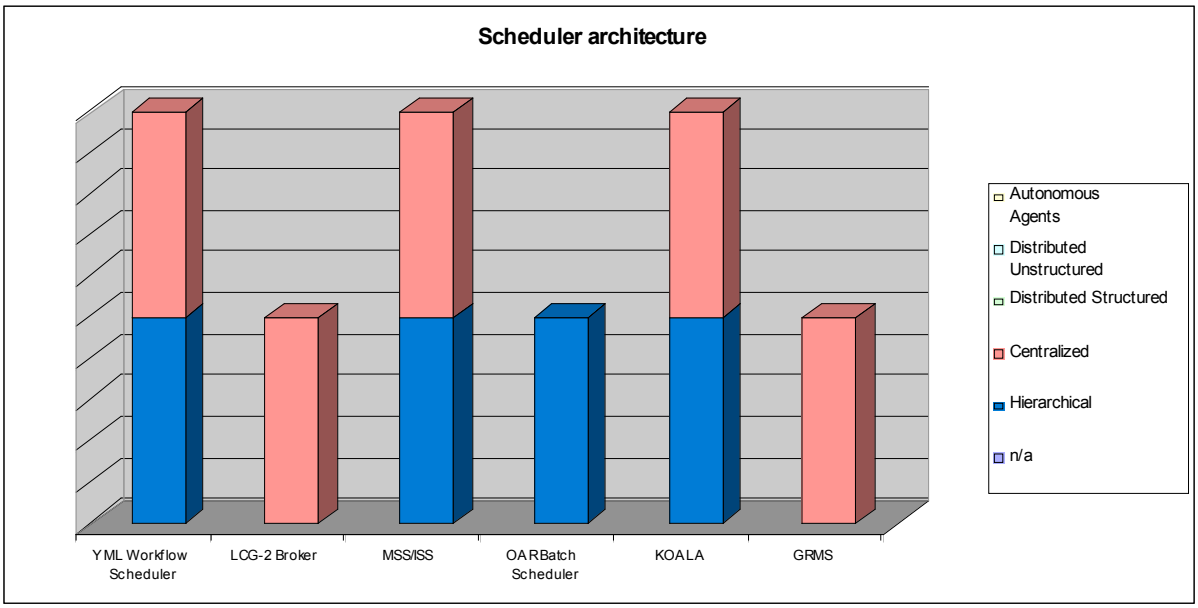
Institution	size of input/output data	Scratch disk on the execution machine	Format of the input/output files
YML Workflow Scheduler	100KB-1MB >1GB	1-100MB >1GB	ASCI Binary XML-based Proprietary standard
LCG-2 Broker	100KB-1MB 1-100MB	>1GB	ASCI Binary
MSS/ISS	>1GB	100MB-1GB >1GB	ASCI International standard Binary Proprietary standard Machine-independent
OAR Batch Scheduler	n/a	n/a	Don't know
KOALA	100MB-1GB	>1GB	ASCI Binary
GRMS	Don't know	Don't know	ASCI Binary XML-based

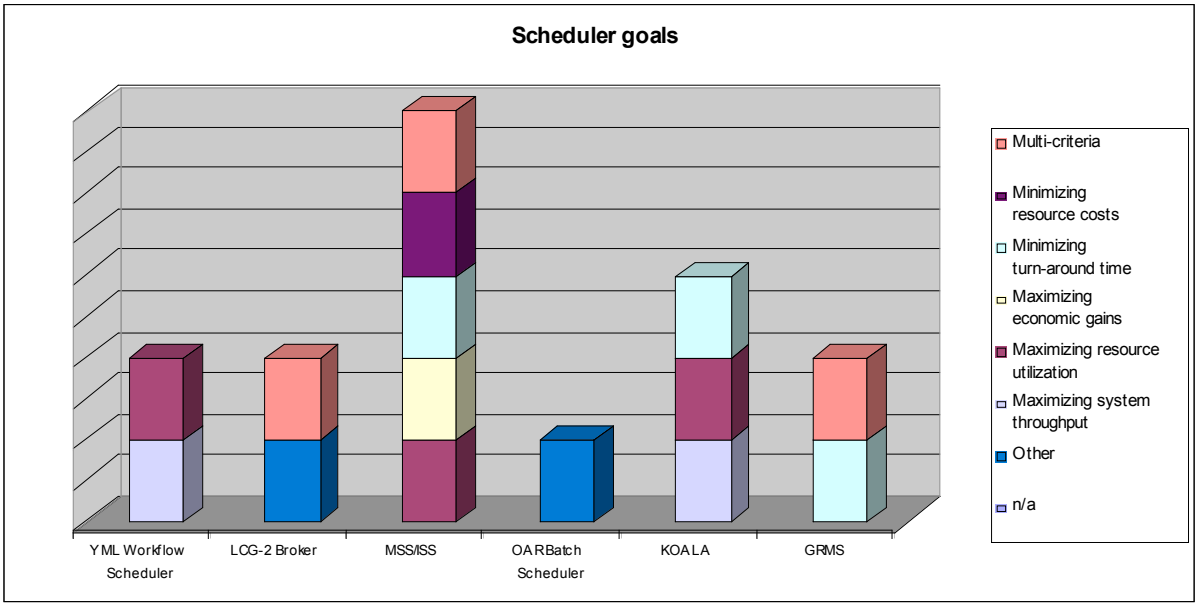
Institution	memory requirements Fixed	memory requirements Virtual	time of application runs
YML Workflow Scheduler	>1GB	1-10MB	<15min 1-6h 2-5days
LCG-2 Broker	n/a	n/a	seconds <15 min 15 min - 1 h
MSS/ISS	100MB-1GB >1GB	100MB-1GB	<15 min 15min-6h 6-12h 12-24h 2-5 days 5-30 days
OAR Batch Scheduler	>1GB	n/a	15min-6h
KOALA	n/a	n/a	n/a
GRMS	>1GB	>1GB	<15min
YML Workflow Scheduler	Don't know	Don't know	Don't know

4.8 Scheduler structure

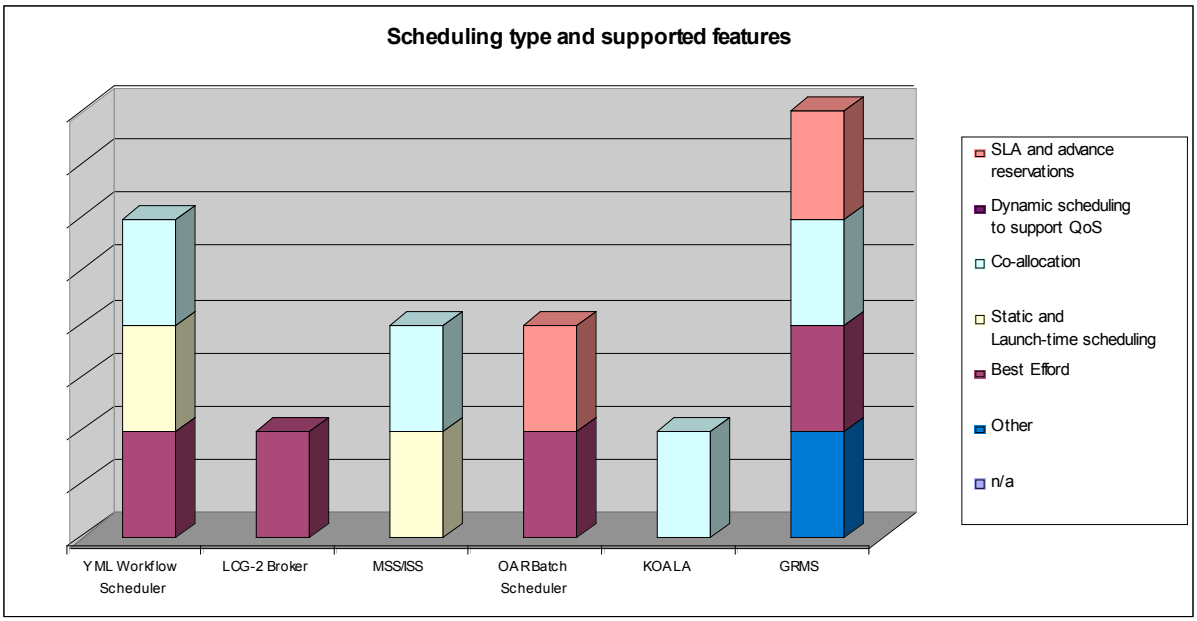
The following results show that most of the systems work in a centralized way. This is expectable as the grid installations only cover single institutions and not a large scale, global infrastructure. This centralized approach

probably limits the scalability in a global environment. However, it needs to be seen whether an additional scheduling level on top of these installations could support a global Grid structure

