The Role of Planning in Grid Computing

Jim Blythe, Ewa Deelman, Yolanda Gil, Carl Kesselman, Amit Agarwal, Gaurang Mehta, Karan Vahi

University of Southern California Information Sciences Institute
4676 Admiralty Way,
Marina del Rey, CA 90292 USA
{blythe,deelman,gil,carl,agarwal,mehta,vahi}@isi.edu

Abstract

Grid computing gives users access to widely distributed networks of computing resources to solve large-scale tasks such as scientific computation. These tasks are defined as standalone components that can be combined to process the data in various ways. We have implemented a planning system to generate task workflows for the Grid automatically, allowing the user to specify the desired data products in simple terms. The planner uses heuristic control rules and searches a number of alternative complete plans in order to find a high-quality solution. We describe an implemented test case in gravitational wave interferometry and show how the planner is integrated in the Grid environment. We discuss promising future directions of this work. We believe AI planning will play a crucial role in developing complex application workflows for the Grid.

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 codes. 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.

Up to now, much of the focus of Grid computing has been on developing middleware, which provides basic functionality such as the ability to query for information about the resources and the ability to schedule jobs onto the resources. With few exceptions, little work has been done in the area of automating job execution. Users still need to discover resources manually and schedule the jobs directly onto the Grid, essentially composing detailed workflow descriptions by hand. This leaves users struggling with the complexity of the Grid and weighing which resources to use, where to run the computations, where to access the data etc.

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.

In this paper, we cast workflow generation as a planning problem, where the goals are the desired data products and the operators are the application components. The declarative representation of actions and search control in domain-independent planners is convenient for representing constraints such as machine characteristics needed for some task or policies on user access to computing resources as well as heuristics such as preferring a high-bandwidth connection between hosts performing related tasks. In addition, our planning-based approach can provide high-quality solutions, in part because it compares a number of alternative solutions and uses heuristics to find good solutions more quickly.

The next section describes the workflow generation problem in a Grid computing infrastructure. We then describe an initial system, not based on planning techniques, that addresses some aspects of the problem. The following section describes our approach using a domain-independent planning system. The output from the planner is a partially-ordered set of tasks assigned to specific computational resources, which is automatically executed on a distributed network through a Grid infrastructure. We also describe our experiences to date in the domain of a gravitational-wave observatory. The following section presents some of the issues for future work, including modeling solution quality, using richer representations of planning knowledge in ontologies, plan monitoring and replanning, and planning under uncertainty.

Workflow Generation

We briefly describe the Grid environment where the jobs are being executed. In the Grid (Foster et al. 01), resources (computational, data and instruments) are distributed in the wide area. To manage the information and interactions with these resources, the Globus toolkit (Globus 02) is deployed on the resources. Globus consists of services, which allow for the discovery of the resources and the scheduling of jobs onto the resources. Globus provides information about locating replicas of files and the means of high-performance data transfer.

The problem

Scientists often seek specific data products, which can be obtained by configuring available application components (programs) and executing them on the Grid. As an example, suppose that the user's goal is to obtain a frequency spectrum of a signal for a given instrument and time frame, placing the results at a given location. In addition, the user would like the results of any intermediate filtering steps performed to be available at another location, perhaps to check the filter results for unusual phenomena or to extract some salient features to the metadata of the final results. The process of mapping this type of user request into jobs to be executed in a Grid environment can be decomposed into two steps, as shown in Figure 1.

Generate an abstract workflow: selecting and configuring application components to form an abstract workflow. Application components are selected based on their specified capabilities and whether they can generate the desired data products. They may require inputs that either exist or need to be planned for in the same way. The resulting abstract workflow specifies the order in which the components must be executed. At this level the components and files are referred to by their logical names. A logical name uniquely identifies a component in terms of its functionality or a data file in terms of its content, but a single logical name can correspond to many actual executables or physical data files in different locations.

Generate a concrete workflow: selecting specific resources, files, and additional jobs required to form a concrete workflow that can be executed in the Grid environment. Each component in the abstract workflow is associated with an executable job by specifying the locations of the physical files of the component and data as well as the resources assigned to it in the execution environment. The chosen resources must meet the computational requirements of the component. Additional jobs may be included in the concrete workflow, for example, to transfer files to the appropriate locations.

