Planning for workflow construction and maintenance on the Grid

Jim Blythe, Ewa Deelman, Yolanda Gil

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

Abstract

We describe an implemented grid planner that has been used to compose workflows and schedule tasks on a computational Grid to solve scientific problems. We then discuss two issues that will demand further attention to make Grid and web service planners a reality. First, the planner must interact not only with external services that are to be composed in the final workflow, but also with external reasoners or knowledge bases containing information that is needed for the planning task, for example resource constraints and policies. Second, the planning system must provide for monitoring and re-planning strategies in order to manage the execution of a workflow in a dynamic environment.

Introduction

Grid computing (Foster & Kesselman 99, Foster et al. 01) promises users the ability to harness the power of large numbers of heterogeneous, distributed resources: computing resources, data storage systems, instruments etc. The vision is to enable users and applications to seamlessly access these resources to solve complex large-scale problems. Scientific communities ranging from high-energy physics (GriPhyN 02), gravitational-wave physics (Deelman et al. 02), geophysics (SCEC 02), astronomy (Annis et al. 02), to bioinformatics (NPACI 02) are embracing Grid computing to manage and process large data sets, execute scientific simulations and share both data and computing resources. Scientific, data intensive applications, such as those outlined above are no longer being developed as monolithic programs. Instead, standalone application components are combined to process the data in various ways. The applications can now be viewed as complex workflows, which consist of various transformations performed on the data. For example, in astronomy, workflows with thousands of tasks need to be executed during the identification of galaxy clusters within the Sloan Digital Sky Survey (Annis et al. 02). Because of the large amounts of computation and data involved, these workflows require the power of the Grid to execute efficiently.

The goal of our work is to automate this workflow generation process as much as possible. Ideally, a user should be able to request data by simply submitting an application-level description of the desired data product. The Grid infrastructure should then be able to generate a workflow by selecting appropriate application components, assigning the required computing resources and overseeing the successful execution. This mapping should be optimized based on criteria such as performance, reliability and resource use. We cast workflow generation as a planning problem, where the goals are the desired data products and the operators are the application components.

The application of planning to the Grid has much in common with web services planning, with its emphasis on composing tasks that are executed on shared, distributed resources, and there is also convergence on the representation of services (Foster et al. 02). Some differences from other web service planning approaches may be found in our current emphasis on Grid applications for scientific and high-performance computing, so that a set of hosts within a virtual organization may be available to execute a component application, and users are highly motivated to provide correct models of these components. In addition, we use the middleware infrastructure provided in Grid environments such as Globus (Globus 02) for resource and data discovery and for execution support.

In the next section we give a short overview of the Grid environment that we use and describe an example application that has driven our work. Next we describe how we have modeled workflow construction for this application as a planning problem, focusing on the integration of the planner with the Grid. In the following section we discuss how the distributed nature of task and service knowledge on the Grid and the web impacts the design of planners for these environments. Next we discuss approaches for allocating hosts to workflows on the grid, when host availability may change during the execution of the workflow.

Overview of Grid environments

Grid environments such as Globus include middleware services that enable users to obtain information about the available resources, component software, data files, and the execution environment. This section describes several of these services, which we have used as sources

of knowledge for our system. More details can be found in (Deelman et al 03a; Deelman et al 03b).

In Grid environments, an application component (e.g., a Fast Fourier Transform or FFT) can be implemented in different source files, each compiled to run in a different type of target architecture. Both executable files and data files can be replicated in various locations, helping to reduce execution time. Each file has a description of its contents in terms of application-specific metadata. A distinction is made between "logical" file descriptions, which uniquely identify the application component or data, and "physical" file descriptions, which in addition uniquely specify the location and name of a specific file. The *Metadata Catalog Service (MCS)* (Chervenak et al. 02b) responds to queries based on application-specific metadata and returns the logical names of files containing the required data, if they already exist. Given a logical file name that uniquely identifies a file without specifying a location, the *Replica Location Service (RLS)* (Chervenak et al. 02a) can be used to find physical locations for the file on the Grid.

The Grid execution environment includes computing and storage resources with diverse capabilities. A specific application may require (possibly indicated in its metadata description) a certain type of resource for execution, for example the existence of certain number of nodes to efficiently parallelize its execution. Executing applications with minimal overhead may require specifying which of the job queues available in the host is more appropriate. The *Monitoring and Discovery service (MDS)* (Czajkowski et al 01) allows the discovery and monitoring of resources on the Grid. Resource matchmakers find resources appropriate to the requirements of application components.