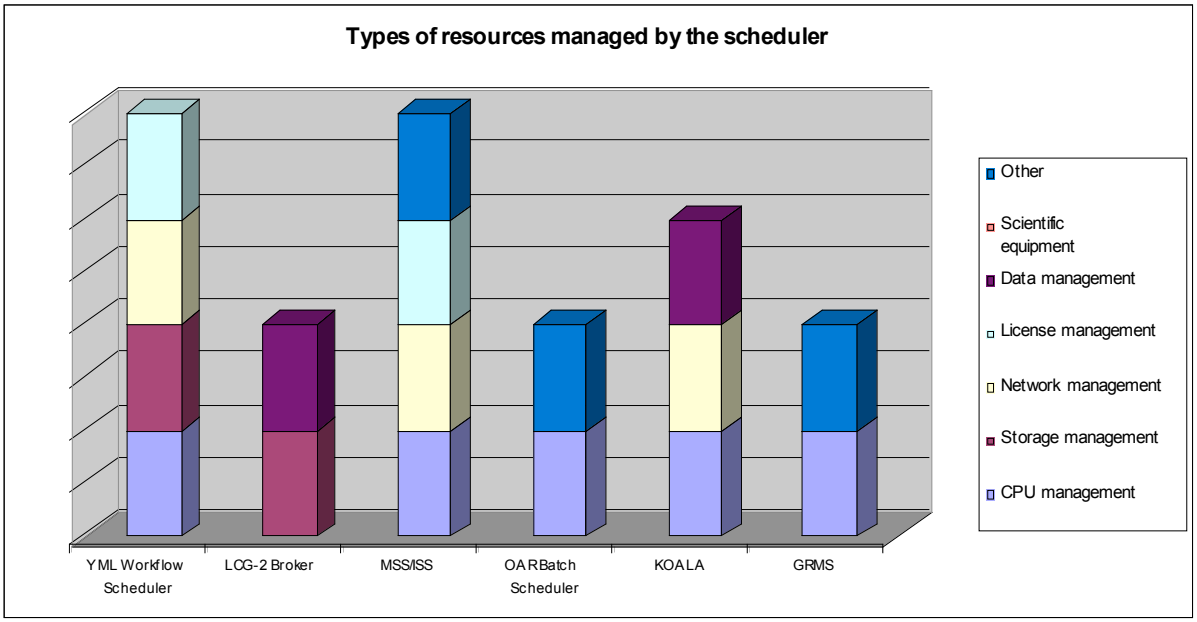




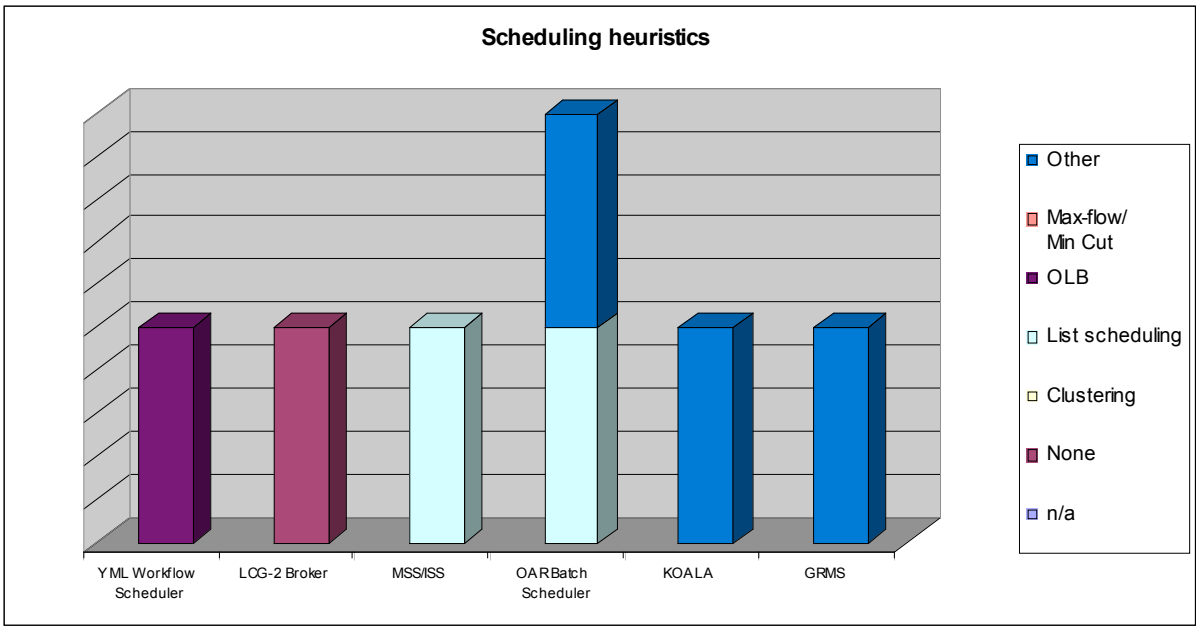
In general, the results allow the conclusion that most systems work comparable to large scale parallel computers or clusters. Most systems optimize turn-around time and resource utilizations. The consideration of cost is not yet well addressed. As the partners come from scientific environments, this is a quite common model as usage cost is often not considered in such systems.



Interestingly, many partners already consider co-allocation, QoS and SLA. The need for supporting these functionality is commonly accepted. However, the perception is usually that this is not yet supported or used on a larger scale with the exception of some commercial Grid service providers. The survey result show that CoreGRID partners are indeed working on or already use such functionality.

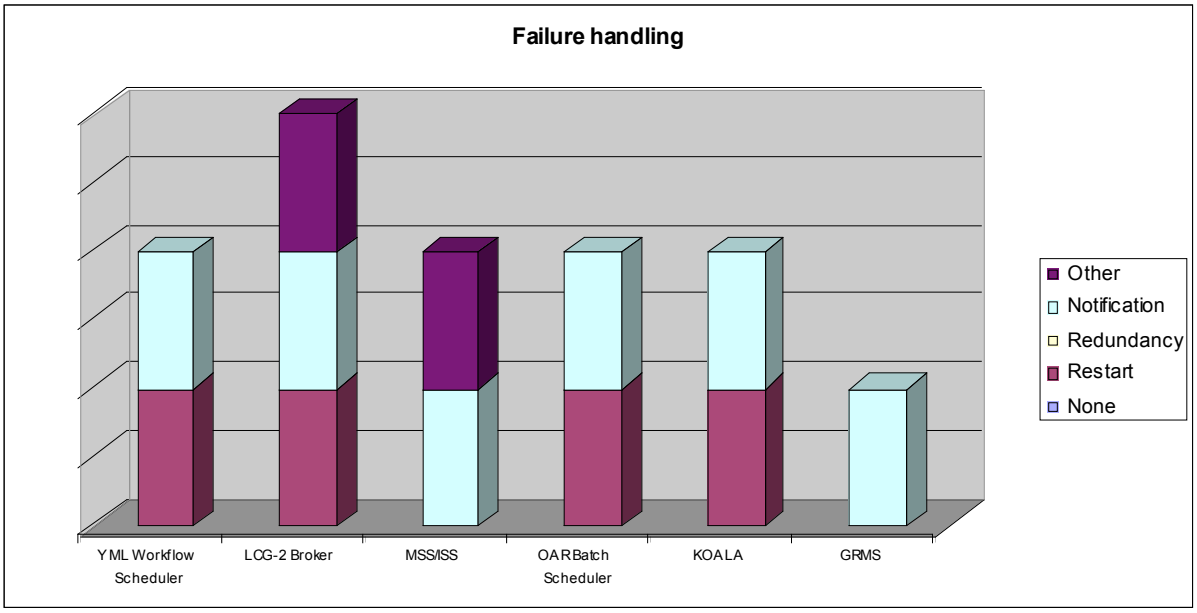


In extension to the co-allocation feature, it can be seen that most CoreGRID partners do consider several resource types. That is, not only CPU-power is considered but also data, storage, network or software.

