Chapter 1

WORKFLOW MANAGEMENT IN GRIPHYN

Ewa Deelman, James Blythe, Yolanda Gil, and Carl Kesselman
USC Information Sciences Institute

Abstract This chapter describes the work done within the NSF-funded GriPhyN project in the area of workflow management. The targeted workflows are large both in terms of the number of tasks in a given workflow and in terms of the total execution time of the workflow, which can sometimes be on the order of days. The workflows represent the computation that needs to be performed to analyze large scientific datasets produced in high-energy physics, gravitational physics, and astronomy. This chapter discusses issues associated with workflow management in the Grid in general and also provides a description of the Pegasus system, which can generate executable workflows.

Keywords: Workflow management, planning, reliability, Grid-enabled scientific applications.

1. Introduction

As the complexity of Grid applications increases, it becomes ever more important to provide a means to manage application composition and the generation of executable application workflows. This chapter describes the GriPhyN project, which explores, among others, issues of workflow management in Grids in support of large-scale scientific experiments.

GriPhyN (Grid Physics Network [Grib]) is an NSF-funded project that aims to support large-scale data management in physics experiments such as high-energy physics, astronomy, and gravitational wave physics in the wide area. GriPhyN puts data both raw and derived under the umbrella of *Virtual Data*. A user or application can ask for a virtual data product using application-specific metadata without needing to know whether the data is available on some storage system or if it must to be computed. To enable the computation, the system needs to have a recipe (possibly in a form of a workflow) that describes the data products.

To satisfy the user's request, GriPhyN will schedule the necessary data movements and computations to produce the requested results. To perform the scheduling, resources need to be discovered, data needs to be located and staged, and the computation needs to be scheduled. The choice of particular computational resources and data storage systems will depend on the overall performance and reliability of entire computation. GriPhyN uses the Globus Toolkit [GLO] as the basic Grid infrastructure and builds on top of it high-level services that can support virtual data requests. These services include request planning, request execution, replica selection, workflow generation, and others.

The following section describes the workflow generation process in general, starting from the application level, which contains application component descriptions, down to the execution level, where the refined workflow is ready for execution. Section 3 provides a brief overview of the issues involved in workflow management in the Grid in general. Particular aspects of the problem such as workflow performance and workflow fault tolerance are addressed. Section 4 describes the Pegasus (Planning for Execution in Grids) [DBG+03b] framework that was developed to provide an experimental platform for workflow management research. Two specific configurations of Pegasus are discussed. The first takes a description of the workflow in terms of logical filenames and logical application component names and produces an executable workflow. The second builds an executable workflow based on metadata that describes the workflow at the application level, allowing scientists to request data in terms that are scientifically meaningful. The chapter concludes with related work on the topic and future research directions.

2. Generating Application Workflows

Scientists often seek specific data products that 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, the user would like the results of any intermediate filtering steps performed to be available in location I, perhaps to check the filter results for unusual phenomena or perhaps to extract some salient features to the metadata of the final results. The process of mapping this type of user request onto jobs to be executed in a Grid environment can be decomposed into two steps, as shown in Figure 1.1.

1 **Generating an abstract workflow:** Selecting and configuring application components to form an abstract workflow. The application components are selected by examining the specification of their capabilities and checking to see if they can generate the desired data products. They

are configured by assigning input files that exist or that can be generated by other application components. The abstract workflow specifies the order in which the components must be executed. More specifically, the following steps need to be performed:

(a) Find which application components generate the desired data products, which in our example is a frequency spectrum of the desired characteristics. Let one such component be C_n. Find which inputs that component needs, check if any inputs are available and if so let the corresponding files be $I_1 \ldots I_j$. If any input is required in formats that are not already available, then find application components that can produce that input, and let one such component be C_{n-1}. This process is iterated until the desired result can be generated from a composition of available components that can operate over available input data, namely $C_1 \ldots C_n$ and $I_1 \ldots I_m$ respectively.

(b) Formulate the workflow that specifies the order of execution of the components $C_1 \ldots C_n$. This is what we call an *abstract workflow*. Please note that at this level the components and files are referred to by their *logical names*, which uniquely identify the components in terms of their functionality and the data files in terms of their content. A single logical name can correspond to many actual executables and physical data files in different locations.

2 **Generating a concrete workflow:** Selecting specific resources, files, and additional jobs required to form a concrete workflow that can be executed in the Grid environment. In order to generate a *concrete workflow*, each component in the abstract workflow is turned into an executable job by specifying the locations of the physical files of the component and data, as well as the resources assigned to the component in the execution environment. Additional jobs may be included in the concrete workflow, for example, jobs that transfer files to the appropriate locations where resources are available to execute the application components. Specifically, the following steps need to be performed:

(a) Find the physical locations (i.e., physical files) of each component $C_1 \ldots C_n$: C_1-$pf \ldots C_n$-pf.

(b) Check the computational requirements of C_1-$pf \ldots C_n$-pf and specify locations $L_1 \ldots L_n$ to execute them according to the required and available resources.

(c) Determine the physical locations of the input data files I_1-$pf \ldots I_m$-pf, and select locations are more appropriate given $L_1 \ldots L_n$.