Jobs that are completely specified for execution are sent to schedulers that manage the resources and monitor execution progress. Condor-G and DAGMan (Frey et al. 01) can be used to request a task to be executed on a resource. Condor-G adds an individual task to a resource's queue, while DAGMan can manage the execution of a partially-ordered workflow by waiting for a task's parents to be completed before scheduling a task.

Given that the planning system decides for the user where to generate the data and what software and input files to use, it is very important to provide the user and others accessing this derived data with its provenance information, or how the data arrived at its current form. To achieve this, we have integrated our system with the Chimera Virtual Data System (Annis et al. 02). Our system generates a Virtual Data Language description of the products produced in the workflow. Chimera uses this description to populate its database with the relevant provenance information.

Scenario: LIGO pulsar search

We have applied this approach in the context of the Laser Interferometer Gravitational Wave Observatory (LIGO), and we use this application to illustrate the work. We have focused on a specific LIGO problem: pulsar search, shown in Figure 2, where Grid resources are required to search for evidence of gravitational waves possibly emitted by pulsars. The data needed to conduct the search is a long sequence (~4 months, $2x10^{11}$ points) of a single channel—the gravitational wave strain channel observed at the LIGO instrument. The output from the observatory is in small segments of many channels that are stacked to make a large frequency-time image, perhaps $4x10^5$ on each side. The pulsar search looks for coherent signals in this image.

The pulsar search is both computation and data intensive and requires more resources than those available within the LIGO Scientific Collaboration. In order to take advantage of the Grid resources, LIGO's existing analysis tools were integrated into the Grid environment. The pulsar search conducted at SC 2002 used LIGO's data collected during the first scientific run of the instrument and targeted a set of 1000 locations of known pulsars as well as random locations in the sky. The results of the analysis were made available to LIGO scientists through the Grid.

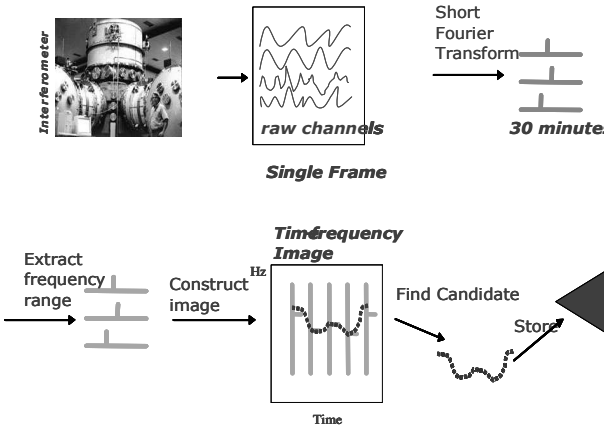

Figure 2: The path of computations required for a pulsar search.

Modeling the task as a planning problem

The problem of assigning a set of coordinated tasks in a workflow and allocating the tasks to available resources is formulated as an AI planning problem as follows. Each application component that may take part in the workflow is modeled as a planning operator. The effects and preconditions of the operators reflect two sources of information: data dependencies between the program inputs and outputs, and resource constraints on the programs, both hardware and software required to run them.

The planner imposes a partial order on the tasks that is sufficient for execution because it models their input and output data dependencies: if the prerequisites of a task are completed before the task is scheduled, then the

information required to run the associated program will be available. Transferring files across the network is also modeled with a planning operator, so any data movement required is accounted for. The data dependencies between tasks are modeled both in terms of metadata descriptions of the information and in terms of files used to represent the data in the system. Metadata descriptions, for example a first-order logic predicate that denotes the result of a pulsar search in a fixed point in the sky across a fixed range of frequencies and at a fixed time, allow the user to specify requests for information without specific knowledge of programs or file systems, with the planner filling in the details of the request. Since the operators also model the files that are created and used by a program, the planner knows how the information is accessed and stored. It can then reason about how tasks can share information, and plan for moving information about the network. The planner's representation of information and files is kept up to date with the state of the Grid, so that its plans are both efficient and directly executable, as we describe below.

