Coordinating Workflows in Shared Grid Environments

Jim Blythe, Yolanda Gil and Ewa Deelman

USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
blythe@isi.edu, gil@isi.edu, deelman@isi.edu

Abstract

Computational grids are characterized by widely distributed computational resources shared by virtual organizations. Users submit workflows of tasks to be executed, some of which may duplicated because of their shared application domain. The problem of allocating tasks to resources in grids has unique characteristics, governed by three features. First, users are generally interested in workflows, coordinated sets of tasks that together achieve some goal, rather than individual tasks. Good allocations of resources will optimize workflow-centered measures of performance, such as makespan and overall resource use, rather than task-centered measures. Second, the allocation process is relatively decentralized, with distributed, autonomous agents as opposed to central control. Third, the products of jobs, files, are easy to replicate and share once created, although transfer times must be taken into account.

We describe several strategies for resource allocation in Grids that differ in the amount of coordination and make an empirical analysis of their performance using a high-fidelity grid simulation environment. This analysis indicates that some coordination is essential, but the most effective strategies depend on global features such as the temporal spacing of different workflow handlers as well as structural features of workflows. We identify situations under which different strategies are appropriate.

Introduction

Computational grids are characterized by widely distributed computational resources shared by virtual organizations, from which independent users submit possibly related tasks. In general, Grid applications today are relatively complex and are not built as monolithic entities, rather they are structured as workflows that consist of individual tasks, which are applications to be run on a host machine. Tasks may have different resource requirements, for example, some may require a supercomputer or a Linux-based cluster of processors, while others can run on a workstation.

Most systems for allocating tasks on grids, such as DAGMan [DAGMan 03], currently allocate each task individually at the time it is ready to run. At this time the necessary files are sent to the host and the job is sent to its queue. We refer to this as the *task-based approach*. Since a host may become unavailable unpredictably, or other tasks may fill its queue, this highly reactive approach can be better than making the decision far in advance. However, the task-based approach is also myopic, and will lead to inefficient execution when tasks occurring later in the workflow have relatively few choices for efficient execution. We explore this in more detail in the next section.

In order to escape this, it is necessary to allocate part of the workflow in advance to be able to detect constraining tasks that will be scheduled later. We also wish to retain the reactivity of the task-based approach. If the time taken to allocate the workflow was not an issue, the workflow could be allocated in full every time a task is submitted to a host, an approach that combines reactivity with look-ahead. However, it is NP-hard to find the optimal allocation of hosts and ordering for a workflow, by a reduction from job-shop scheduling, so we cannot expect to find a complete schedule often.

In this paper we explore different approaches that combine a fast allocation algorithm, based on an approach used in job-shop scheduling, with information about other agents to reduce the number of re-allocations that are required. We limit our attention to interactions caused when multiple agents submit tasks, and ignore uncertainty from other sources. We explore the benefits of the approach within a discrete-event Grid simulator. The contributions of this paper include a framing of task allocation on Grids at the workflow level, a fast workflow allocation scheme and an investigation of communication strategies for coordinating workflow allocation.

In the next section we describe the Grid workflow allocation problem and relate it to topics in multi-agent systems. In the following section we describe the simulator and use it to show the myopic nature of task-by-task scheduling. Next we introduce an allocation algorithm based on local improvement with multiple restarts and use the simulator to explore different strategies for re-allocating based on the activities of other agents. The final sections discuss our results and related work.

Coordinating Grid Workflows

The coordination of grid workflows can be seen as a multi-agent planning problem [Durfee 01], where each agent builds a plan as it designs and submits its own workflow. Coordination among these agents is useful because they contend for the resources where the workflows are to be executed. As background and justification for the studies discussed in this paper, this section discusses typical properties of workflows for grid computing applications, the kinds of resources that they require, and the characteristic execution environment that grids provide. By understanding the specific needs of grid workflow coordination problems we can best relate them to other work on multi-agent planning, resource allocation and scheduling, and dynamic multi-agent coordination.

