Planning and Metadata on the Computational Grid

Jim Blythe, Ewa Deelman, Yolanda Gil

USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina Del Rey, CA 90292
{blythe, deelman, gil}@isi.edu

Abstract

Grid computing provides key infrastructure for distributed problem solving in dynamic virtual organizations. It has been adopted by many scientific projects, and industrial interest is rising rapidly. However, grids are still the domain of a few highly trained programmers with expertise in networking, high-performance computing, and operating systems. We have been working in capturing knowledge and heuristics about how to select application components and computing resources, and using that knowledge to generate automatically executable job workflows for a grid. Our system is implemented and integrated with a grid environment where it has generated dozens of workflows with hundreds of jobs in real time. In order to be applicable to a wide range of existing and new grid applications, the planner needs to be able to work with varying levels of semantic information for processes and the information they consume and create. We discuss our experiences dealing with different levels of data and describe a planning-based system that can provide different levels of support based in the information available.

Introduction

Once the realm of high-performance computing for scientific applications, grid computing is arising as key enabling infrastructure for resource sharing and coordinated problem solving in dynamic multi-institutional virtual organizations (Foster et al. 01). Grids build over networking technology to provide middleware components such as locating files over a network of computers, scheduling the distributed execution of jobs, and managing resource sharing and access policies (Foster and Kesselman 99). The need of scientific communities to interconnect applications, data, expertise, and computing resources is shared by other application areas, such as business, government, medical care, and education (Foster et al. 02, Waldrop 03).

Unfortunately, grid computing is today far from reach from regular computer users. Users interact with grids by sending a specification in the form of a detailed executable script of which jobs should be run on which computers using which physical files and sometimes the specific scheduler in the host computer where jobs are to be submitted for execution. We believe that this need not be so. Our work aims to develop intelligent middleware components that encapsulate the expertise required to use grids. In (Blythe et al. 03a), we outlined the use of AI planning techniques to automatically generate executable job workflows from high-level specifications of desired results. In (Blythe et al. 03b) we described the knowledge and heuristics used in the system and how the planner is integrated within a grid environment to extract relevant knowledge from existing grid middleware.

Although grid environments form the application domain for this work, it has more general applications for composition of web services. Indeed, many grid toolkits are now adopting the Open Grid Services Infrastructure (Foster et al. 02) in which grid services are versions of web services. The issues of semantic descriptions of component and data information used in planners discussed in this paper are highly relevant in both domains.

One of our application domains is the Laser Interferometer Gravitational Wave Observatory (LIGO, www.ligo.caltech.edu) aimed at detecting gravitational waves predicted by Einstein's theory of relativity. The planner can be given the high-level goal of making a pulsar search in certain areas of the sky for a time period. It has been used to generate workflows for 58 pulsar searches, scheduling over 330 jobs and over 460 data transfers, consuming over eleven CPU hours on resources distributed over the Grid.

Our solution for LIGO used semantic characterizations of the processes in the workflow and of the files they require and produce in order to compose the workflow automatically. However, some of the benefits of AI planning and scheduling can be realised in domains where such characterizations are not yet available. Moreover, greater improvements in efficiency are possible with semantic characterizations of information that can describe related groups of files, or that can be realized by any one of a group of files. In this paper we explore these benefits and describe a planning-based solution that can provide incremental payoff as the level of semantic information is increased about a problem domain on the grid.

Below we outline some of the challenges of using the Grid today and the benefits of our approach. We then show the kinds of knowledge available in current Grid infrastructure and capabilities, using as an example the specific Grid environment of the LIGO application. We describe the

Copyright © 2003, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

system that we have developed, how it extracts the knowledge available in the current Grid and uses it to generate complete executable workflows, and how the workflows are translated into the scripting language required by the Grid environment. We finish with a discussion of our future work and the potential of AI technology for Grid computing.