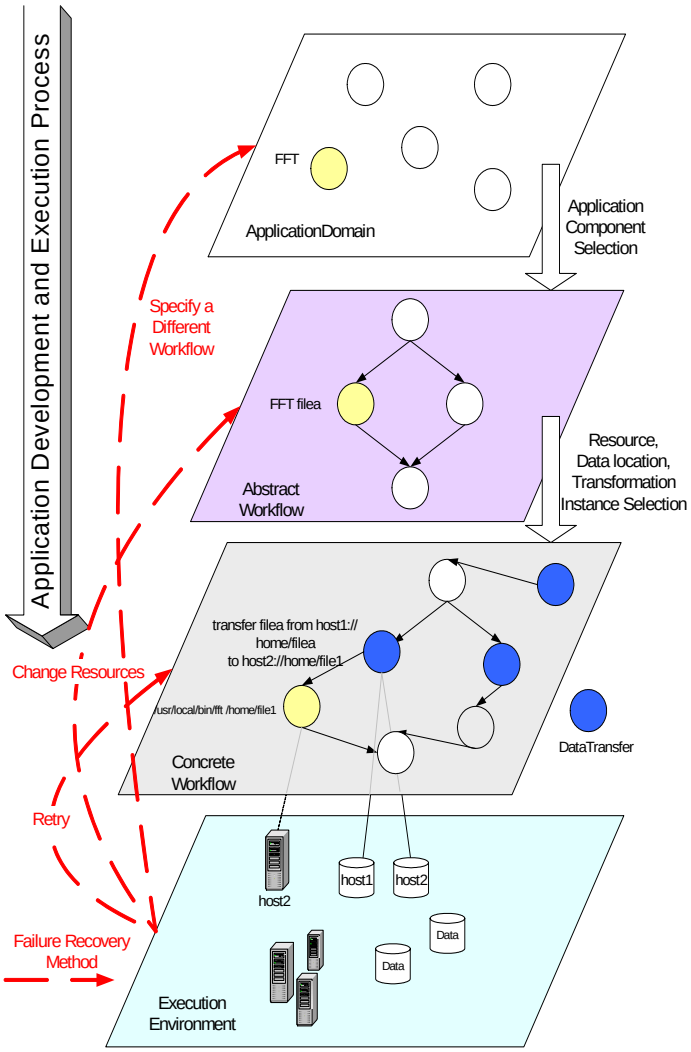

Figure 1: The Process of Developing Data Intensive Applications for Grid Environments.

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

Usability: Users are currently required to have extensive knowledge of the Grid computing environment and its middleware functions. For example, the user needs to understand how to query an information service such as the Monitoring and Directory Service (MDS) (Czajkowski et al. 01), to find the available and appropriate computational resources for the computational requirements of a component. The user also needs to query the Replica Location Service (RLS) (Chervenak et al. 02) to find the physical locations of the data.

Complexity: In addition to requiring scientists to become Grid-enabled users, the process may be complex and time consuming. At each step, the user must make choices between alternative application components, files, or locations. The user may reach a dead end where no solution can be found, which would require backtracking to undo some previous choice. Many different interdependencies may occur among components, and as a result it may be hard to determine which choice should be changed and which option will lead to a feasible solution.

Solution cost: Lower cost solutions are highly desirable in light of the high cost of some of the computations and users' resource access limitations. Because finding any feasible solution is already time consuming, users are unlikely to explore alternative workflows that may reduce execution cost.

Global cost: Because many users are competing for resources, minimizing cost within a community or a virtual organization (VO) is desirable (Foster et al. 01). 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 user's workflows and executed only once.

Reliability of execution: In today's Grid framework, when the execution of a job fails the recovery consists of resubmitting that job for execution (In Figure 1 this is shown as the "retry"). However, it is also desirable to be able to choose a different set of resources when tasks fail. This process needs to be performed at the abstract workflow level and may require choosing a new abstract workflow. Currently there is no mechanism for opportunistically redoing the remaining tasks in the workflow to adapt to the dynamic environment. Moreover, if any job fails repeatedly the system should assign an alternative component to achieve the same overall user goals. This would need to be performed at the application level, where there is an understanding of how different application components relate to each other.

While addressing the first three points would enable wider accessibility of the Grid to users, the last two simply cannot be handled by individual users and will probably need to be addressed at the architecture level. In addition, there are many access control policies that limit user's access to resources, and these must be taken into account in order to accommodate as many users as possible while they are contending for limited resources.

There are three different levels of abstraction that a user can use to specify a workflow. At the lowest level (concrete workflow) the user needs to specify explicit data movements and the exact executables and resources to be used. At the abstract workflow level the user needs only specify the workflow using logical files and logical component names. Finally at the application level, the user needs to specify only the metadata describing the desired data products.

Level	Specification example	Specification detail
Application	Frequency spectrum of a signal for a given instrument and time frame	Application-specific metadata
Abstract workflow	FFT file1	Logical file names, logical component names
Concrete workflow	Gridftp host1://home/file1 host2://home/file1 /bin/fft –i file1	Resource-level physical files and executables

Table 1: Levels of abstraction used to describe workflows

Original Workflow Generators

In the first approach taken by the group, the work is divided between two separate programs, an *Abstract Workflow Generator* (AWG) and a *Concrete Workflow Generator* (CWG). AWG uses rewrite rules to choose executable programs that can be used to produce required files, using Chimera's Virtual Data Language (Foster et al. 02). The program requires the rules to be specified in terms of logical file names, so rules must typically be specified for each request for a file. AWG takes does not check whether a file already exists, and so its resulting abstract workflow can be larger than necessary.

