

Automatically Composed Workflows for Grid Environments

Jim Blythe, Ewa Deelman, and Yolanda Gil, *USC Information Sciences Institute*

Once the realm of high-performance computing for scientific applications, grid computing is rising as a key enabling infrastructure for resource sharing and coordinated problem solving in dynamic multi-institutional virtual organizations.¹ Grids build over networking technology to provide middleware support such as locating files

over a network of computers, scheduling the distributed execution of jobs, and managing resource sharing and access policies.² 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.^{3,4}

Unfortunately, grid computing today is far from the reach of regular computer users. To use a grid, many users must provide a detailed executable script specifying the jobs that should run, the hosts and files to use, and sometimes the specific scheduler in the host computer. Our research aims to develop intelligent middleware components that encapsulate the expertise required to use grids. Earlier, we used AI planning techniques to automatically generate executable job workflows from high-level specifications of desired results.⁵ We integrated our planner in a grid environment to extract relevant knowledge from existing grid middleware.⁶

One of our application domains is the Laser Interferometer Gravitational Wave Observatory (LIGO, www.ligo.caltech.edu), which aims to detect pulsars. The planner can be given the high-level goal of making a search in certain areas of the sky over a certain time period. Our solution for LIGO uses semantic descriptions of the workflow processes and of the files they require and produce, to compose the workflow automatically.

However, domains where such semantic descriptions aren't yet available can still benefit from AI planning and scheduling. Moreover, greater improvements in efficiency are possible using semantic descriptions of the information content of a group of related files, or of content that any one of a group of files can provide.

Motivation

Scientists often seek specific *data products*, which they can obtain by configuring available *application components* and executing them on the Grid. For 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 results, the user might have requirements on intermediate steps. For example, the user might want the results of any intermediate filtering steps to be available in location *I*, perhaps to examine them for unusual phenomena.

Today, users must transform this kind of high-level requirement into a workflow of jobs that they can submit for execution on the Grid. Each job must specify which files contain the code to run. To do this, the user must map the high-level requirements to available application components (for example, the fast Fourier transform coded by Caltech's group, version 3.0 or higher) and select a file location from the many available replicas of the code in various locations. The job also specifies the host on which it should run, on the basis of the code requirements (for example, the code must be compiled for a message-passing interface and parallelized to run on a tightly coupled architecture, preferably with more than five nodes) and on the basis of the user access policies for computing and storage resources. An executable workflow also includes jobs to transfer input data and application component files to the execution location.

The need for automation

Although grid middleware allows for discovery of the available resources and of the replicated data's locations, users currently must carry out all these steps manually. Automating this process is not only

This planning system uses heuristics to select application components and computing resources to help generate executable workflows for a grid, and provides different levels of support, depending on the information available.