Motivation

Scientists often seek specific *data products*, which can be obtained by configuring available *application components* and executing them on the Grid. As an example, suppose that the user's goal is to obtain a frequency spectrum of a signal S from instrument Y and time frame X, placing the results in location L. In addition to these stated desired results, the user may have additional requirements on intermediate steps. For example, the user may want the results of any intermediate filtering steps to be available in location I, perhaps to examine the filter results to check for unusual phenomena.

Today, users have to transform this kind of high-level requirement into a workflow of jobs that can be submitted for execution on the Grid. Each job must specify which files contain the code to be run, selected by mapping the high level requirements above to available application components (e.g., a Fast Fourier Transform coded by Caltech's group, version 3.0 or higher) and selecting a physical file from the many available replicas of the code in various locations. The job also specifies the location (or host) where it should be run, based on the code requirements (e.g., code is compiled for MPI, parallelized to run on tightly-coupled architecture, preferably with more than 5 nodes) and on user access policies to computing and storage resources. An executable workflow also includes jobs to move input data and application component files to the execution location.

Although Grid middleware allows for discovery of the available resources and of the locations of the replicated data, users are currently responsible for carrying out all of these steps manually. There are several reasons why automating this process is not only desirable but necessary:

1. **Usability:** Users are required to have extensive knowledge of the Grid computing environment and its middleware functions. For example, the user needs to query the Replica Location Service (RLS) (Chervenak et al. 02a) to find the physical locations of input data files.
2. **Complexity:** In addition to requiring scientists to become Grid-enabled users, the process may be complex and time consuming. The user must make many choices when alternative application components, files, or locations are available. The user may reach a dead end where no solution can be found, which would require backtracking to undo some previous choice.
3. **Solution cost:** Lower cost solutions are highly desirable in light of the high cost of some of the computations, the user's limitations in terms of resource access and bandwidth variability. Because finding any feasible solution is already time consuming, users are unlikely to explore alternative workflows that may reduce execution cost.
4. **Global cost:** Because many users are competing for resources, it is desirable to minimize cost within a community. This requires reasoning about individual user's choices in light of other user's choices, such as possible common jobs that could be included across workflows and executed only once. In addition, policies that limit user's access to resources should be taken into account in order to accommodate as many users as possible while they are contending for limited resources.
5. **Reliability of execution:** In today's Grid framework, when the execution of a job fails the job is resubmitted for execution on the same resources. Better recovery mechanisms would be desirable that consider alternative resources and components as the grid changes.

While addressing the first three points would enable wider accessibility of the Grid to users, the latter two simply cannot be handled by individual users and will likely need to be addressed at the architecture level.

Our approach is twofold. First, we use *declarative representations of knowledge* involved in each choice of the workflow generation process. This includes knowledge about how application components work, characteristics and availability of files, capabilities of the resources available, access control policies, etc. Second, this *knowledge is uniformly available to the system* at any point during workflow generation. This allows the system to make decisions and assignments in a flexible manner that

- takes into account previous and future choices, searching for a low-cost workflow configuration that satisfies the requirements from the user,
- is feasible in the given execution environment, and
- can adapt to changes in the overall system state.

Figure 1 illustrates our approach. Users provide high level specifications of desired results, as well as constraints on the components and resources to be used. These requests and preferences are represented in the knowledge base. The Grid environment contains middleware to find components that can generate desired results, the input data that they require, to find replicas of component files in specific locations, to match component requirements with resources available, etc. The knowledge currently used by Grid middleware (resource descriptions, metadata catalogs to describe file contents, user access rights and use policies, etc) would also be incorporated in the knowledge base. The system would generate workflows that have iteratively add details to the workflow based on the execution of the initial portions of it and the current state of the execution environment.

