Pegasus: Mapping Scientific Workflows onto the Grid

Ewa Deelman[1], James Blythe[1], Yolanda Gil[1], Carl Kesselman[1], Gaurang Mehta[1], Sonal Patil[1], Mei-Hui Su[1], Karan Vahi[1], Miron Livny[2]

[1] Information Sciences Institute, University of Southern California, Marina Del Rey, CA 90292
{deelman, blythe, gil, carl, gmehta, sonal, mei, vahi}@isi.edu
[2] Computer Sciences Department, University of Wisconsin, Madison, WI 53706-1685
miron@cs.wisc.edu

Abstract

In this paper we describe the Pegasus system that can map complex workflows onto the Grid. Pegasus takes an abstract description of a workflow and finds the appropriate data and Grid resources to execute the workflow. Pegasus is being released as part of the GriPhyN Virtual Data Toolkit and has been used in a variety of applications ranging from astronomy, biology, gravitational-wave science, and high-energy physics.

1 Introduction

Grid technologies are changing the way scientists conduct research, fostering large-scale collaborative endeavors where scientists share their resources, data, applications, and knowledge to pursue common goals. These collaborations, defined as Virtual Organizations (VOs) [Foster 2001], are formed by many scientists in various fields, from high-energy physics, to gravitational wave physics to biologists. For example, the gravitational-wave scientists from LIGO and GEO [Abramovici 1992] have formed a VO that consists of scientists in the US and Europe, as well as compute, storage and network resources on both continents. As part of this collaboration, the data produced by the LIGO and GEO instruments and calibrated by the scientists are being published to the VO using Grid technologies.

Collaborations also extend to Virtual Data, where data refers not only to raw published data, but also data that has been processed in some fashion. Since data processing can be very expensive, sharing these data products can save the expense of performing redundant computations. The concept of Virtual Data was first introduced within the GriPhyN project (www.griphyn.org). An important aspect of Virtual Data are the applications that produce the desired data products. In order to discover and evaluate the validity of the Virtual Data products, the applications are also published within the VO.

Taking a closer look at the Grid applications, they are no longer monolithic codes, rather they are being built from existing application components. In general, we can think of applications as being defined by workflows, where the activities in the workflow are individual application components and the dependencies between the activities reflect the data and/or control flow dependencies between the components.

In general, a workflow can be described in an abstract form, in which the workflow activities are independent of the Grid resources used to execute the activities. We denote this workflow an abstract workflow. Abstracting away the resource descriptions allows the workflows to be portable. One can describe the workflow in terms of computations that need to take place without identifying particular resources that can perform this computation. Clearly, in a VO environment, this level of abstraction allows for easy sharing of workflow descriptions between VO participants.

This level of abstraction also enables the workflows to be efficiently mapped onto the existing Grid resources at the time that the workflow activities can be executed. It is possible that users may develop workflows ahead of a particular experiment and then only during the experiment's runtime are the workflows executed. Since the Grid environment is very dynamic, and the resources are shared among many users, it is impossible to optimize the workflow from the point of view of execution ahead of time. In

fact, one may want to make decisions about the execution locations and the access to a particular (possibly replicated) data set as late as possible.

In this work we refer to the executable workflow as the concrete workflow (CW). In the CW the workflow activities are bound to specific Grid resources. The concrete workflow also includes the necessary data movement to stage data in and out of the computations. Other nodes in the CW also may include data publication activities, where newly derived data products are published into the Grid environment.

In the paper we focus on the process of mapping abstract workflows to their concrete forms. In particular, we present Pegasus, which stands for Planning for Execution in Grids. We describe the current system and the applications that use it. The current system is semi-dynamic, in that the workflows are fully mapped to their concrete form when they are given to Pegasus. We also explore a fully dynamic mode of mapping workflows using a combination of technologies such as Pegasus and the Condor's workflow executioner, DAGMan [DAGMan]. In this approach DAGMan is used to guide the workflow mapping process.