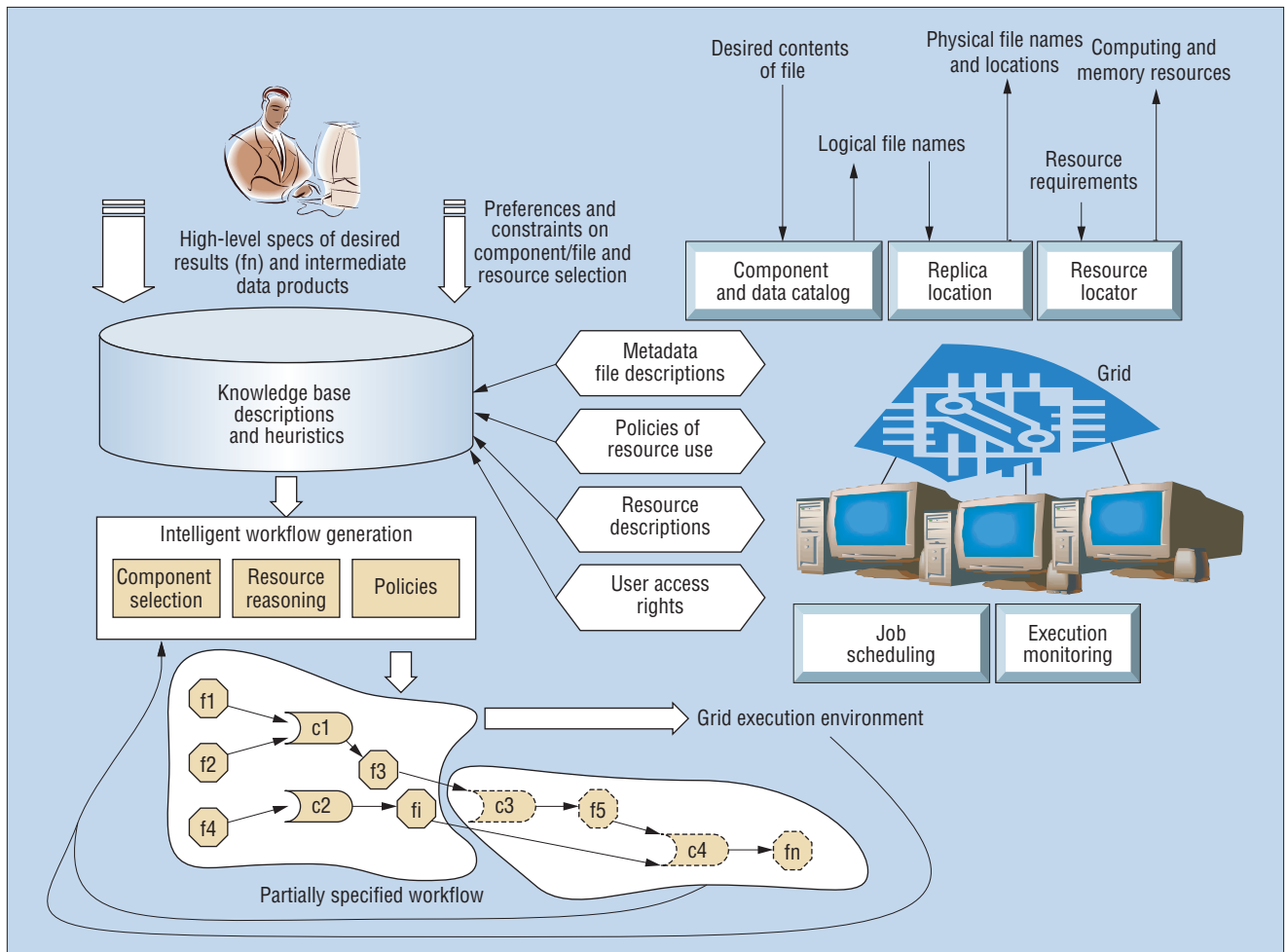


Figure 1. An architecture for workflow generation on the Grid.

desirable but also necessary for five reasons.

The first is usability. Currently, users must have extensive knowledge of the grid computing environment and its middleware functions. For example, the user must query the *Replica Location Service* (RLS)⁷ to find the input data files' physical locations.

The second reason is complexity. In addition to requiring scientists to become grid-enabled users, the process might be complex and time consuming. The user must make many choices when alternative application components, files, or locations are available. The user might reach a dead end where he or she can't find a solution, which would require backtracking to undo some previous choice.

The third reason is solution cost. Lower-cost solutions are highly desirable in light of the high cost of some computations, the user's limited access to resources, and bandwidth variability. Because finding any feasible solution is already time consuming, users won't be likely to explore alternative

workflows that could reduce execution cost.

The fourth reason is global cost. Because many users are competing for resources, minimizing cost within a community is desirable. This requires reasoning about one user's choices in light of other users' choices, such as possible common jobs that could be included across workflows and executed only once. In addition, policies that limit a user's access to resources should be taken into account to accommodate as many users as possible while they're contending for limited resources.

The last reason is reliability of execution. In today's Grid framework, when a job's execution fails, the job is resubmitted for execution on the same resources. Recovery mechanisms that consider alternative resources and components as the grid changes are desirable.

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

Our approach to automation

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, and access control policies.

Second, the planner can access this knowledge at any point during workflow generation. This lets the planner 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 user requirements
- Is feasible in the given execution environment
- Can adapt to changes in the overall system state

Figure 1 illustrates our approach. Users provide high-level specifications of desired

