Resource Management Systems: Scheduling of Resource-Intensive Multi-Component Applications

Deliverable of the *Grid.it* project - WP8

R. Baraglia, S. Orlando, R. Perego

April 27, 2004

Abstract

Contents

1 Introduction 2

2 Grid scheduling 3

3 General Architecture of a Grid Scheduler 5

4 Grid Scheduling Systems for multi-component applications 7

4.1 The AppLes project . . . 7

4.2 The GrADS project . . . 10

4.2.1 Launch-time scheduling . . . 11

4.2.2 Rescheduling . . . 11

4.2.3 Meta-scheduling . . . 12

4.3 Grid workflows . . . 12

5 Assessment and future work 14

5.1 Summary of the planned activities . . . 15

5.2 Experimenting with application level schedulers for skeleton-based programs . . . 16

5.2.1 Performance models and mapping algorithms . . . 16

5.2.2 Launch-time scheduling vs. Rescheduling . . . 17

5.2.3 Scheduling Policies . . . 18

5.3 Workflow scheduling . . . 18

1 Introduction

A Grid is a dynamic, seamless, integrated computational and collaborative environment. Grids integrate networking, communication, computation and information to provide a virtual platform for computing and data management. Machines in a Grid are typically grouped into autonomous administrative domains that communicate via high-speed communication links. For a good description of Grid technology refer to [14, 11].

A Grid Resource Management System (RMS) is central to the operation of a Grid, and its basic functions are to accept requests for resources and assign specific machine resources to a request from the overall pool of grid resources for which a user has access permission. The RMS is constituted by the middleware, tool and services that allow to disseminate resource information, discover suitable resources and scheduling resources for job execution. The design of a RMS have to consider aspects such as: site autonomy, heterogeneity, extensibility, allocation/co-allocation, scheduling, and online control [13, **?**].

> *Site autonomy.* Computing resources are geographically distributed under different ownerships each having their own access policy, cost and various constraints. Every resource owner will have a unique way of managing and scheduling resources and the grid schedulers have to ensure that they do not conflict with resource owners policies.
>
> *Heterogeneity.* The solution of complex problems can require various kind of resources located on different sites that can use different operating systems as well as different local resource management systems which lead to significant differences in functionality.
>
> *Extensibility.* A resource management solution must support the frequent development of new domain-specific management structures, without requiring changes to code installed at participating sites.
>
> *Allocation/Co-allocation.* Applications have computational requirements that can be satisfied by using one or more resources that could be allocated simultaneously at several sites. Site autonomy and possibility of failure during allocation introduces a need for specialized mechanisms for allocating resources, initiating computation on those resources, and monitoring and managing those computations.
>
> *Scheduling.* To achieve high performance, efficient mechanism to assign job/applications tasks to the selected resources and to schedule on them their execution are needed.
>
> *Online control.* Many applications can require to adapt their execution to resource availability, in particular when application requirements and resource characteristics change during application execution.

In the past several RMS were proposed, but currently no one supports a full set of functionalities as required by a Grid RMS. Some, such as Condor [23] supports site autonomy, but not co-allocation or online control, others, such as Legion, supports online control and policy extensibility, but not the heterogeneous substrate or co-allocation problems [12]. In [16] a taxonomy to survey existing Grid resource management implementations can be found.

Even if, in the broadest sense, any application can be a Grid application, in practice, applications that can take more advantage in using the Grid are loosely-coupled, computational intensive, component based, and, often, multidisciplinary. Components can be sequential or parallel, written using different programming languages, and may be distributed across a wide-area network. In order to completely exploit the computational peculiarities of a Grid, an application has to be *grid-aware*. A grid-aware application is one that at run-time can exploit the RMS to identify Grid characteristics, and then dynamically reconfigure resource requirements and possibly the application structure to maintain the required performance. Moreover, the application should be scalable and portable.

To reach application goals such as the desired application performance and scalability, complex distributed application schedulers that exploit and extend concepts introduced by projects such as AppLes [5] are needed. Such schedulers have to be able to collect information describing the computational resources in the 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. Projects like GrADS [9] adopts a predictive approach, according to which current and historical information are taken into account in order to estimate the resource state.

As regards scheduling policy adopted within an RMS, we can distinguish two main ones: system oriented and application oriented. *Job schedulers* usually adopts system oriented policies, trying to optimize system throughput. On the other hand, *application schedulers* try to optimize application-specific metrics like completion time. While the goal of a job scheduler is to maximize the overall resource utilization, application schedulers may try to acquire much more resources than the ones each application is able to efficiently utilize. As usual, there is a tradeoff between the two approach. Several authors believe that schedulers should provides interfaces whereby external agents can changes the scheduling policy if needed. A similar approach is being to be adopted by the GrADS project, where the *metascheduler* service is an attempt to find a tradeoff 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.

The rest of this report is organized as follows. Section 2 introduces the Grid scheduling issues, while Section 3 discusses the general architecture of a Grid Scheduler. Then Section 4 surveys the main research projects related Grid scheduling of multi-component applications. Finally Section 5 discusses some research proposals for the next year of the project.

2 Grid scheduling

A Grid schedulers has to compute one or more schedules for input lists of jobs, subject to constraints that can be specified at launching-time. Such constraints can also affect the run-time of the job, if resource selection and scheduling decisions are taken by a grid-aware support of the application.

The structure 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* [23]. 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 Condor [23], LSF[2], and Condine [1]. It is not suitable for Grid RMSs as they are expected to deal with local policies imposed by resource owners. The Decentralized model seems to better fit for a typical Grid environment. In this model the schedulers interact among themselves in order to decide which resources should be allocated to the jobs being 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 schedulers has to enforce. However, because the status of remote jobs and resources is not available at single location, the possibility of generating highly efficient schedules is questionable. The Hierarchical model also fits 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), and seems to better suit grid systems. The scheduler at the top of the hierarchy is called super-scheduler/resource broker, which interacts with local schedulers in order to decide schedules.

For scheduling resource-intensive parallel applications, projects like AppLeS and GrADS worked on a Decentralized approach, where specific *Grid application schedulers*, tailored on particular applicative classes, are adopted. The idea here is that each application scheduler works on predictive estimations of the system load, and identifies the best set of resources using a performance model and considering user-provided requirements and application-specific metrics, e.g., the minimization of application completion time. On the other hand, job schedulers for parallel application have to multiplex different applications on a set of shared resources, and usually consider the parallel applications as black-boxes, without considering that the application can be reconfigured to run on less or more resources on the basis of their availability. Without this application-based configuring phase, it is possible that applications can experience large slowdown in time-shared systems, and high queue waiting times in space-spared systems.