2 Pegasus

Pegasus is designed to map abstract workflows onto the Grid environment [Deelman 2003a, Deelman2003b]. The abstract workflows can be constructed by using Chimera [Foster 2002] or can be written directly by the user. The inputs to the Chimera system are partial workflow descriptions that describe the logical input files, the logical transformations (application components) and their parameters, as well as the logical output files produced by these transformations. Following is a simple example of partial workflow descriptions expressed in Chimera's Virtual Data Language (VDL).

```
TR preprocess( output b[], input a )  {
        argument = "-a top -T60";   argument = " -i "${input:a};    argument = " -o "${output:b}; }
TR findrange( output b, input a2, input a1, none name="findrange", none p="0.0" ) {
        argument arg = "-a "${none:name} " -T60";   argument = " -i "${input:a1}" "${input:a2};
        argument = " -o "${output:b};   argument = " -p "${none:p}; }
TR analyze( output b, input a[] ) {
        argument arg = "-a bottom -T60"; argument = " -i "${input:a};   argument = " -o "${output:b};}

DV top->preprocess( b=[ @{output:"f.b1":"true"}, @{output:"f.b2":"true"} ],   a=@{input:"f.a"}   );
DV left->findrange(b=@{output:"f.c1":"true"},  a2=@{input:"f.b2":"true"},  a1=@{input:"f.b1":"true"},
         name="left", p="0.5" );
DV right->findrange( b=@{output:"f.c2":"true"},  a2=@{input:"f.b2":"true"}, a1=@{input:"f.b1":"true"},
        name="right", p="1.0" );
DV bottom->analyze(  b=@{output:"f.d"},  a=[ @{input:"f.c1"}, @{input:"f.c2"} ] );
```

"TR" denotes the template for the transformation, including the definition of the expected number of input and output files as well as the definition of the transformation's parameters. "DV" is an instantiation (derivation) of the TR template and includes the logical input and output filenames. The DVs are uniquely named, here for illustration purposes we added the words *left*, *right*, etc. to indicate the final position of these derivations in the abstract workflow.

Given a particular VDL and a desired set of logical output filenames, Chimera produces an abstract workflow by matching the names of the input and output files, starting from the user-specified output filenames. In the example above, let's assume that the user wanted to obtain file *f.c2*. Given the VDL above, Chimera would chain the partial workflows based on the input/output files and would produce the following abstract workflow.

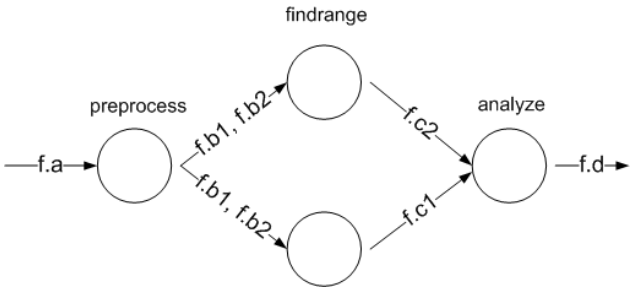

Figure 1: Abstract Workflow.

The resulting abstract workflow is specified in XML in the form of a DAX (DAG XML description.) Some users choose to write the DAX directly, especially if the list of possible VDL definitions is very long. An example of such application is Montage (www.ipac.caltech.edu), an astronomy application, where mosaics of the sky are created based on user requests. In case of Montage it is not realistic to pre-populate the system with all the possible VDL definitions. Additionally, some pre-processing of the request needs to be performed to pick the appropriate parameters for the montage computation. Below is part of a DAX for the VDL expressed above. It shows the definition of the final activity *analyze* in the workflow and defines its dependency on the two *findrange* nodes that have ids *ID000002* and *ID000003*.

```
<job id="ID000004" name="analyze" level="1" dv-name="bottom">
   <argument>-a bottom -T60 -i <filename file="f.c1"/> <filename file="f.c2"/> -o <filename file="f.d"/></argument>
   <uses file="f.c1" link="input" ……>
   <uses file="f.c2" link="input" ……>
   <uses file="f.d" link="output"……>
 </job>
…..
<child ref="ID000004">
   <parent ref="ID000002"/>
   <parent ref="ID000003"/>
 </child>
```

Whether the input comes through Chimera or is given directly by the user, Pegasus requires that it is in specified in DAX format. Based on this specification, Pegasus produces a concrete (executable) workflow that can be given to Condor's DAGMan [Frey 2001] for execution.

2.1 Mapping Abstract Workflows onto the Grid

The abstract workflows describe the computation in terms of logical files and logical transformations and indicate the dependencies between the workflow components. Mapping the abstract workflow description to an executable form involves finding the resources that are available and can perform the computations, the data that is used in the workflow, and the necessary software.