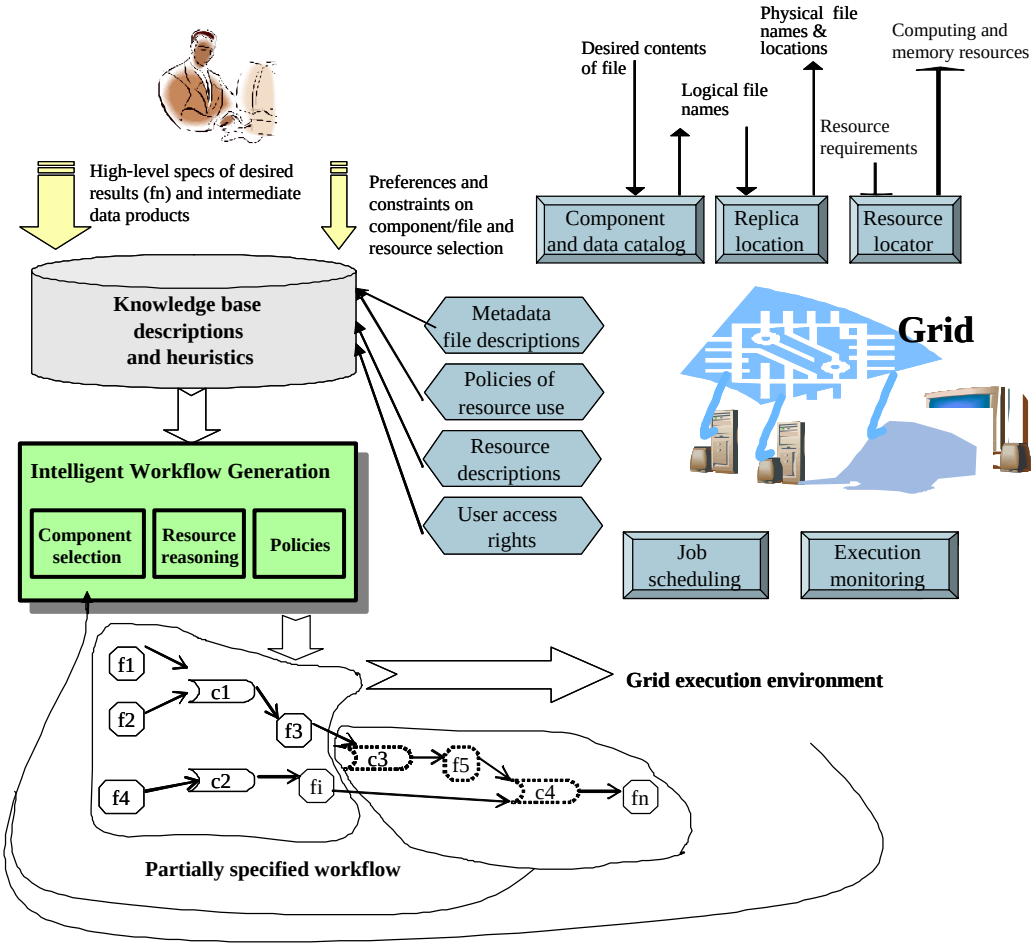

Figure 1: application in the grid environment

Much of the knowledge concerning descriptions of components, resources and the system's state is available from a variety of Grid middleware, as we describe in the next section. However, one must be an experienced Grid user to run jobs on the Grid, which means that much additional knowledge needs to be represented about what terms mean and how they related to one another. For example, an application component may be available in a file where it has been compiled for MPI. MPI is a Message Passing Interface, which means that the source includes calls to MPI libraries that will need to be available in the host computer where the code is to be run. Even a simple piece of Java code implies requirements in the execution host, namely that the host can run JVM (Java Virtual Machine). Our contribution is to organize this knowledge and reason about it within a uniform framework.

Overview of the Grid environment

Grid environments, such as Globus (Globus 02), include middleware services that enable users to obtain information about the resources available, 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. Exact replicas of the executable file can be stored in many locations, which helps reduce execution time. Data files can also be replicated in various locations. 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 a 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

Our techniques are general, but we have applied them 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.