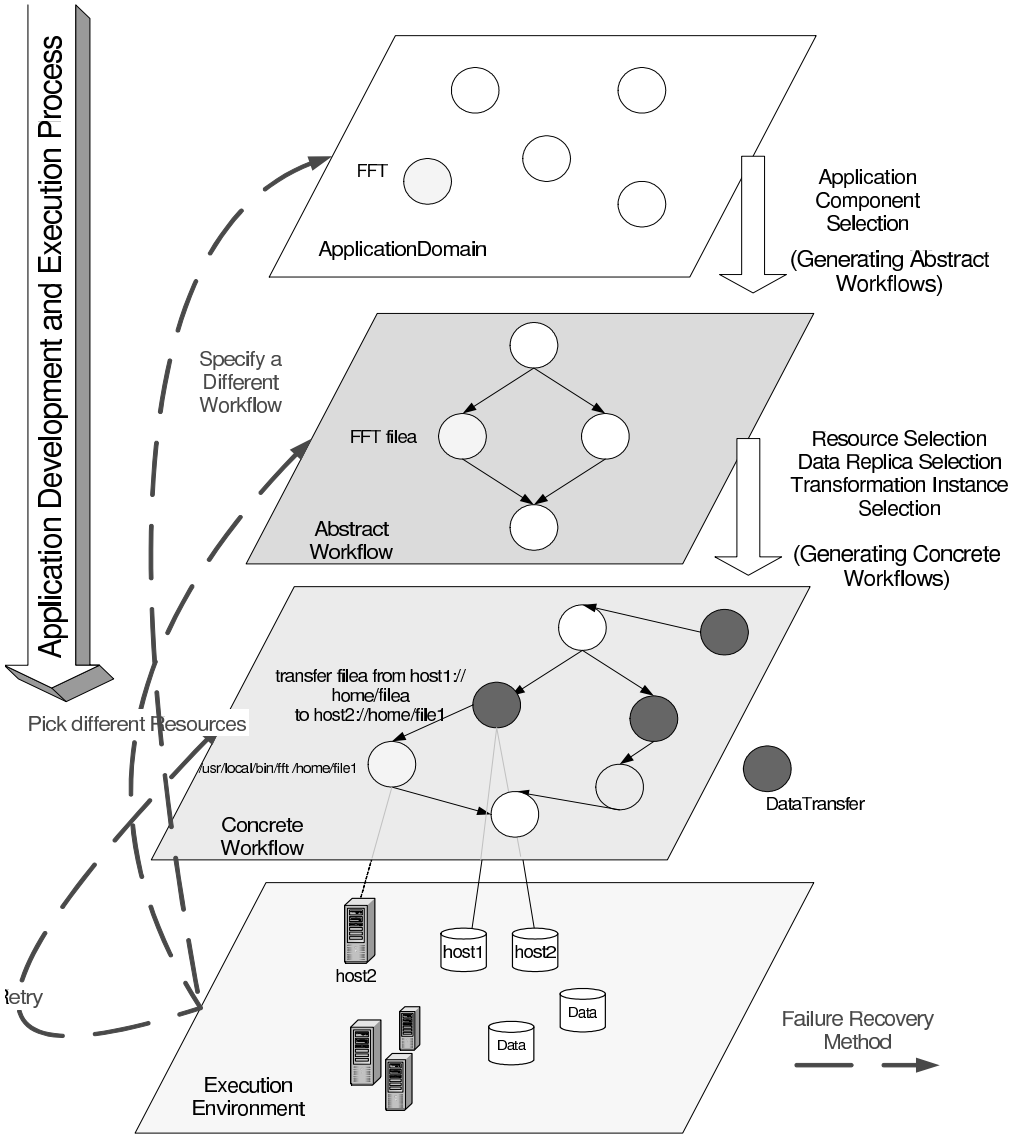

Figure 1.1. The process of developing data intensive applications for Grid environments.

(d) Augment the workflow description to include jobs $K_1 \ldots K_{m+n}$ to move component and input data files (C_1-$pf \ldots C_n$-pf and I_1-$pf \ldots I_m$-pf) to the appropriate target locations $L_1 \ldots L_n$.

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 important reasons that automating this process not only desirable but necessary:

- Usability: Users are 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 Discovery Service (MDS) [CFFK01], to find the available and appropriate computational resources for the computational requirements of a component (step 2b). The user also needs to query the Replica Location Service (RLS) [CDF+02] to find the physical locations of the data (step2c).

- Complexity: In addition to requiring scientists to become Grid-enabled users, the process of workflow generation may be complex and time consuming. Note that in each step the user makes 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. Many different interdependencies may occur among components, and as a result it may be difficult to determine which choice of component and or resource to change and what would be a better option that leads to a feasible solution.

- Solution cost: Lower cost solutions are highly desirable in light of the high cost of some of the computations and the user's limitations in terms of resource access. 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) [FKT01] is desirable. This requires reasoning about individual user choices in light of all other user choices. It may be desirable to find possible common jobs that can be included across user workflows and executed only once.

While addressing the first three points above would enable wider accessibility of the Grid to users, the last point, minimizing global cost, simply cannot be handled by individual users and will likely need to be addressed at the architecture level. In addition, there are many policies that limit user access to

resources, and that needs to be taken into account to accommodate as many users as possible as they contend for limited resources.