Scheduling algorithms are classified as *static* or *dynamic*. In the former the mapping decisions are taken before executing an application and are not changed until the end of the execution. Also launching-time scheduling for parallel applications, where configuration and mapping decisions are taken before running the application, can be classified as static scheduling policies. In the latter the mapping decisions are instead taken while the program is running. Since static mapping usually does not imply overheads on the execution time of the mapped application, more complex mapping solutions than the dynamic ones can be adopted. When the characteristics of a parallel application, such as task computational cost, amount of data exchanged among tasks, and task dependencies, are known before the application execution, a static mapping approach can be profitably exploited. Otherwise, a dynamic approach has to be adopted, where a monitoring system helps the application scheduler to reschedule the application to respect a given performance contract.

While launch-time reconfiguration of a parallel application only requires malleable code, - i.e. applications that can be configured without recoding in order to exploit a different number of resources – dynamic scheduling need grid-aware supports – i.e. such

application have to be based on a run-time support that can manage the rescheduling and the interaction with the grid RMS.

3 General Architecture of a Grid Scheduler

In [28] Schopf delineates the general architecture of a Grid scheduler, also termed super-scheduler or broker. The phases of this scheduler are illustrated in Figure 1. Grid scheduling involves three main phases: resource discovery, which generates a list of potential resources; information gathering about those resources and selection of a good set of them; and job execution, which includes file staging and cleanup.

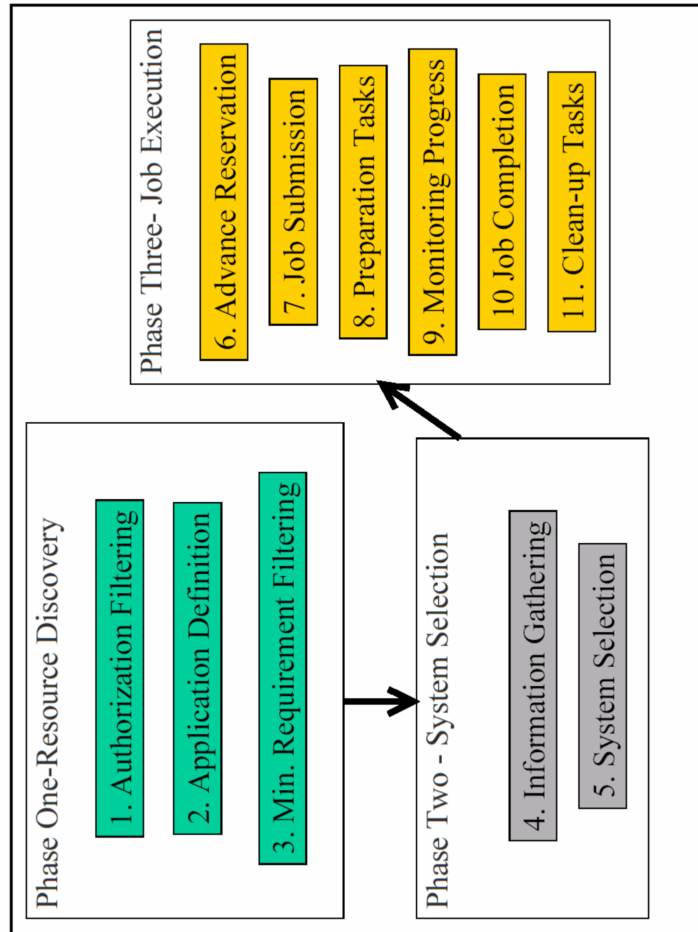


Figure 1: Architecture of a general Grid scheduler.

In the following of this section we'll use this scheme as a general framework for

discussing Grid scheduling issues, independently of the policies adopted, the kind of applications, etc.

Resource Discovery. The first phase, i.e. resource discovery, is perhaps the less studied in the scheduling community. This phase requires a standard way to express application requirements with respect to the resource information stored in the Grid Information System. It is thus needed an agreed-upon schema to describe the attributes of the systems, in order for different systems to understand what the values mean. 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. For example, in the GrADS project [8] the list of machines where to map applications is currently obtained directly from the user. They claim that once secure MDS publishing mechanisms are available, user account information can be published directly in the MDS and retrieved automatically by the scheduler.

Resource selection. Resource selection usually occurs after the first resource discover phase. While the first phase filters out unsuitable resources, this phase should determine from this large list the best (set of) resource(s) chosen to map the application.

This selection requires to gather detailed dynamic information from resources, e.g. by accessing local GRIS of Globus middleware, or querying performance prediction systems such as Network Weather System (NWS) [26]. 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 the application.

While resource selection can be quite simple for sequential jobs, this selection can become particularly complex for multi-component parallel applications. In [17], Liu and Foster introduce a constraint language for the selection of resources for parallel jobs. The language extends the ClassAds one used by the Condor's matchmaking mechanism. The important idea is the set-extension of the ClassAds language, in order to express constraints for the selection of "set of resources" suitable for co-allocating parallel applications. For example, the authors are interested in expressing collective constraints on the set of resources to select, such as the aggregate memory size of a set of workstations. Another important argument discussed in this paper is the criticism to the method currently used to express the description of the resources, published using Grid services like MDS, and used by the scheduler to select suitable resources. This mechanism is considered unflexible and not suitable with respect to the autonomous nature of Grid resources. For example, the owner of a resource might want to allow access only to users belonging to a certain group or able to pay a fee. It was exactlt this observation that leds Raman et al. [25] to propose that resource retrieval and selection should be treated as a bilateral matching process.

The problem of selecting and co-allocating set of resources for parallel applications has been recently discussed by Berman *et al.* in [8]. They face the problem by first subdividing the machines into disjoint subsets, or sites. In this way they introduce a hierarchical layering into the "completely connected" network graph. They try to determine clusters of machines (sites) on the basis of network bandwidth dconsiderations,

where the network delays within each cluster are presumably lower than the network delays between clusters. In the selection of the best set of machines, their mapping algorithm considers this hierarchy, by trying to maximize the usage within each site. For example, by trying to allocate a program within a single site, if it is possible and profitable. While in their first implementation Berman *et al.* only group machines into the same site when they share the same domain name, they plan to consider more sophisticated approaches based on [29].