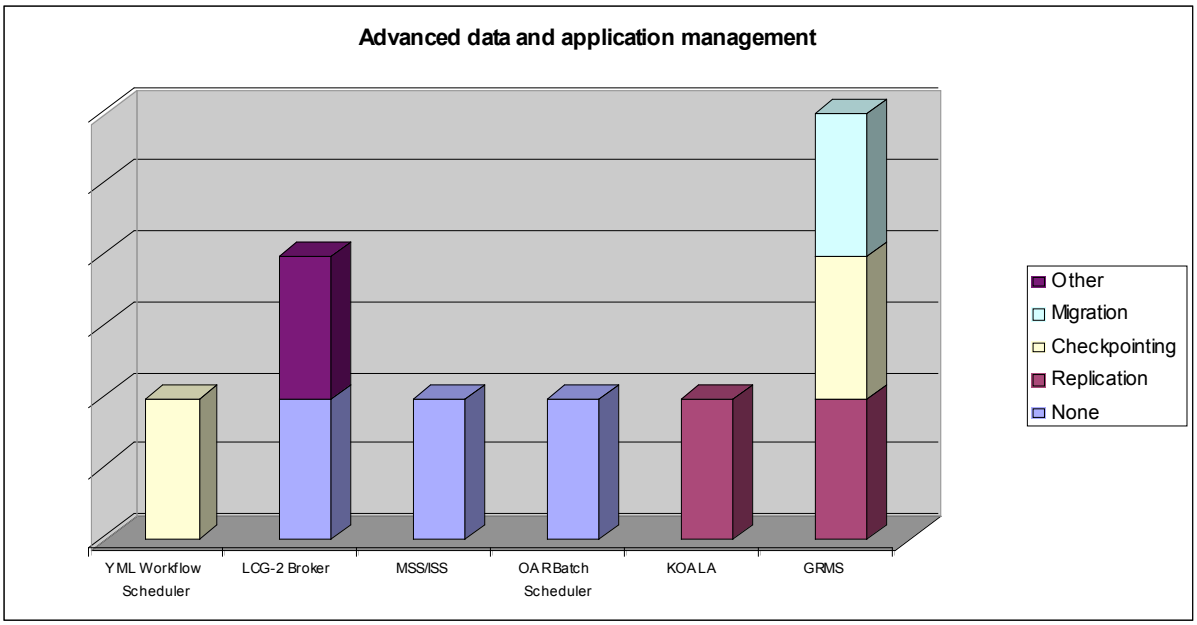


The questions in regard to the scheduling heuristics showed that the use of advanced scheduling heuristics is quite limited:

- Most partners use a list scheduling
- or rely on the underlying queuing system which is usually also a list scheduling heuristic.

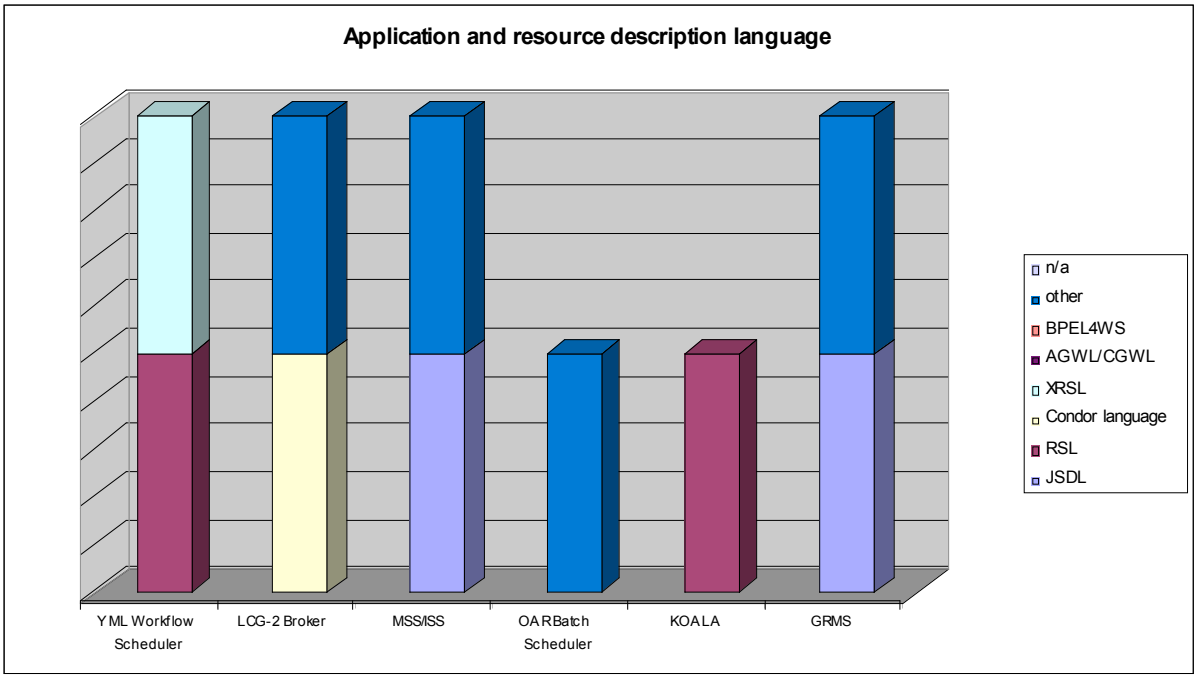


Failure handling is usually limited to notification and/or a restart of a job. No installation applied redundancy strategies.



However, some installation would have been able to support checkpoint restart which could have been used for improved failure handling. It needs further investigation whether check-point restart is used to dynamically migrate jobs or change the resource footprint of a job during runtime.