Grid computing provides middleware for submitting and executing complex job workflows in a distributed set of computing resources. Jobs may need to be executed in different hosts, depending on the computational requirements of the code to be executed as well as the memory requirements of its inputs or outputs. Upon the execution of a job, data may be produced as output that is required by another job as input. To make this data flow possible, workflows include not only code execution jobs but also jobs that move data from a storage resource to a host or from a host to another host. The structure of workflows is often represented as a Directed Acyclic Graph (DAG), which simplifies the underlying job scheduling mechanisms.

We refer to a grid testbed as a specific installation of a grid in a specific set of resources. Resources in grids include 1) computational resources, such as host computers where jobs can be executed, 2) network resources, such as a link connecting two computers to enable data transfer jobs, 3) storage resources, to retrieve data needed for a computation or to store data that results from a computation, 4) instruments that are manipulated through the execution of a job that indicates its setup and captures the results as data. Different grid testbeds may contain different subsets of these resource types, each with different capabilities.

Each job in a workflow has a set of resources assigned to it. For example, a job that executes code will have a host assigned to it. For a set of requirements of that given piece of code, there may be a lot of flexibility in terms of what hosts can be assigned for that job. Some resource assignments may be better than others, for example assigning a Linux cluster to execute a simple piece of code that is not parallelized is ok as long as there are no single-CPU machines that are free or as long as there are no other jobs with parallelized code waiting for execution.

Exploring the space of tradeoffs and making reasonable resource assignments for job workflows has been a focus of some recent work [Blythe et al 03]. However, resources may be scarce in some grid testbeds and fair allocation of resources to jobs is a crucial issue.

But there is another issue that affects resource allocation in grids. Suppose that two jobs that are submitted as part of two entirely different workflows generate the same data. Why execute both? More generally, a job may be submitted to generate a large data set when a significant subset of that data is already available and only a smaller portion may need to be computed from scratch. There are many possibilities to reduce the amount of redundant computation on separately submitted workflows.

An important area of research in grid computing is resource reservations and provisioning ahead of time when it is possible to anticipate a surge in the need for a particular kind of resource in a testbed. If the workflows could be designed in coordination and ahead of time, it would be possible to have good estimates of the resource requirements for a certain time horizon. This kind of resource management in grids is very important, since the organizations that create a testbed may have flexibility and willingness to make extra resources available if the requests are made in a timely and justified manner.

Coordinating the design and execution of grid workflows has three advantages: 1) enable fair assignment of computing resources, 2) avoid redundant or overlapping jobs, 3) facilitate advanced reservations and provisioning of adequate pools of resources. The rest of this paper explores these and other coordination issues in the specific context of grid workflows.

Experimental design

In the following sections we use a discrete-event simulator to explore strategies for allocating and coordinating between workflows. The simulator models hosts, files and workflows consisting of DAGs of tasks. A task requires input files and produces output files when it is run. Each host has a single dedicated queue of tasks. When a task comes to the front of the queue, it is executed as long as all its input files are present on the host. The runtime of a job is controlled by its length and the speed factor of the host, and is assumed to be deterministic and known to the allocation algorithms. When a job completes, its output files are created on the host. Files may be transferred in parallel with task execution and the speed of transfer is controlled by file size and bandwidth between the two hosts. Each workflow is managed by a handler that ensures that a job's input files are present on a host before submitting the job (typically by executing the child jobs in the workflow and transferring the files if necessary). The handler is similar to workflow managers available in grid toolkits, such as DAGMan [DAGMan 02].

In our initial experiments we modeled a variable number of handlers that are each assigned tree-shaped, three-level workflows with two separate branch parameters *b* and *c* and an index *i*. The workflow defined by these parameters is shown in Figure 1.