An additional issue is the reliability of execution. When the execution of the job fails, in today's Grid framework the recovery consists of resubmitting that job for execution on the same resources. (In Figure 1.1 this is shown as retry.) However, it is also desirable to be able to choose a different set of resources when tasks fail. This choice needs to be performed at the abstract workflow level. Currently, there is no mechanism for opportunistically re-running the remaining tasks in the workflow to adapt to the dynamic situation of the environment. Moreover, if any job fails repeatedly it would be desirable for the system to assign an alternative component to achieve the same overall user goals. This approach would need to be performed at the application level where there is an understanding of how different application components relate to each other.

3. Workflow Management Issues in Grids

Grid computing has taken great strides in the last few years. The basic mechanisms for accessing remote resources have been developed as part of the Globus Toolkit and are now widely deployed and used. Among such mechanisms are: Information services, which can be used to find the available resources and select the resources which are the most appropriate for a task; Security services, which allow users and resources to mutually authenticate and allow the resources to authorize users based on local policies; Resource management, which allows for the scheduling of jobs on particular resources; and Data management services, which enable users and applications to manage large, distributed, and replicated data sets.

With the use of the above mechanisms, one can manually find out about the resources and schedule the desired computations and data movements. However, this process is time consuming and can potentially be complex. As a result, it is becoming increasingly necessary to develop higher-level services that can automate the process and provide an adequate level of performance and reliability. Among such services are workflow generators, which can construct concrete or abstract workflows or both, and executors, which given a concrete workflow execute it on the specified Grid resources.

When generating workflows, one can take into account metrics such as the overall runtime of the workflow, and the probability that the workflow will be executed successfully. An important aspect of obtaining desired results from workflow execution is the relationship between workflow generator and workflow executor. In the following section we explore this interaction.

3.1 Planning for Optimal Performance

In this section, we focus on the behavior of a workflow generator (here referred to as the planner) and its interaction with an executor, such as Condor-G/DAGMan [FTLT01]. The relationship and interfaces between the planner and the executor are important both from the standpoint of planning and fault tolerance. For this discussion, we assume that multiple requests to the system are handled independently.

We first describe two ways that the process of workflow generation and execution can vary, and use them to characterize the different interactions between the planner and executor. Decisions about the workflow, such as the resource on which to run a particular task, can be made with reference to information about the whole workflow or just the task at hand. The former we refer to as a *global decision* and the latter as a *local decision*. Typically, executors make local decisions about tasks, while some planners make global decisions. Decisions can be made as soon as possible (*early*) or just before the task is executed (*in-time*). Typically, an executor makes any decisions just before task execution. To reason globally, a planner needs to make decisions before any tasks are executed, but they may be revisited nearer the execution time.

One can imagine two extremes for how work is divided between the planner and executor: on one hand, the planner can make a fully detailed workflow based on the current information about the system that includes where the tasks need to execute, the exact location from where the data needs to be accessed for the computation, etc., leaving very few decisions to the executor. We call this approach *full-plan-ahead* , and it can be seen to result in an early, global decision-maker. At the other extreme, the planner can leave many decisions up to the executor. It can, for example, give the executor a choice of compute platforms to use, a choice of replicas to access, etc. At the time the executor is ready to perform the computation or data movements, the executor can consult the information services and make local decisions (here called *in-time local scheduling*).

The benefit of the full-plan-ahead approach is that the planner can aim to optimize the plan based on the entire structure of the workflow. However, because the execution environment can be very dynamic, by the time the tasks in the workflow are ready to execute, the environment may have changed so much that the execution is now far from the expected performance. Additionally, the data may no longer be available at the location assumed by the planner, leading to an execution-time error. Clearly, if the planner constructs fully instantiated plans, it must be able to adapt to the changing conditions and be able to quickly re-plan.

Faults due to the changing environment are far less likely to occur when the executor is given the freedom to make decisions as it processes the abstract

workflow, as in the in-time-scheduling approach. At the time a task is to be scheduled, the executor can use the information services to find out about the state of the resources and the location of the data and make a locally optimal decision. However, because the executor does not use global information about the request, it can make potentially expensive decisions.

Another approach is *in-time global scheduling*. Here, the planner provides an abstract workflow description to the executor, which in turn contacts the planner when it is ready to schedule a task and asks for the execution location for the task. The planner can at that time make a decision that uses global information about the workflow. In the meantime, the planner can continually plan ahead, updating its views of the state of the Grid. If the resources available to the planner itself are not able to support continuous planning, the planner can run periodically or at a time when a new decision needs to be made. Because the planner can make decisions each time a task is scheduled, it can take many factors into consideration and use the most up-to-date information. The drawback, however, is that the control may be too fine, and may result in high communication overheads and a large amount of computation due to re-planning.

Clearly, there is no single best solution for all applications, since these approaches can have very different characteristics. For example, consider a data-intensive application, where the overall runtime is driven by data movement costs, in which some task A requires file A_i as input and produces output file B_i that is input for task B. If B can only run in a few locations, and B_i is large relative to A_i, then a local decision-maker such as the in-time local scheduler is likely to create a very poor plan, because it will first choose a location to run A that is close to its required input A_i, without regard to the requirement to then move the output B_i to a location where B can run. Global reasoning is required to find the best workflow.