Pegasus consults various Grid information services to find the above information (see Figure 3). Pegasus uses the logical filenames referenced in the workflow to query the Globus Replica Location Service (RLS) [Chervenak 2002] to locate the replicas of the required data. We assume that data may be replicated in the environment and that the users publish their data products into RLS. Given the set of logical filenames, RLS returns a corresponding set of physical file locations. After Pegasus produces new data products, it registers them into the RLS as well (unless the user does not want that to happen.) Intermediate data products can be registered as well.

In order to be able to find the location of the logical transformations defined in the abstract workflow, Pegasus queries the Transformation Catalog (TC) [Deelman 2001] using the logical transformation names. The catalog returns the physical locations of the transformations (on possibly several systems) and the environment variables necessary for the proper execution of the software.

Pegasus queries the Globus Monitoring and Discovery Service (MDS) [Czajkowski 2001] to find the available resources, their characteristics such as the load, the scheduler queue length, and available disk space. The information from the TC is combined with the MDS information to make scheduling decisions. When making resource assignment, Pegasus prefers to schedule the computation where the data already exist, otherwise it makes a random choice or uses a simple scheduling techniques.

Additionally, Pegasus uses MDS to find information about the location of the gridftp servers [Allcock 2001] that can perform data movement, job managers [Czajkowski 1991] that can schedule jobs on the remote sites, storage locations, where data can be pre-staged, shared execution directories, the RLS where new data can be registered into, site-wide environment variables, etc. This information is necessary to produce the submit files that describe the necessary data movement, computation and catalog updates.

2.2 Pegasus' Workflow Reduction

The information about the available data can be used to optimize the concrete workflow from the point of view of Virtual Data. If data products described within the AW already exist, Pegasus can reuse them and thus reduce the complexity of the CW. 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 may have already materialized (available on some storage system) part of the entire required dataset. If this information is published into the RLS, Pegasus can utilize this knowledge and obtain the data, thus avoiding possibly costly computation. As a result, some components that appear in the abstract workflow do not appear in the concrete workflow.

Pegasus 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.

The final result produced by Pegasus is an executable workflow that identifies the resources where the computation will take place, the data movement for staging data in and out of the computation, and registers the newly derived data products in the RLS.

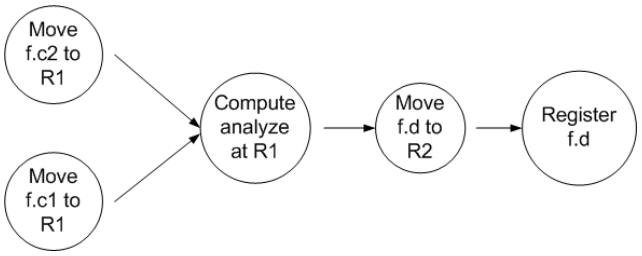

Figure 2: Concrete Workflow.

Following the example above, if files *f.c1* and *f.c2* have already been computed, then the abstract workflow is reduced to just the *analyze* activity and a possible concrete workflow would be as shown above.

2.3 Workflow Execution

The concrete workflow produced by Pegasus is in a form of submit files that are given to DAGMan for execution. The submit files indicate the operations to be performed on given remote systems and the order in which the operations need to be performed. Given the submit files DAGMan submits jobs to

Condor-G [Frey 2001] for execution. DAGMan is responsible for enforcing the dependencies between the jobs defined in the concrete workflow.

In case of job failure, DAGMan can retry a job a given number of times or if that fails, DAGMan generates a rescue DAG that can be potentially modified and resubmitted at a later time. Job retry is useful for applications that have intermittent software problems and in condor pools, where there is a likelihood of running on another host at the next try. The rescue DAG is useful in cases where the failure was due to lack of disk space that can be reclaimed or in cases where totally new resources need to be assigned for execution.

3 Application Examples

The GriPhyN Virtual Data System (VDS) that consists of Chimera, Pegasus and DAGMan has been used to successfully execute both large workflows with an order of 100,000 jobs with relatively short runtimes [Annis 2002] and workflows with small number of long-running jobs [Deelman2003a]. Figure 3 depicts the process of workflow generation, mapping and execution. The user specifies the VDL for the desired data products, Chimera builds the corresponding abstract workflow representation. Pegasus maps this AW to its concrete form and DAGMan executes the jobs specified in the Concrete Workflow.