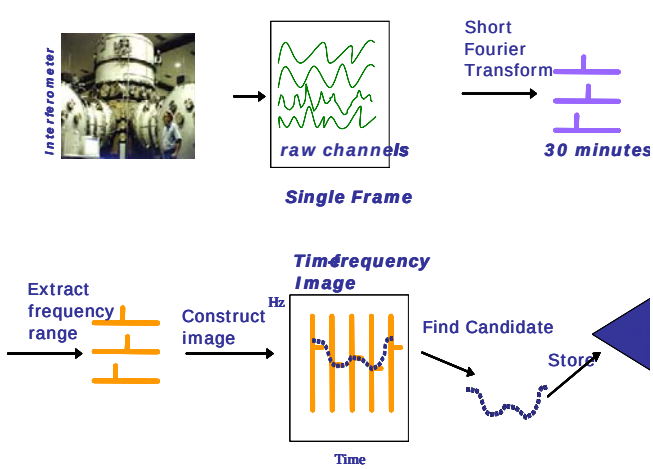

Figure 2: The computational pathway required for a pulsar search.

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.

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 requirements of the programs, both hardware and software.

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. File transfer across the network is also modeled with a planning operator, accounting for any required data movement. 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.

The initial state given as input to the planner captures information from several sources, including (1) hardware resources available to the user, described using the CIM ontology (CIM 02), (2) estimates of bandwidths between the resources.and (3) relevant data files that have already been created, their locations and metadata descriptions.

Our aim is for this information to be extracted automatically. At present, the process is partially automated, 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 using Prodigy (Veloso et al. 95), see (Blythe et al. 03a).

We now describe the implementation of our planning-based solution in more detail. 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.

Integration with the Grid environment

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 the 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 final 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.

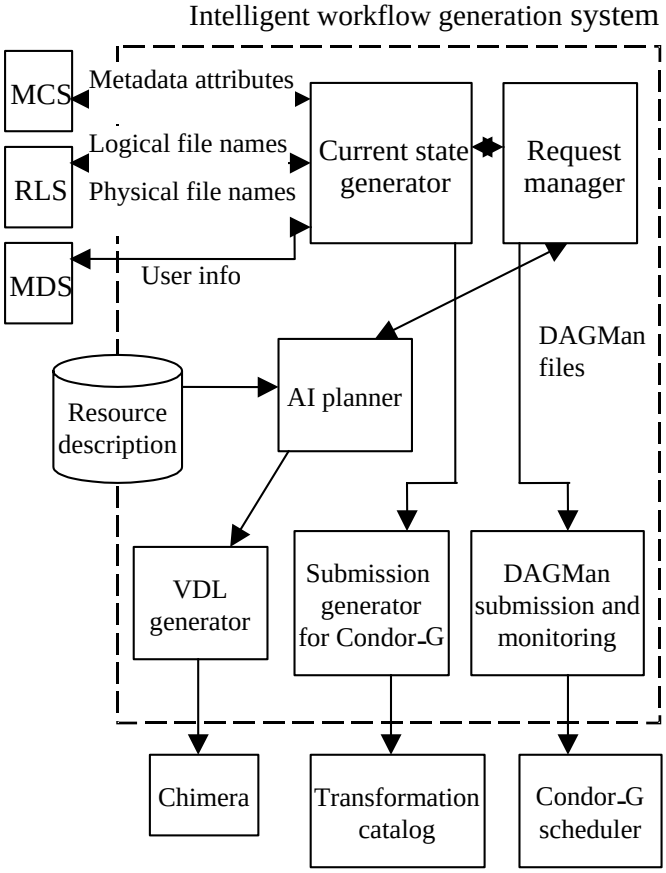

Figure 3: Architecture of the planning system and its interactions with other Grid-based services in Pegasus.