To decide on the best strategy, however, we also need to talk about how quickly the Grid environment changes. If the availability of the needed resources changes quickly enough that an early decision for where to execute B may be poor or invalid by the time A completes, and if B can run at many locations or B_i is relatively small, then in-time local scheduling would be better than full-ahead-planning. If the domain changes quickly and requires global scheduling, then in-time global scheduling is probably the best strategy. However, it is also possible that global scheduling itself will take significant time to complete compared with the savings it produces. So if the planning time is large compared with the execution savings, in-time local scheduling or full-ahead-planning may still be preferred. Similar arguments can be made for compute-intensive applications.

Another factor in workflow management is the use of reservations for various resources such as compute hosts, storage systems, and networks. As these

technologies advance, we believe that the role of full-plan-ahead and in-time global scheduling systems will increase.

Up to now, we have considered only the case where the workflow management system handles only one request at a time. The problem becomes more complex when the system is required to find efficient workflows for multiple requests and to accommodate various usage policies and community and user priorities. In this case, full-plan-ahead planners have the the advantage of being able to compare and in some cases optimize end-to-end workflows (through exhaustive search.) However, full-plan-ahead planners still face the challenge of needing to be able to react to the changing system state.

The nature of this problem seems to indicate that the workflow management system needs to be flexible and adaptable in order to accommodate various applications behavior and system conditions. Because an understanding of the application's behavior is crucial to planning and scheduling the execution, application performance models are becoming ever more necessary.

3.2 Planning for Fault Tolerance

Performance is not the sole criterion for workflow generation. Fault tolerance of the workflow execution also needs to be considered. The abstract workflow generator can construct a few different workflows, allowing the lower levels to decide which abstract workflow to implement. For example one workflow can rely on data that is available only from very unreliable resources, whereas another can use resources that are expected to be reliable. In that case, the concrete workflow generator may decide to instantiate the second abstract workflow.

The concrete workflow generator itself may provide multiple concrete workflows, leaving it up to the executor to choose between them, or allowing it to execute alternative workflows in cases of a failure of a particular concrete workflow. The generator may also include "what if" scenarios in the concrete workflows. The executor would then execute the dependent nodes in the workflow based on the success or failure (or possibly some other metric) of the execution of the parent nodes. The generator can also instantiate the abstract workflow based on the information it has about the reliability of the system components. To achieve this, monitoring of the behavior of various system components needs to be performed. This information can also be used by the executor, which can replicate tasks if it is scheduling jobs on compute resources viewed as unreliable or on compute resources connected by a poorly performing network. The execution environment itself can incorporate fault tolerance measures, by checkpointing the state and providing restart capabilities. The executor may also want to adapt the checkpointing interval for the jobs based on which resources the job is running.

So far we have discussed the general issues in workflow management in GriPhyN. In the following sections we focus on the first implementation of a workflow generation system.

4. Pegasus-Planning for Execution in Grids

Pegasus, which stands for Planning for Execution in Grids, was developed at ISI as part of the GriPhyN project. Pegasus is a configurable system that can map and execute complex workflows on the Grid. Currently, Pegasus relies on a full-ahead-planning to map the workflows. As the system evolves, we will incorporate in-time local scheduling and in-time global scheduling as well.

Pegasus and its concrete workflow generator has been integrated with the GriPhyN Chimera system [F+02]. Chimera provides a catalog that can be used to describe a set of application components and then track all the data files produced by executing those applications. Chimera contains the mechanism to locate the recipe to produce a given logical file in the form of an abstract workflow. In the Chimera-driven configuration, Pegasus receives an abstract workflow (AW) description from Chimera, produces a concrete workflow (CW), and submits the CW to Condor-G/DAGMan for execution. The workflows are represented as Directed Acyclic Graphs (DAGs). The AW describes the transformations and data in terms of their logical names; it has information about all the jobs that need to be performed to materialize the required data.

The Pegasus *Request Manager* sends this workflow to the *Concrete Workflow Generator* (CWG.) The latter queries the Globus Replica Location Service (RLS) to find the location of the input files, as specified by their logical filenames in the DAG. The RLS returns a list of physical locations for the files.

The information about the available data is then used to optimize the concrete workflow from the point of view of Virtual Data. The optimization is performed by the *Abstract DAG Reduction* component of Pegasus. If data products described within the AW are found to be already materialized, Pegasus reuses them and thus reduces the complexity of the CW.

The following is an illustration of the reduction algorithm. Figure 1.2 shows a simple abstract workflow 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*.

If we assume that $F.c2$ is already available on some storage system (as indicated by the RLS), the CWG reduces the workflow to three components, namely *Extract*, *Resample*, and *Concat*. It then adds the transfer nodes for transferring $F.c2$ and $F.a$ from their current locations. It also adds transfer

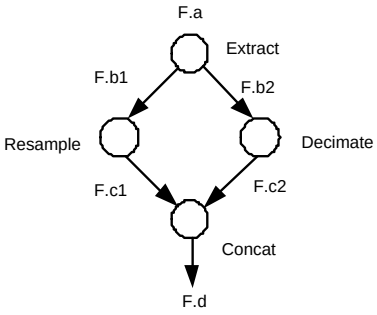

Figure 1.2. An example abstract workflow.