Finally, in order to finally select the actual subset of systems to run a parallel application, several researchers claim that the optimal choice can be only done by exploiting a (more or less accurate) performance model. See sections 4.1 and 4.2 for a description of this selection strategy.

Job Execution The last phase of the scheduling architecture regards job execution. This phase can be very complex, since the preparation of a job run can require various intermediary steps, like staging of files, advance reservation, etc.

One of the main activities shown in Figure 1 regards the monitoring of the progress of application execution. A user, when a job is not making sufficient progress, may stop the job and reschedule it by returning to Step 4. Such rescheduling is significantly harder for parallel job executing on multiple sites. Dynamic scheduling is needed it that case.

4 Grid Scheduling Systems for multi-component applications

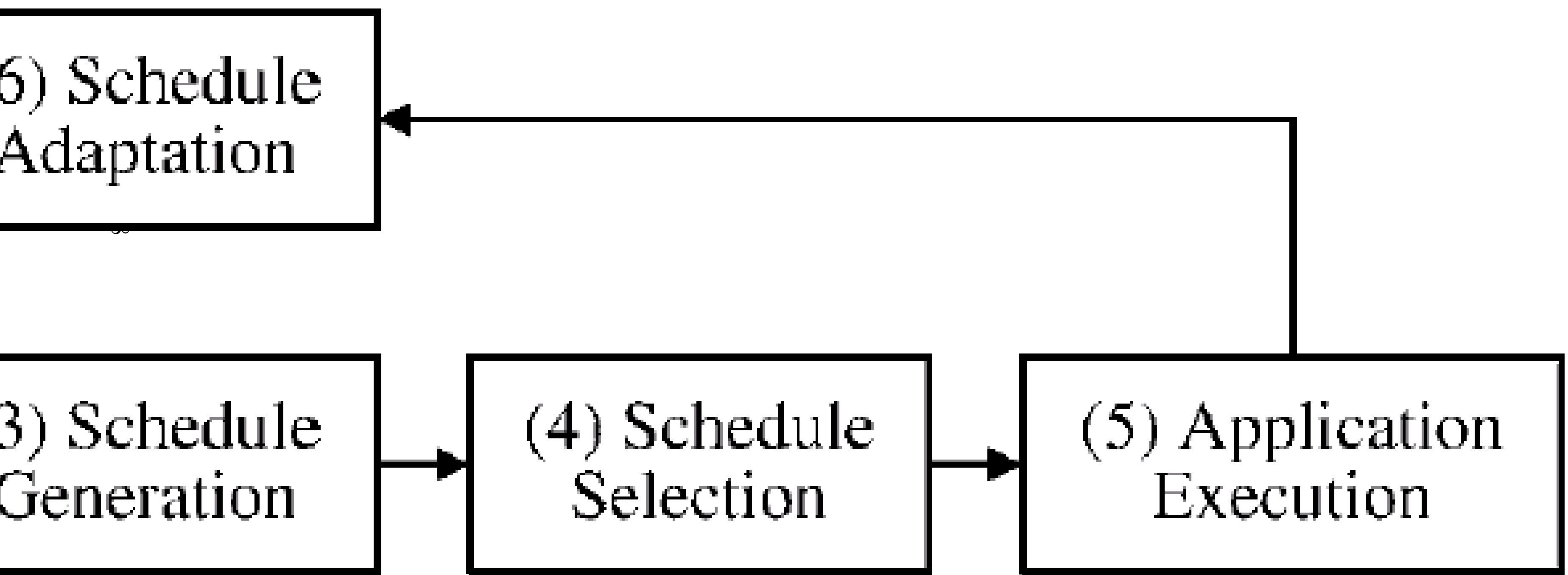
In this section we give a brief overview of some current Grid scheduling efforts in the area of multi-components resource-intensive parallel applications, surveying the main results of the AppLes and GrADS projects. Finally we introduce the concept of Grid workflow, as presented in literature, and some of the approaches currently followed for their scheduling.

4.1 The AppLes project

AppLeS (Application Level Scheduling) is a project leaded by F. Berman at the University of California, S. Diego. It is a metodology for adaptive application scheduling on heterogeneous computing platforms. The AppLes approach exploits static and dynamic resource information, performance predictions, application and user-specific information, and scheduling techniques that adapt "on-the-fly" to application execution.

In Figure 2 the phases in the Apples scheduling methodology are shown.

As we can see, the System Selection phase of the general scheduler architecture of Figure 1, in AppLeS is split into three sub-phases: (2) Resource selection, (3) Schedule generation, and, (4) Schedule selection. During sub-phase (2) the resources enabled to run the application are selected according to application-specific resource selection models. To this end, AppLeS uses information carried out by the Network Weather Service (NWS) performance monitor (NWS is a distributed system that periodically monitors and dynamically forecasts the performance various network and computational resources can deliver over a given time interval). An ordered list of viable


6) Schedule
Adaptation
3) Schedule
Generation
(4) Schedule
Selection
(5) Application
Execution

resources is finally produced. In (3) a performance model is applied to determine a set of candidate schedules for the application on the selected resources (for any given set of resources, many schedules may be possible). In (4) the schedule that best matches the chosen performance criteria is selected.

The AppLeS approach requires to integrate in the application a scheduling agent which must be customized according to application features. In order to make easier this customization, *templates* to be applied to classes of applications with common characteristics were introduced. Templates for parameter sweep applications (APST), master/worker applications (AMWAT), and for scheduling moldable jobs on space-shared parallel supercomputers (SA) are currently available.

- Parameter sweep applications are structured as sets of computational tasks that are mostly independent. The APST template provides two distinct processes: a daemon, which is in charge of deploying and monitoring applications, and a client which is essentially a console that can be used either interactively or from scripts. The user can invoke the client to interact with the daemon to submit requests or to check application progress. The user describes the program to run and the needed resources by a XML file, and sends such file to the daemon by a client. The daemon locates the programs to run, selects the required computational resources and according to a scheduling policy move programs and input data, if necessary, on the selected resources. At the end of the application execution the output files are transferred back to the user. Scheduling decisions are taken by dynamically generating a Gantt chart to schedule unassigned tasks between scheduling events. The ASTP software compute the next scheduling event and accordingly creates or updates a Gantt Chart. Then, for each computation and file transfer currently underway, it computes an estimate of its completion time and fill in the corresponding slots in related Gantt Chart. Then, it selects a subset of the tasks that have not started execution, and until each host has been assigned enough work, it heuristically assigns tasks to hosts, filling in slots of the Gantt Chart. Finally, it implements the computed scheduling. The heuristics implemented for scheduling independent tasks are: Max-min, Min-min, Sufferage and XSufferage [21, 18].
- The AMWAT template provides an API for structuring applications with a single master process which controls the flow of computation that is performed on one or more remote worker processes.