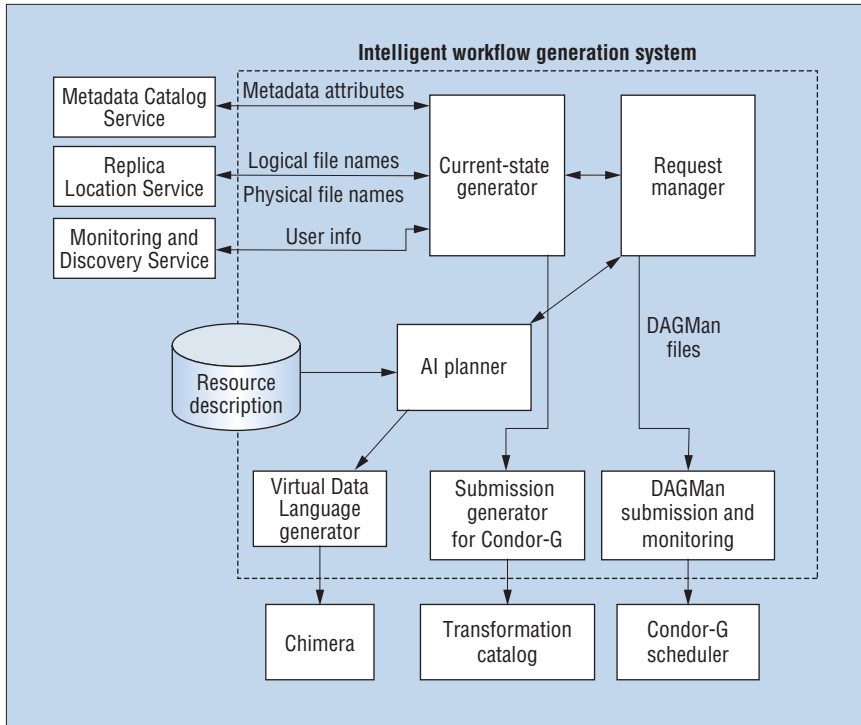


Figure 2. The planner's architecture and its interactions with other grid-based services in Pegasus.

results, as well as constraints on the components and resources to use. These requests and preferences are represented in the knowledge base. The grid environment contains middleware to find components that can generate the desired results, to find the input data that they require, to find replicas of component files in specific locations, to match component requirements with available resources, and so on. The knowledge base also incorporates some knowledge that middleware currently uses (resource descriptions, metadata catalogs to describe file contents, user access rights and use policies, and so on). Workflow generation and maintenance components update workflow during execution as new information becomes available.

Although much of the necessary knowledge is available from middleware (as we describe in the next section), running jobs on a grid requires considerable additional knowledge. For example, an application component might be available in a form that's compiled for a message-passing interface. So, the component makes calls to MPI libraries that will need to be available in the host computer where the component is to run. Similarly, a piece of Java code implies requirements on the execution host—namely,

that the host can run Java Virtual Machine. Our approach organizes this knowledge and reasons about it in a uniform framework.

Grid environments

As we mentioned before, middleware services such as those that the Globus toolkit (www.globus.org) provides help users obtain information about available resources, component software, data files, and the execution environment. We've used several of these services as knowledge sources for our planner^{8,9} as part of its integration in the Pegasus system (<http://pegasus.isi.edu>) for planning and execution in grids (see Figure 2).

In grid environments, an application component (for example, a fast Fourier transform) can be implemented in different source files, each compiled to run on 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 application-specific metadata, which is basically a collection of terms of predicate logic capturing the meaning, provenance, and other aspects of the data. *Logical file descriptions* uniquely identify the application component

or data; *physical file descriptions* uniquely specify the location and name of a specific file on a host. The *Metadata Catalog Service* (MCS) responds to queries that are based on application-specific metadata and returns the logical names of files containing the required data, if they exist.^{10,11} Given a logical file name that uniquely identifies a file without specifying a location, the RLS can find physical locations for the file on a grid.⁷

The grid execution environment includes computing and storage resources with diverse capabilities. A specific application might require a certain type of resource for execution—for example, a certain number of nodes to efficiently parallelize its execution. This might be indicated in its metadata description. Executing applications with minimal overhead might require specifying which job queue available in the host is more appropriate. The *Monitoring and Discovery Service* (MDS)¹² allows discovery and monitoring of grid resources. Resource match-makers find resources appropriate to the application components' requirements.

Pegasus sends jobs that are completely specified for execution to schedulers that manage the resources and monitor execution progress. For example, users can employ Condor-G and DAGMan (Directed Acyclic Graph Manager)¹³ to request a task to be executed on a resource. Condor-G adds an individual task to a resource's queue, while DAGMan manages the execution of a partially ordered workflow by waiting for a task's parents to complete before scheduling the task.

Given that our planner decides for the user where to generate the data and what software and input files to use, it's important to provide the user and others accessing this derived data with its provenance information, or to explain how the data arrived at its current form. To achieve this, we've integrated our system with the Chimera Virtual Data System.¹⁴ Our system generates a Virtual Data Language description of the products generated in the workflow. Chimera uses this VDL description to populate its database with the relevant provenance information.

LIGO pulsar search

Our techniques are general, but we've applied them in the context of LIGO. We've focused on pulsar search (see Figure 3), where grid resources must search for evidence of gravitational waves possibly emitted by pulsars. The data needed to conduct

the search is a long sequence (approximately four months and 2×10^{11} points) of a single channel—the gravitational wave strain channel observed at the LIGO instrument. The observatory’s output is in small segments of many channels that are stacked to make a large time-frequency image, perhaps 4×10^5 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 in the LIGO Scientific Collaboration. To exploit the grid resources, LIGO’s existing analysis tools were integrated into a grid environment. The executable workflows we created were applied to LIGO data collected during the instrument’s first scientific run. These workflows targeted a set of 1,000 locations of known pulsars as well as random locations in the sky. The analysis results were made available to LIGO scientists through the grid.