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

Practical issues in using an AI planner

AI planning is an appropriate tool for constructing workflows because it allows a declarative representation of workflow components with separate search regimes. Heuristic control for constructing good workflows or

evenly sampling the set of possible workflows can also be modeled declaratively. In order to make the most efficient use of AI planning, it was necessary to integrate the planner with specialized sub-solvers that were more efficient for certain sub-problems. Similar approaches have been taken to integrate AI planning with scheduling systems (Myers et al. 01).

In this case, a user request for a pulsar search might lead the planner to schedule up to 400 separate short Fourier transform (SFT) tasks on the available machines. These tasks are identical except for their input parameters, are independent of one another and all need to be completed before a concatenation task is applied to their results: a situation sometimes called 'parameter sweep'. It is more efficient for the planner to consider the separate instances of the SFT program running on one host as a single instance, and use a separate routine to assign the SFT instances to the available hosts. This routine balances the workload while taking into account the availability of any already existing input files on the Grid. For the planner, reasoning at this slightly higher level of abstraction required re-formulating the operator for the SFT from one that models single instances of the program into one that models multiple instances. This was done by hand but we plan to automate the process, since this situation is common.

In the LIGO application, the planner returned the first plan generated, using local heuristics aimed at generating a plan with low expected run-time. The planner can also be used to evaluate all possible plans and return one with the lowest run-time according to its estimates, or as an anytime algorithm, searching for better plans and returning the best found when queried. To make the estimates, a routine is attached to each operator to estimate its run-time as a function of its input data and the chosen host. The local estimates are combined into an estimate for the whole plan based on the partial order of tasks. In principle the estimates can also be used with partial plans in order to perform an A* search for the best plan, but this is not currently implemented.

Executing the plan on the grid

Once the plan is completed, it is sent to the Request Manager as a partially-ordered set of tasks. Any tasks that are combined in the planner, for example the SFT construction tasks in the LIGO scenario, are represented separately. The partial order is used to oversee the execution of the plan as a workflow on the Grid, but two steps must first be taken to complete the workflow.

The plan includes steps to create any needed data products, and these will be stored in files on the Grid. Within the plan, these data products are referred to by their metadata descriptions, and another set of queries to the Metadata Catalog Service is made to create the appropriate logical file names and enter their associated metadata attributes. These files will be created on the host machines where the programs are to be run, but some of them may need to be moved to other storage machines for long-term availability and registered to services like the Replica Location Service so they can be found and re-used in future requests, if the user so chooses. The Request Manager adds the necessary steps to the workflow to store and register these files.

The completed workflow is then submitted to DAGMan for execution. DAGMan keeps track of task dependencies, and schedules each task on the required machine when the parent tasks have completed.

Experiences with the planner

The planning system described above was shown at the Super Computing conference SC '02 in November, where it was used to create and execute workflows in the pulsar search domain, using approximately ten machines and clusters of different architectures and computing and storage resources at Caltech, the University of Southern California and the University of Wisconsin, Milwaukee. During the conference it was used to perform 58 pulsar searches, scheduling over 330 tasks and over 460 data transfers, consuming over eleven hours of runtime on high performance computing resources from several organizations. Since then the system was used to generate additional workloads that resulted in 185 pulsar searches, 975 tasks being executed and 1365 data files transferred. The total runtime was close to 100 hours.

Due to the interest from the physics-based user community, the demonstrators were asked at the conference if they could include an alternative algorithm for the pulsar search task that used different resource types, routines and support files. Although the authors of the planning domain were not present, it was possible for them to define additional planning operators for these routines and describe the new hosts in the resource file. The system was then able to create and execute workflows using either the original or the new algorithm, and could choose the most appropriate one depending on the availability of hosts or data products. Our collaborators from the LIGO project expressed great interest in this work and we aim for this initial implementation to become the foundation of a system with which they can perform production-level analysis.