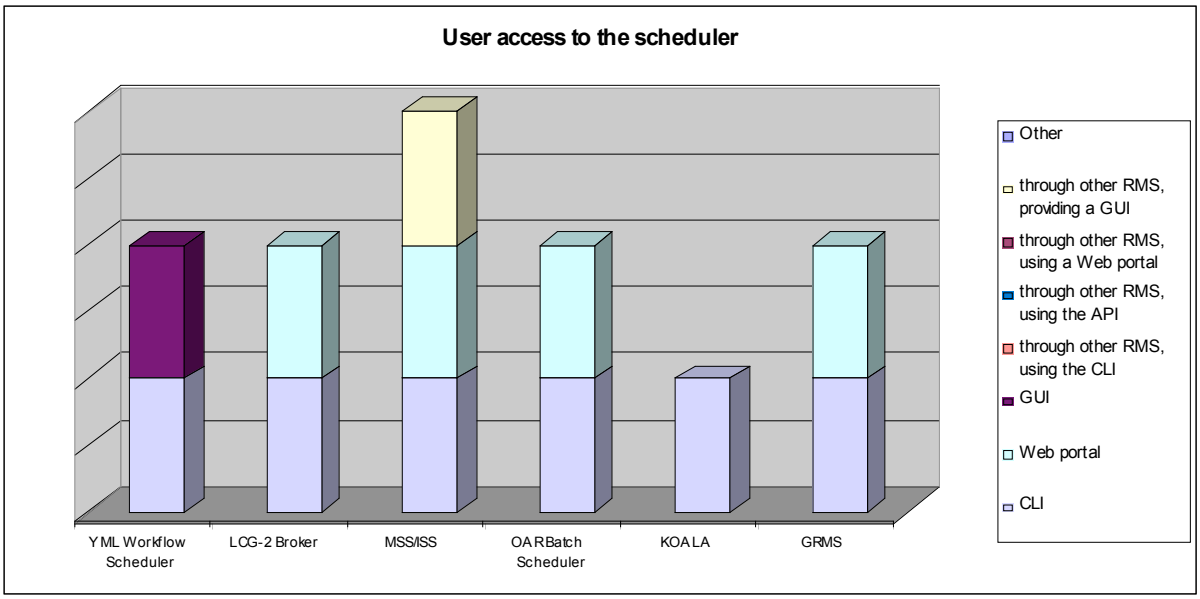
4.9 Scheduler implementation, deployment, and user support



Many partner used a customized description model. Notably, a small subset applies JSDL which is often considered as the upcoming standard.

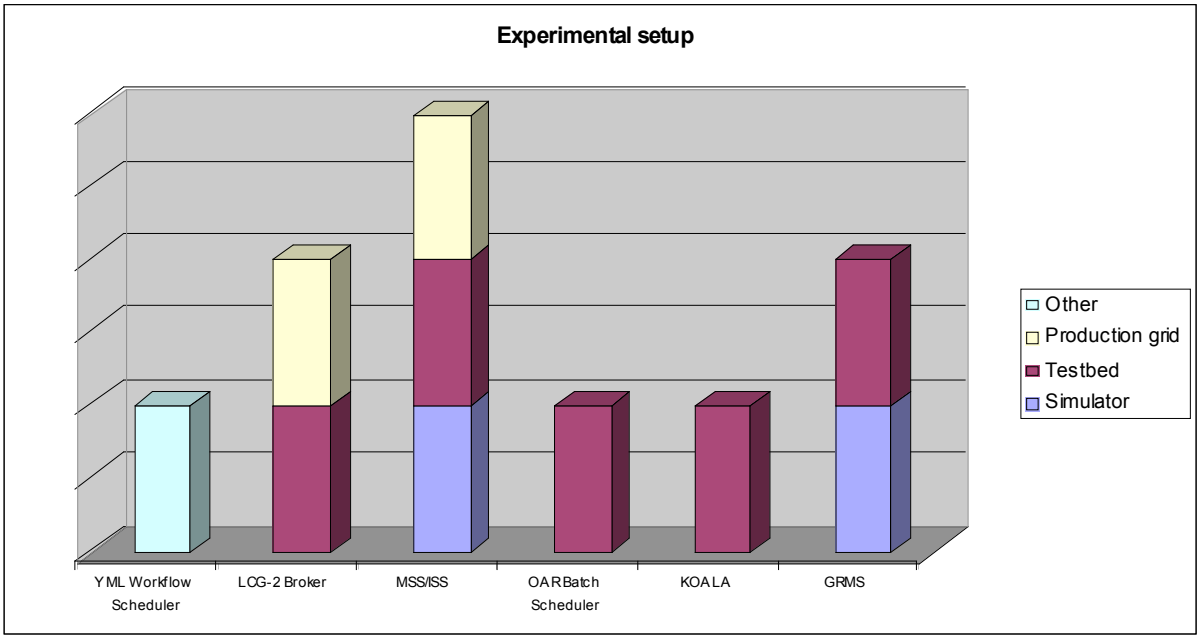
Dependencies on and collaboration with other services

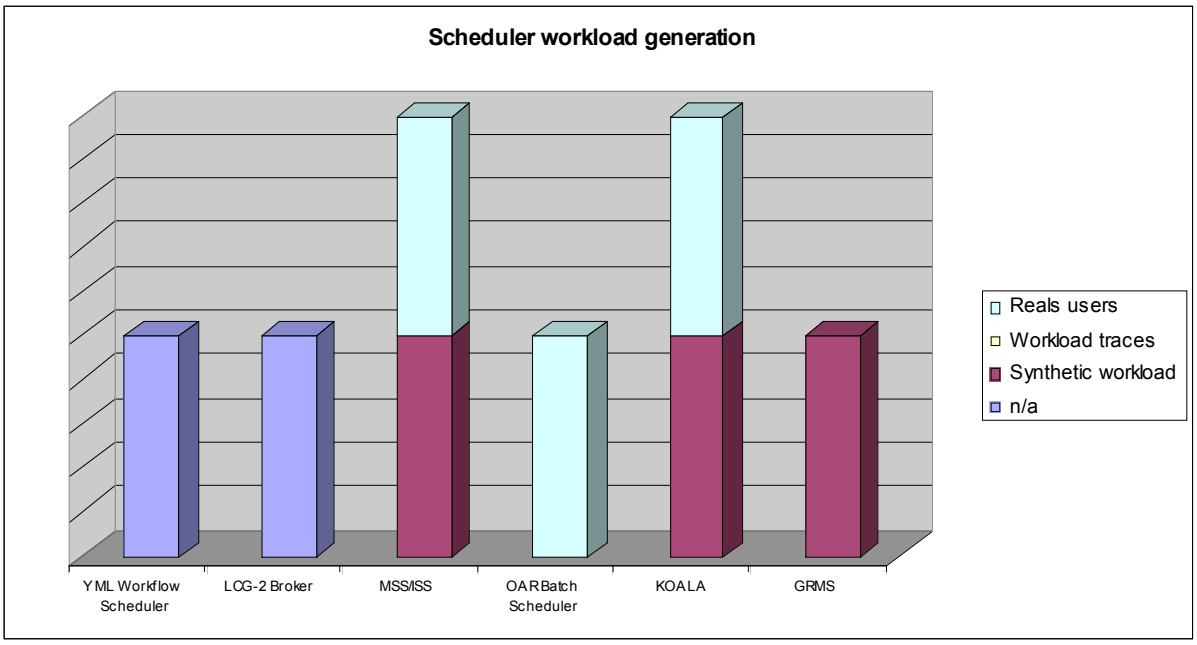
Institution	Information Service	Monitoring Service	Security Service	Brokerage Service	Data Transfer Service	Accounting and Billing Service	File replica catalog service
YML Workflow Scheduler		X					
LCG-2 Broker	Service of LCG-2 (BDII)				Service of LCG-2		X
MSS/ISS	General Information service	VAMOS	SSL Certificates	Grid Broker	UNICORE related one		
OAR Batch Scheduler		Ganglia and Monica (custom)					
KOALA			GSI		GridFTP		
GRMS	Globus MDS, iGrid, prediction system (GPRES)	Mercury	GAS		GridFTP, SRB, DMS (developed in PSNC)		