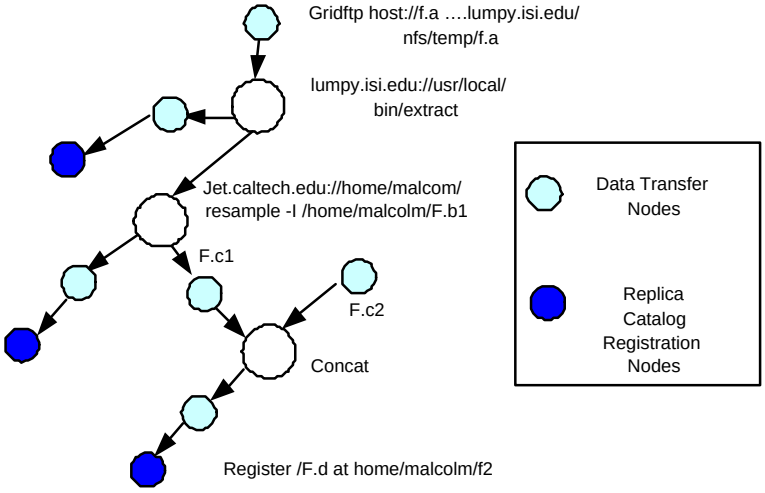

Figure 1.3. An example reduced, concrete workflow.

nodes between jobs that will run on different locations. For example, if the executables for *Resample* and *Concat* are available at two different locations, *Fc.1* will have to be transferred. Finally, the CWG adds output transfer nodes to stage data out and registration nodes if the user requested that the resulting data be published and made available at a particular location. The concrete workflow for this scenario is shown in Figure 1.3. Once the workflow is instantiated in the form of a DAG, software such as DAGMan and Condor-G is used to schedule the jobs described in the nodes of the DAG onto the specified resources in their specified order.

In general, the reduction component of Pegasus assumes that it is more costly to execute a component (a job) than to access the results of the component if that data is available. For example, some other user in the VO may

have already materialized part of the entire required data set. If this information is published into the RLS, then Pegasus can utilize this knowledge and obtain the data thus avoiding possible costly computations. As a result, some components that appear in the abstract workflow do not appear in the concrete workflow.

During the reduction, the output files of the jobs that do not need to be executed are identified. Any antecedents of the redundant jobs that do not have any unmaterialized descendants are removed. The reduction algorithm is performed until no further nodes can be reduced.

The CWG component of Pegasus maps the remaining abstract workflow onto the available resources. Currently the information about the available resources is statically configured. However, at this time, we are incorporating the dynamic information provided by the Globus Monitoring and Discovery Service (MDS2) into the decision-making process.

The CWG also checks for the feasibility of the abstract workflow. It determines the root nodes for the abstract workflow and queries the RLS for the existence of the input files for these components. The workflow can only be executed if the input files for these components can be found to exist somewhere in the Grid and are accessible via a data transport protocol, such as GridFTP [ABB+02]. The *Transformation Catalog* [DKM01] is queried to determine if the components are available in the execution environment and to identify their locations. The Transformation Catalog performs the mapping between a logical component name and the location of the corresponding executables on specific compute resources. The Transformation Catalog can also be used to annotate the components with the creation information as well as component performance characteristics. Currently the CWG picks a random location to execute from among the possible locations identified by the catalog.

Transfer nodes are added for any of these files that need to be staged in, so that each component and its input files are at the same physical location. If the input files are replicated at several locations, the CWG currently picks the source location at random.

Finally transfer nodes and registration nodes, which publish the resulting data products in the RLS, are added if the user requested that all the data be published and sent to a particular storage location.

In order to be able to execute the workflow, Pegasus' *Submit File Generator* generates submit files which are subsequently sent to DAGMan for execution. Their execution status is then monitored within Pegasus.

In the abstract-workflow-driven configuration, Pegasus has been shown to be successful in mapping workflows for very complex applications such as the Sloan Digital Sky Survey [A+02] and the Compact Muon Source [DBG+03a]. Pegasus took the abstract workflow with hundreds of nodes, instantiated it, and oversaw its successful execution on the Grid.

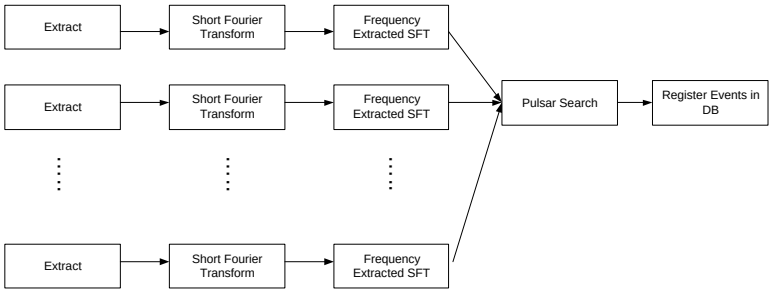

Figure 1.4. The stages in the LIGO pulsar search.

4.1 Planning at the Level of Metadata

Pegasus can also be configured to generate a workflow from a metadata description of the desired data product. This description is application-specific and is meaningful to the domain expert. This configuration of Pegasus was applied to one of the physics projects that are part of GriPhyN, the Laser Interferometer Gravitational-Wave Observatory, (LIGO [LIG, A^+92, BW99]). LIGO is a distributed network of interferometers whose mission is to detect and measure gravitational waves predicted by general relativity, Einstein's theory of gravity. One well-studied source of gravitational waves is the motion of dense, massive astrophysical objects such as neutron stars or black holes. Other signals may come from supernova explosions, quakes in neutron stars, and pulsars.