The planning operators also model constraints on the resources required to perform the desired operations. Hardware constraints may include a particular machine type, minimum physical memory or hard disk space available, or the presence of a certain number of nodes in a distributed-memory cluster. Software constraints may include the operating system and version, the presence of scheduling software on the host machine and the presence of support environments, for example to run Java. In our work on the LIGO scenario, it was sufficient to model requirements on the scheduling software present, because of the close relationship between the software and the hardware configurations of the machines involved. More recently, we have described different hardware requirements on operators using the CIM model (CIM, 02).
The initial state given as input to the planner captures information from several sources:

1. Hardware resources available to the user described using the CIM ontology, and estimates of bandwidths between the resources.
2. Relevant data files that have already been created and their locations.

Our aim is for this information to be extracted automatically. At present, some is automatically extracted and some is hand-coded, as we describe below.

The goal given to the planner usually represents a meta-data request for information and a location on the network where the data should be available. If there is a preference, the goals can also be used to specify programs or host machines to be used, for intermediate or final steps.

In addition to operators, an initial state and goals, our implemented workflow planner also uses search control rules, to help it quickly find good solutions based on preferences for resources and component operators, and to help it search the space of all plans more efficiently in order to find high-quality plans given more search time. For more details on the planning domain specification and its implementation, see (Blythe et al. 03a; Blythe et al. 03b).

Integration with the Grid environment

We now describe in more detail the integration of the planning system with existing Grid services. The use of the planner can be divided into three phases: preparing the input problem specification for the planner, practical considerations for using AI planning in this problem domain, and interpreting the output plan as an executable workflow.

Two modules shown in Figure 3 provide input for the planner: the Current State Generator, which produces the initial state description, and the Request Manager, which produces the goal description from a user request. The Current State Generator makes use of two tools that have been independently built for the Grid: the Metadata Catalog Service and the Replica Location Service.

Given knowledge of which data products already exist and where they are located, the planner can choose whether to transfer existing data across the network or re-create it closer to the point where it is needed. This choice is made either by search control heuristics or by simulating a number of plans and picking the one with the best expected run time.

An important design decision in our current implementation was whether to encode information about all possible required data products in the initial state before planning begins, or allow the planner to query for the existence of data products while planning. Although the planner is capable of making these queries, we chose to gather the information before planning because the potentially large number of queries about files can then be combined, reducing bandwidth and the load on the MCS and RLS. The data products to be queried are decided by the Current State Generator based on the goal description. This could be done through a static analysis of the planning operators, but is currently hard-coded.

Once the file information is retrieved, it is sent to the AI planner, along with the goal from the Request Manager. The planner merges this information with a static file describing available resources to create the initial state and goals used for planning. Our aim in the near future is to use the Globus Monitoring and Discovery Service to retrieve information about computer hosts, rather than use a static file, and also to use the Network Weather Service (Wolski 97) to retrieve timely information about bandwidth estimates between resources. We also intend to use a metadata service to retrieve information about Grid users including their preferences and their access to resources.

The planning operators are stored separately. We are currently investigating ways to generate the operators from metadata and resource information about the application components.

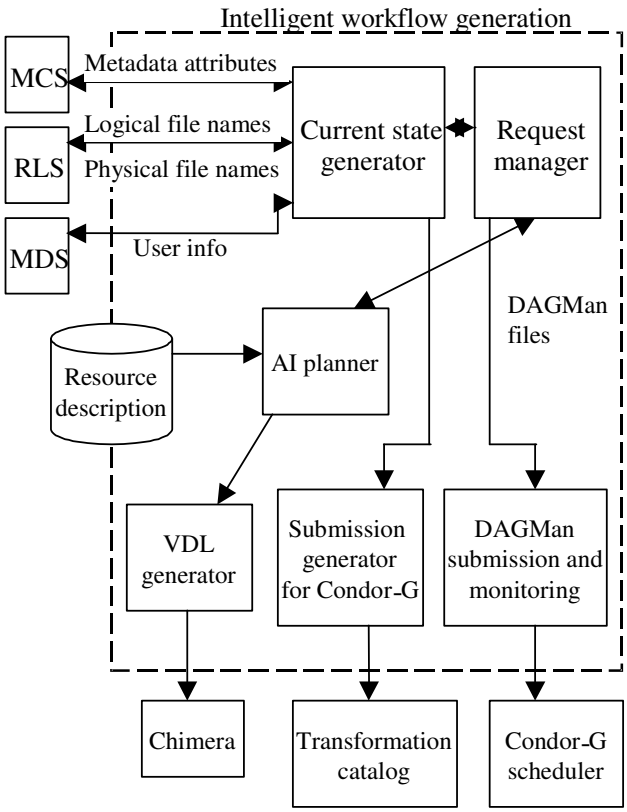

Figure 3: Architecture of the planning system and its interactions with other Grid-based services.