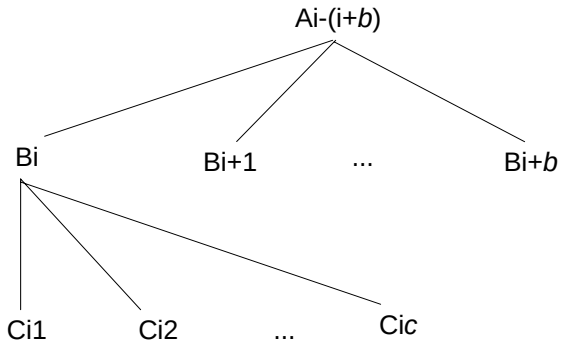

Figure 1. Schematic of parameterized workflows for initial experiments

Each workflow consists of a top-level job Ai-(i+*b*), *b* second-level jobs B*i* to B(*i*+*b*) and *bc* third-level jobs, C*ij*, $1 \le i \le b$, $1 \le j \le c$, where C*ij* is a child job of B*i*. Each job creates a unique file of the same name when run, which is a required input file for the parent job. By varying *b* and i-min – *i*-max, we can control the expected proportion of jobs that overlap between any pair of agents.

A grid is a complex environment with a large number of variables, and many aspects of grid computation will not be captured in this family of workflows. However, it reflects experiences with real-world grid applications such as LIGO [Abramovic 92] and Montage [Montage 03] in a single framework. In this model, the C*ij* tasks can be viewed as data and the B*i* tasks represent intermediate computational results. These are often shared between different workflows within a virtual organization. In many applications, such as LIGO for example, data collected by the instruments is calibrated and then processed with widely-used signal processing applications such as Fourier transforms. The calibrated and frequency data are often not the final product desired by the gravitational-wave scientists but are used in more complex analysis. Since data collected during the experiments spans observation time intervals of two to four weeks, the intermediate data products tend to span a similar time frame and are therefore common to many workflows. We have observed a similar reuse of intermediate results in the Montage application that constructs custom-mosaics of sky images on demand. One source of complexity in that application is the choice of parameters that are used to project the individual images. The projection is also the most computationally expensive part of the Montage. Usually, professional astronomers pick these parameters based on the specific hypothesis they want to explore through this visualization. It is rarely the case that individual astronomers would pick the exact same set of parameters. However, Montage is also widely used by amateur astronomers, for whom a standard parameter set is sufficient and thus provides a good source of reuse of the projected images.

Task-based allocation

We model the behavior of a task-based allocation approach such as Condor with the following algorithm: given a task *t* to allocate, consider every candidate host *h*, and estimate the time *c(h,t)* at which *h* would be able to complete *t*. Let *q(h)* be the time at which *h* will finish every task on its current queue, and let *f(t)* be the earliest time by which the files required for *t* can be transferred to *h*, and let *r(h,t)* be the estimate of the runtime of *t* on *h*. Then

$$C(h,t) = r(h,t) + \max(q(h), f(t))$$

The task-based approach selects a host *a(t)* that minimizes this completion time:

$$a(t) = \text{argmin}(h)\ c(h,t).$$

This local approach to scheduling is typical in Grid applications, but can often lead to sub-optimal decisions as illustrated by the following set of problems. Each problem is a workflow of the form shown in Figure 1 with $b = 3$ and $c = 2$. One of the files, C*ij*, has size 5 while all other files have size 1. Its consuming task, B*i*, can be run on only a subset of the hosts. When task-based allocation is used, the execution time depends on the proportion of hosts on which B*i* can run, because the producing task C*ij* will be allocated to an arbitrary host. In the next section we describe a workflow allocation algorithm that considers the global effects of task allocations on the whole workflow. This algorithm finds an optimal allocation in all instances of this problem.

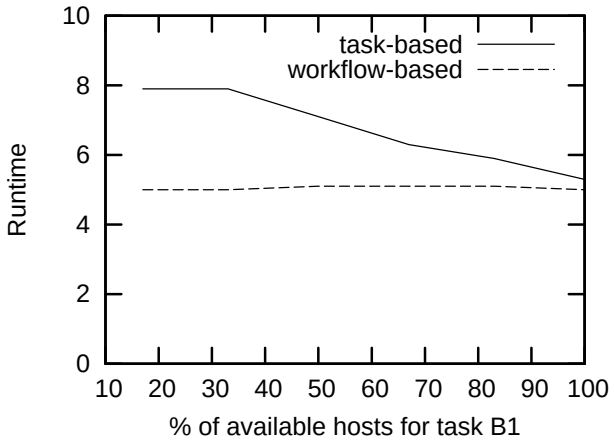

Figure 2. The average makespan found by task-based allocation for the workflow, shown with the average makespan found using the workflow allocation method. Each data point is the average of 10 trials.