Modeling workflow composition as a planning problem

Here’s how we formulate assignment of a set of coordinated tasks in a workflow and allocation of the tasks to available resources as an AI planning problem.

Planning operators

We model each application component that might take part in the workflow as a planning operator. The operators’ effects and preconditions reflect two information sources: data dependencies between the program inputs and outputs, and the programs’ hardware and software resource requirements.

The planner imposes a partial order on the tasks that’s sufficient for execution because it models their input and output data dependencies. If a task’s prerequisites are completed before the task is scheduled, the information required to run the associated program will be available. To account for any required data movement, we also use a planning operator to model file transfer across the network. We model the data dependencies between tasks 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 term in predicate logic denoting the result of a pulsar search in a fixed point in the sky across a fixed range of frequencies at a fixed time—let the user specify requests for information without specific knowledge of programs or file systems, with the planner filling in the request’s details.

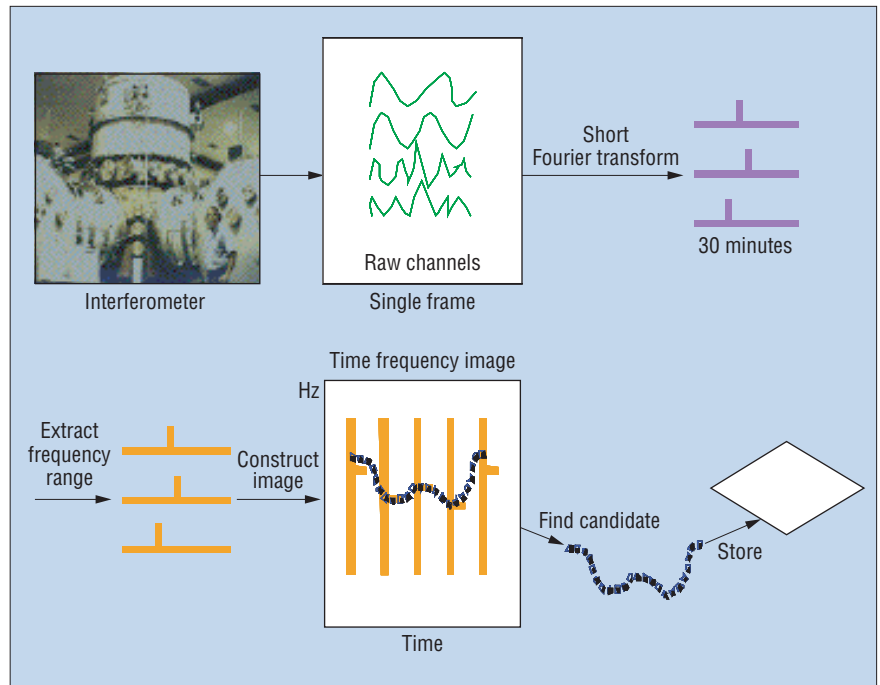


Figure 3. The computational pathway required for a pulsar search.

Because the operators also model the files that a program creates and uses, the planner knows how the information is accessed and stored. It can then reason about how tasks can share information and can plan for moving information around the network. The planner’s representation of information and files can be kept up to date with the state of the Grid by making queries to the MCS and RLS, so that its plans are directly executable and efficient.

The planning operators also model constraints on the resources required to perform the desired operations. Hardware constraints might include a particular machine type, the minimum RAM or hard disk space available, or the presence of a certain number of nodes in a distributed-memory cluster. Software constraints might 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

The planner receives as input an initial state that captures information from several sources, including

- Hardware resources available to the user, described using the CIM (Common Information Model) schema (www.dmtf.org/standards/standard_cim.php)
- Estimates of bandwidths between the resources
- Relevant data files that have already been created, and their locations and metadata descriptions

Our objective is automatic extraction of this information. At present, the process is partially automated, as we describe later in this article.

Goals

The goal given to the planner usually represents a metadata request for information and a location on the network where the data should be available. Additional goals can also specify the programs or host machines to use, for intermediate or final steps.

Search control rules

These rules help the planner quickly find good solutions based on preferences for resources and component operators. They also help the planner search the space of all plans more efficiently to find high-quality plans, given more search time. Details on the planning-domain specification and its implementation using the PRODIGY architecture¹⁵ appear elsewhere.⁵

Using the planner

Our use of the planner has three phases:

1. Preparing the input problem specification
2. Practical considerations for using AI planning for workflow composition
3. 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* produces the initial state description, and the *request manager* produces the goal description from a user request. The current-state generator uses the MCS and RLS.