Synthesizing planning knowledge from distributed sources

Planning systems have typically made the assumption that all the required information is stored in local files or memory throughout the duration of the planning task. Examples of systems that relax this assumption include those that reason about uncertainty (Boutilier et al. 99, Blythe 99) or work in teams and communicate with their peers (Tambe et al. 99). Similar information gathering in web services was explored in the Optop planner (McDermott 02). In these cases, however, the action models are assumed to be fixed, as are the domain modeling primitives for state descriptions, goals and plan metrics. Web service and Grid planners that integrate resource discovery must work with a set of operators that may change as resources are discovered or become unavailable. We expect that domain modeling primitives will also be dynamic when these planners reason with distributed sources of knowledge about constraints and preferences associated with tasks, resources and users.

The impact that this will have on planners has been little discussed in the literature to this point. However it will have a significant effect both on architectural choices for web or grid enabled planning systems and also on appropriate planning algorithms and the guarantees they can provide. Here we briefly discuss why and how planning knowledge may be distributed, and the issues that are presented for planning systems.

Relevant knowledge for a planning episode, including constraints and preferences as well as object-level information, is likely to be stored at several different locations based both on the distributed execution environment and on the different kinds of knowledge being stored. On the grid, for example, services that allow a task to be scheduled on a host machine are independent of the task that is to be run: rather, they accept and schedule arbitrary tasks as long as the executable can be addressed. Thus, in order for a planning system to reason about a task allocated to a host using a single operator, information from at least two separate sources, about tasks and about hosts, must be combined to produce the operator description. While the planner could in principle enter a dialog with each host to decide where each task can be allocated, it will be necessary for the planner to reason declaratively about the host constraints to perform efficiently. Figure 4 describes other sources of knowledge that are relevant during the process of composing and executing a workflow.

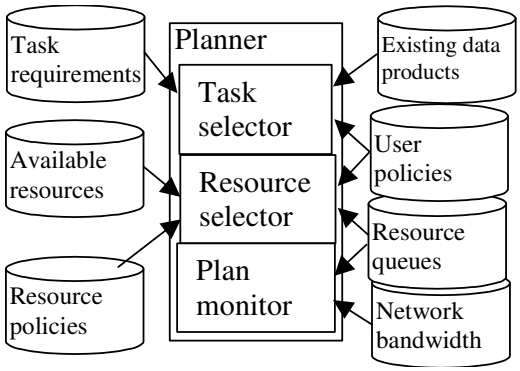

Figure 4: Sources of knowledge that must be combined by a workflow planner and execution monitor.

Knowledge about host usage policies will be stored by each available host and made available through a discovery process. Currently, most policy information on the Grid is typically not codified, but informally known to users and built implicitly into scripts or other representations that are used to compose tasks. In our initial working LIGO application, for example, available hosts are declared by the user and automatically added to the planner's initial state when the planner starts.

Resource policies, once encoded, are likely to be represented in a constraint or rule-based format that is independent of any particular planning system, *e.g.* (Tangmunarunkit et al. 03). There are two ways a planning component could handle knowledge represented in such a format. First, it may be possible to re-write the rules or constraints into the planner's internal format. For

example, suppose we have a collection of rules, each of which can conclude separately that a user can access a resource. Each could be represented as an operator that adds this conclusion, and the conclusion could be made a precondition of all operators that assign tasks to resources. This approach clearly requires a mapping from the semantics of the rule or constraint representation into the planner's reasoning capabilities. This mapping must not only adequately capture the inferential power of the representation, but must also allow any object-level information that is accessed by the policy reasoner to be represented in the planner. If this can be done, however, it has the advantage that the planner can reason about aspects of policy as they arise in the context of the problem, propagating and reformulating constraints as needed.