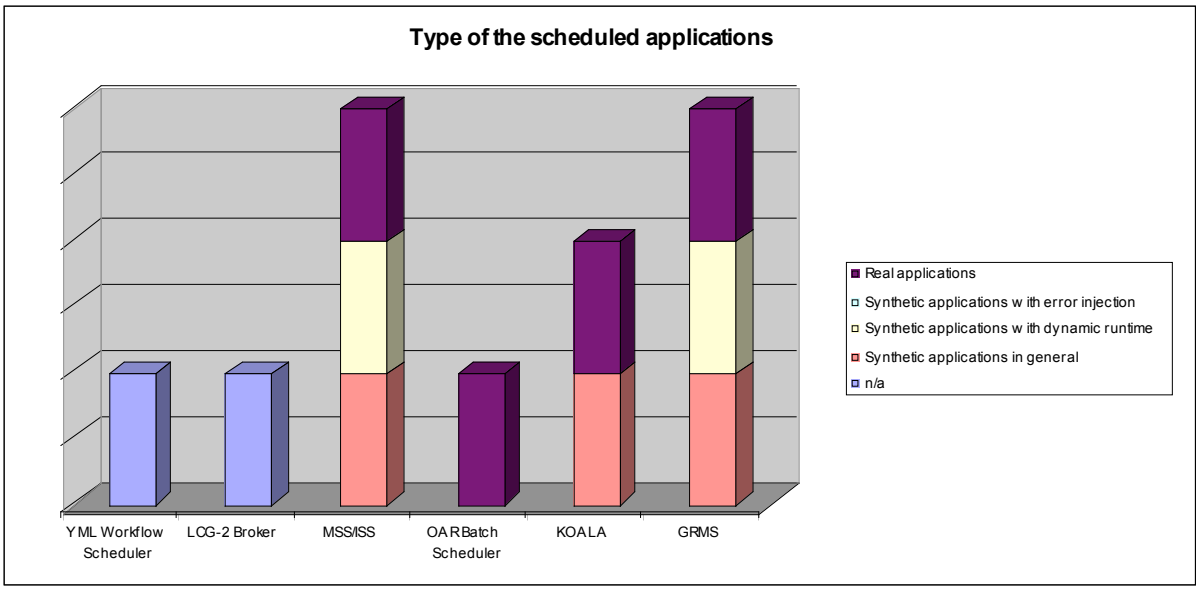

User access to the scheduler
YML Workflow Scheduler
LCG-2 Broker
MSS/ISS
OAR Batch Scheduler
KOALA
GRMS
Other
through other RMS, providing a GUI
through other RMS, using a Web portal
through other RMS, using the API
through other RMS, using the CLI
GUI
Web portal
CLI

4.10 Scheduler evaluation techniques

The questions on the evaluation approach for schedulers showed that many partners actually apply their solution on their test or production systems. 50% of the partners use – at least partially – a simulator to evaluate, analyze and fine-tune their scheduling solution. Most partner use the actual workload on their system to evaluate the system. A significant part also artificial workloads, either from models or older traces.







5 Summary

In this document we provided an overview on multi-level scheduling models as they are investigated by the CoreGRID partners in the Virtual Institute on Resource Management and Scheduling. Besides a general discussion on the scheduling problem, existing solutions have been presented. Moreover, a survey has been conducted to learn about the environment and the model of the existing solutions. The results show that quite different approaches are currently applied. However, several research groups have been established in CoreGRID to work on common aspects to improve the available scheduling landscape. The presented document gives a suitable overview to better exploit the partner expertise and stipulate more collaboration among them.

6 References

[1] YML Website, http://www.prism.uvsq.fr/cni/yml.

[2] O. Delannoy and S. Petiton, A Peer to Peer Computing Framework: Design and Performance Evaluation of YML, ISPDC/HeteroPar proceedings, IEEE Computer Society Press, 2004.

[3] S. Noél, O. Delannoy, N. Emad, P. Manneback, S. Petiton, A multilevel scheduler for the Grid computing YML framework, submitted to CoreGRID Workshop on Grid Middleware, Dresden, August 2006.

[4] W. Allcock, J. Bresnahan, I. Foster, L. Liming, J. Link, and P. Plaszczac. GridFTP Update. Technical report, January 2002.

[5] S. Ananad, S. Yoginath, G. von Laszewski, and B. Alunkal. Flow-based Multistage Co-allocation Service. In In Proc.of the International Conference on Communications in Computing, pages 24–30, Las Vegas, 2003. CSREA Press.

[6] S. Banen, A.I.D. Bucur, and D.H.J. Epema. A Measurement-Based Simulation Study of Processor Co-Allocation in Multicluster Systems. In D.G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, 9th Workshop on Job Scheduling Strategies for Parallel Processing, volume 2862 of LNCS, pages 105–128. Springer-Verlag, 2003.

[7] A.I.D. Bucur and D.H.J. Epema. The Performance of Processor Co-Allocation in Multicluster Systems. In In Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the GRID (CCGrid2003), pages 302–309. IEEE Computer Society Press, 2003.

[8] A.I.D. Bucur and D.H.J. Epema. Trace-Based Simulations of Processor Co-Allocation Policies in Multiclusters. In In Proc. of the 12th IEEE International Symposium on High Performance Distributed Computing(HPDC12), pages 70–79. IEEE Computer Society Press, 2003.

[9] Karl Czajkowski, Ian T. Foster, and Carl Kesselman. Resource Co-Allocation in Computational Grids. In In Proc. of the 8th IEEE International Symposium on High Performance Distributed Computing(HPDC-8), pages 219–228. IEEE Computer Society Press, 1999.

[10] H.H. Mohamed and D.H.J. Epema. An Evaluation of the Close-to-Files Processor and Data Co-Allocation Policy in Multiclusters. In In Proc. of IEEE International Conference Cluster Computing 2004. IEEE Computer Society Press, September 2004.

[11] H.H Mohamed and D.J.H. Epema. Experiences with the KOALA Co-Allocating Scheduler in Multiclusters. In In Proc. of the 5th IEEE/ACM International Symposium on Cluster Computing and the GRID (CCGrid2005), pages 784– 791. IEEE Computer Society Press, 2005.

[12] Rob V. van Nieuwpoort, Jason Maassen, Rutger Hofman, Thilo Kielmann, and Henri E. Bal. Ibis: an Efficient Java-based Grid Programming Environment. In ACM Java Grande ISCOPE 2002 Conference, pages 18–27, Nov 2002.

[13] R.V. van Nieuwpoort, J. Maassen, H.E. Bal, T. Kielmann, and R. Veldema. Wide-Area Parallel Programming Using the Remote Method Invocation Model. Concurrency: Practice and Experience, 12(8):643–666, 2000.

[14] Web-site. Mpich-g2. http://www3.niu.edu/mpi/.

[15] Web-site. The Distributed ASCI Supercomputer (DAS). http://www.cs.vu.nl/das2.

[16] Web-site. The Globus Toolkit. http://www.globus.org/.

[17] Jon B. Weissman and Andrew S. Grimshaw. A Federated Model for Scheduling in Wide-Area Systems. In 5th IEEE International Symposium on High Performance Distributed Computing (HPDC-5), pages 542–550. IEEE Computer Society Press, 1996.

[18] Jon B. Weissman and Xin Zhao. Scheduling Parallel Applications in Distributed Networks. Cluster Computing, 1(1):109–118, 1998.

[19] D. Erwin, ed. UNICORE Plus Final Report – Uniform Interface to Computing Resources. UNICORE Forum e.V., ISBN 3-00-011592-7, 2003.

[20] J. Schopf. Ten Actions when Grid Scheduling. In Grid Resource Management (J. Nabrzyski, J. Schopf and J. Weglarz, eds.), pages 15-23, Kluwer Academic Publishers, 2004.

[21] OpenPBS Batch Processing and Resource Management System. Web site, 22 November 2005, <http://www.openpbs.org/>.

[22] I. Foster, A. Roy, and V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. In 8th International Workshop on Quality of Service (IWQOS 2000), pages 181-188, June, 2000.

[23] A. Andrieux et. al., Web Services Agreement Specification, July, 2006. On-line: <https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.graap-wg/docman.root.current drafts/doc13652>.