Planning with varying levels of semantic information

We have recently been working to apply the same techniques in several other domains on the Grid, including galaxy morphology, image processing and new problems related to LIGO. While semantic information is required to define the application components with full generality for the planner, in some domains information about the input and output requirements of application domains is readily available only at the level of logical file names, for example through the Virtual Data Language VDL (Annis et al. 02). Using this information, the planning system is

just as powerful for problems for which input-output relations are defined, the only difference is in the generality of the specification. We have therefore created a translation module to automatically create planning domains from VDL descriptions. Given separately defined information about the resource requirements of the components, the resulting planning domain behaves identically to our hand-coded LIGO domain for problems within its scope. Without this resource information, the planner creates a workflow that does not assign components to resources, but is otherwise correct.

In some domains, a very large performance improvement can be realized using semantic descriptions, effectively making planning practical. As in the LIGO domain described above, file-based information is often re-arranged and re-packaged between application components, and this can result in a large number of files required as input to a component. In one real-world case where the planning domain was created from VDL, 60 operators were created that required 1,000 input files, and one was created that required 60,000.

These files can often be viewed as created by an iterative process, which implies that they can be given semantic descriptions in which they are represented by contiguous points in space defined by the semantic information. For example, if the 60,000 files are generated from two parameters, x and y, that range from 1 to 200 and 1 to 300 respectively, then each file might be described by the predicate `(f x y)`, and the collection can be represented by a predicate that implicitly encodes the set, for example `(range f 1 200 1 300)`.

We have implemented a domain-independent set representation convention of this type into the planning system in order to represent large groups of files efficiently. At the same time, some of the application component definitions are lifted to work with sets rather than individual files. However in order to complete a general solution, a domain-independent library of operators for reasoning about sets encoded as ranges must be added to the planner. This is because (1) the initial state may specify that certain files exist, but not explicitly that ranges exist, (2) existing files may cover some but not all of a required range of files, and (3) components may have a maximum set size for which they can be scheduled on one machine which may not match the required range.

Our philosophy is, as before, that a user should not have to think about the re-packaging of data that is required in order to coordinate workflows on a grid. Therefore we have encoded a library of axioms for reasoning about ranges of files into the planner as domain-independent operators that are added during the creation of the planning domain. Currently the planner is able to solve problems efficiently with the following set of axioms:

- The ‘empty’ range of files always exists
- If a range of files exists, then so does any subrange
- If two contiguous ranges of files exist, then so does their concatenation as a single range of files
- If every individual file defined by a range exists, then so does that range of files

We are further exploring this representation to ensure the completeness and efficiency of the planner when working with ranges of files. The work is a good example of how domain-independent but specialized reasoning can be of great benefit in planning and scheduling problems in grid and web environments.

Related Work

AI planning has been used to compose component programs for image processing to achieve an overall goal (Lansky et al. 95, Chien and Mortensen 96). These systems face similar issues in modeling components for planners, but do not handle distributed resources on a network, or attempt to improve plan runtime. McDermott (02) applies planning to the problem of web service composition, which shares with this domain the problem of composing software components in a distributed environment where many components are not directly under the planner's control although the work does not address resource requirements and use. Other projects use knowledge bases to facilitate the use of the Grid. The MyGrid project (Wroe et al. 03) uses DAML+OIL (Horrocks 02) and DAML-S (Ankolekar et al. 01) to describe application components as semantic web services. These descriptions are used to support matching and discovery of components through a description logic reasoner. Our work is complementary in that it uses the descriptions of the components to generate end-to-end workflows.

Future work

Our initial steps developing this application have shown the usefulness of a planning approach to workflow construction, and of declarative representations of knowledge uniformly available in the Grid. We have also identified a number of issues that we will explore in our future work in this area.

Reasoning about entire workflows allows us to find a globally optimal solution that may not be possible if we seek a locally optimal allocation for each component task. However, a relatively long-term plan may be far from optimal or unachievable in practice because the computational environment can change rapidly while the plan is executed. Scheduling of tasks may simply fail, and resources may become unavailable or be swamped when needed, bandwidth conditions may change and new data may render some later steps pointless.