CWG performs the mapping from an abstract workflow to a concrete workflow. It finds physical locations for both components and data, finds appropriate computer resources to execute the components and generates an executable workflow of jobs that can be submitted to the Grid. It determines whether files in the abstract workflow already exist and, if so, removes unnecessary jobs to produce them. However, CWG performs no search, and makes a random choice whenever several alternatives are possible (e.g., alternative physical files, alternative resources). Therefore the final result is a feasible solution and not necessarily a low-cost one.

As an example, Figure 2 shows a simple abstract workflow that might be produced by AWG, in which the logical component *Extract* is applied to an input file with a logical filename *F.a*. The resulting files, with logical filenames *F.b1* and *F.b2*, are used as inputs to the components identified by logical filenames *Resample* and *Decimate* respectively. Finally, the results are *Concatenated*. CWG locates existing files by querying a Grid service and

removes the *Decimate* step, since F.c2 has been created previously. It assigns computer resources to the nodes and adds appropriate nodes for file transfer to yield the concrete workflow shown in Figure 3. This is submitted to Grid workflow programs for execution.

This solution produces a feasible workflow, querying the existing services for existing files. CWG has been successfully used in mapping and executing workflows for the Compact Muon Solenoid detector (Wulz 98), executing 678 jobs over 7 days. During that time a total of 350 CPU/days of computing power was used and a total of 200GB of data was produced. For more details on the approach and its application, see (Deelman et al. 03a).

However, these workflow generators require specifying explicit files, while a user should be able to directly request the information that the files contain, for example a Fourier transform of data from a particular location and time. We refer to this information as the file metadata. Moreover the generators do not attempt to optimize the workflow for time or reliability, and the split between abstract and concrete workflow generation introduces a barrier for optimization. Nor do they consider network bandwidth, scheduler queue size, resource reliability, speed or available memory, or check for access control policies. A more powerful approach is warranted to address aspects such as these. We next describe a solution using AI planning techniques that uses metadata, integrates abstract and concrete workflow creation and searches for a globally optimal solution. The declarative nature of the planning domain makes it easier to represent criteria based on bandwidth and resource characteristics, some of which are represented in the current version.

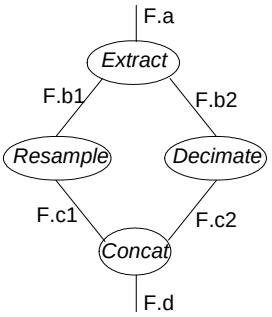

Figure 2: abstract workflow

Planning Solution

In this section we describe how we have framed workflow generation (WG) as a planning problem. Later we describe further work that is needed in planning to improve the task modeling and the generated solution.

Formulating workflow generation as a planning problem

WG models the application components along with data transfer and data registration as operators. Each operator's parameters include the host where the component is to be run, so a ground plan corresponds to a concrete workflow. In addition, some of the effects and preconditions of the operators capture the data produced by components and their input data dependencies. As a result the planner can also create an abstract workflow. The state information used by the planner includes a description of the available resources and the relevant files that have already been created. The input goal description can include (1) a metadata specification of the information the user requires and the desired location for the output file, (2) specific components to be run or (3) intermediate data products. Several issues make this application domain challenging, we touch upon them as we describe the domain model in more detail.

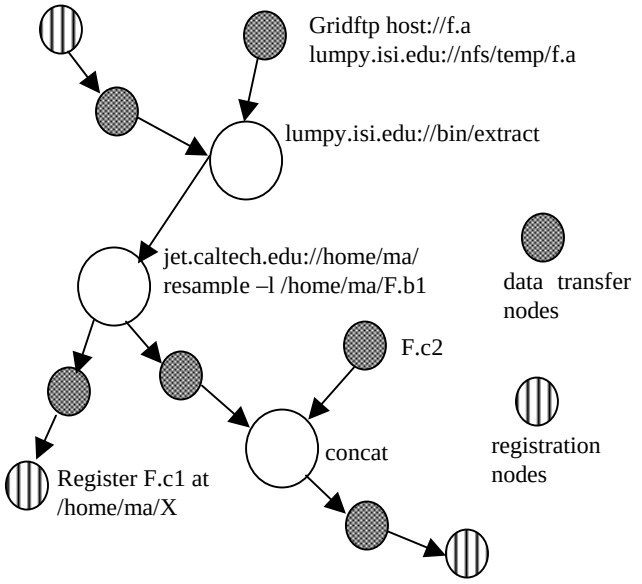

Figure 3: Concrete workflow. Specific hosts are described and nodes for data transfer and registration are added.