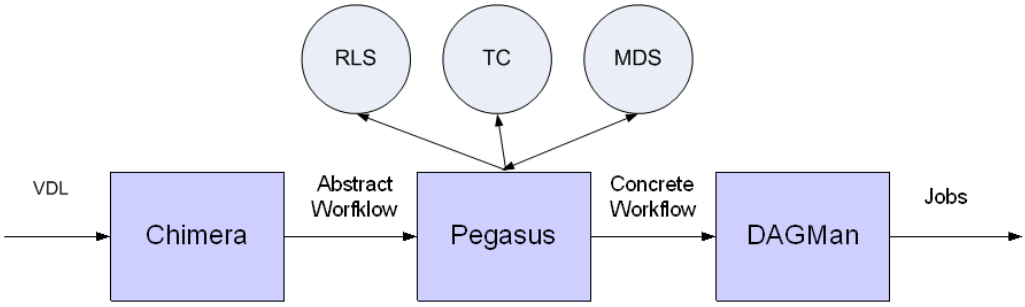

Figure 3: Components of a Workflow Generation, Mapping and Execution System.

Pegasus and DAGMan were able to map and execute workflows on a variety of platforms: condor pools, clusters managed by LSF or PBS, TeraGrid hosts (www.teragrid.org), and individual hosts. Below, we describe some of the applications that have been run using the VDS.

Bioinformatics and Biology

One of the most important bioinformatics applications is BLAST which consists of a set of sequence comparison algorithms that are used to search sequence databases for optimal local alignments to a query. ANL scientists used the VDS to perform two major runs. One consisted of 60 genomes and the other of 450 genomes, each composed of 4,000 sequences. The runs produced on the order of 10,000 jobs and approximately 70GB of data. The execution was performed on a dedicated cluster. A speedup of 5-20 times were achieved using Pegasus and DAGMan not because of algorithmic changes, but because the nodes of the cluster were used efficiently by keeping the submission of the jobs to the cluster constant.

Another application that uses the VDS is the tomography application, where 3D structures are derived from a series of 2D electron microscopic projection images. Tomography allows for the reconstruction and detailed structural analysis of complex structures such as synapses and large structures like dendritic spines. The tomography application is characterized by the acquisition, generation and processing of extremely large amounts of data, upwards of 200GB per run.

Astronomy

Astronomy applications that were executed using the VDS fall into the category of workflows with a large number of small jobs. Among such applications are Montage and Galaxy Morphology.

Montage is a grid-capable astronomical mosaicking application. It is used to reproject, background match, and finally mosaic many image plates into a single image. Montage has been used to mosaic image plates from synoptic sky surveys, such as 2MASS in the infrared wavelengths. Montage is being developed under the ESTO CT project by a team that includes Caltech IPAC, Caltech CACR, and JPL.

Figure 4 shows snapshot of a small Montage workflow that consists of 1200 executable jobs. In the case of Montage, the application scientists produce their own abstract workflows without using Chimera, because they need to tailor the workflow to individual requests.