Workflow allocation on the Grid

Allocating the whole workflow avoids the myopia of the task-based scheduler. However, it can be too expensive to find a global allocation that will perform well. This is an even greater problem when other agents are using hosts and creating files, since the allocation must be frequently updated. We introduce a local search algorithm for workflow allocation and show that it is fast enough to use for re-allocating workflows with up to a hundred tasks. Under the assumption that other agents are independently creating some of the files of interest, we explore how to decide when to re-allocate the workflow to maintain good performance.

Greedy randomized adaptive search for workflow allocations

To create a workflow, we follow the generalized GRASP procedure (Greedy randomized adaptive search) [Resende and Ribeiro 02], which has been shown to be effective for job-shop scheduling [Binato et al., 01]. On each iteration, an initial allocation is constructed in a greedy phase, and then a number of local modifications are considered by swapping pairs of tasks. On each pass, our initial allocation algorithm computes the tasks whose children have already been scheduled, and considers every possible host for each such task. For each (task, host) pair, the algorithm computes the increase to the current makespan of the workflow if the task is allocated to this host. Let I-min be the lowest increase found in all pairs and I-max be the largest. The algorithm picks one pair at random from those whose increase I is less than I-min + w (I-max – I-min) for some width parameter w, $0 \leq w \leq 1$. It then repeats until all tasks are allocated.

The difficulty of finding an efficient assignment will depend on such factors as the number of available hosts, the distribution of their CPU speeds, the size of the files created and the distribution of bandwidths. In our experiments, we allow file size, job length and host speed to vary according to a uniform probability distribution and initially assume that bandwidth is the same constant value between all pairs of hosts. In practice, this method finds good assignments with a time limit of 10 seconds for workflows containing up to 100 tasks on a Pentium 4 at 2.6 GHz with 512 MB RAM. Figure 2 shows the amount of CPU time taken on average to find the best workflow for a subclass of workflows for which the optimal is known. Each point in the graph is the average of 5 trials. The search time increases quadratically because the initial allocation phase is quadratic in the number of tasks. For this class of workflows, the algorithm always returns an optimal allocation and it can be seen that a time cutoff of 10 seconds is sufficient for workflows of up to 100 tasks. In the experiments that follow, the workflows submitted by individual agents contained fewer tasks.

Coordination strategies for Grid workflow allocators

Workflows submitted by different agents on a Grid may have conflicting resource requirements, and may also present an opportunity to share common tasks. The greater their autonomy, the greater the opportunity for workflow managers to minimize conflicts and to maximize the work that is shared between them. Possible strategies include re-ordering tasks to avoid waiting for a conflicted resource, re-allocating a task to a different resource or choosing a different task to achieve the same information goal. In general, however, an agent may have limited freedom due to external constraints placed on the workflow, possibly from the user. In these experiments we assume that agents may re-order tasks or re-allocate them to alternative hosts, but may not assign different tasks from those in their assigned workflow.

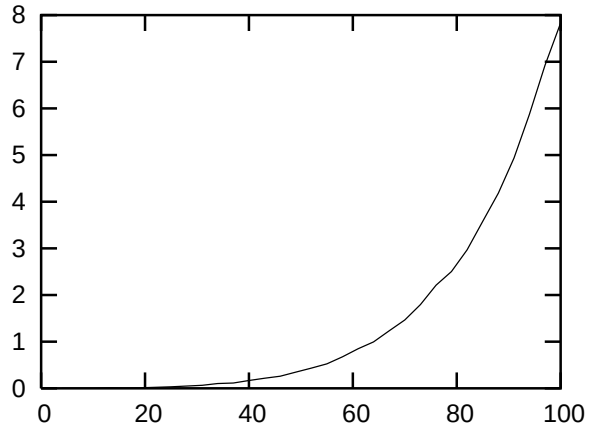

Figure 2. The time take to find the plan with shortest makespan with the GRASP algorithm in the case with uniform host speed, plotted against the number of jobs in the workflow.