In our initial work, we are using the Prodigy planner (Veloso et al. 95), because search heuristics play an important role in WG and Prodigy has an expressive language for search control. We also tested versions of the domain with the more recent planner FastForward (Hoffman & Nebel 01) and found Prodigy to be competitive for our purposes, similarly to the experiences of Aler & Borrajo (02).

State information

The planner's world state includes information about resources. Some state information changes slowly if at all, such as the operating system or total disk space installed on a resource, and some of the information can change in seconds or minutes, such as the available memory or queue length. In the long run the planner may need to reason about how the information can change over time, but in our initial implementation we only model the type of a

host, network bandwidths and file information. This information is captured once at the planner's startup. In general, thousands or millions of files may be available, while only a relatively small number are relevant to the current plan. The planner can handle this by requesting the relevant information while planning, but currently we filter the set of files before planning begins.

It is useful for the planning state to include metadata about the files for several reasons. As mentioned, the planner can assume the task of creating both the abstract and concrete workflows. It is also more appropriate to reason at the level of the metadata than at the level of the files that represent that data content. Instead of searching for a file with appropriate characteristics, the components are linked to the characteristics themselves. This also avoids quantifying over the set of existing files, which may change during planning as objects are created and destroyed.

Goal statements

In most planning applications, goals refer to properties that should be true after the plan has been executed. For WG, such goals include having a file described by the desired metadata information on some host. However, it is also sometimes useful to specify goals that refer to intermediate components or data products, or for registering certain files. Thus the goal statement can, in effect, specify a partial plan.

In principle, the goals given to the planning system may be those of a single user or the aggregated goals of a group of users, although we have not explored the latter case. In that case, the planner may be able to create a more efficient plan for the overall computations required by exploiting any synergy in the users' goals.

Operator descriptions

The operators represent the concrete execution of a component on a particular host to generate a particular file or move a file across the network. Their preconditions represent both the data dependencies of the component, in terms of the input information required, and the feasible resources for running the component, including the type of resource. These operators capture information similar to that represented in Chimera's Virtual Data Language (Foster et al. 02), such as the name of the component and its parameters. However, the operators also contain the additional information about the preconditions necessary for the use of the component, and describe the effect of executing the component on the state of the system, such as the consumption of resources. We are currently adding further information about resource requirements, such as minimal physical memory or hard disk space.

Plans generated in response to user requests may often involve hundreds or thousands of files and it is important to manage the process of searching for plans efficiently. If a component needs to be run many times on different input files, it is not useful for the planner to explicitly consider different orderings of those files. Instead the planner reasons about groups of files that will be treated identically. An auxiliary routine allocates the files to different groups, looking for a locally optimal allocation. Since the number of input files or groups may vary by component and even by invocation, the preconditions are modeled using quantification over possible files.

Below is an example of an operator representing a frequency extraction component, translated to PDDL from Prodigy's action language. The operator is defined for a set of input files and describes these files as well as the resulting file in terms of metadata, such as *start-time* and *end-time*, which define the interval of time of the signal over which the extraction is taken. The predicates that are negated in the preconditions are function calls used as generative filters in Prodigy. The operator also captures the notion of the availability of the component on a resource (*host*). The effects show the creation of the output file on the chosen host.

```
(:action frequency-extract
  :parameters (?host - Host
       ?file-group - File-Group
       ?start-time - Number
       ?end-time - Number
       ?channel - Channel
       ?instrument - Instrument
       ?format - File-Format
       ?f0 - Number
       ?fN - Number
       ?sample-rate - Number)
  :precondition
    (forall
      (?sft-file-group - File-Group
       ?file-start-time - Number
       ?file-end-time - Number)
      (or
        (not (sft-range-for-sub-sft
               ?start-time ?end-time
               ?channel ?instrument))
        (not (start-time-for-sft-range
               ?sft-file-group ..))
        (not (end-time-for-sft-range
               ?sft-file-group ..))
        (and
         (sft-group
          ?file-start-time ?file-end-time
          ?channel ?instrument FRAME
          ?sample-rate ?sft-file-group)
         (at ?sft-file-group ?host))))
  :effect
    (and (sub-sft-group
           ?start-time ?end-time
           ?channel ?instrument ?format
           ?f0 ?fN ?sample-rate
           ?file-group)
         (created ?file-group)
         (at ?file-group ?host)))
```

Solution space and plan generation strategy

Most planning systems are designed to produce a feasible plan given constraints on the possible actions, but do not attempt to optimize any measure of plan quality. In WG there may be many feasible plans and it is important to find a high-quality solution. The measure of a plan's quality may include several dimensions, including the *performance* in terms of the overall expected time to satisfy the user request, the *reliability* in terms of probability of failures and their impact on performance, and issues of *policy*, for example not expending too much of a user's allowance on some precious resource if cheaper resources would be adequate. Helping users manage the tradeoff between these dimensions is a topic of future work. Our current system attempts to minimize the overall runtime of the plan. We can estimate the run-time of the plan based both on the expected run-time of individual components on the allocated resources and on the expected transfer time for files around the network.