LIGO scientists have developed applications that look for gravitational waves possibly emitted by pulsars. The characteristics of the search are the position in the sky, frequency range of the expected signal, and time interval over which to conduct the search. LIGO scientists would like to make requests for data using just such metadata.

To support this type of request, the pulsar application is decomposed into several stages (each an application in itself). Some of these stages may contain thousands of independent applications. The first stage (Figure 1.4) consists of extracting the gravitational wave channel from the raw data. Some processing geared towards the removal (cleaning) of certain instrumental signatures also needs to be done at that time. The pulsar search is conducted in the frequency domain; thus, Fourier Transforms, in particular Short Fourier Transforms (SFTs), are performed on the long duration time frames (second stage). Since the pulsars are expected to be found in a small frequency range, the frequency interval of interest is extracted from the STFs (third stage). The resulting power spectra are used to build the time-frequency image, which is analyzed for the presence of pulsar signatures (fourth stage). If a candidate signal with a good signal-to-noise ratio is found, it is placed in LIGO's event

database (final stage). Each of the stages in this analysis can be described uniquely by LIGO-specific metadata, as seen in Table 1.1. For example, the SFT is described by the channel name and the time interval of interest. These descriptions need to made available to the workflow generation system, here Pegasus, which translates them into abstract and/or concrete workflows.

Stage	Metadata
Extract	Channel name, time interval
SFT	Channel name, time interval
Frequency Extracted SFT	Channel name, time interval, frequency band
Pulsar Search	Channel name, time interval, frequency band, location in the sky

Table 1.1. Metadata describing the LIGO pulsar search stages.

4.2 Metadata-driven Pegasus

Given attributes such as time interval, frequency of interest, location in the sky, etc., Pegasus was configured to support the LIGO pulsar search. Details about the search can be found in [D+02]. Pegasus is currently able to produce any virtual data products present in the LIGO pulsar search. Figure 1.5 shows Pegasus configured for such a search.

Pegasus receives the metadata description from the user. The *Current State Generator* queries the *Metadata Catalog Service* (MCS) [C+02], to perform the mapping between application-specific attributes and logical file names of existing data products (step 3 in Figure 1.5). The MCS, developed at ISI, provides a mechanism for storing and accessing metadata, which is information that describes data files or data items. The MCS allows users to query based on attributes of data rather than names of files containing the data. In addition to managing information about individual logical files, the MCS provides management of logical collections of files and containers that consist of small files that are stored, moved, and replicated together.

The MCS can keep track of metadata describing the logical files such as:

- Information about how data files were created or modified, including the creator or modifier, the creation or modification time, what experimental apparatus and input conditions produced the file, or what simulation or analysis software was run on which computational engine with which input parameters, etc.

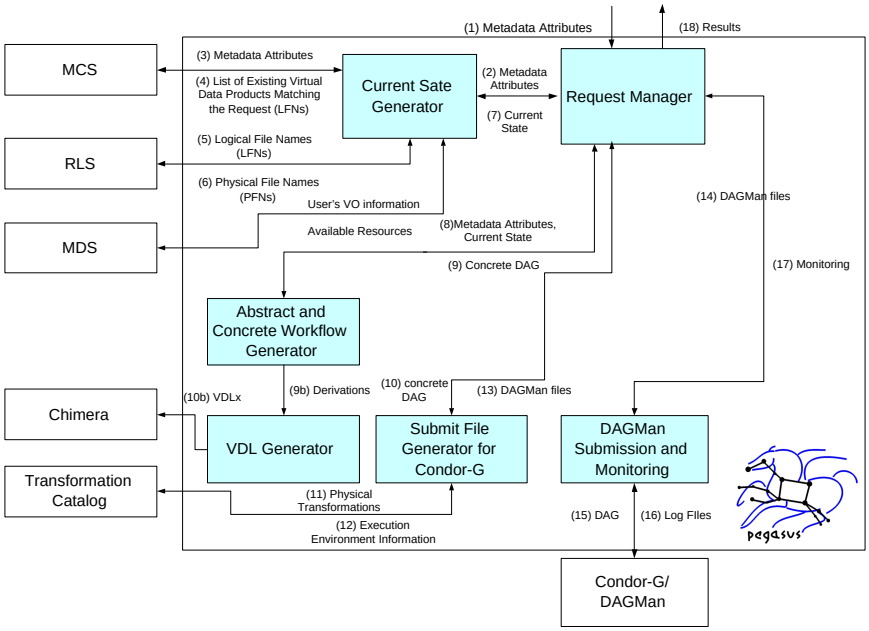

Figure 1.5. Pegasus configured to perform the LIGO pulsar search, based on application-specific metadata.

- A description of what the data stored in a file represent, for example, time series data collected at a particular instrument.

Once the metadata for a product is specified, MCS determines the corresponding logical file and determines the metadata and logical filenames for all other sub-products that can be used to generate the data product, and returns them to the Current State Generator as shown in Figure 1.5. The Current State Generator then queries the RLS to find the physical locations of the logical files (step 5). We are also interfacing the Current State Generator to MDS2 to find the available Grid resources. Currently, this information is statically configured. The metadata and the current state information are then passed to the *Abstract and Concrete Workflow Generator* (ACWG).