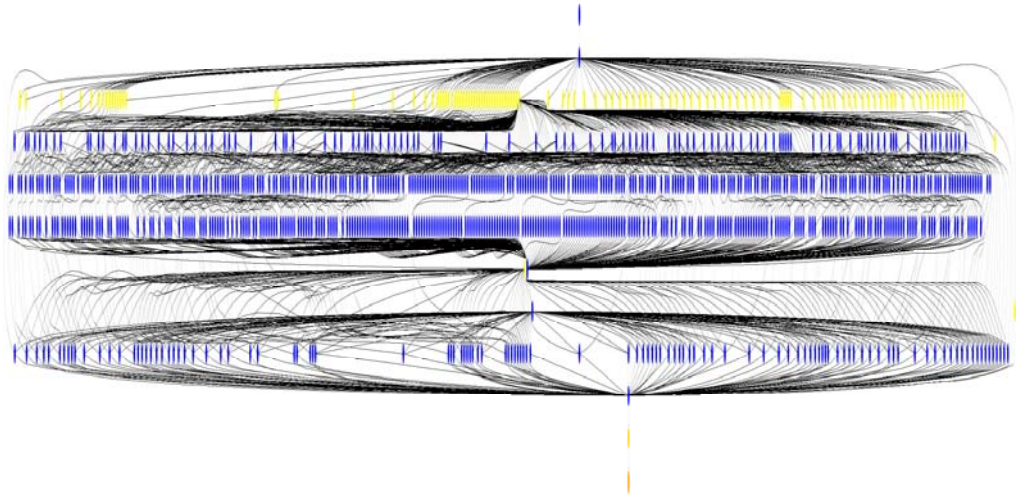

Figure 4: Montage workflow produced by Pegasus. The light colored-nodes represent data stage-in and the dark colored nodes, computation.

The Galaxy morphology application [Deelman 2003c] is used to investigate the dynamical state of galaxy clusters and to explore galaxy evolution inside the context of large-scale structure. Galaxy morphologies are used as a probe of the star formation and stellar distribution history of the galaxies inside the clusters. Galaxy morphology is characterized in terms three parameters that can be calculated directly from an image of the galaxy such as average surface brightness, concentration index, asymmetry index. The computational requirements for calculating these parameters for a single galaxy are fairly light; however, to statistically characterize a cluster well, the application needs to calculate our parameters for the hundreds or thousands of galaxies that constitute the galaxy cluster.

High-Energy Physics

High-energy physics applications such as Atlas and CMS [Deelman 2003a] fall into the category of workflows that contain few long running jobs. A variety of different use-cases exist for simulated CMS data production. One of the simpler use-cases is known as an *n-tuple-only production* that consists of a five stage computational pipeline. These stages consist of a *generation* stage that simulates the underlying physics of each event and a *simulation* stage that models the CMS detector's response to the events. Additional stages are geared towards formatting the data and construction of an “image” of what the physicist would "see" as if the simulated data were actual data recorded by the experimental apparatus. In the final stage user-specific information is selected and a convenient, easy to use file that can be analyzed by a researching physicist is created. In one of the CMS runs, over the course of 7 days, 678 jobs of 250 events each were submitted using the VDS. From these jobs, 167,500 events were successfully produced using approximately 350 CPU/days of computing power and producing approximately 200GB of simulated data.

Gravitational-Wave Physics

The Laser Interferometer Gravitational-Wave Observatory (LIGO) [Abramovici 1992, Deelman 2003a] 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 of the hypotheses is that gravitational wave signals may come from supernova explosions, quakes in neutron stars, and pulsars.

Gravitational waves interact extremely weakly with matter, and the measurable effects produced in terrestrial instruments by their passage are expected to be miniscule. In order to establish a confident detection or measurement, a large amount of auxiliary data is acquired and analyzed along with the strain signal that measures the passage of gravitational waves. The LIGO workflows aimed at detecting gravitational waves emitted by pulsars are characterized by many medium and small jobs. In a Pegasus run conducted at SC 2002, over 58 pulsar searches were performed resulting in a total of 330 tasks, 469 data transfers executed and 330 output files. The total runtime was 11:24:35.

4 Just In-time Planning

In the Grid, resources are often shared between users within a VO and across VOs as well. Additionally, resources can come and go, because of failure or local policy changes. Therefore, the Grid is a very dynamic environment, where the availability of the resources and their load can change dramatically from one moment to the next.

Even if a particular environment is changing slowly, the duration of the execution of the workflow components can be quite large and by the time a component finishes execution, the data locations may have changed as well as the availability of the resources. Choices made ahead of time even if still feasible may be poor.

Clearly, software that deals with executing jobs on the Grid needs to be able to adjust to the changes in the environment. In this work, we focus on providing adaptivity at the level of workflow activities. We assume that once an activity is scheduled on a resource, it will not be preempted and its execution will either fail or succeed.

Up to now, Pegasus was generating fully specified, executable workflows based on an abstract workflow description. The new generation of Pegasus takes a more "lazy" approach to workflow mapping and produces partial executable workflows based on already executed tasks and the currently available Grid resources.

In order to provide this level of just in-time planning we added a new component to the Pegasus system: the partitioner that partitions the abstract workflow into smaller partial workflows. The dependencies between the partial workflows reflect the original dependencies between the tasks of the abstract workflow. Pegasus then schedules the partial workflows following these dependencies. The assumption, similar to assumption made in the original version of Pegasus is that the workflow does not contain any cycles. Figure 5 illustrates the partitioning process.