Given knowledge of which data products already exist and their location, the planner can choose whether to transfer existing data across the network or re-create it closer to where it's needed. The planner makes this choice either explicitly by search control rules or implicitly by creating a number of candidate plans and picking the one with the best expected runtime.

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 to let the planner query for the existence of data products while planning. Although the planner can make the queries, we chose to gather the information before planning because this lets us combine the potentially large number of queries about files, reducing bandwidth and the load on the MCS and RLS. The current-state generator decides which data products to query on the basis of the goal description. This could be done through a static analysis of the planning operators but is currently hard coded.

Once the current-state generator retrieves the file information, it sends the information to the AI planner, along with the goal from the request manager. The AI planner merges this information with a static file describing available resources to create the final initial state and goals used for planning. Our aim for the near future is to use the MDS to retrieve information about computer hosts, rather than use a static file, and to use the Network Weather Service¹⁶ to retrieve timely information about bandwidth estimates between resources. We also intend to integrate user profile information and user-provided input indicating user preferences and access to resources.

The planning operators are stored sepa-

rately. We're investigating ways to generate the operators from metadata and resource information about the application components. We describe one implemented approach in a later section.

Practical considerations. AI planning techniques allow a declarative representation of workflow components with separate search regimes. They can also declaratively model heuristics for constructing good workflows or evenly sampling the set of possible workflows. To make the most efficient use of AI planning, we have to integrate the planner with specialized subsolvers that were more efficient for certain subproblems. Other researchers have taken similar approaches to integrate AI planning with scheduling systems.¹⁷

AI planning techniques allow a declarative representation of workflow components with separate search regimes.

For the LIGO application, for example, a user request for a pulsar search might lead the planner to schedule up to 400 separate short Fourier transform tasks on the available machines. These SFT tasks are identical except for their input parameters and are independent of one another. All these tasks must complete before the planner applies a concatenation task to their results—a situation sometimes called a *parameter sweep*. In this case, it's more efficient for the planner to consider all the instances of the SFT program that are assigned to one host as a task group, without explicitly reasoning about the instances. A separate routine assigns the SFT instances to the task groups. This simple routine is a specialized subsolver that balances the workload while taking into account any input files that already exist on the grid. Reasoning at this slightly higher level of abstraction in the planner required reformulating the operator for the SFT from one that models single instances of the program to one that models multiple instances. This is an example of changing the level of semantic detail in the planner, as we describe in more detail in a later section.

In the LIGO application, the planner re-

turns the first plan generated, using local heuristics aimed at generating a plan with low expected runtime. The planner can also either evaluate all possible plans and return one with the lowest runtime according to its estimates, or act as an anytime algorithm, searching for better plans and returning the best found when queried. To make the estimates, we attach a routine to each operator to estimate its runtime as a function of its input data and the chosen host. The local estimates are combined into an estimate for the whole plan on the basis of the partial order of tasks. In principle, the planner could also use the estimates with partial plans to perform an A* search for the best plan, but we haven't implemented this.

Executing the plan on the grid. Once the plan is complete, the AI planner sends it to the request manager as a partially ordered set of tasks. Any tasks that the planner has combined—for example, the SFT construction tasks in the LIGO scenario—are represented separately. The planner uses the partial order to oversee the plan's execution as a workflow, but two steps are first necessary to complete the workflow.

The first step is to create any needed data products and store them in files on the grid. The plan, which includes steps to create the data products, refers to them by their metadata descriptions. The planner makes another set of queries to the MCS 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 run. However, some of the files might need to be moved to other storage machines for long-term availability and registered to services such as the RLS so that they're available for reuse in future requests. The request manager adds the necessary steps to the workflow to store and register these files.

In the second step, the request manager submits the completed workflow 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

To create and execute workflows for pulsar searches, our planner used approximately 10 machines and clusters of different architectures and computing and storage resources at Caltech, the University of Southern Cali-

fornia, and the University of Wisconsin, Milwaukee. The planner performed 185 pulsar searches, scheduling 975 tasks and making 1,365 data transfers. The total runtime was close to 100 hours.

Owing to the interest from the physics-based user community, we were asked at a conference demonstration to include an alternative algorithm for the pulsar search task that used different resource types, routines, and support files. We were able to define additional planning operators for these routines and describe the new hosts in the resource file. The planner could then create and execute workflows using either the original or the new algorithm and could choose the most appropriate workflow 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've recently been applying these techniques in several other domains, including galaxy morphology,¹⁸ astronomy,¹⁹ and tomography. While defining the application components with full generality for the planner requires semantic information, in some domains information about the application components' input and output requirements is readily available only at the level of logical file names. For example, the VDL characterizes components with a set of clauses, where each clause relates a group of output files to the input files and program parameters required to produce them.¹⁴

We've created a translation module to automatically create planning domains from VDL descriptions. Given this information, the planner can compose workflows as before, as long as the goal is a specific logical file name rather than a metadata description. However, the domain is typically less general than when we use metadata, because we must mention each potential goal explicitly. Given separately defined information about the components' resource requirements, the resulting workflows behave identically to our hand-coded LIGO domain for problems within the reduced domain. Without this resource information, the planner creates a workflow that doesn't assign components to resources but is otherwise correct.