An alternative approach is to have the planner access an external knowledge service that answers queries based on the policy rules. This approach has the advantage that no mapping is required, so the planner can make use of a knowledge-based policy system even if we are ignorant of the representation and inference schemes that it uses. However, this approach may have efficiency problems because (1) the planner may need to post a large number of queries as it performs search and (2) the planner may rely on regression techniques, for example, that are not supported by such a query interface.

We intend to explore these two approaches for integrating planners with heterogeneous reasoners and knowledge bases distributed in a Grid or on the web. It is quite likely that a combination of both approaches will be used. A large body of work in the semantic web (Berners-Lee et al. 01) aims to allow knowledge to be shared between different reasoners, and representations to share rule-based knowledge and operator knowledge are becoming well used (Ankolekar et al. 01), however the implications of these approaches for planning systems has not been studied.

Workflow scheduling and maintenance

A complete planning-based solution for Grid or web services composition must address execution as well as construction of workflows. Grid and web planning systems make decisions in dynamic environments in which the services that are composed as part of a plan may become unavailable during its execution. Clearly, a successful workflow system must include a plan monitoring component, that checks the availability of a service on a host when or before a task is due to be run on it, and re-submits to an alternative host if necessary. Two metrics of workflow execution are the reliability of the workflow, measured for example as the average number of times a job fails and must be re-submitted, and the expected execution time. In this section we first discuss processor allocation in static environments, and then extend to dynamic environments.

Finding an optimal allocation of processors for tasks in a workflow is NP-hard (Papadimitriou and Yannakis 90) and tools must focus on finding reasonable heuristics or on identifying families of problems that can be solved efficiently. On many current Grid-based systems, an abstract workflow is first created as a DAG whose nodes represent tasks, with a directed edge from task *T1* to *T2* if *T1* must be completed before *T2* begins. No allocation of tasks to machines is made in the abstract DAG, however, and a separate service, for example DAGMan (Frey et al. 01) is used to schedule tasks on machines, making a local decision for each task as it becomes ready to be executed. The default rule in DAGMan is greedy, scheduling the most expensive task on the best available processor at each decision point.

This is a reasonable default, but the local decision rule can lead to arbitrarily poor assignments of tasks. Consider the abstract DAG shown in Figure 5, with three tasks. Task A and B can be started immediately, and task C can be started after task B ends. Task A will take 3 time units, task B will take 1 and task C will take 5. Suppose there are two processors available, one (called *fast*) that takes the times as shown to make a computation and one (called *slow*) that takes twice the time. Using the default local rule, a scheduler would allocate task A to processor *fast* and tasks B and C to processor *slow*. This allocation will take 12 time units to complete. The optimal allocation uses processor *fast* for tasks B and C and takes 6 time units. An identical argument can be made to show that one must reason about the whole DAG in order to allocate processors to optimize for reliability, if each processor has a different, stationary probability of failure over a given time interval.

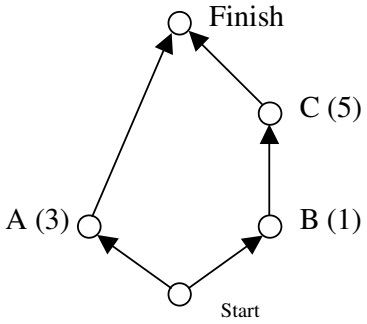

Figure 5. An abstract DAG showing three tasks and their execution times. Arrows point upwards, meaning that lower tasks must be completed before higher tasks in the DAG are started.

The above argument makes use of the parallelism in the DAG. Even if a workflow is completely serialized, however, a greedy allocation may be poor if the tasks are coupled in other ways. For example, if the tasks will be performed in order A, B, C and a user can use at most 5 time units on the fast processor, a greedy scheduler might allocate the fast processor to tasks A and B, leaving task C to the slow processor and resulting in a total execution time of 14, while the best is 13. In a more faithful model,

where each task can only be run on a subset of the available machines, the greedy approach may lead the scheduler to fail to complete a workflow that can be completed within the resource limitations. When we also consider the amount of data that needs to be transferred between tasks, bandwidth differences create dependencies between machines that lead to similar non-local effects. Finally, all these considerations are made for a single DAG, however there may be alternative compositions of services to achieve the same goal, leading to multiple possible DAGs, any of which might lead to the optimal allocation.