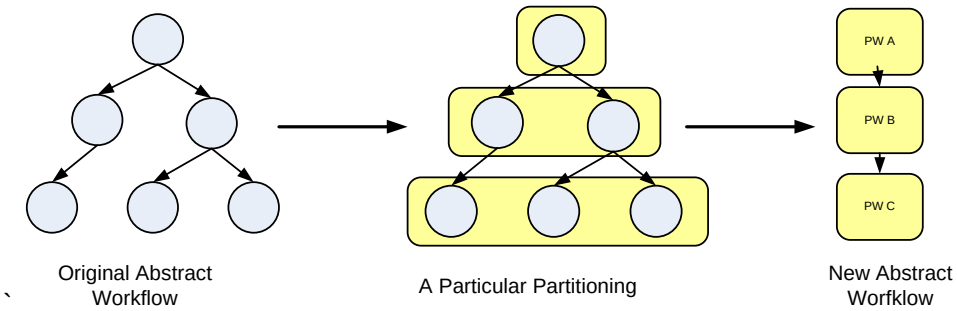

Figure 5: The New Abstract to Concrete Workflow Mapping.

The original workflow is partitioned according to a specified partitioning algorithm. The result is a workflow, where the elements are partial workflows. The particular partitioning algorithms shown in Figure 5 simply partitions the workflow based on the level of the node in the Abstract Workflow. Investigating various partitioning strategies is the focus of our future work.

Once the partitioning is performed, Pegasus maps and submits the partial workflows to the Grid. If there is a dependency between two partial workflows, Pegasus is made to wait (by DAGMan) to map the dependent workflow until the preceding workflow has finished executing.

DAGMan is used to drive the in-time planning process by making sure that Pegasus does not refine a partial workflow until the previous partial workflow successfully finished execution. The following figure shows the DAG that is submitted to DAGMan for execution.

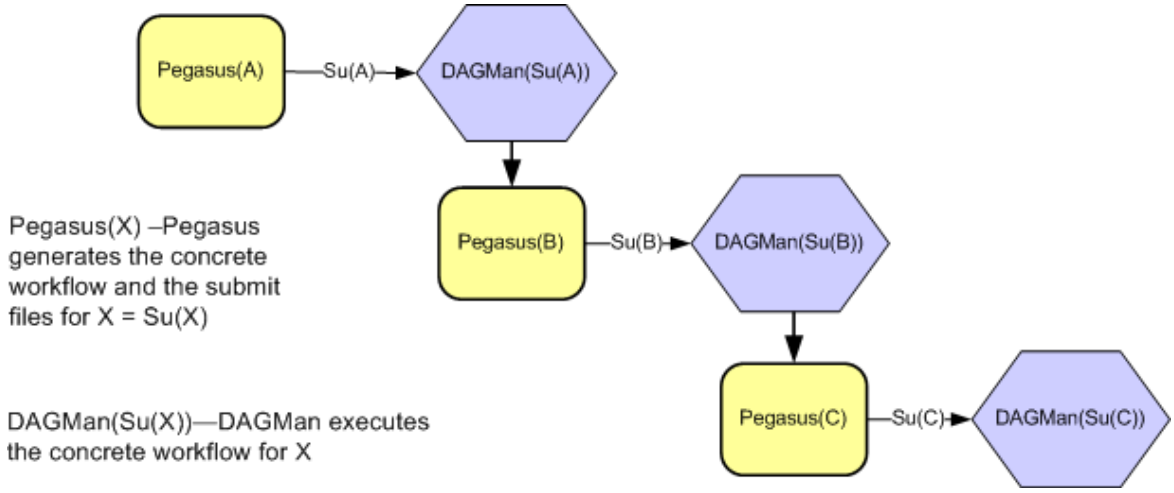

Figure 6: Just In-Time Workflow Mapping.