  AMWAT achieves application portability by providing common interfaces to various services such as interprocess communication and process invocation. In order to achieve good performance under different conditions, several scheduling strategies are provided by this remplate. Thesestrategies are: one-time fixed allocation (FIXED) strategy, Self Scheduling (SS) [22], Fixed Size Chunking (FSC) [7], Guided Self Scheduling (GSS) [6], Trapezoidal Self Scheduling (TSS) [30], and Factoring (FAC2) [27]. While FIXED, SS, and FSC are examples of allocation strategies which apply the same block size throughout an application run, GSS, TSS, and FAC2 are examples of strategies which utilize decreasing block sizes as the application progresses.
- The SA (Supercomputer AppleS) template allows user-directed scheduling of moldable jobs (i.e., jobs that can be executed with any of a collection of possi-

ble partition sizes) in a batch-scheduled, space-shared, back-filling environment. Such environments are common in production supercomputer centers and include MPPs scheduled by EASY [79], the Maui Scheduler [80], and LSF [81]. The user provides SA with a set of possible partition sizes that can be used to submit a given moldable job. SA uses simulations to estimate the turn-around time of each potential request based on the current state of the supercomputer and then forwards to the supercomputer the request with the smallest expected turn-around time. SA does not always select the best request because the execution times of the jobs already in the system are not known (request times are used as estimates) and future arrivals can affect jobs already in the system. However, SA chooses close to an optimal request for most jobs and its pick is generally considerably better than the user's choice.

4.2 The GrADS project

The goal of the Grid Application Development Software Project (GrADS)[1] [9, **?**] is to realize a Grid system, by providing tools, such as problem solving environments, Grid compilers, schedulers, performance monitors, to manage all the stages of application development and execution. Using GrADS the user will only concentrate on high-level application design without putting attention to the peculiarities of the Grid computing platform used.

The GrADS system is composed of three main components: Program Preparation System (PPS), Configurable Object Program (COP), and Program Execution System (PES). The PPS component handles application development, composition, and compilation. To develop their Grid application, users interact with a high-level interface providing a problem solving environment, which permits the integration of the application source code, software components and library modules. Then, the resulting application is passed to a specialized GrADS compiler that generares an intermediate representation code and a configurable object program (COP). The COP encapsulates all results (e.g. application performance models and the intermediate application representation code) of the PPS phase for later usage. The PES components provides on-line resource discovery, scheduling, binding, application performance monitoring, and rescheduling.

To execute an application, the user submits parameters of the problem such as problem size to the GrADS system. The PPE component receives the COP as input and, at this stage, the scheduler carries out an application-appropriate schedule. The binder is then invoked to perform a final, resource-specific compilation of the intermediate representation code. Next, the executable is launched on the selected Grid resources and a real-time performance monitor is used to track program performance and detect violation of performance guarantees. Performance guarantees are formalized in a performance contract. In the case of a performance contract violation, the rescheduler is invoked to evaluate alternative schedules.

The scheduler is a key component of the GrADS system. In GrADS, scheduling decisions are taken by exploiting application characteristics and requirements in order to

[1]Sponsored by the Next Generation Software (NGS) program of the National Science Foundation. Istitutions involved: Rice University, UCSD, University of Tennessee, University of Chicago, Indiana University, University of Houston, ISI-USC, and UIUC

obtain the best application execution time. Monitoring and Discovery Service (MDS) [?], Network Weather Service (NWS) [26, 33], ClassAds/Matchmaking approach [24], Globus Toolkit [12] and GridFTP for file transfer [32] are the existing software components used to implement the GrADS system.

GrADS supports three scheduling phases: Launch-time scheduling, Rescheduling, and Meta-scheduling.

4.2.1 Launch-time scheduling

Launch-time scheduling is used before application start to determine how the current application execution should be initially mapped to available Grid resources. This technology, the research on which is leaded by the same people of the AppLeS project [8], is perhaps the most mature one in GrADS. The scheduler takes by a user the list of the machines he/she wants to use and then it collects static and dynamic information describing the required machines by querying MDS and NWS, respectively. Then, according to the application requirements, the machines that are not usable to run the application are discarded. Then the information describing the remaining machines is used by a search procedure to carry out scheduling. In GrADS two search procedures: Resource-aware, and Simulated annealing have been developed.

The performance model is used by the search procedure to estimate the application performance when it is executed on a specific set of resources (some metrics such as application execution or turnaround time, and system throughput can be used to this end). The performance model takes as input a proposed schedule, the application requirements and characteristics, and Grid resources information, and returns the predicted execution time for that schedule. On the basis of the estimated cost for different schedules mapping decisions are taken. In GrADS three mapping strategies, called Equal Allocation, Time Balancing, and Data Locality are investigated. The choice of the mapping stategy is function of the application considered. The Equal Allocation is the default and it allocates data evenly among the selected machines. The goal of Time Balancing is to balance the work among the processors used in order to remove a performance constraining due to the slowest processors. The Data Locality strategy tries to reduce the data movement between machines.

4.2.2 Rescheduling

Rescheduling consists in modifying the application schedule during execution in order to sustain good performance for long running applications when other applications introduce load variations in the system. Rescheduling can include changing the machines on which the application is executing (migration) or changing the mapping of data and/or processes to those machines (dynamic load balancing). In GrADS rescheduling solutions are introduced only for iterative applications, and rescheduling decisions are evaluated at each iteration.

The solutions proposed in GrADS use application and resource sensors to monitor application execution progress and resouce usage. Application sensors are co-located with application processes to monitor application progress, while resource sensors are located on the machines on which the application is executing, as well as the other machines available to the user. When performance falls below expectations, the rescheduler must determine whether rescheduling is profitable, and if so, what new schedule

should be used. In GrADS two rescheduling solutions called Application Migration and Process Swapping have been experimented. To implement these solutions two user-level checkpointing libraries were provided. The Application Migration was implemented adopting the stop/start approach. When a running application is signaled to migrate, all application processes checkpoint their internal state and shutdown. To this end, a user-level checkpointing library called Stop Restart Software (SRS) has been developed [**?**].The rescheduled execution is launched by restarting the application, which then reads in the checkpointed data and continues its execution.