AI-based planning technologies are used to construct both the abstract and concrete workflows [BDG+03, BDGK03]. The AI-based planner models the application components along with data transfer and data registration as operators. The parameters of the planner operators include the location where the component can be run. Some of the effects and preconditions of the operators can capture the data produced by components and their input data dependencies. State information used by the planner includes a description of the available resources and the files already registered in the RLS. 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. The planner generates the concrete workflow (in the form of a DAG) necessary to satisfy the user's request. The planner takes into account the state of the network to come up with an efficient plan, and in some domains finds an optimal plan using an exhaustive search that is made feasible by careful pruning. This planner also reuses existing data products when applicable.

The plan generated specifies the sites at which the job should be executed and refers to the data products in terms of metadata. This plan is then passed to the submit file generator for Condor-G (step 10). The submit file generator determines first the logical names for the data products by querying the MCS and then the physical names by querying the RLS. In addition, it queries the Transformation Catalog to get the complete paths for the transformations at the execution locations described in the concrete DAG.

Pegasus also contains a Virtual Data Language generator that can populate the Chimera catalog with newly constructed derivations (step 9b). This information can be used later to obtain provenance information about the derived data products.

In this configuration, Pegasus similarly generates the necessary submit files and sends the concrete workflow to DAGMan for execution (step 15).

As a result of execution of the workflow, the newly derived data products are registered both in the MCS and RLS and thus are made available to subsequent requests.

The metadata-driven Pegasus configuration that supported the LIGO pulsar search was first demonstrated at the SC 2002 conference held in November in Baltimore, MD. For this demonstration the following resources were used:

- Caltech (Pasadena, CA): Data Storage Systems and the LIGO Data Analysis System (LDAS), which performed some of the stages of the LIGO analysis shown in Figure 1.4.
- ISI (Marina del Rey, CA): Condor Compute Pools, Data Storage Systems, Replica Location Services and Metadata Catalog Services.
- University of Wisconsin (Milwaukee, WI): Condor Compute Pools and Data Storage Systems.

The requests for pulsar searches were obtained using an automatic generator that produced requests both for approximately 1300 known pulsars as well as for random point-searches in the sky. A user was also able to request a specific pulsar search by specifying the metadata of the required data product through a web-based system. Both the submission interfaces as well as all the compute and data management resources were Globus GSI (Grid Security

Infrastructure) [WSF+03] enabled. Department of Energy Science Grid [Gria] issued X509 certificates were used to authenticate to all the resources.

During the demonstration period and during a subsequent run of the system, approximately 200 pulsar searches were conducted (both known as well as random), generating approximately 1000 data products involving around 1500 data transfers. The data used for this demonstration was obtained from the first scientific run of the LIGO instrument. The total compute time taken to do these searches was approximately 100 CPU hrs. All the generated results were transferred to the user and registered in the RLS; the metadata for the products placed in the MCS as well as into LIGO's own metadata catalog. Pegasus also generated the corresponding provenance information using the Virtual Data Language and used it to populate in the Chimera Virtual Data Catalog.

The job execution was monitored by two means. For each DAG, a start and end job was added, which logged the start time and the end time for the DAG into a MySQL database. This information was then published via a web interface. Another script-based monitor parsed the Condor log files at the submit host to determine the state of the execution and published this information onto the web.

5. Related Work

In the area of AI planning techniques, the focus is on choosing a set of actions to achieve given goals, and on scheduling techniques that focus on assigning resources for an already chosen set of actions. Some recent approaches in scheduling have had success using iterative refinement techniques [SL94] in which a feasible assignment is gradually improved through successive modifications. The same approach has been applied in planning and is well suited to ACWG, where plan quality is important [AK97]. Some work has been done on integrating planning and scheduling techniques to solve the joint task [M+01].

Central to scheduling large complex workflows is the issue of data placement, especially when the data sets involved are very large. In CWG, we give preference to the resources where the input data set is already present. Ranganathan and Foster, in [RF02a, RF02b] and Chapter **??**, study the data in the Grid as a tiered system and use dynamic replication strategies to improve data access, and achieve significant performance improvement when scheduling is performed according to data availability while also using a dynamic replication strategy.

While running a workflow on the Grid makes it possible to perform large computations that would not be possible on a single system, it leads to a certain loss of control over the execution of the jobs, as they may be executed in different administrative domains. To counter this, there are other systems try to provide Quality of Service guarantees required by the user while submitting

the workflow to the Grid. Nimrod-G [ABG02, BAG00] uses the information from the MDS to determine the resource that is selected to meet the budget constraints specified by the user, while Keyani et al. [KSW02] monitors a job progress over time to ensure that guarantees are being met. If a guarantee is not being met, schedules are recalculated.

Each of the systems mentioned above are rigid because they use a fixed set of optimization criteria. In GriPhyN we are developing a framework for a flexible system that can map from the abstract workflow description to its concrete form and can dynamically change the optimization criteria.

6. Future Directions

The efforts in workflow planning presented in this chapter lay the foundation for many interesting research avenues.

As mentioned in Section 3.2, fault tolerance is an important issue in the Grid environment, where runtime failures may result in the need to repair the plan during its execution. The future work within Pegasus may include planning strategies that will design plans to either reduce the risk of execution failure or to be salvageable when failures take place. Current AI-based planners can explicitly reason about the risks during planning and searching for reliable plans, possibly including conditional branches in their execution. 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.