In our initial approach, we seek high-quality plans with a combination of local search heuristics, aimed at preferring good choices for individual component assignments, and an exhaustive search for a plan that minimizes the global estimated run-time. Both aspects are necessary: without the global measure, several locally optimal choices can combine to make a poor overall plan because of conflicts between them. Without the local heuristics, the planner may have to generate many alternatives before finding a high quality plan.

These local heuristics are represented explicitly in the planner using search control rules (Veloso et al. 95). As the planner searches for a solution, it repeatedly chooses a goal to address, an operator to achieve the goal and parameter assignments for the operator. For each choice, the planner may need to backtrack and examine several alternatives in order to find a feasible plan, and search further to find the best plan. Search control rules specify options that should be exclusively considered at any choice point in the search algorithm. They can also change the order in which options are considered. The rules can refer to information about the current state when the choice is made, or to other goals in the planner. For example, a rule can be used to prefer to allocate a component to a location with a higher-bandwidth connection to the location at which the component's output is needed. This rule is applicable in almost any WG problem. Application-specific rules can also be defined. For example, the following control rule would force the planner to choose a host to perform the pulsar search that is in the same location as a host that can execute the FFT component, if possible.

```
(control-rule
  select-nearby-mpi-for-pulsar-search
  (if (and (current-operator pulsar-search)
           (true-in-state
             (available fft ?fft-host))
           (true-in-state
             (physically-at ?fft-host ?loc))
           (true-in-state
             (physically-at ?mpi ?loc))
           (type-of-object ?mpi Mpi)))
  (then select bindings ((?host ?mpi))))
 ;;; (?host is a parameter of the pulsar-search operator)
```

The planner is able to produce results similar to CWG in several test scenarios using only 3 operators and 2 control rules, although it currently supports a broader range of problems using 9 operators and 17 control rules. It takes around a tenth of a second to find its first solution in a problem with around 400 files and 10 locations, requiring 800 separate components, and around 30 seconds to exhaustively search the solutions for this problem. In this domain the time to find the first plan will scale linearly in the number of files and resources, although of course this cannot be guaranteed in other domains.

Case study: LIGO

The LIGO, or Laser Interferometer Gravitational-Wave Observatory project aims to detect gravitational waves predicted by Einstein. Theoretically, these can be used to detect astronomical objects and events such as binary pulsars, mergers of black holes or 'starquakes' in neutron stars. Searching for these objects requires, among other things, a Fourier analysis of a particular set of frequencies over some time frame. To conduct a pulsar search, for example, the user must find a number of files of raw data output corresponding to this time frame, extract the required channel, concatenate the files and make a series of Fourier transforms (FT) on the result. The desired frequencies must then be extracted from the set of FT output files, and processed by a separate program that performs the pulsar search.

In a typical pulsar search, the user may require thousands of Fourier transforms, some of which may have already been performed and stored at some location in the Grid. For good performance, this work must be divided among the suitable hosts that are available on the Grid, taking into account their different speeds and currently queued tasks. The results must be marshaled to one host for frequency extraction, and the final search must be executed on a different host because of the program requirements. In all, many gigabytes of data files may be generated, so a fast-running solution must take the bandwidth between hosts into account.

We have implemented a workflow generator called Pegasus that uses the planning approach described in the last section, and applied it to the LIGO domain. The system is operational and has generated workflows that

have been executed on the Grid. Figure 4 shows a simplified version of the Pegasus architecture, abstracting from the actual Grid services and schedulers that are used. More details can be found in (Deelman et al. 03b). In a run conducted at the Supercomputer conference in November 2002, for example, the compute and storage resources were at five locations distributed across the USA. Over 58 pulsar searches were performed resulting in a total of 330 tasks, 469 data transfers and 330 output files. The total runtime for scheduled tasks was over 11 hours. We briefly describe some of the issues that arose in integrating the planner into this environment.

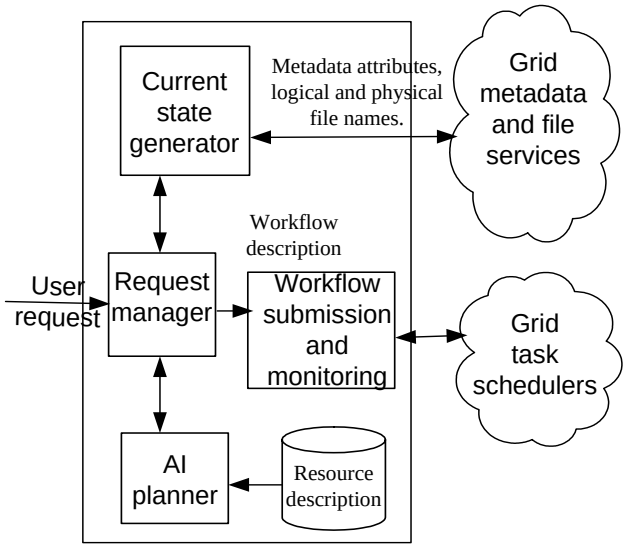

Figure 4: Simplified Pegasus architecture. The Request Manager oversees creation of the current state, calling the Planner and submitting the workflow for execution on the Grid.