To reduce application migration overhead, the Process Swapping solution has been introduced. To enable swapping, the application is launched on more machines than those actually used for the computation. Some of these machines become part of the computation (the active set) while some do nothing initially (the inactive set). During execution, the system periodically checks the performance of the machines and swaps loaded machines in the active set with faster machines in the inactive set. This approach requires little application modification and it reduces the introduced overhead, but, on the other hand, it is less flexible than the Application Migration one. The processor pool is limited to the original set of machines and there is no incorporated support for modifying data allocation.

4.2.3 Meta-scheduling

The launch-time scheduler elaborates an application at a time without considering the presence of other applications in the system. Decisions taken to schedule an application can thus limit or deny the execution of other applications that are subsequently submitted to the GrADS system. To limit these kind of problems, a metascheduler [**?**] has been implemented. This metascheduler collects information describing all applications in the system, and by exploiting this global knowledge, tries to balance the needs of the applications and the overal performance of the system. The metascheduler is strucured according to four components: database manager, permission service, contract developer, and contract negotiator. The fisrt one acts as a data repository storing information about all applications in the system (e.g. status of applications, launch-time application-level schedules, and predicted application execution costs). The database manager is queried by the other components to make scheduling decisions. The primary objective of the permission service is to accommodate as many applications in the system as possible while respecting the constraints of the resource capacities. The permission service thus decides if allowing or denying the execution of an application.

The request for the execution of an application is rejected if the scheduling decisions are based on outdated resource information, or if its execution can severely impact the performance of other executing applications. Moreover, the permission service can also decide to preempt an executing application to improve the performance contract of a new application. This kind of scheduling decisions tries to provide high performance to the individual applications and to increase the throughput of the system.

4.3 Grid workflows

A central concept for the deployment of Grid software components is the adoption of a *workflow* description model to specify the coordination level of the activities/services in

execution on the Grid. Workflows capture the linkage of constituent services together in a hierarchical fashion to build larger composite services. In several papers workflows captures the "programming the Grid" concept, and encompasses a broad range of approaches with names like "Service Orchestration", "Service or Process Coordination", "Service Conversation", "Web or Grid Scripting". While the *activities/services* are units of work to be performed by agents, computers, communication links etc., the *process description* of a workflow is a structure describing both the activities to be executed and the order of their execution [19].

Traditionally, workflows are encountered in business management, office automation, or production management. Grid workflows have several peculiar characteristics. In a high-performance computing grid, we are mostly concerned with computational tasks that execute on heterogeneous resources. Grid workflow tasks are thus heavy computational processes, parallel jobs running on multitude of processors, terabyte-size data transfers to visualization servers at different sites, and archiving large data sets to mass storage. Thus *Resource allocation* and *workflow scheduling* are particularly critical aspects of the Grid-based workflow enactment. Moreover, because Grids provide heterogeneous platforms, different batch schedulers, and varying site policies, it is extremely difficult for users to deal directly with the different interfaces.

In this section we will survey the main approaches to workflow scheduling currently followed in the Grid community. After that we will assess our view and describe the research directions we are going to follow.

In the scientific Grid community, the component model and the related workflow concepts often correspond to merely determining the execution order of sequences of tasks, which simply read/write raw files. Many scientific applications are built, in fact, by composing legacy software components, often written in different languages, where the seamless interface is realized through raw permanent streams (files). The most common Grid workflow can thus be modelled as simple Task (Directed Acyclic) Graphs (DAGs), where the order of execution of tasks (modelled as nodes) is determined by dependencies (in turn modelled as directed arcs). Each DAG node represents the execution of a component, characterized by a set of attributes such as an estimate of its cost and possible requirements on the target execution platform, while DAG directed edges represent data dependencies between specific application components. Data dependencies will be usually constituted by large files written by a component and required for the execution of one or more other components of the application.

Even if several projects have addressed the composition of specific sequences of Grid tasks, and several groups have developed visual user interfaces to define the linkage between components, currently there is no consensus on how workflow should be expressed. Within this framework, the most notable example is the Directed Acyclic Graph Manager (DAGMan) for Condor [31], i.e. the well known workload management system for compute-intensive jobs. Condor is a major effort to reach *high throughput computing* on distributed resources. It features resource management and scheduling for jobs and data on large collections of computing elements. It also offers very effective scheduling strategies for parameter-sweep applications. DAGMan is a Condor meta-scheduler, that submits jobs to the high-throughput Condor scheduler in an order represented by a DAG and processes the results. The user can submit a DAG by specifying its jobs and their dependencies with a simple scripting language. The DAGMan meta-scheduler processes the DAG dynamically, by sending to the condor

scheduler the jobs as soon as their dependencies are satisfied and they become ready to execute.

Besides DAGs, UNICORE [10] provides more sophisticated control facilities in the workflow language. Constructs such as Do-N, Do-Repeat, If-then-else, and Hold-Job have been defined and integrated into the abstract job description and the client UNICORE GUI. In particular, Do-N forces the repetition of an activity N times, where N is fixed at submission time, Do-Repeat repeats an activity until a condition evaluated at runtime becomes true, If-Then-Else executes one of two activities depending on a condition evaluated at runtime, while Hold-Job suspends an activity until a given time/date has passed.

The problem of scheduling DAG workflows has been recently addressed by Ken Kennedy et al. [**?**] within the GrADS infrastucture. They proposes the adoption of a two-steps, in-advance, static scheduling policy for each DAG. In other words, the devised an application scheduler for a single DAG that statically takes scheduling decision. First, for each component of the DAG the available resources are ranked to reflect their expected performance in executing the specific component. Then, on the basis of the ranks estimated for each component and resource, a performance matrix is built and used by some known heuristics to obtain a mapping of components to resources. Known heuristics (min-min, min-max, sufferage) to optimize the mapping of independent jobs are used when several DAG's nodes become runnable upon the scheduling of their parents in the graph. The author claim that static knowledge of DAG tasks' costs and their dependencies allows global optimizations to be devised that cannot be addressed by dynamic approaches such that followed by DAGMan. Unfortunately, precisely estimate component execution cost is usually very hard. In fact, the computational cost of a component often depends not only on the size of data, but also on run-time parameters provided by the user. Consider for example an Association Rule Mining (ARM) analysis: its complexity not only depends on the size of the input dataset, but also on the user-provided support and confidence thresholds. Moreover, the correlations between the items present in the various transactions of a dataset largely influences the number and the maximal length of the rules found by an ARM tool [**?**]. Therefore, it becomes difficult to predict in advance either the computational and input/output costs, or the size of the output data.