Exhaustive search would of course find the optimal allocation for problem descriptions that model machines with different speeds that can accept different subsets of tasks and have different user access polices and network connections with different bandwidths. However, this can quickly become intractable even in a static environment where host availability and idle times do not change. In a dynamic environment, the system needs to make time-dependent decisions when an initial workflow allocation is found to be inappropriate during execution. For this reason, a two-phase approach has been widely used, in which an abstract allocation of some sort is created before execution and used to guide online processor allocation during execution.

Within this framework, many tradeoffs are possible between the amount of time spent in each of the phases, and the way that the initial structure guides online allocation decisions. We have already discussed the use of DAGMan with abstract workflows, which is an example of this approach. Another example is provided by clustering techniques in processor allocation in multiprocessors (Liou and Palis 97).

In this approach, before execution it is assumed that tasks can run on any host, that there is a constant finite bandwidth between any two different hosts and that all hosts are equally fast. These assumptions work well for allocating tasks on a multiprocessor, for which the approach was developed. It is also assumed that the number of available processors is unknown before execution. The initial structure is a clustering of tasks in the DAG, such that tasks in the same cluster will be performed on the same processor. For example in the DAG in Figure 5, with a non-zero communication cost between Tasks B and C, the optimal clustering will be {A} and {B, C}.

When the DAG is to be executed, processors are allocated to clusters rather than individual tasks. If there are more clusters than processors, a greedy approach can be used to merge clusters attempting to balance processor load or communication traffic. For the DAG in Figure 5 and the two processors discussed earlier, a greedy allocation method would assign the cluster containing tasks B and C to the fast processor, yielding the optimal allocation. In general, it is NP-hard to find an optimal clustering, so heuristics are used. The optimal clustering may not lead to the optimal processor allocation, since constraints on processors and non-uniform bandwidths have been ignored until the time of execution, when decisions are made locally.

Clustering is an interesting approach to dividing the computational effort between reasoning before execution and reasoning during execution. Essentially it ignores availability of non-uniform processors before execution, but considers the DAG's critical path and task communication requirements. We plan to further explore the tradeoffs in the division of labor in a two-phase planning and processor allocation approach. For example, in *full-ahead* planning, the structure created before execution is a fully specified candidate plan, created by our planner following local heuristics and trying as many alternative global solutions as time allows. At execution time, the processor specified for a task is used if available, and a greedy substitution is made otherwise. This approach has the advantage that it reasons about the whole workflow to allocate processors and the concrete plan is very fast to use if it is still appropriate. However, re-planning is likely to be required more often than with more abstract initial structures. An interesting problem is how to compile the dependencies that might be found between tasks during global reasoning into constraints that can be applied locally when re-planning is done.

A similar tradeoff between initial plan structures and execution is discussed in (Jonsson et al. 00) for the RAX system. Their theoretical treatment is very general, however in the implemented system, the initial structure is completely specified except for temporal intervals, leading to an approach similar to full-ahead planning.

Conclusions

Planning for grid applications has many similarities with web services planning as well as widely-used middleware infrastructures such as Globus. Our implemented system for planning on the grid offers a platform grounded in real daily applications that can be used to explore the decision space for both architecture and algorithms for planning systems on the grid. In this paper we described two key issues that need to be addressed in the near future: interaction of planners with other reasoning KBs or services, and workflow monitoring and maintenance on the Grid. We are interested both in developing sound principles for web and grid planners and in putting useful AI-based systems in the hands of grid users, and look forward to many other challenges in this exciting area.

Acknowledgements

We gratefully acknowledge helpful discussions on these topics with our colleagues, including Carl Kesselman, Hongsuda Tangmunarunkit and Karan Vahi. This research was supported in part by the National Science Foundation under grants ITR-0086044 (GriPhyN) and EAR-0122464 (SCEC/ITR), and in part by an internal grant from USC's Information Sciences Institute.

References

J. Annis, Y. Zhao, J. Voeckler, M. Wilde, S. Kent, and I. Foster, "Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey," Technical Report GriPhyN-2002-05, 2002.

Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., Zeng, H., DAML-S: Semantic Markup for Web Services, Proceedings of the International Semantic Web Working Symposium (ISWWS), 2001.

Annis, J., Zhao, Y., Voeckler, J., Wilde, M., Kent, S. and Foster, I., Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey. in *Supercomputing*. 2002. Baltimore, MD.