The initial state is divided into two components, based on how rapidly the information changes. Relatively stable information such as the available hosts on the Grid, their computational resources and operating systems, is stored in a persistent file, while more transient state information, such as data files that have already been created, is gathered from Grid services by the Current State Generator and sent as part of each planning request to the Planner. We intend the Planner to query Grid services directly for existing files, available resources, host idle times and network bandwidth conditions as the information is found to be relevant during planning, but currently information about files is sent with the request and no other state information is used.

There are typically many FT tasks required in a plan and relatively few hosts that are suitable to run these tasks. Rather than search the possible assignments of tasks to machines, the planner uses an auxiliary routine to allocate the tasks that attempts to balance the workload of the hosts according to their different capabilities. It is not uncommon for planners to make use of auxiliary routines such as this to solve real-world problems, for example (Nau et al 95) describes a similar partnership for planning and scheduling in manufacturing domains.

The planner models the expected run-time of each step in order to estimate the expected runtime of the plan, based on a critical path through the partial order of components. (Although Prodigy generates totally-ordered plans, the partial order can be recovered from the causal structure.) Multiple plans are produced and the best according to the runtime estimate is returned. The final plan is converted into a detailed task specification that can be executed by a Grid service that monitors the hosts and ensures that all necessary tasks are completed prior to starting a new task.

Benefits of AI planning

Any workflow generation tool is a significant benefit to the scientist, who no longer need compose the required tasks and allocate them to hosts on the Grid by hand. We focus on the benefits of planning over the existing workflow generation approach, described earlier. First, the planner allows the user to express goals in terms of metadata, or information about the data required, rather than the logical file names. For example, the planner's top-level goal might be a pulsar search specifying the location, time, channel, instrument and settings to use. Second, the planner uses an explicit, declarative representation for workflow constraints such as program data dependencies and host constraints, and user access constraints. This makes it easier to add and modify these constraints, and to construct applications out of reusable information about the Grid and the hosts available, as we describe in the next section.

Third, the planner creates a number of alternative plans and either returns the best according to some quality criterion, or returns a set of alternatives for the user to consider. This is possible because the planner is quite efficient in this domain: a feasible plan involving hundreds of FTs can be found in under a second on a 2 GHz Pentium 4 PC. We currently use the estimated expected runtime as the quality criterion as mentioned above.

The Grid as a Test bed for Planning Research

Finding good abstract and concrete workflows involves a wide range of issues that have been investigated by the planning community, including hierarchical planning, temporal reasoning and scheduling, reasoning about resources, planning under uncertainty and interleaving planning and execution. Although we have already shown several advantages from using planning techniques for workflow generation, we anticipate more as we begin to incorporate some of the existing techniques we mention here. In addition, this list, by no means exhaustive, highlights the potential of the workflow generation problem as a test application for planning research. In the near future we plan to evaluate approaches such as plan reuse and planning under uncertainty to increase the level of WG's performance and sophistication. We also plan to

investigate the applicability of our approach to service-level composition.

Plan Quality

The workflow produced by the AI planner must be of sufficiently high quality, where the quality metric is likely to include a number of dimensions whose relative importance may vary with the application area, the user and even the specific application. These dimensions will include the overall expected runtime of the workflow, a probability of successful execution and a distribution of possible runtimes, the use of computer or data resources that are costly or restricted for the user, and application-dependent preferences on data sources and component programs. The tradeoffs between these different dimensions will be hard to predict in general for a partial plan, which is why our approach is to generate a number of alternative complete plans and test them against a global quality measure as well as using local search control. In the future, we would like to handle the requests of several users simultaneously, increasing the benefits of optimization and also making tradeoffs more complex.

Most of the work in plan quality focuses on plan length, or a sum of operator costs as the metric (Estlin & Mooney 97) although others have used more general approaches, *e.g.* (Perez 95). Some recent approaches in scheduling have had success using iterative refinement techniques (Smith & Lassila 94) in which a feasible assignment is gradually improved through successive modifications. The same approach has been applied in planning (Ambite & Knoblock 97) and is well suited to seeking high-quality plans in WG. Some work has been done on integrating planning and scheduling techniques to solve the joint task (Myers et al. 01).

A research area that is likely to be effective for this problem is the reuse of previously computed plans. Case-based planning is a powerful technique to retrieve and modify existing plans that need slight changes to work in a new situation (Veloso 94, Kambhampati 89). These approaches have potential for workflow generation because the network topology and resource characteristics are likely to be fairly stable and therefore high-quality solutions, which may take time to generate from first principles, will be good starting points for similar problems in the future.