For some domains, semantic descriptions

can considerably improve performance, effectively making planning practical. As in the LIGO domain, file-based information is often rearranged and repackaged between application components, which can result in a large number of files required as input to a component. A case in which we created the planning domain from VDL descriptions produced 60 operators that required 1,000 input files and one operator that required 60,000 input files.

Because we can often view these files as being created by an iterative process, we can represent them compactly by explicitly representing the iteration parameters. Suppose, for example, that the 60,000 files are generated from two parameters, x and y , whose values range from 1 to 200 and 1 to 300, respectively. Then, a predicate $(f\ x\ y)$ can describe

For some domains, semantic descriptions can considerably improve performance, effectively making planning practical.

each file, and a predicate that implicitly encodes the set—for example, $(\text{range } f\ 1\ 200\ 1\ 300)$ —can represent the collection.

We've used a domain-independent convention of this type to represent large groups of files efficiently in the planner. We also modified some application component definitions to work with sets rather than individual files. However, to complete a general solution, the planner must reason about operations on sets. There are several reasons for this. First, the initial state might specify that all the files in some range exist but not explicitly specify that the range exists. The planner must be able to infer this if needed. Second, existing files might cover some but not all of a required range of files, and the planner must reason about which files are missing. Third, there might be a maximum number of files for which a component can be scheduled on one machine, which might be smaller than the range the user desires. The planner must therefore be able to reason about the subsets of input files that computations require and to infer that those subsets exist.

Our philosophy is, as we mentioned before, that a user shouldn't have to think about the repackaging of data that's necessary to coor-

dinate workflows on a grid. So, we've encoded a library of axioms for reasoning about ranges of files as domain-independent operators. We include these operators in the planning domain when we create it. Currently, the planner can solve problems efficiently with these four 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 their concatenation also exists as a single range of files.
- If every individual file defined by a range exists, then so does that range of files.

We're further exploring this representation to ensure the planner's completeness and efficiency when working with ranges of files. This research is a good example of how domain-independent but specialized reasoning can help planning and scheduling problems in Grid and Web environments.

Our initial work in applying knowledge-based techniques to make grid computing more transparent and accessible has led to promising results and an encouraging response from the user community. We're further testing the approach's generality by developing applications for high-energy physics and with earthquake simulations for the Southern California Earthquake Center (www.scec.org). If this approach is successful, it will significantly help bring grid-based computing's benefits to a much wider user base. (For a look at other research related to ours, see the sidebar.)

Although our research focuses on the domain of grid environments, it has more general applications for composition of Web Services. Indeed, many grid toolkits are adopting the Open Grid Services Architecture,³ in which grid services are an evolution of Web Services. The issues of semantic descriptions of component and data information used in planners discussed in this article are relevant to both domains.

We've identified several issues that we'll explore in future research. For instance, reasoning about entire workflows lets us find a globally optimal solution that might not be possible if we seek a locally optimal allocation for each component task. However, a relatively long-term plan might be far from opti-

Related Work

Researchers have used AI planning to compose component programs for goal-directed image processing.^{1,2} These systems face similar issues in modeling components for planners but don't handle distributed resources on a network or attempt to improve plan runtime. Drew McDermott applies planning to Web Service composition, which shares with the domain of grid applications the problem of composing software components in a distributed environment where many components aren't directly under the planner's control.³ However, this research doesn't address resource requirements and usage.

Several projects have recently demonstrated planning techniques for Web Services.⁴⁻⁷ Other projects use knowledge bases to facilitate Grid use. The myGrid project⁸ uses OWL (www.w3.org/TR/owl-ref) and OWL-S⁹ to describe application components as Semantic Web Services. These descriptions support matching and discovery of components through a description logic reasoner. Our work is complementary in that it uses the component descriptions to generate end-to-end workflows.