We intend to incorporate Grid monitoring services in our framework to continually monitor the environment as the plan is executed, and repair or recompute the plan if needed. We will initially exploit the fact that plan creation in this domain is fast compared with execution, so one can continually re-plan as the situation changes, and always schedule the next task from the latest available plan. Other strategies for plan monitoring, re-planning and reactive planning are also applicable, as are strategies to predict and avoid likely sources of failure (Boutilier et al. 98).

Conclusions

Our initial work in applying knowledge-based techniques to make Grid computing more transparent and accessible has led to interesting results and an encouraging response from the user community. In addition to considering the challenges listed above, we are currently testing the generality of the approach by developing applications for high-energy physics and with earthquake simulations for the Southern California Earthquake Center (http://www.isi.edu/ikcap/scec-it/). If successful, this approach will be of significant help in bringing the benefits of Grid-based computing to a much wider base of users. Many additional AI techniques will be useful towards this goal, including scheduling and resource reasoning, ontologies and description logic reasoning, multi-agent systems, and reasoning about uncertainty.

Acknowledgments. We gratefully acknowledge helpful discussions on these topics with our colleagues, including Ann Chervenak, Carl Kesselman, Jihie Kim, Paul Rosenbloom, Tom Russ and Hongsuda Tangmunarunkit. We thank Gaurang Mehta, Gurmeet Singh and Karan Vahi for the development of the demonstration system. We also thank the following LIGO scientists for their contribution to the grid-enabled software: Kent Blackburn, Phil Ehrens Scott Koranda, Albert Lazzarini, Mary Lei, Ed Maros, Greg Mendell, Isaac Salzman and Peter Shawhan. This work was supported by the National Science Foundation under grants ITR-0086044 (GriPhyN) and EAR-0122464 (SCEC/ITR), and by an internal grant from USC's Information Sciences Institute.

References

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., 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) 2003a

Blythe, J., Deelman, E., Gil, Y., Kesselman, C., Transparent Grid Computing, a Knowledge-based Approach in Proc. *Innovative Applications of Artificial Intelligence*, (IAAI) 2003b

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

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

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.

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.

Chien, S. A. and H. B. Mortensen, "Automating Image Processing for Scientific Data Analysis of a Large Image Database," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18 (8): pp. 854-859, August 1996.

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, 2003a

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

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

Foster, I., C. Kesselman, and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 2001.

Foster, I., C. Kesselman, J. Nick, S. Tuecke. Grid Services for Distributed System Integration. *Computer*, 35(6), 2002.

Frey, J., Tannenbaum, T., Foster, I., Livy, 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

Horrocks, I., DAML+OIL, a Reasonable Web Ontology Language, *Proceedings of EDBT 2002*, LNCS 2287, 2-13, 2002

Lansky, A., L. Getoor, M. Friedman, S. Schmidler, N. Short, The COLLAGE/KHOROS Link: Planning for Image Processing Tasks, *AAAI Spring Symp. on Integrated Planning Applications*, 1995

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

Myers, L. and Smith, S. and Hildum, D. and Jarvis, P. and de Lacaze, R. Integrating Planning and Scheduling through Adaptation of Resource Intensity Estimates. *Proceedings of the 6th European Conference on Planning* (ECP), 2001

Veloso, M., J. Carbonell, A. Perez, D. Borrajo, E. Fink, and J. Blythe. (1995) Integrating Planning and Learning: The PRODIGY Architecture. *Journal of Theoretical and Experimental AI*, 7(1), 1995.

Waldrop, M, 2003 Grid Computing, in "10 Emerging Technologies That Will Change the World". MIT Technology Review, February 2003

Wolski, R., Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service, in Proc. *6th IEEE Symp. on High Performance Distributed Computing*. 1997: Portland, Oregon.

Wroe, C., R. Stevens, C. Goble, A. Roberts, and M. Greenwood. (2003). "A Suite of DAML+OIL ontologies to describe bioinformatics web services and data". To appear in *Journal of Cooperative Information Science*.