Another area of future research is the exploration of the use of ontologies to provide rich descriptions of Grid resources and application components. These can then be used by the workflow generator to better match the application components to the available resources.

Acknowledgments

We would like to thank Gaurang Mehta, Gurmeet Singh, and Karan Vahi for the development of the Pegasus system. We also gratefully acknowledge many helpful and stimulating discussions on these topics with our colleagues Ian Foster, Michael Wilde, and Jens Voeckler. The Chimera system mentioned in this paper has been jointly developed by the University of Southern California Information Sciences Institute, the University of Chicago, and Argonne National Laboratory. Finally, we would like to thank the following LIGO scientists for their contribution to the development of the Grid-enabled LIGO software: Kent Blackburn, Phil Ehrens, Scott Koranda, Albert Lazzarini, Mary Lei, Ed Maros, Greg Mendell, Isaac Salzman, and Peter Shawhan. This research was supported in part by the National Science Foundation under grants ITR-0086044(GriPhyN) and EAR-0122464 (SCEC/ITR).

References

[A+92] A. Abramovici et al. Ligo: The laser interferometer gravitational-wave observatory (in large scale measurements). *Science*, 256(5055), 1992.

[A+02] J. Annis et al. Applying chimera virtual data concepts to cluster finding in the sloan sky survey. In *Supercomputing*., 2002.

[ABB+02] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke. Gridftp protocol specification. Technical report, Global Grid Forum, September 2002.

[ABG02] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the nimrod-g resource broker. *Future Generation Computer Systems*, XXX:XXX, 2002.

[AK97] J. e. L. Ambite and C .A. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In *Proceednigs of Fourteenth National Conference on Artificial Intelligence*, 1997.

[BAG00] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*. IEEE Computer Society Press, USA, 2000.

[BDG+03] Jim Blythe, Ewa Deelman, Yolanda Gil, Carl Kesselman, Amit Agarwal, Gaurang Mehta, and Karan Vahi. The role of planning in grid computing. In *International Conference on Automated Planning and Scheduling*, 2003.

[BDGK03] Jim Blythe, Ewa Deelman, Yolanda Gil, and Carl Kesselman. Transparent grid computing: a knowledge-based approach. In *Innovative Applications of Artificial Intelligence Conference*, 2003.

[BW99] B. C. Barish and R. Weiss. Ligo and the detection of gravitational waves. *Physics Today*, 52(10):44, 1999.

[C+02] A. Chervenak et al. A metadata catalog service for data intensive applications. Technical report, GriPhyN, 2002.

[CDF+02] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, B. Tierney, and Giggle. Giggle: A framework for constructing scalable replica location services. In *Proceedings of Supercomputing 2002 (SC2002)*, 2002.

[CFFK01] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing*, August 2001.

[D+02] E. Deelman et al. Griphyn and ligo, building a virtual data grid for gravitational wave scientists. In *11th Intl Symposium on High Performance Distributed Computing*, 2002.

[DBG+03a] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbee, Richard Cavanaugh, and Scott Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1, 2003.

[DBG+03b] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Scott Koranda, Albert Lazzarini, and Maria Alessandra Papa. From metadata to execution on the grid: The pegasus pulsar search. Technical Report 2003-15, GRIPHYN, http://www.griphyn.org/documents, 2003.

[DKM01] Ewa Deelman, Carl Kesselman, and Gaurang Mehta. Transformation catalog design for griphyn, prototype of transformation catalog schema. Technical Report 2001-17, GRIPHYN, http://www.griphyn.org/documents, 2001.

[F+02] I. Foster et al. Chimera: A virtual data system for representing, querying, and automating data derivation. In *14th International Conference on Scientific and Statistical Database Management (SSDBM'02)*, 2002.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal*

of High Performance Computing Applications, 15(3):200–222, 2001. http://www.globus.org/research/papers/anatomy.pdf.

[FTLT01] J. Frey, T. Tannenbaum, M. Livny, and S. Tuecke. Condor-g: A computation managament agent for multi-institutional grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*, 2001. San Francisco, California.

[GLO] Globus. http://www.globus.org.

[Gria] DOE Science Grid. http://www.doegrids.org.

[Grib] Grid physics network (griphyn). http://www.griphyn.org.

[KSW02] P. Keyani, N. Sample, and G. Wiederhold. Scheduling under uncertainty: Planning for the ubiquitous grid. In *Proceedings of the Fifth International Conference on Coordination Models and Languages, COORD 2002*. Stanford Database Group., 2002.

[LIG] Ligo - laser interferometer gravitational wave observatory. http://www.ligo.caltech.edu.

[M+01] K. Myers et al. Integrating planning and scheduling through adaptation of resource intensity estimates. In *Proceedings of the 6th European Conference on Planning (ECP-01)*, 2001.

[RF02a] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data intensive applications. In *International Symposium for High Performance Distributed Computing (HPDC-11).*, 2002.

[RF02b] K. Ranganathan and I. Foster. Identifying dynamic replication strategies for a high-performance data grid. In *Proceedings of the Second International Workshop on Grid Computing*, 2002.

[SL94] S. F. Smith and O. Lassila. Toward the development of mixed-initiative scheduling systems. In *Proceedings ARPA-Rome Laboratory Planning Initiative Workshop*, 1994. Tucson, AZ.

[WSF+03] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Cajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Gsi: Security for grid services. In *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, 2003.

Index