[24] VIOLA – Vertically Integrated Optical Testbed for Large Application in DFN. Web site, 22 November 2005, <http://www.viola-testbed.de/>.
[25] G. Quecke and W. Ziegler. MeSch – An Approach to Resource Management in a Distributed Environment. In Proc. of 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000), Volume 1971 of Lecture Notes in Computer Science, pages 47-54, Springer, 2000.
[26] R. Menday and Ph. Wieder. GRIP: The Evolution of UNICORE towards a Service-Oriented Grid. In Proc. of the 3rd Cracow Grid Workshop (CGW'03), Cracow, PL, Oct. 27-29, 2003.
[27] The Globus Toolkit 4.0. Project documentation web site, 22 November 2005, <http://www.globus.org/toolkit/docs/4.0/>.
[28] Ph. Wieder and W. Ziegler. Bringing Knowledge to Middleware -The Grid Scheduling Ontology. In First Core-GRID Workshop, Springer, 2005, to appear.
[29] V. Keller, K. Cristiano, R. Gruber, P. Kuonen, S. Maffioletti, N. Nellari, M.-C. Sawley, T.-M. Tran, Ph. Wieder, and W. Ziegler. Integration of ISS into the VIOLA Meta-scheduling Environment. In Proc. of the Integrated Research in Grid Computing Workshop, Pisa, IT, November 28-20, 2005, to appear.
[30] O. Waeldrich, Ph. Wieder, and W. Ziegler, A Meta-Scheduling Service for Co-allocating Arbitrary Types of Resources. In Proc. of the Second Grid Resource Management Workshop (GRMWS'05) in conjunction with Parallel Processing and Applied Mathematics: 6th International Conference, PPAM 2005, Lecture Notes in Computer Science, Volume 3911, Springer, R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski (eds.), pp. 782-791, Poznan, Poland, September 11-14, 2005. ISBN: 3-540-34141-2.
[31] Ian Foster and Carl Kesselman. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco, CA, 1999.
[32] Fran Berman and Geoffrey C. Fox and Anthony J. G. Hey. Grid Computing Making the Global Infrastructure a Reality, Wiley Series in Communications Networking & Distributed Systems, 2003
[33] Ian Foster. The anatomy of the grid: Enabling scalable virtual organizations. In Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing, London, UK, 2001. Springer.
[34] R. Buyya, H. Stockinger, J. Giddy, and D. Abramson. Economic models for management of resources in peer-to-peer and grid computing. In Proceedings of the SPIE International Conference on Commercial Applications for High-Performance Computing, Denver, USA, August 20-24 2001.
[35] Luis M. Camarinha-Matos and Hamideh Afsarmanesh (eds). Infrastructures for virtual enterprises: Networking industrial enterprises. In IFIP TC5 WG5.3 / PRODNET Working Conference on Infrastructures for Virtual Enterprises (PRO-VE '99), Porto, Portugal, October 27-28 1999. Kluwer.
[36] A. Keller M. Hovestadt, O. Kao and A. Streit. Job scheduling strategies for parallel processing. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, 9th International Workshop, JSSPP, Seattle, USA, June 24 2003. Springer.
[37] R. Buyya, D. Abramson, and J. Giddy. An Economy Driven Resource Management Architecture for Global Computational Power Grids. In Proc. of International Conference on Parallel and Distributed Processing Techniques and Applications, 2000.
[38] H. El-Rewini, T. G. Lewis, and H. H. Ali. Task scheduling in parallel and distributed systems. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.
[39] Ludek Matyska Pavel Fibich and Hana Rudov. Model of grid scheduling problem. In AAAI Press Technical Reports, editor, In AAAI05 Workshop on Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing, 2005.
[40] Gridware, Codine/GRD. http://www.gridware.com.
[41] Load Sharing Facility (LSF). http://www.platform.com.
[42] Condor, www.cs.wisc.edu/**condor**/
[43] R. Yahyapour, Implications on Resource Brokerage and Scheduling in Grids, ParCo 2003, 2 - 5 September 2003.
[44] J. Nabrzyski, J.M. Schopf, and J. Weglarz, editors. Grid resource management: state of the art and future trends. Kluwer Academic Publishers, Norwell, MA, USA, 2004.
[45] H. Dail, F. Berman, and H. Casanova. A modular scheduling approach for grid application development environments. Journal of Parallel and Distributed Computing, 63(5), 2003.
[46] J. Sgall. Online scheduling – a survey. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1997.
[47] D. G. Feitelson, Larry Rudolph, and et al. Parallel job scheduling - a status report.
[48] A. Feldmann, J. Sgall, and S. Teng. Dynamic scheduling on parallel machines, 1991.
[49] A. Fiat and G.J. Woeginger, editors. Online Algorithms, The State of the Art (the book grow out of a Dagstuhl Seminar, June 1996), Lecture Notes in Computer Science. Springer, 1998.
[50] Y. Bartal, A. Fiat, H. Karlo_, and R. Vohra. New algorithms for an ancient scheduling problem. 1992.

[51] M. R. Garey and Ronald L. Graham. Bounds for multiprocessor scheduling with resource constraints. SIAM J. Comput., 1975.
[52] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour. the design and evaluation of job scheduling algorithms, 1999.
[53] D. A. Lifka. The anl/ibm sp scheduling system. In IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, London, UK, 1995, Springer-Verlag.
[54] T. L. Casavant and J.G. Kuhl. A taxonomy of Scheduling in General-Purpose Distributed Computing Systems, IEEE Transactiond on Software Engineering, Vol. 14. No. 2, February 1988.
[55] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Journal of Future Generation Computing Systems, 15(5-6), 1999.
[56] H. Dail, O. Sievert, F. Berman, H. Casanova, A. YarKhan, S. Vadhiyar, J. Dongarra, C. Liu, L. Yang, D. Angulo, and I. Foster. Scheduling in the grid application development software project, 2003.
[57] M. Iverson, F. Ozguner, and G. Follen. Parallelizing existing applications in a distributed heterogeneous environment, 1995.
[58] T. Kawaguchi and S. Kyan. Worst case bound of an lrf schedule for the mean weighted flow-time problem. SIAM J. Comput., 1986.
[59] U. Schwiegelshohn. Preemptive weighted completion time scheduling of parallel jobs, In European Symposium on Algorithms, 1996.
[60] U. Schwiegelshohn, W. Ludwig, J. L.Wolf, J. Turek, and P. S. Yu. Smart bounds for weighted response time scheduling. SIAM J. Comput., 1998.
[61] H. Topcuoglu, S. Hariri, and M.Y. Wu. Performance-e_ective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 2002.
[62] J. Turek, U. Schwiegelshohn, J. L. Wolf, and P. S. Yu. Scheduling parallel tasks to minimize average response time. In SODA '94: Proceedings of the fifth annual, ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.
[63] C. Weng and X. Lu. Heuristic scheduling for bag-of-tasks applications in combination with qos in the computational grid. Future Gener. Comput. Syst., 2005.
[64] CoreGRID Deliverable, "Proposed Grid scheduling architecture", D.RMS.2, 2006.
[65] Global Grid Forum, "Defining the Grid: A Roadmap for OGSA(TM) Standards", H. Kishimoto, J. Treadwell (editors), Global Grid Forum Informational Document (GFD.53), 2005.
[66] Grid Resource Management System (GRMS), http://www.gridlab.org/grms
[67] HPC-Europa, http://www.hpc-europa.org
[68] Kurowski K., Nabrzyski J., Oleksiak, A., Węglarz, J., 2003, Multicriteria Aspects of Grid Resource Management, In Grid Resource Management edited by J. Nabrzyski, J. Schopf, and J. Węglarz, Kluwer Academic Publishers, Boston/Dordrecht/London.
[69] Kurowski, K., Ludwiczak, B., Nabrzyski, J., Oleksiak, A., Pukacki, J. Improving Grid Level Throughput Using Job Migration and Rescheduling Techniques in GRMS. *Scientific Programming*. *IOS Press*. Amsterdam The Netherlands 12:4 (2004) 263-273
[70] K. Kurowski, A. Oleksiak, J. Nabrzyski, A. Kwiecien, M. Wojtkiewicz, M. Dyczkowski, F. Guim, J. Corbalan, J. Labarta, Multi-criteria Grid Resource Management using Performance Prediction Techniques, *Integrated Research in Grid Computing*, Springer
[71] K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Węglarz, Scheduling Jobs on the Grid – Multicriteria Approach, to appear in *Computational Methods in Science and Technology*, OWN, Poznań
[72] Ralf Gruber, Vincent Keller, Pierre Kuonen, Marie-Christine Sawley, Basile Schaeli, Ali Tolou, Marc Torruella, and Trach-Minh Tran (2005), "Intelligent GRID Scheduling System", PPAM 2005, Poznan, Poland, Lecture Notes in Computer Sciences (Springer) 3911, p. 751-757
[73] Pierre Manneback, Guy Bergère, Nahid Emad, Ralf Gruber, Vincent Keller, Pierre Kuonen, Sébastien Noël, and Serge Petiton (2005), "Towards a scheduling policy for hybrid methods on computational Grids", CoreGRID Meeting, Pisa (28-30 November, 2005), to appear in Lecture Notes in Computer Sciences (Springer)
[74] Ralf Gruber, Pieter Volgers, Alessandro De Vita, Massimiliano Stengel, and Trach-Minh Tran, "Parameterisation to tailor commodity clusters to applications", Future Generation Computer Systems 19 (2003) 111-120
[75] Ralf Gruber, Vincent Keller, Michela Thiémard, Oliver Wäldrich, Philipp Wieder, Wolfgang Ziegler, and Pierre Manneback, "Integration of Grid Cost Model into ISS/VIOLA Meta-Scheduler environment", (Dresden, 2006) to appear.