In order for an agent to modify its workflow to reduce conflicts or increase synergy with another agent, it needs information about that agent's activities. It is not practical to assume a centralized control system, where the goals of all the agents and conditions of all resources are considered by one planning system, because of the widespread nature of Grids. Instead we consider what individual agents may observe in their environment.

In the simplest case, the agent might search for files that it would otherwise have to create, either locally or across the Grid, and either during the task allocation phase or continually during execution. If these files are found, the agent can remove a subset of its workflow tasks, and may re-allocate the remaining tasks to reflect the newly available resources and file locations.

While files are the results of past actions, the agent may also consider the intended future actions of other agents

and plan either to use files that it believes will be created or to avoid hosts that are likely to be congested. Once agents are aware of each other's intentions they may further negotiate where tasks of shared interest take place or who takes responsibility.

Here we explore the relative benefits of these styles of communication experimentally using a discrete event simulator that models the agents, tasks, hosts, files and transfers between hosts. One reason to do this is to gain insight into the behavior of a rich but constrained set of agents. Another reason is to explore the benefits that may come with potential modifications to Grid environments. For example, most Grid toolkits today can easily support agents looking for files via services such as the Globus toolkit's RLS [Chervenak 02]. Workflow handlers that continually monitor their environment are expected in the near future, followed by agents able to communicate with each other about their intended tasks. Reservation systems for hosts are not currently supported by most Grid toolkits, but are currently under consideration.

Coordinating via files

We first explore the improvements in the makespans when agents ignore the intended tasks of other agents but make use of existing files. We consider a number of agents submitting workflows of the type described earlier, described by width parameters *b* and *c* and by index *i*. Two such agents, with top-level tasks *Ai* and *Aj*, will have subtasks in common whenever $| i - j | \leq b$. In each trial, *n* agents are instantiated, with fixed parameters *b* and *c*, and with *i* chosen randomly with uniform distribution between *i*-min and *i*-max. By varying *b* and i-min – *i*-max, we can control the expected proportion of jobs that overlap between any pair of agents.

There is a random pause between the times that agents i and i+1 begin executing their workflows, uniformly distributed between 0 and some fixed maximum. Recall that agents submit tasks to the queue of the designated host only when all of the files required are already present on that host. This approach guarantees that the task can be executed by the host as soon as it reaches the front of the queue. It also implies that tasks will be released on the grid gradually as a workflow is executed, so that tasks from several agents are likely to be interleaved.

We consider three levels of communication between the agents. In the first case, handlers pre-allocate their workflows and do not search for files across the Grid, but will remove a task from the workflow if the file that it creates already exists on either the host where it would be created or where it would be used. In this case a common task will be duplicated unless the producing or consuming hosts for the two instances have an intersection.

In the second case, handlers delay allocation until their workflow begins execution and search the Grid for existing files that are required for their tasks. When a file is found, they may choose either to use it, transferring it from its current host, or to run the task to re-create it. The latter may be preferable if the transfer time, governed by file size and bandwidth, is high compared to the task's running time.

Figure 3 shows the cumulative runtime of the workflows submitted by these handlers over time. Their runtimes gradually decrease as more pre-existing files are found. The delayed handlers show optimal behavior after some time, since each workflow contains one task that must be executed.