Given this DAG, DAGMan (instance #1) first calls Pegasus on one partition of the abstract workflow, partition *A*. Pegasus then generated the concrete workflow and produces the submit files necessary for the execution of that workflow through DAGMan, these files are named *Su(A)* in the figure above. Now the first instance of DAGMan calls a new instance of DAGMan (instance #2) with the submit files *Su(A)*. This is reflected in the *DAGMan(Su(A))* node in the figure above; it is a nested call to DAGMan within DAGMan. Once the second instance of DAGMan concludes successfully, implying that the concrete workflow corresponding to the partial abstract workflow *A* has successfully executed, the first instance of DAGMan call Pegasus with the abstract workflow B, and the process repeats until all the partitions of the workflow are refined to their concrete form and executed.

Initial results of using this approach for gravitational wave applications proved that the approach is viable. Although we have not yet performed a formal study, it is clear that there are benefits of just in-time planning. For example, assume that a resource fails during the execution of the workflow. If the workflow was fully scheduled ahead of time to use this resource, the execution will fail. However, if the failure occurs at the partition boundary, the new partition will not be scheduled onto the failed resource. An interesting area of future research is to evaluate various workflow partitioning algorithms and their performance based on the characteristics of the workflows and the characteristics of the target execution systems.

5 Related Work

There have been a number of efforts within the Grid community to develop general-purpose workflow management solutions.

WebFlow [Fox 1998] is a multileveled system for high performance distributed computing. It consists of three layers. The top layer consists of a web based tool for visual programming and monitoring. It provides the user the ability to compose new applications with existing components using a drag and drop capability. The middle layer consists of distributed web flow server implemented using java extensions to httpd servers. The lower layer uses the Java CoG Kit to interface with the Grid [Laszewski 2001] for high

performance computing. Webflow uses GRAM as the interface between webflow and the Globus Toolkit. Thus, Webflow also provides a visual programming aid for the Globus toolkit.

GridFlow [Cao 2003] has a two-tiered architecture with global Grid workflow management and local Grid sub workflow scheduling. GridAnt [gridant] uses the Ant [ant] workflow processing engine. Nimrod-G [Buyya 2000] is a cost and deadline based resource management and scheduling system. The Accelerated Strategic Computing Initiative Grid [Beiriger 2000] distributed resource manager includes a desktop submission tool, a workflow manager and a resource broker. In the ASCI Grid software components are registered so that the user can ask "run code X" and the system finds out an appropriate resource to run the code. Pegasus uses a similar concept of virtual data where the user can ask "get Y" where Y is a data product and the system figures out how to compute Y. Almost all the systems mentioned above except GridFlow use the Globus Toolkit [globus] for resource discovery and job submission. The GridFlow project will apply the OGSA [ogsa] standards and protocols when their system becomes more mature. Both ASCI Grid and Nimrod-G uses the Globus MDS [Fitzgerald 1997] service for resource discovery and a similar interface is being developed for Pegasus. GridAnt, Nimrod-G and Pegasus use GRAM [Czajkowski 1998] for remote job submission and GSI [Welch 2003] for authentication purposes. GridAnt has predefined tasks for authentication, file transfer and job execution, while reusing the XML-based workflow specification implicitly included in ant, which also makes it possible to describe parallel and sequential executions.

The main difference between Pegasus and the above systems is that while most of the above system focus on resource brokerage and scheduling strategies Pegasus uses the concept of virtual data and provenance to generate and reduce the workflow based on data products which have already been computed earlier. It prunes the workflow based on the assumption that it is always more costly the compute the data product than to fetch it from an existing location. Pegasus also automates the job of replica selection so that the user does not have to specify the location of the input data files. Pegasus can also map and schedule only portions of the workflow at a time, using just in-time planning techniques.

6 Future Directions

Pegasus, a Grid workflow mapping system presented here has been successfully used in a variety of applications, from astronomy, biology and physics. Although Pegasus provides a feasible solution, it is not necessarily a low cost one in term of performance. One step towards performance optimization and reliability is the work that we have been doing in terms of just in-time planning, where resources are identified as needed, before the jobs can be scheduled. An aspect of the work we plan to focus on in the near future are the various partitioning methods that can be applied to dividing workflows into smaller components. Obviously the type of partitioning performed has a great impact on the performance of the entire workflows. In the future work, we also plan to investigate scheduling techniques that would schedule partial workflows onto the Grid.