Moreover, a grid environment is intrinsically dynamic. Site and communication link performance may vary rapidly in an unpredictable way thus vanishing the efforts done to statically optimize the overall DAG scheduling.

5 Assessment and future work

This section deals with our future work within the WP8 of the *Grid.it* project. Before outlining this, however, we would like to assess the literature on Grid RMSs, and remark some promising research direction to investigate.

First of all, we can say that the skeleton-based programming model chosen in our project should permit to better derive performance models to devise specialized application schedulers. The benefits of integrating with applications a scheduling agent, which must be customized according to application features, was formerly observed by the AppLes researchers. Moreover, to make feasible this Grid programming method-

ology, *templates* to be applied to classes of applications with common characteristics were introduced in AppLes. Note that the concept of template is similar to the skeleton one pursued in our project.

Similarly to GrADS, both launching-time and dynamic scheduling approaches should be investigated in our project. While the first one only requires a parallel code that can be configured at launching-time, the last one may require the following features:

- The ability of the application code (run-time support) to balance the load between (all, or a subset of all) the resources allocated at loading-time. Note that this feature is also needed in case the application has on board an agent able to acquire or release resources on the basis of monitoring and performance model of the application. Consider that, even when it is not profitable to modify resource allocation at run-time, a non-intrusive load balancing policy should be useful to solve sudden changes in the resources' capacities.
- The ability to completely freeze the application, and continue the execution on a different set of resources. Such technique could be useful when the current set of resource are becoming globally inefficient or failed. Hence, this checkpointing features should be also necessary to guarantee some dependability features. A point to investigate is the seamless integration of user-level checkpoint library in the run-time support of our skeleton-based language. Finally, note the the problem of choosing another set of resources to continue the running of an application is similar to a launch-time scheduling one.

A point to be deeply investigated is also the integration of dynamic information sources into the Grid resource information service, and how we can count on this information to devise good scheduling techniques. Mainly because the dynamic information on the resource loads may become quickly stale and unstable, forecasting based on historical logs may become more predictive and stable. Several projects thus investigated the use of a prediction system such as NWS. The experiments presented in literature are however limited, and further investigations should be carried out on this topic to better understand pro/cons of this approach vs simpler ones.

Finally, since Grid workflows are considered a powerful way to orchestrate Grid computation, we think that their expressive power and scheduling issues need to be deeply investigated in this project. We noted that in the Grid community the concept of workflow often corresponds to simple DAGs, where arcs represent (data) dependencies between component, so we think that it is better to consider DAGs as the fist step of our research.

5.1 Summary of the planned activities

The main activities on which we plan to concentrate our efforts in the next year are the following:

- Definition of a set of *synthetical* kernel applications, developed according to a set of predefined skeletons and their compositions, developed in order to adhere to known execution/communication costs and performance models on a given base architecture (e.g., an unloaded cluster, composed by identical workstations, interconnected by a given network LAN). The idea is to use them as benchmarks

for our RMS. Moreover, using the same approach used to build the benchmark library, it should be possible to construct other kernels that mimic the behavior of some use-case applications developed in the *Grid.it* project.

- Develop and experimentally evaluate launching-time configuration and application scheduling strategies based on specific policies, tailored for the various parallelism schemas. The evaluation of the policies will be done using the benchmark library above.
- Start with the first experiences in dynamic resource management, based on application rescheduling and load balancing, in order to evaluate feasibility of grid-awareness.

Moreover we plan to study the scheduling of simple workflow of jobs, organized as simple DAGs, where jobs can access large files. We want to study the strategies for allocating workflows using a centralized job scheduler, or separated application schedulers.

5.2 Experimenting with application level schedulers for skeleton-based programs

To experiment our application-level schedulers, we would like to create a set of synthetical benchmark applications for which we know how to optimize the mapping, once a suitable set of resources are selected. In some cases, when the application contains non-uniformity, we can postulate to know average and variance of some costs, on the basis of which to optimize the resource usage. Since we need to build this set of synthetical applications which adhere to specific performance models, we also need a methodology to build them.

We would like to evaluate our methods for launching-time configuration and mapping of our benchmarks in the context of a real, non-dedicated Grid, where systems for monitoring and prediction of performance are available. This means that an application scheduler has to base its choices on a system like NWS to make careful choices, even if we plan to consider various relaxations of our methodology for mapping Grid applications. For example, we can renounce to take into account accurate knowledge on the status of Grid system, or we can relax/symplify the application performance model.

5.2.1 Performance models and mapping algorithms

The selection of a suitable set of resources for a skeleton-based application and their composition requires to have a performance model of the various components of an application. In our simplified view, a component can only be a pipeline stage (or filter) with input/output streams. The performance information we have about each components regards communication/computation bandwidth and memory usage. Such filters can be composed as an ordinary simple pipeline schema, or as a more complex generalized/generic pipeline graph.

We assume to also have information about the stateless behavior of each stage, so that it can be replicated at launching time according to a farm skeleton. Moreover,

some stages may give the opportunity of exploiting data parallelism, so that we assume that the component can be configured at launching time for exploiting multiple heterogeneous nodes, with suitable data allocation strategies aimed at balancing the load.

The mapping algorithms we plan to exploit are very simple. First of all, strictly-coupled parallel components (such as data parallel ones) should be mapped as much as possible on "close" computational nodes. This concept requires to be better specified, but the rough idea is that the mapping algorithm has also to exploit information about the network topology of the target Grid. While task parallelism usually entails loosely-coupled components, which might be mapped to nodes connected by a long latency (low bandwidth) network, data parallelism usually requires to allocate components on clusters (or multiple clusters interconnected by an efficient network). Another qualitative consideration, to be transformed into quantitative ones in order to be exploited by our mapping algorithm, is concerned with the load variance registered by our Grid sensors. It is well known, in fact, that (strictly-coupled synchronized) data parallel skeletons should require co-scheduling to avoid waiting delay in the synchronization. Since co-scheduling is not supported by current operating systems, while our target systems would be an ordinary time shared parallel system without advance reservation services, we had to avoid to co-allocate part of a data parallel components on nodes that can suddenly become highly loaded, thus slowing down the rest of the computation due the global synchronizations that unfortunately characterize this skeleton.