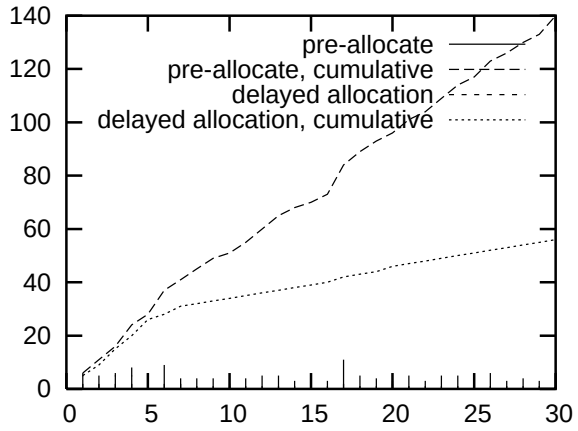

Figure 3. Cumulative runtimes for eager and delayed allocation of workflows. In each case, the workflow is allocated only once.

When several agents are submitting tasks at the same time, a file that is required by one agent may become available after it begins executing. In the third case, when an agent begins to execute its workflow it continues to monitor for files of interest. When one is found, it re-allocates all the remaining tasks in the workflow, since after the duplicated task is skipped, subsequent tasks may need to be relocated to where the file is found, or the host on which the original task was to be run may be available for other tasks.

In this experiment there is negligible improvement from monitoring for files, because typically only one or two agents are simultaneously executing workflows, and therefore very few overlap in both time and in the tasks they submit. When workflows overlap in time, however, their handlers must continue to monitor the grid after beginning execution in order to take advantage of shared tasks. We demonstrate this in the case where all the workflows of the previous example are submitted at the same time.

Figure 4 shows the extra improvement due to continuously monitoring tasks in this scenario. The size of the improvement depends on the number of workflows with overlapping tasks that are running simultaneously.

Coordinating via intended tasks

The previous section showed the benefits for agents in searching the Grid for files that were created earlier by other agents, and continually monitoring for new files. However, redundant work is still done when an agent submits a task to a queue while a similar task is queued or under execution elsewhere on the grid. This problem can be avoided if agents are allowed to inspect the workflows that have been planned but only partially executed by other handlers, using a server similar to RLS that shares workflows rather than files. The agent may choose to use a data product in place of its own task that has yet to be produced but is intended by another agent.

We tested this approach with the same set of workflows, also shown in Figure 4. The approach leads to the best runtime performance even though the agents made no re-allocations. It can be seen that coordination via sharing information about intended tasks in workflows has the potential to make grid applications more efficient, although we are not aware of work in this area currently within the grid community. Notice that task allocations built in this way may not run as predicted, since the agent posting the task may re-allocate it to a different host.

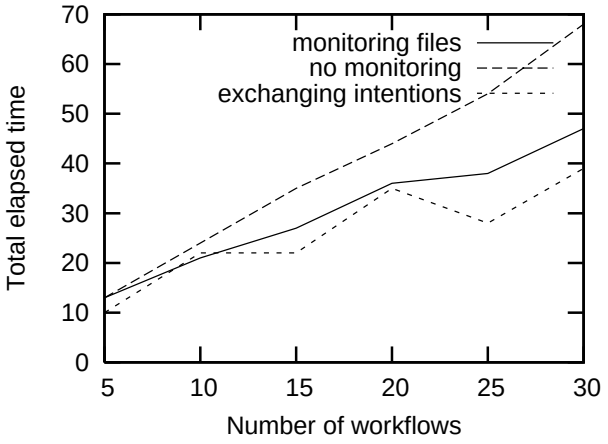

Figure 4. Elapsed time for workflows submitted simultaneously, using different coordination methods.

Related work

Bidot et al. [03] used a simulator to investigate heuristics for determining when to re-schedule a job-shop scheduling problem with uncertain activity durations. The heuristics are based on the difference between the predicted and observed makespans. Boloni and Marinescu [02] describe a general technique to help a grid workflow allocator find solutions that are more robust to discrepancies in predicted and observed runtime of the components. In contrast, our techniques rely on knowledge about the task products, and are opportunistic rather than aiming to avoid or recover from failures. Maclaren et al. [04] describe a recently started project to explore scheduling for Grids based on an expressive reservation language.