Ontologies and Reuse of Planning Knowledge

Although much work needs to be done in the area of workflow generation, we believe that the current framework is a good foundation for developing more sophisticated techniques, which will make use of an increasing amount of information about the applications and the execution environment. Figure 5 shows additional sources of information that we would like to integrate within the workflow generation process in our future work. At the application level, we can describe the application components as services, which can be composed into new more sophisticated services. We plan to augment service-based component descriptions by developing ontologies of application components and data, which will describe the service behavior and add semantic meaning to the service interactions. Ontologies will allow us to generate abstract workflows more flexibly from user requirements that may be partially complete or specified at higher levels of abstraction than the current service descriptions. Additional information provided by performance models of the services can guide the initial composition.

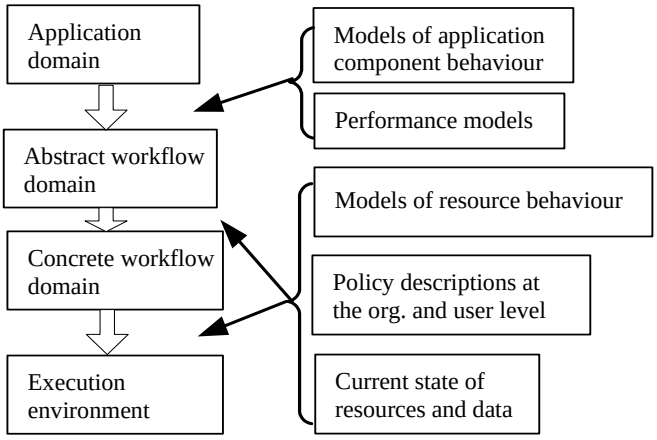

Figure 5 The workflow mapping process and the information and models required.

Ontologies will also play a very important role in generating concrete workflows. Ontologies of Grid resources will allow the system to evaluate the suitability of given resources to provide a particular application service instance. The resources that are to be allocated to various tasks can often be characterized in a domain-independent way by how they are used. For example, a computer system becomes available again once a task has been completed but a user's allocation of time on a particular machine is permanently depleted. Ontologies of resources capture these qualities *e.g.* (Smith & Becker 97, Gil & Blythe 00). Such ontologies, along with others that can capture computer system capabilities and job requirements, are key in building planning domains quickly and reliably from generic components. However, there has been little work in this area of engineering planning domains, although an example is (Long & Fox 00).

Fault-tolerant planning

In the simplest case, the planner creates a plan that is subsequently executed without a hitch. Often, however, run-time failures may result in the need to repair the plan during its execution. Planning systems can also design plans that either reduce the risk of execution failure or are more likely to be salvageable when failures take place, by reasoning explicitly about the risks during planning and searching for reliable plans, possibly including conditional branches in their execution (Boutilier, Dean et al. 1999),

(Blythe 1999). Some planners delay building parts of the plan until execution, in order to maintain a lower commitment to certain actions until key information becomes available. These approaches are likely to have high impact in the Grid computing domain, since its decentralized nature means many factors are beyond the control of the planning agent. Some resources may fail to complete tasks that are assigned to them, or may suffer long delays. In addition, network bandwidth may change greatly in a short period of time. However current techniques for handling uncertainty have high complexity, and are not useable when more than a few potential failure points need to be considered.

Multi-agent planning

When several users make workflow requests, each is likely to use a personal planning agent because of the distributed nature of the Grid. Improvements both to individual solutions and to global resource usage will be made if planners with overlapping goals can locate each other and agree to pool some of their users' resources. Issues in how such planners could locate one another, communicate shared goals and formulate, agree and commit to a shared plan have been studied in work on multi-agent planning (Tambe et al. 99) which we expect to be highly relevant to this domain.

Discussion

We have described an application of planning techniques to workflow generation on the computational grid. Key features in our approach are the use of application meta-data to describe user goals and component inputs and outputs, explicit representation of constraints both in operators and control rules, and searching a number of plans to find a high-quality solution. The planning representation also allows access policies and user preferences to be represented. The planner-based approach allows users to specify goals in terms of required metadata and finds a solution that can be executed on the Grid in time comparable to the existing tools, and with significantly better performance. A contribution of this work is the full integration of the planner in an end-to-end system, Pegasus, that constructs workflows that are executed on the Grid.

Other work in task scheduling on the Grid has focused on individual tasks, while we believe it is necessary to consider the entire workflow to optimize performance. AppLeS (Berman and Wolski 96) uses a performance metric which is tuned to each application, while the Workflow Management Package (Giacomini and Prelz 01) uses a resource broker that integrates several Grid services.