7 Acknowledgements

This paper includes work carried out jointly within the CoreGRID Network of Excellence funded by the European Commission's IST programme under grant #004265.
Some of the referenced work reported in this paper is supported by a grant from SUN MICROSYSTEMS. In addition some of the work is funded by the German Federal Ministry of Education and Research through the VIOLA project under grant #01AK605L.

Appendix A. Questionnaire

The basis of the discussion on existing scheduling models was a survey that was conducted among the partner in the Institute on Resource Management and Scheduling who work on scheduling models. This survey gives the partners a good overview on the features and requirements of the different approaches.
For the sake of completeness, we present in the following the questionnaire as it was distributed. We look at some selected results in Section 4.

```
Questionnaire of Task 6.2 in WP6
--------------------------------
Please note that all the data provided in this questionnaire will be anonymized.

0) Introduction
         1. User profile
                Please tell us a few words about yourself and your group.

                a. Name: _____________________________

                b. Institution (please specify at least name and country):
                        _____________________________

                c. Size of the group (check one)
                   [ ] 1-2 [ ] 3-10 [ ] 10-20 [ ] 20-50 [ ] +50

                d. Size of the group working directly with the scheduler
                   (please check one)
                   [ ] 1-2 [ ] 3-10 [ ] 10-20 [ ] 20-50 [ ] +50

                e. Fields of work for which the group makes use of the scheduler
                   (please check one or more)
                   [ ] Geophysics
                   [ ] Physics, other than Geophysics
                   [ ] Chemistry
                   [ ] Computing and Computer Engineering
                   [ ] Bioinformatics
                   [ ] Food Engineering
                   [ ] Medical imaging
                   [ ] Astronomy
                   [ ] Weather and Climate
                   [ ] Automotive
                   [ ] Mathematics
                   [ ] Telecoms
                   [ ] other, please specify _____________________

                f. Contact information:
                   email: _________________

                g. Future contact
                   [ ] Do NOT contact me anymore with this kind of questions.

         2. Scheduler profile
                Please tell us a few words about the scheduler you are using.
                c. Author(s):
                d. Web site:
                e. Short description:
                ____________________________________________________________
                ____________________________________________________________
                ____________________________________________________________
                f. Your relationship with the scheduling project
                  (please check one or more)
                g. [ ] Developer [ ] User [ ] Not related


1) Grid Type

          1. Grid model
```

a. What is the grid model of the environment you work with?
[ ] Computational grid
    [ ] High performance computing
    [ ] High throughput computing
[ ] Data grid
    [ ] Data intensive computing
[ ] Service grid
    [ ] On-demand computing
    [ ] Collaborative computing
    [ ] Multimedia computing

b. What is the span of the grid environment you work with?
[ ] Private grid
[ ] Test grid
[ ] Departamental grid
[ ] Company/organization grid

c. How many other groups have access to your grid environment?

From your country (please check one):
[ ] Don't know
[ ] 0
[ ] 1-2
[ ] 3-10
[ ] >10

From Europe (please check one):
[ ] Don't know
[ ] 0
[ ] 1-2
[ ] 3-10
[ ] >10

From Asia (please check one):
[ ] Don't know
[ ] 0
[ ] 1-2
[ ] 3-10
[ ] >10

From US (please check one):
[ ] Don't know
[ ] 0
[ ] 1-2
[ ] 3-10
[ ] >10

Total (please check one):
[ ] Don't know
[ ] 0
[ ] 1-2
[ ] 3-10
[ ] 10-20
[ ] >20

2. Core middleware services
What are the core middleware services of the environment you work with?
[ ] None
[ ] Resource registration and discovery, please specify
    _____
[ ] Resource allocation, please specify _____
[ ] Monitoring, please specify _____
[ ] Accounting, please specify _____
[ ] Security
    [ ] Authentication

```
            [ ] Authorization/access control
            [ ] Credential validation and transformation
        [ ] Data access/management
        [ ] Other, please specify _____________________________

3. Environment rules
        What are the access and the usage rules of the environment you
        work with?  (i.e., where and when are you allowed to run your
        jobs? (SLAs))
        [ ] Please specify _____________________________

4. Environment size and structure

        a. Clusters / Units
        How many clusters and/or functional units (e.g., supercomputer)
        are in your grid environment
        and what is their type?
        (e.g., 5 x clusters, homogeneous nodes, homogeneous software,
        different gateways)
        [ ] Please specify _____________________________

        a. User interface/Gateway
        How many machines act as user interface/gateway and of what type
        are they? (e.g., 2 x Intel Xeon 2.4GHz per cluster)
        [ ] Please specify _____________________________

        b. Computing nodes
        How many computing nodes are present in the environment, how are
        they distributed, and what is their type?
        (e.g., 200 nodes, each node 2 x Intel P-IV 3GHz)
        [ ] Please specify _____________________________

        c. Storage devices
        How many storage devices are present and what is their type?
        (e.g., 1 x tape/1TB/access time: 1 hour, each node 1 x HDD 250GB)
        [ ] Please specify _____________________________

        d. Network
        What are the types of the used networks?
        (e.g. Myrinet 10GB/Gigabit Ethernet intra-cluster, 10 GB WAN
        inter-cluster )
        [ ] Please specify _____________________________


2) Type of Applications

        1. Unitary application type

            Unitary applications are applications that represent single
            schedulable units.
            You can submit unitary applications as they are to the local
            resource manager.
            Examples: a sequential application written in Python, a parallel
            MPI application.

            What are the types of unitary applications you use? (please check
            one or more)

            [ ] Type 1: Sequential jobs
            [ ] Type 2: Fixed-size distributed/parallel computing;
                        please specify the number of used CPUs (please check
                        one or more):
                        [ ] 2-8 [ ] 9-32 [ ] 33-64 [ ] 65-128 [ ] 129-512
                        [ ] +512

            [ ] Type 3: Moldable/Evolving computing
                        please specify the minimum and the maximum numbers of
```

```
                used CPUs (please check one or more):
                min: [ ] 1 [ ] 2-8 [ ] 9-32 [ ] 33-64 [ ] 65-128
                           [ ] 129-512 [ ] +512
                max:       [ ] 2-8 [ ] 9-32 [ ] 33-64 [ ] 65-128
                           [ ] 129-512 [ ] +512

[ ] Type 4: High-throughput computing
[ ] Type 5: On-demand computing
[ ] Type 6: Data-Intensive computing
[ ] Type 7: Collaborative computing
[ ] Type 8: other(s),
    please specify _____________________________________________
```

2. Composite application type

Composite applications, as opposed to unitary applications, comprise multiple schedulable units.
You need to submit composite applications to a specialized scheduler, or to a scheduler wrapper, e.g., DAGMan or Nimrod on top of Condor, in order to have them run.
Examples: a parameter sweep comprising a sequential application written in Python, a DAG comprising sequential and parallel applications.