Berners-Lee, T., James Hendler and Ora Lassila. "The Semantic Web" *Scientific American*, May 2001.

Blythe, J. "Decision-Theoretic Planning," *AI Magazine*, vol. 20, 1999.

Blythe, J., Deelman, E., Gil, Y., Kesselman, C., Agarwal, A., Mehta, G., Vahi, K., The Role of Planning in Grid Computing, in Proc. *Intl. Conf. on AI Planning and Scheduling*, (ICAPS) 2003

Blythe, J., Deelman, E., Gil, Y., Kesselman, C. Transparent Grid Computing: a Knowledge-Based Approach, *Proc. Innovative Applications of Artificial Intelligence Conference* (IAA) 2003

Boutilier, C., Dean, T., Hanks, S., Decision-Theoretic Planning: Structural Assumptions and Computational Leverage, *Journal of Artificial Intelligence Research*, 11, 1-94, 1999.

Chervenak, A., E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripenu, B. Schwartzkopf, H. Stockinger, K. Stockinger, B. Tierney (2002). Giggle: A Framework for Constructing Scalable Replica Location Services. in *Supercomputing*. 2002. Baltimore, MD.

CIM, 2002, http://www.dmtf.org/standards/standard_cim.php

Chervenak, A., Deelman, E., Kesselman, C., Pearlman, L. and Singh, G., A Metadata Catalog Service for Data Intensive Applications. 2002, *GriPhyN technical report*, 2002-11.

Czajkowski, K., Fitzgerald, S., Foster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing. in *10th IEEE International Symposium on High Performance Distributed Computing*. 2001: IEEE Press.

Deelman, E., J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments", *Journal of Grid Computing*, vol. 1, 2003

E. Deelman, K. Blackburn, P. Ehrens, C. Kesselman, S. Koranda, A. Lazzarini, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, and R. Williams., "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists," *11th Intl Symposium on High Performance Distributed Computing*, 2002.

Deelman, E., et al., From Metadata to Execution on the Grid: The Pegasus Pulsar Search. 2003, GriPhyN 2003-15.

I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999.

I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of High Performance Computing Applications*, vol. 15, pp. 200-222, 2001

Foster, I., Kesselman, C., Nick, J., Truecke, S., The physiology of the grid: an open grid services architecture for distributed systems integration. Open Grid Service Infrastructure WG, Global Grid Forum, 2002

Frey, J., Tannenbaum, T., Foster, I., Livny, M., Tuecke, S., Condor-G: A Computation Management Agent for Multi-Institutional Grids. in *10th International Symposium on High Performance Distributed Computing*. 2001: IEEE Press.

Globus, 2002 www.globus.org

GriPhyN 2002, www.griphyn.org.

Jonsson, A., Morris, P., Muscettola, N., Rajan, K., Smith, B., Planning in interplanetary space: theory and practice, *International Conference on Artificial Intelligence Planning Systems*, (AIPS) 2000

Liou, J. and Palis, M. A comparison of general approaches to multiprocessor scheduling, *International Parallel Processing Symposium* (IPPS) 1997

McDermott, D. "Estimated-Regression Planning for Interactions with Web Services". in *Sixth International Conference on Artificial Intelligence Planning Systems*, (AIPS) 2002.

McIlraith, S. and Son, T., "Adapting Golog for Composition of Semantic Web Services". in *Eighth International Conference on Knowledge Representation and Reasoning,* 2002

NPACI 2002, "Telescience, https://gridport.npaci.edu/Telescience/."

Papadimitriou, C. and Yannakis, M., Towards an architecture-independent analysis of parallel algorithms, SIAM J. on Computing, 19:2 (1990), 322-328

Tambe, M., Adibi, J., Alonaizon, Y., Erdem, A., Kaminka, G., Marsella, S. and Muslea, I. Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence, volume 110, pages 215-240, 1999*.

SCEC 2002. Southern California Earthquake Center's Community Modeling Environment, http://www.scec.org/cme/.

Tangmunarunkit, H., Decker, S., Kesselman, C. Ontology-based Resource Matching---The Grid meets the Semantic Web. *Semantics in Peer2Peer and Grid Computing Workshop*, *International World Wide Web Conference*, 2003.