A number of AI planning techniques have been used for composing software components, for example in image processing (Lansky et al. 95, Chien and Mortensen 96; Golden and Frank 02). These systems face similar issues in modeling components for planners, but do not handle distributed resources on the network or attempt to improve plan runtime. McDermott (02) and McIlraith and Son (02) apply 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. Many of the issues that we have described here are also very important for web services composition. We believe the family of Grid application domains can inform a wide range of research interests in AI planning and Grid-related ontologies.

Acknowledgements

We gratefully acknowledge many helpful and stimulating discussions on these topics with our colleagues, including Ann Chervenak, Jihie Kim, Paul Rosenbloom, Tom Russ and Hongsuda Tangmunarunkit. 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

R. Aler and D. Borrajo. On control knowledge acquisition by exploiting human-computer interaction. *Sixth International Conference on AI Planning And Scheduling*, 2002.

J. L. Ambite and C. A. Knoblock, "Planning by Rewriting: Efficiently Generating High-Quality Plans," *Fourteenth National Conference on Artificial Intelligence*, 1997.

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.

F. Berman and R. Wolski, "Scheduling from the Perspective of the Application", *High Performance Distributed Computing*, 1996

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

C. Boutilier, T. Dean, and S. Hanks, "Planning under uncertainty: structural assumptions and computational leverage," *Journal of Artificial Intelligence Research*, vol. 11, 1999.

A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripenu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney, "Giggle: A Framework for Constructing Sclable Replica Location Services.," *Proceedings of Supercomputing 2002* (SC2002), 2002.

S. Chien and H. 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.

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

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.

E. Deelman, 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, to appear, 2003

E Deelman, J. Blythe, Y. Gil, C. Kesselman, "Pegasus: Planning for Execution in Grids", GriPhyN technical report, GRIPHYN-2002-20 (www.griphyn.org).

T. Estlin and R. Mooney , "Learning to Improve Both Efficiency and Quality for Planning" *Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

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

I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation," *Scientific and Statistical Database Management*, 2002.

F. Giacomini and F. Prelz, "Definition of Architecture, Technical Plan and Evaluation Criteria for Scheduling, Resource Management, Security and Job Description", EDG Workload Management Draft, 2001

Y. Gil and J. Blythe, "PLANET: A Shareable and Reusable Ontology for Representing Plans," AAAI Workshop on Representational Issues for Real-world Planning Systems, 2000

Globus 02, www.globus.org

K. Golden and J. Frank, "Universal Quantification in a Constraint-Based Planner". in *Sixth International Conference in AI Planning And Scheduling*, 2002

GriPhyN 02, www.griphyn.org.

J. Hoffmann, B. Nebel, The FF Planning System: Fast Plan Generation Through Heuristic Search, in: *Journal of Artificial Intelligence Research*, Volume 14, 2001, Pages 253 - 302.

S. Kambhampati, "Flexible Reuse and Modification in Hierarchical Planning: a Validation-Structure Based Approach", PhD Thesis, University of Maryland, 1989

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

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

K. Myers, S. Smith, D. Hildum, P. Jarvis, and R. d. Lacaze, "Integrating Planning and Scheduling through Adaptation of Resource Intensity Estimates," *Sixth European Conference on Planning*, 2001.

D. Long and M. Fox, "Recognizing and Exploiting Generic Types in Planning Domains," *Fifth International Conference on AI Planning and Scheduling*, 2000

D. McDermott, "Estimated-Regression Planning for Interactions with Web Services". in *Sixth International Conference in AI Planning and Scheduling*, 2002

NPACI 02, "Telescience, https://gridport.npaci.edu/Telescience/."

D. Nau, W. Regli, and S. Gupta. "AI Planning Versus Manufacturing-Operation Planning: A Case Study." *International Joint Conference on Artificial Intelligence*, 1995.

M. A. Pérez., "Learning Search Control Knowledge to Improve Plan Quality", PhD thesis, School of Computer Science, Carnegie Mellon University, July 1995

SCEC 02. Southern California Earthquake Center's Community Modeling Environment, http://www.scec.org/cme/.

S. F. Smith and M. Becker, "An Ontology for Constructing Scheduling Systems," AAAI Spring Symposium on Ontological Engineering, Stanford University, 1997.

S. F. Smith and O. Lassila, "Toward the Development of Mixed-Initiative Scheduling Systems," in *Proceedings ARPA-Rome Laboratory Planning Initiative Workshop*. Tucson, AZ, 1994

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

M. M. Veloso, *Planning and Learning by Analogical Reasoning*: Springer Verlag, December 1994.

M. Veloso, J. Carbonell, A. Perez, D. Borrajo, E. Fink, and J. Blythe, "Integrating Planning and Learning: The PRODIGY Architecture," *Journal of Experimental and Theoretical AI*, vol. 7, pp. 81-120, 1995.

C.-E. Wulz, "CMS - Concept and Physics Potential," *Second Latin American Symposium on High Energy Physics* (II-SILAFAE), San Juan, Puerto Rico, 1998.