What are the types of composite applications you use?
(please check one or more)

```
[ ] None
[ ] Chains-of-tools,
    [ ] composed of the unitary application types specified in the
         previous section
    [ ] composed of other types of applications:
        _____________________ (please specify)
[ ] Parameter sweeps,
    [ ] composed of the unitary application types specified in the
        previous section
    [ ] composed of other types of applications:
         _____________________ (please specify)
[ ] Simple Workflows (at most Direct Acyclic Graph structure,
    branches)
    [ ] composed of the unitary application types specified in the
        previous section
    [ ] composed of other types of applications:
         _____________________ (please specify)
[ ] Complex Workflows (e.g., Direct Cyclic Graph structure, loops)
    [ ] composed of the unitary application types specified in the
        previous section
    [ ] composed of other types of applications:
         _____________________ (please specify)
```

3. Libraries

What are the specific libraries you are using in your applications?

```
[ ] None

a. Grid-related (please check one or more)
[ ] Globus CoG
[ ] GAT
[ ] UNICORE
[ ] other, please specify _____________________________

b. Communication (please check one or more)
[ ] MPI,
    version [ ] 1, [ ] 2, [ ] don't know
```

provider [ ] MPICH, [ ] LAM [ ] other,
please specify: _____
[ ] Java RMI
[ ] PVM
[ ] OpenMP
[ ] Ibis
[ ] other, please specify

4. Resource requirements

a. What is the size and format of your input/output data requirements?

Total size (please check one or more):
[ ] Don't know
[ ] 0-10KB
[ ] 10-100KB
[ ] 100KB-1MB
[ ] 1-100MB
[ ] 100MB-1GB
[ ] >1GB

Scratch disk on each of the execution machine(s)
(please check one or more):
[ ] Don't know
[ ] 0-10KB
[ ] 10-100KB
[ ] 100KB-1MB
[ ] 1-100MB
[ ] 100MB-1GB
[ ] >1GB

Format of the input/output files (please check one or more):
[ ] Don't know
[ ] ASCI [ ] International standard
[ ] Binary [ ] Proprietary standard
[ ] XML-based
[ ] Machine-independent

b. What are your memory requirements? (please check one or more)

Fixed
[ ] Don't know
[ ] 0-10MB
[ ] 10-100MB
[ ] 100MB-1GB
[ ] >1GB

Virtual
[ ] Don't know
[ ] 0-10MB
[ ] 10-100MB
[ ] 100MB-1GB
[ ] >1GB

b. What is the time your applications run?
(please check one or more)
[ ] Don't know
[ ] seconds
[ ] <15 minutes
[ ] 15 minutes - 1 hour
[ ] 1-6 hours
[ ] 6-12 hours
[ ] 12-24 hours
[ ] 2-5 days
[ ] 5-30 days
[ ] >30 days

3) Scheduler structure

1. Scheduler architecture

What is the architecture of the scheduler you are using?
[ ] Don't know
[ ] Hierarchical
[ ] Centralized
[ ] Distributed Structured
[ ] Distributed Unstructured
[ ] Autonomous Agents
[ ] Other, please specify _____

2. Scheduler goals

What are the goals that the scheduling process can optimize for?
[ ] Don't know
[ ] Maximizing system throughput
[ ] Maximizing resource utilization
[ ] Maximizing economic gains
[ ] Minimizing the turn-around time/elapsed time for an application
[ ] Minimizing resource costs
[ ] Multi-criteria, please specify _____
[ ] Other, please specify _____

3. Scheduling type
What are the features supported by the scheduler?
[ ] Don't know
[ ] Best Effort
[ ] Static/Launch-time scheduling
[ ] Co-allocation
[ ] Dynamic scheduling to support QoS
[ ] type of SLA, reservations
[ ] Other, please specify _____

4. Types of resource management
What kind of resource management is the scheduler capable of?
[ ] Don't know
[ ] CPU management
[ ] Storage management
[ ] Network management
[ ] License management
[ ] Data management
[ ] Scientific equipment, please specify _____
[ ] Other, please specify _____

5. Scheduling heuristics
What are the heuristics applied by the scheduler?
[ ] Don't know
[ ] None
[ ] Clustering
[ ] List scheduling
[ ] OLB
[ ] Max-flow/Min Cut
[ ] Other, please specify _____

6. Failure handling
What is the support offered by the scheduler to cope with failures?
[ ] None
[ ] Restart

[ ] Redundancy
[ ] Notification
[ ] Other, please specify _____

7. Advanced data and application management
What advanced data and application management support is offered by the scheduler?
[ ] None
[ ] Replication
[ ] Checkpointing
[ ] Migration
[ ] Other, please specify _____

4) Scheduler implementation, deployment, and user support

1. Application and Resource Description Languages

What are the application and/or resource description languages understood by the scheduler?
[ ] JSDL
[ ] RSL
[ ] Condor language
[ ] XRSL
[ ] AGWL/CGWL
[ ] BPEL4WS
[ ] other, please specify

2. Dependency on/Collaboration with other services

What are the services on which the scheduler depends or with whom it collaborates?
(from the ones specified in Sec.1.3, or proprietary)
[ ] Information Service, please specify _____
[ ] Monitoring Service, please specify _____
[ ] Security Service, please specify _____
[ ] Brokerage Service, please specify _____
[ ] Data Transfer Service, please specify _____
[ ] Accounting and Billing Service,
please specify _____

3. Scheduler access

How do you make use of the scheduler's features?
[ ] Direct invocation, through the command-line interface
[ ] Direct invocation, through a web portal (using my browser)
[ ] Direct invocation, through a custom tool providing a graphical user interface
[ ] Indirectly, through another resource management tool, through the command-line interface
[ ] Indirectly, through another resource management tool, through a web portal (using my browser)
[ ] Indirectly, through another resource management tool, through a custom tool providing a graphical user interface
[ ] Somebody else submits my jobs
[ ] Other, please specify _____

4. Scheduler administration, maintenance, and support

Considering that administrators are the employees whose main task is the scheduler's administration, maintenance, and user support, how many administrators handle the configuration and the maintenance of the scheduler,and who are they affiliated to?

Number of administrators (please check one)
[ ] Don't know
[ ] 1

[ ] 2-5
[ ] 5-10
[ ] >10

Administrator's affiliation (please check one or more)
[ ] Don't know
[ ] There is only one administrator
[ ] Every organization participating in the grid has 1 delegated administrator
[ ] Every organization participating in the grid has 2-5 delegated administrators
[ ] Other, please specify _____

5) Scheduler evaluation techniques

1. Experimental setup

What is the experimental setup used to evaluate your scheduler?
[ ] Don't know
[ ] Simulator
[ ] Testbed
[ ] Production grid
[ ] Other, please specify _____

How is the scheduler workload generated?
[ ] Don't know
[ ] Synthetic workload
[ ] Workload traces
[ ] Real users
[ ] Other, please specify _____

What is the type of the scheduled applications?
[ ] Don't know
[ ] Synthetic applications
    [ ] Dynamic runtime
    [ ] Error injection
    [ ] Other, please specify _____
[ ] Real applications

2. Statistical soundness of the testing procedure
What are the statistical methods that were applied for validating your scheduler's performance evaluation?
[ ] None
[ ] T-Tests
[ ] F-Tests
[ ] Analysis of variance
[ ] Other, please specify _____