References

1. S.A. Chien and H.B. Mortensen, "Automating Image Processing for Scientific Data Analysis of a Large Image Database," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 8, 1996, pp. 854-859.
2. A. Lansky et al., "The COLLAGE/KHOROS Link: Planning for Image Processing Tasks," *Proc. AAAI Spring Symp. Integrated Planning Applications*, AAAI Press, 1995.
3. D. McDermott, "Estimated-Regression Planning for Interactions with Web Services," *Proc. 6th Int'l Conf. Artificial Intelligence Planning Systems (AIPS 2002)*, AAAI Press, 2002, pp. 204-211.
4. T. Kichkaylo and V. Karamcheti, "Optimal Resource-Aware Deployment Planning for Component-Based Distributed Applications," to be published in *Proc. 13th Int'l Symp. High Performance Distributed Computing (HPDC 2004)*, IEEE CS Press, 2004.
5. J. Kim, Y. Gil, and M. Spraragen, "A Knowledge-Based Approach to Interactive Workflow Composition," *Proc. ICAPS Workshop Planning and Scheduling for Web and Grid Services*, 2004, pp. 44-53.
6. U. Kuter et al., "Information Gathering During Planning for Web Services Composition," *Proc. ICAPS Workshop Planning and Scheduling for Web and Grid Services*, 2004, pp. 54-61.
7. M. Pistore et al., "Planning and Monitoring Web Service Composition," *Proc. ICAPS Workshop Planning and Scheduling for Web and Grid Services*, 2004, pp. 70-77.
8. R. Stevens, A. Robinson, and C. Goble, "myGrid: Personalised Bioinformatics on the Information Grid," *Bioinformatics*, vol. 19, Supplement 1, 2003, pp. i302-i304; http://bioinformatics.oupjournals.org/cgi/content/abstract/19/suppl_1/i302.
9. A. Ankolekar et al., "DAML-S: Web Service Description for the Semantic Web," *Proc. 1st Int'l Semantic Web Conf. (ISWC 2002)*, LNCS 2342, Springer-Verlag, 2002, pp. 348-363.

workflow life cycle will require the integration of many additional AI techniques, including scheduling and resource reasoning, ontologies and description logic reasoning, multiagent 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 developing the demonstration system. We also thank Laser Interferometer Gravitational Wave Observatory scientists Kent Blackburn, Phil Ehrens, Scott Koranda, Albert Lazzarini, Mary Lei, Ed Maros, Greg Mendell, Isaac Salzman, and Peter Shawhan for their contribution to the grid-enabled software. National Science Foundation grants ITR-0086044 (Grid Physics Network) and EAR-0122464 (Southern California Earthquake Center/Information Technology Research) and an internal grant from the University of Southern California's Information Sciences Institute supported this work.

References

1. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int'l J. Supercomputer Applications*, vol. 15, no. 3, 2001; www.globus.org/research/papers/anatomy.pdf.
2. I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999.
3. I. Foster et al., "Grid Services for Distributed System Integration," *Computer*, vol. 35, no. 6, 2002, pp. 37-46.
4. M. Waldrop, "Grid Computing," in "10 Emerging Technologies That Will Change the World," *MIT Technology Rev.*, Feb. 2003, pp. 33-49.
5. J. Blythe et al., "The Role of Planning in Grid Computing," *Proc. 13th Int'l Conf. AI Planning and Scheduling (ICAPS 2003)*, AAAI Press, 2003, pp. 153-163.
6. J. Blythe et al., "Transparent Grid Computing: A Knowledge-Based Approach," *Proc. 15th Innovative Applications of Artificial Intelligence Conf. (IAAI 2003)*, AAAI Press, 2003, pp. 57-64.
7. A. Chervenak et al., "Giggle: A Framework for Constructing Scalable Replica Location Services," *Proc. ACM/IEEE SC 2002 Conf. (Supercomputing 2002)*, IEEE CS Press, 2002, p. 58.
8. E. Deelman et al., "Mapping Abstract Complex Workflows onto Grid Environments," *J. Grid Computing*, vol. 1, no. 1, 2003, pp. 25-39.
9. E. Deelman et al., "From Metadata to Execu-

mal or unachievable in practice because the computational environment can change rapidly while the plan executes. Scheduling of tasks might simply fail, and resources might become unavailable or be swamped when needed. Bandwidth conditions might change, and new data might render some later steps pointless.

We intend to incorporate grid-monitoring services in our framework that continually monitor the environment as the plan executes and that repair or recompute the plan if needed. We'll initially exploit the fact that plan creation

in this domain is fast compared with execution, so that these services can continually replan as the situation changes and can always schedule the next task from the latest available plan. Other strategies for plan monitoring, replanning, and reactive planning are applicable, as are strategies to predict and avoid likely sources of failure.²⁰

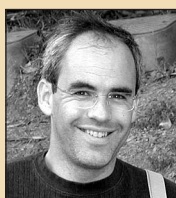
This research has demonstrated the usefulness of a planning-based approach to workflow construction and of declarative representations of data shared between several components in the Grid. Providing support throughout the

tion on the Grid: The Pegasus Pulsar Search,” GriPhyN tech. report 2003-15, Grid Physics Network, 2003; www.isi.edu/~deelman/griphyn_2003_15.pdf.