Acknowledgments

Many GriPhyN members need to be thanked for their contributions and discussions regarding Pegasus. Many application scientists have also contributed to the successful application of Pegasus to their application domain. For the Galaxy Morphology: G. Greene, B. Hanisch, R. Plante, and others. For Montage: Bruce Berriman, John Good, Joseph C. Jacob, Daniel S. Katz, and Anastasia Laity; For BLAST: Natalia Matlsev, Michael Milligan, Veronika Nefedova, Alexis Rodriguez, Dinanath Sulakhe, Jens Voeckler, Mike Wilde; For LIGO: Bruce Allen, Kent Blackburn, Albert Lazzarini, Scott Koranda, and Maria Alessandra Papa; For Tomography: Mark Ellisman, Steve Peltier, Abel Lin, Thomas Molina; and For CMS: Adam Arbree and Richard Cavanaugh.

References:

[Abramovici 1992] A. Abramovici, W. E. Althouse, and e. al., "LIGO: The Laser Interferometer Gravitational-Wave Observatory," *Science*, vol. 256, pp. 325-333, 1992.

[Allcock 2001] W. Allcock, J. Bester, et al., "Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing," Proceedings of the IEEE Mass Storage Conference, pp. 13-28 April 2001.

[Annis 2002] J. Annis, Z. Yong, J. Voeckler, M. Wilde, S. Kent, and I. Foster, "Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey.," SC 2002, Baltimore, MD.

[ant] http://ant.apache.org

[Beiriger 2000] Beiriger, J., Johnson, W., Bivens, H., Humphreys, S. and Rhea, R., "Constructing the ASCI Grid". In Proc. 9th IEEE Symposium on High Performance Distributed Computing, 2000, pp. 193-200.

[Buyya 2000] Buyya, R., Abramson, D. and Giddy, J. "Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid", HPC Asia 2000, 2000, China.

[Cao 2003] J. Cao, et al. "WorkFlow Management for Grid Computing." In Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 198-205, 2003.

[Chervenak 2002] A. Chervenak, E. Deelman, et al., "Giggle: A Framework for Constructing Scalable Replica Location Services," Proceedings of Supercomputing 2002 (SC2002), Baltimore, MD. 2002.

[Czajkowski 1998] K.Czajkowski, et al. "A Resource Management Architecture for Metacomputing Systems." Proc.IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[Czajkowski 2001] K. Czajkowski, S. Fitzgerald, et al., "Grid Information Services for Distributed Resource Sharing," Proc. of the 10th IEEE International Symposium on High Performance Distributed Computing, pages: 181 -194, 2001.

[DAGMan] Condor Team. "The directed acyclic graph manager." www.cs.wisc.edu/condor/dagman, 2002

[Deelman 2001] E. Deelman, C. Kesselman, et al., "Transformation Catalog Design for GriPhyN," Technical Report GriPhyN-2001-17, 2001.

[Deelman 2003a] E. Deelman, et al. "Mapping Abstract Complex Workflows onto Grid Environments," *Journal of Grid Computing*, vol. 1, pp. 25-39, 2003.

[Deelman 2003b] E. Deelman, J. Blythe, Y. Gil, Carl Kesselman, "Workflow Management in GriPhyN", Chapter in "The Grid Resource Management", Kluwer 2003.

[Deelman 2003c] E. Deelman et al. "Grid-Based Galaxy Morphology Analysis for the National Virtual Observatory," Proceedings of SC 2003.

[Fitzgerald 1997] S. Fitzgerald, et al.. A Directory Service for Configuring High-Performance Distributed Computations. Proc. 6th IEEE Symposium on High-Performance Distributed Computing, 1997.

[Foster 2001] 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.

[Frey 2001] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids.," *Cluster Computing*, vol. 5, pp. 237-246, 2002.

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

[Fox 1998] http://www.supercomp.org/sc98/TechPapers/sc98_FullAbstracts/Akarsu809

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

[gridant] http://www-unix.globus.org/cog/projects/gridant

[Laszewski 2001] von Laszewski, G. I. Foster, J. Gawor, P. Lane, "A Java Commodity Grid Kit**,"** *Concurrency and Computation: Practice and Experience*, pages 643-662, Volume 13, Issue 8-9, 2001.

[ogsa] www.globus.org/ogsa

[Welch 2003] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Cajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke. Security for Grid Services. Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), pp. 48 -57, June 2003, IEEE Press.