5.2.2 Launch-time scheduling vs. Rescheduling

Till now we have dealt with launching time scheduling decisions. The problems with Grid is that, for long running applications, launching time optimization cannot suffice. Moreover, when costs are not available at all, such optimizations may become impossible.

The solution to these issues is to introduce dynamic policies within the skeleton supports. For example, a load balancing between the resources assigned at launching time. The application scheduler could assign more resource that those strictly needed, while the application (the skeleton support) could balance the run-time support the resource

The overheads of rescheduling can be high: monitoring for and evaluating the need to reschedule to fulfill a given performance contract is a very complex process. When a rescheduling event is initiated, migration of application processes or reallocation of data can be very expensive operations. Without careful design, rescheduling can thus hurt application performance.

Data from the resource sensors can be used to evaluate various schedules, but the re-scheduler must also consider (i) the cost of moving the application to a new execution schedule and (ii) the amount of work remaining in the application that can benefit from a new schedule.

The most transparent migration solution would involve an external migrator that, without application knowledge, freezes execution, records important state such as register values and message queues, and restarts the execution on a new processor. Unfortunately, this is not yet feasible as a general solution in heterogeneous environments. So the application-level approach still seem the most feasible solution.

The approach currently followed by GrADS is based on rescheduling actuators located on each processor. To initiate schedule modification, the rescheduler contacts these actuators, which use user-level mechanisms to initiate the actual migration [**?**]. Application support for migration is the most important part of such rescheduling system. An interesting research topic in our project regards the evaluation of a transparent integration of a checkpoint library in the support of a skeleton based language.

An alternative schema to guarantee good levels of performance when the system characteristics change at run-time is to assign at launching time more resources than the needed ones to each application. While an application starts running using some of the total computational resources assigned, it can easily add/remove resources from its working set to accomodate changes in the system feature. Clearly, the application must be able to balance the load among the computational resource currently used.

The first candidate skeleton applications for testing dynamic resource management are data-parallel (long running) iterative applications, where rescheduling decisions can naturally taken at each iteration/synchronization. Some load balancing techniques for these applications have been introduced and evaluated by the authors of this report [**?**, **?**]. In [**?**], we also studied the exploitation of such load balancing technique in a shared cluster, a platform similar to our Grid platform of reference.

5.2.3 Scheduling Policies

Even if application scheduling usually exploits a greedy policy, which tries to assign as much resources as possible to reduce the single application latency, we would like to experiment application policies that tries to find a tradeoff between user requirements and system one. In other words, we want to avoid to assign too many resources if they are not well exploited by the application. For example, at launching time a mapping decision could be to assign resources till it is possible to maintain the *efficiency* of the parallel application high enough. The threshold of the efficiency could be chosen as a function of the system load: low when the system is under-loaded, and high when the system is becoming overloaded. We would like to evaluate this solution, and compare it with an approach based on a hierarchical metascheduler as in GrADS.

5.3 Workflow scheduling

Since the Condor project [31] and the GrADS one [**?**] are approaching the problem of the workflow/DAG scheduling using different approaches, a completely dynamic vs a completely static one, we intend to evaluate them and other approaches with a simulator. Note that this simulation infrastructure can also be used to evaluate other scheduling approaches for different applications classes.

In more detail, we plan to design and evaluate various strategies for allocating DAG workflows submitted to an on-line meta-scheduler which takes its scheduling decisions by considering:

- meta-information associated with DAG nodes. Meta-information includes a set of attributes describing component costs and requirements. Requirements might be memory occupation, OS type, use of particular libraries, uri of (possibly replicated) input files etc. The cost is a possible imprecise estimate of job execution

time modelled by an average value and a variance with respect to a known architecture of reference;

- meta-information associated with DAG directed edges. This information regards job data dependencies and it is weighted on the basis of an estimate of the size of raw file(s) produced and consumed by the parent job and son job, respectively;
- information about the actual power and current status of grid computational and communication resources; Such information originate from both static and dynamic sources, where dynamic information is periodically refreshed by interacting with grid monitoring tools such as NWS;
- historical knowledge of past scheduling decisions made;
- logs recording events such as actual starting and ending time of previously scheduled jobs. Since both the cost estimates of job scheduled in the past and dynamic information about the grid status may be imprecise, the event system allows the meta-scheduler to update and refine its forecasting.

Figure 3 shows the high-level architecture of the on-line grid workflow scheduler we are going to design. Experiments and evaluations will be conducted by adopting a simulation approach. We are currently setting up our simulation environment based on the *GridSim Toolkit* [**?**]. The *Event Listener* and *Grid Status Manager* modules have been already implemented, along with the software layer that allows to instantiate the grid infrastructure by means of the GridSim toolkit. DAGS are specified by using a formalism inspired by the DAGMan scripting language, while the hierarchical configuration of the grid (number of sites, resourse and communication capabilities of the machines and networks within each site) is taken from a simple XML file.

References

[1]

[2] Load Sharing Facility (LSF). http://www.platform.com.

[3] Angulo, D. and Foster, I. and Liu, C. and Yang, L. Design and Evaluation of a Resource Selection Framework for Grid Applications. In *http://www.globus.org/research/papers.html*, 2002.

[4] R. Baraglia et al. AssistConf: a Grid configuration tool for the ASSIST parallel programming environment. In *Proc. of IEEE Euromicro 03*, 2003.

[5] F. Berman, R. Wolski, H. Casanova, et al. Adaptive Computing on the Grid Using AppLeS. *IEEE Trans. on Parallel and Distrib. Systems*, 14(5), 2003.

[6] D.J. Kucks C.D. Polychronopoulos. Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers. *IEEE Trans. Parallel and Distributed Systems*, 36(12), 1987.

[7] A. Weiss C.P. Kruskal. Allocating Independent Subtasks on Parallel Processors. *IEEE Trans. Ssoftware*, 11(10), 1985.

[8] H. Dail, F. Berman, and H. Casanova.