Discussion

We introduced a discrete-event grid simulator and used it to explore task-based and workflow-based allocation methods as well as different coordination strategies among distributed workflow schedulers. Task-based scheduling, where tasks are allocated to resources when they are ready to be executed and without reference to subsequent tasks, are reactive but can be myopic. Workflow-based strategies consider the entire workflow but can be expensive, especially since the allocation should be updated when new data products are produced by other workflow handlers on the grid. We showed that monitoring for specific data products on the grid can be effective in reducing the duplicated work as well as the number of re-allocations required, and that a central repository for workflows under execution can provide greater improvements.

Our approach uses a search algorithm for workflow allocations that combines randomized greedy search with local improvements. This algorithm is fast enough to support re-allocations when new files are found and provides a good global solution for moderately sized workflows. The results we have shown are equally applicable to other allocation algorithms, such as genetic algorithms or simulated annealing. The local search aspect of the algorithm seems well suited to repairing a workflow when required, rather than building a new one, and we intend to investigate this in the near future.

The simulations we showed here give an indication of the range of behaviors the approaches will exhibit. However, the precise levels of improvement or ranges of behavior are dependent on various features in the domain, such as file sizes or host speeds, and we only draw conclusions about general behaviors. We plan to complement these experiments with others that are more closely tied to real grid applications from our work.

We have recently finished the implementation of a more detailed grid simulator that more accurately models networks and distributed sites each containing several hosts. On this platform we are investigating scheduling techniques that simultaneously consider all the tasks whose children have been submitted to the Grid. These techniques form a middle ground to the approaches described in this paper, of allocating either a single task or the entire workflow, and will be compared with them in a variety of scenarios.

Acknowledgments

We are grateful to Jaskaran Singh, Karan Vahi and Anirban Mandal for discussions of scheduling approaches for Grid applications. This research was supported in part by the National Science Foundation under grant ITR-0086044 (GriPhyN).

References

A. Abramovici, W. E. Althouse, and e. al., "LIGO: The Laser Interferometer Gravitational-Wave Observatory (in Large Scale Measurements)," Science, vol. 256, pp. 325-333, 1992.

Bidot, J., Laborie, P., Beck, C. and Vidal, T., Using Simulation for Execution Monitoring and On-Line Rescheduling with Uncertain Durations, *ICAPS Workshop on Plan Execution*, 2003

S. Binato, W.J. Hery, D. Loewenstern, and M.G.C. Resende. A GRASP for job shop scheduling.In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys on metaheuristics*, pages 59-79. Kluwer Academic Publishers, 2001.

Blythe, J., Deelman, E., Gil, Y. and Kesselman, C., Transparent Grid Computing: A Knowledge-Based Approach, IAAI 2003

Boloni, L. and Marinescu, D., *Robust scheduling of meta-programs*, Journal of Scheduling, 2002

DAGMan Condor Team. The directed acyclic graph manager. http://www.cs.wisc.edu/condor/dagman, 2002.

Chervenak, A., E. Deelman, et al. Giggle: A Framework for Constructing Scalable Replica Location Services, *Proceedings of Supercomputing 2002* (SC2002), Baltimore, MD. 2002.

Durfee, E. Distributed Problem-Solving and Planning In M. Luck, V. Marik, O. Stepankova, and R. Trappl (eds.), *Multiagent Systems and Applications: Selected tutorial papers from the Ninth ECCAI Advanced Course* (ACAI 2001) and AgentLink's Third European Agent Systems Summer School (EASSS 2001), pages 118-149, Springer-Verlag Lecture Notes in AI 2086, Berlin 2001.

MacLaren , J., Sakellariou, R., Krishnakumar, K., Garibaldi, J. Ouelhadj, D. Towards Service Level Agreement Based Scheduling on the Grid, *ICAPS 04 workshop on planning and scheduling for web and grid services*, 2004

Montage 2003, http://montage.ipac.caltech.edu/

Resende, M. and Ribeiro, C, Greedy Randomized Adaptive Search Procedures, State-of-the-art Handbook in Metaheuristics, Glover and Kochenberger, eds., Kluwer Academic Publishers, 2002