10. E. Deelman et al., “Grid-Based Metadata Services,” to be published in *Proc. 16th Int’l Conf. Scientific and Statistical Database Management (SSDBM 04)*, IEEE Press, 2004.
11. G. Singh et al., “A Metadata Catalog Service for Data Intensive Applications,” *Proc. ACM/IEEE SC 2003 Conf. (Supercomputing 2003)*, IEEE CS Press, 2003, p. 33.
12. K. Czajkowski et al., “Grid Information Services for Distributed Resource Sharing,” *Proc. 10th IEEE Int’l Symp. High Performance Distributed Computing (HPDC 2001)*, IEEE CS Press, 2001, pp. 181–194.
13. J. Frey et al., “Condor-G: A Computation Management Agent for Multi-Institutional Grids,” *Proc. 10th Int’l Symp. High Performance Distributed Computing (HPDC 2001)*, IEEE CS Press, 2001, pp. 55–66.
14. I. Foster et al., “A Virtual Data System for Representing, Querying and Automating Data Derivation,” *Proc. 14th Conf. Scientific and Statistical Database Management (SSDBM 02)*, IEEE Press, 2002, pp. 37–46.
15. M. Veloso et al., “Integrating Planning and Learning: The PRODIGY Architecture,” *J. Theoretical and Experimental Artificial Intelligence*, vol. 7, no. 1, 1995, pp. 81–120.
16. R. Wolski, “Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service,” *Proc. 6th IEEE Symp. High Performance Distributed Computing (HPDC 1997)*, IEEE CS Press, 1997, pp. 316–325.
17. K. Myers et al., “Integrating Planning and Scheduling through Adaptation of Resource Intensity Estimates,” *Proc. 6th European Conf. Planning (ECP 2001)*, 2001; www-2.cs.cmu.edu/afs/cs/project/ozone/www/jfacc/ecp01/ecp01.html.
18. E. Deelman et al., “Grid-Based Galaxy Morphology Analysis for the National Virtual Observatory,” *Proc. 2003 ACM/IEEE SC Conf. (Supercomputing 2003)*, IEEE CS Press, 2003, p. 47.
19. G.B. Berriman, “Montage: A Grid Enabled Image Mosaic Service for the National Virtual Observatory,” *Proc. Astronomical Data Analysis Software & Systems XIII (ADASS XIII)*, ASP Conf. Series vol. CS 314, Astronomical Soc. of the Pacific, 2003; www.isi.edu/~deelman/adass_03.pdf.
20. C. Boutilier, T. Dean, and S. Hanks, “Decision-Theoretic Planning: Structural Assumptions and Computational Leverage,” *J. Artificial Intelligence Research*, vol. 11, 1999, pp. 1–94.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.

The Authors



Jim Blythe is a computer scientist in the University of Southern California’s Information Sciences Institute. His research interests include planning, workflows in grid applications, and knowledge acquisition for problem solving and planning. He received his PhD in computer science from Carnegie Mellon University. Contact him at the USC Information Sciences Inst., 4676 Admiralty Way, Marina del Rey, CA 90292-6696; blythe@isi.edu; www.isi.edu/~blythe.



ences Inst., 4676 Admiralty Way, Ste. 1001, Marina del Rey, CA 90292; deelman@isi.edu.

Ewa Deelman is a research team leader at the Center for Grid Technologies at the University of Southern California’s Information Sciences Institute and an assistant research professor in USC’s Computer Science Department. Her research interests include the design and exploration of collaborative scientific environments based on grid technologies, with particular emphasis on managing large amounts of data and metadata as well as workflow management. At ISI, she’s part of the Globus project, which designs and implements middleware for the Grid. She received her PhD in computer science from Rensselaer Polytechnic Institute. Contact her at the USC Information Sciences Inst., 4676 Admiralty Way, Ste. 1001, Marina del Rey, CA 90292; deelman@isi.edu.



Yolanda Gil is an associate division director at the University of Southern California’s Information Sciences Institute and a research associate professor in the university’s Computer Science Department. She’s the principal investigator of several projects and conducts research in interactive knowledge capture, intelligent user interfaces, planning, and knowledge-intensive applications in defense and sciences. She received her PhD in computer science from Carnegie Mellon University. Contact her at the USC Information Sciences Inst., 4676 Admiralty Way, Marina del Rey, CA 90292; gil@isi.edu.

What have we *done for you lately?*



We publish **IEEE Intelligent Systems** as a service to our readers. With each issue, we strive to present timely articles and departments with information you can use.

How are we doing?

Send us your feedback, and help us tailor the magazine to you!

Write us at

Intelligent @computer.org