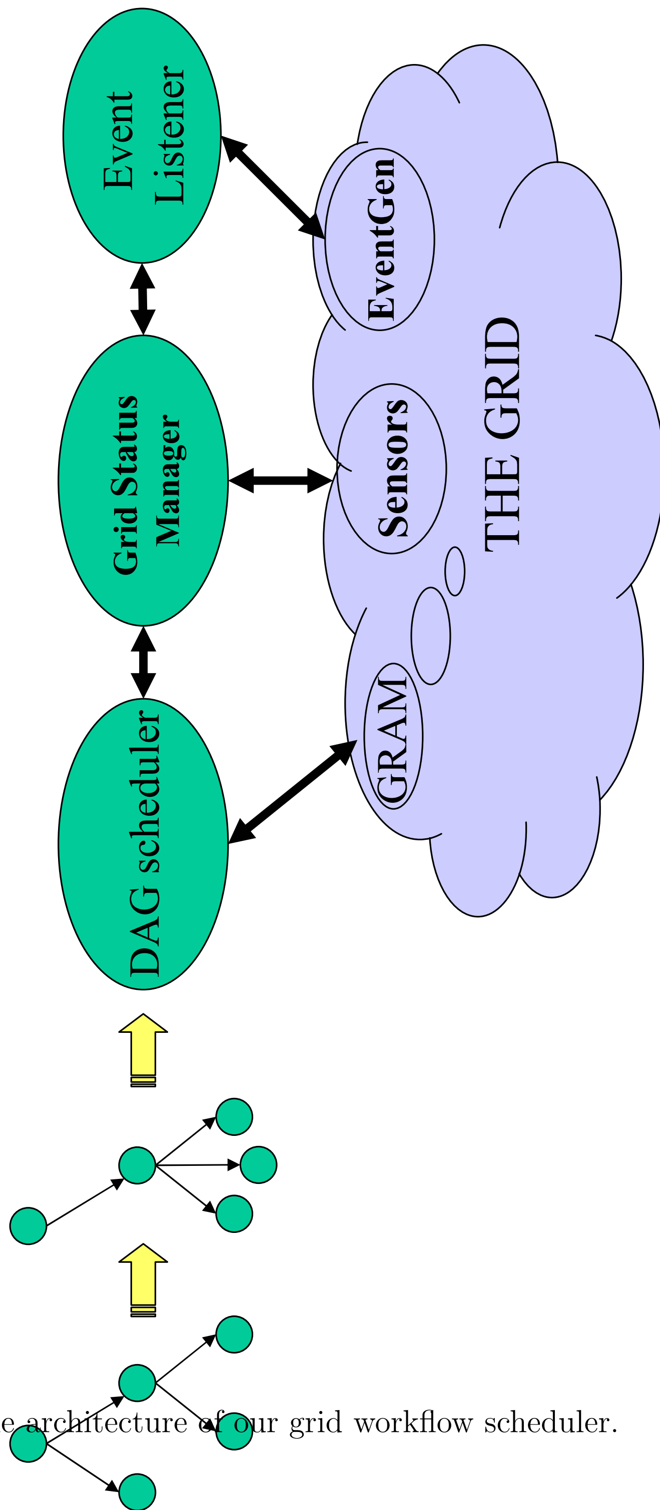


Figure 3: The architecture of our grid workflow scheduler.

[9] 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. In J. Schopf M. Stroinski J. Weglarz, J. Nabrzyski, editor, *Grid Resource Management*. Kluwer, 2003.

[10] Dietmar Erwin. *Unicore Plus Final Report*. 2003. http://www.unicore.org/ forum/ documents.htm.

[11] A. J. G. Hey F. Berman, G. C. Fox. *Grid Computing: Making the Global Instrastructure a Reality*. Willey Series in Communications Networking and Distributed Systems, 2003.

[12] I. Foster and C. Kesselman. The globus project: A status report. In *Proceedings of the 7th Heterogeneous Computing Workshop, IEEE Press*, 1998.

[13] C. Kesselman I. Foster and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 14(3), 2001.

[14] C. Kesselman (eds) I. Foster. *The Grid: Blueprint for a future computing infrastructure*. Morgan Kaufmann, 1999.

[15] I. Foster K. Czajkowski, S. Fitzgerald and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the 10th IEEE 5gmposium on High-Performance Distributed Computing*, 2001.

[16] Rajkumar Buyya Klaus Krauter and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for dis- tributed computing. *Software Practice and Experience*, 32(2), 2002.

[17] C. Liu and I. Foster. A Constraint Language Approach to Grid Resource Selection. Technical Report TR-2003-07, Department of Computer Science, University of Chicago, 2003.

[18] H. J. Siegel D. Hensgen R. Freud M. Maheswaran, S. Ali. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In *Proc. 8th Heterogeneous Computing Workshop*, 1999.

[19] Dan C. Marinescu. A grid workflow management architecture. 2003. Proposal to the GCE and the GSM Research Groups. Available at http:// www-unix.gridforum.org/ mail_archive/ gce-wg/ 2002/ Archive/ msg00402.html.

[20] S. Melody, J. Schopf, and Zhang X. Grid Searcher. In *http://people.cs.uchicago.edu/ hai/GridSearcher/overview.html*, 2002.

[21] C. E. Kim O. H. Ibarra. Heuristics algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, 24(2), 1977.

[22] P.-C. Yew P. Tang. Processor Self-Scheduling for Multiple Nested Parallel Loops. In *Proc. Intl Conf. Parallel Processing*, 1986.

[23] D. Abramson R. Buyya 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 (PDPTA 2000)*, 2000.

[24] M. Livny R. Raman and M. Solomon. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing*, 2(2), 1999.

[25] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL.*

[26] Neil Spring Rich Wolski and Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6), 1999.

[27] R.N. Uma J. Wein S. Flynn Hummel, J. Schmidt. Load-Sharing in Heterogeneous Systems via Weighted Factoring. In *Proc. Eighth Ann. ACM Symp. Parallel Algorithms and Architectures*, 1996.

[28] J. M. Schopf. General Architecture for Scheduling on the Grid. Technical Report ANL/MCS-P1000-1002, Argonne National Laboratory, 2002.

[29] M. Swany, R., and Wolski. Building performance topologies for computational grids.

[30] L.M. Ni T.H. Tzen. Trapezoidal Self-Scheduling: A Practical Scheme for Parallel Compilers. *IEEE Trans. Parallel and Distributed Systems*, 4(1), 1993.

[31] D. Thain, T. Tannenbaum, and M. Livny. *Grid Computing: Making The Global Infrastructure a Reality*, chapter 11 - Condor and the Grid, pages 299–335. John Wiley, 2003.

[32] J. Bresnahan A. Chervenak I. Foster C. Kesselman S. Meder V. Nefedova D. Quesnel W. Allcock, J. Bester and S. Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Computing*, 28(5), 2002.

[33] Rich Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. *Journal of Cluster Computing*, 1, 1998.