# Pegasus: Planning for Execution in Grids

Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman
University of Southern California, Information Sciences Institute

## 1 Motivation

Grid computing has made great progress 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 allow for the discovery and monitoring of resources. The information provided can be used to find the available resources and select the resources which are the most appropriate for the task.
- Security services, which allow users and resources to mutually authenticate and allows the resources to authorize users based on local policies.
- Resource management, which allows for the scheduling of jobs on particular resources.
- Data management services, which enable users and applications to manage large, distributed and replicated data sets. Some of the available services deal with locating particular data sets, others with efficiently moving large amounts of data across wide area networks.

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 quiet complex. As the result it is becoming increasingly necessary to develop higher level services which can automate the process and provide an adequate level of performance and reliability.

## 2 Issues in Mapping Workflows onto the Grid

In general we can think of applications as being composed of application components. The process of application development (shown in Figure 1) can be described as follows. The application components are selected, their input and output file names are identified by their logical names (names that uniquely identify the content of the file, but not its location), and the order of the execution of the components is specified. As a result, we obtain an abstract workflow (AW), where the behavior of the application is specified at an abstract level.

Next this workflow needs to be mapped onto the available Grid resources, performing resource discovery and selection. Finally the resulting concrete workflow (CW) is sent to the executor for execution. In this section, we focus on the behavior of the concrete workflow generator (CWG) and its interaction with an executor, such as for example Condor-G/DAGMan [1].

Important issues are the relationship and interfaces between the planner and the executor, both from the standpoint of planning and fault tolerance. For this discussion, we assume that multiple requests to the system are handled independently.

One can imagine two extremes: on one hand, the planner can make an exact plan of computation based on the current information about the system. The planner would decide where the tasks need to execute, the exact location from where the data needs to be accessed for the computation etc… 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 planning decisions (in-time scheduling).

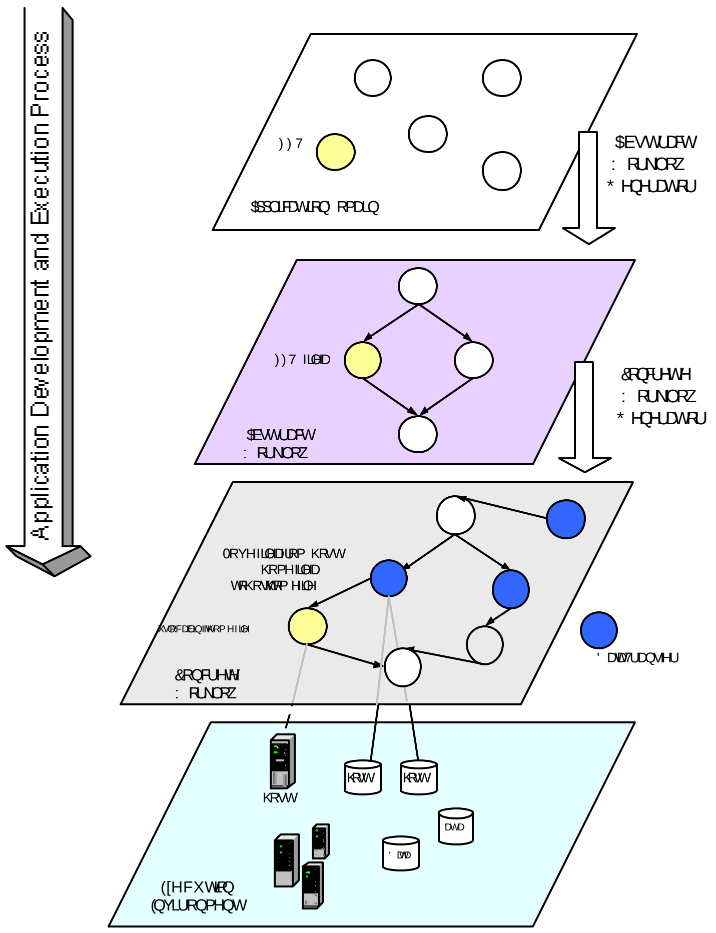

Application Development and Execution Process
))7
$SSOLFDWLRQ 'RPDLQ
$EVWUDFW :RUNIORZ *HQHUDWRU
))7 ILOHD
$EVWUDFW :RUNIORZ
&RQFUHWH :RUNIORZ *HQHUDWRU
0RYH ILOHD IURP KRVW KRVW
&RQFUHWH :RUNIORZ
'DWD7UDQVIHU
KRVW
([HFXWLRQ (QYLURQPHQW

**Figure 1: General view of application development in Grids.**

The benefit of the first approach (we term it full-plan-ahead), is that the planner can aim to optimize the plan based on the entire structure of the DAG, however, because the execution environment is very dynamic, by the time the tasks in the DAG are ready to execute, the environment might have changed so much that the execution is now far less optimal. Additionally, the data assumed to be at a certain location, might not be longer available and thus an error would be detected during execution. If the planner constructs full plans, it must be able to adapt to the changing conditions and to be able to quickly re-plan.

Fault 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. 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 have global information about the request it could make potentially expensive decisions.

Another approach is deferred scheduling, where the executor and the planner work together to come up with a plan. The planner provides an abstract workflow description to the executor, which, when it is ready to schedule a task, contacts the planner and asks for the execution location for the task. The planner can at that time make a decision, which would take into account the global information it has. Because the planner can make decisions at 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 might be too fine, and can 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 can have very different characteristics. For example, if we consider data-intensive applications, where the overall runtime is driven by data movement costs, then the full-plan-ahead planner can minimize the overall data movement by picking appropriate compute resources, resources "close" to the data. An in-time scheduler can also schedule computation near the input data but without the knowledge of the overall data flow it will only try to achieve local optimization and might come up with an execution which is poor overall.

For applications, which are compute-intensive, in-time-scheduling might be sufficient and optimal because the best compute resources can be found at a given moment in time and the time to stage the data is negligible.

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 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 optimize across multiple requests and accommodate various usage policies and community and user priorities. In this case, full-plan-ahead planners have the advantage of being able to optimize the end-to-end workflows, however still facing the challenge of being 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 the understanding of the application's behavior is crucial to the ability of planning and scheduling of the execution, application performance models are becoming ever more necessary.

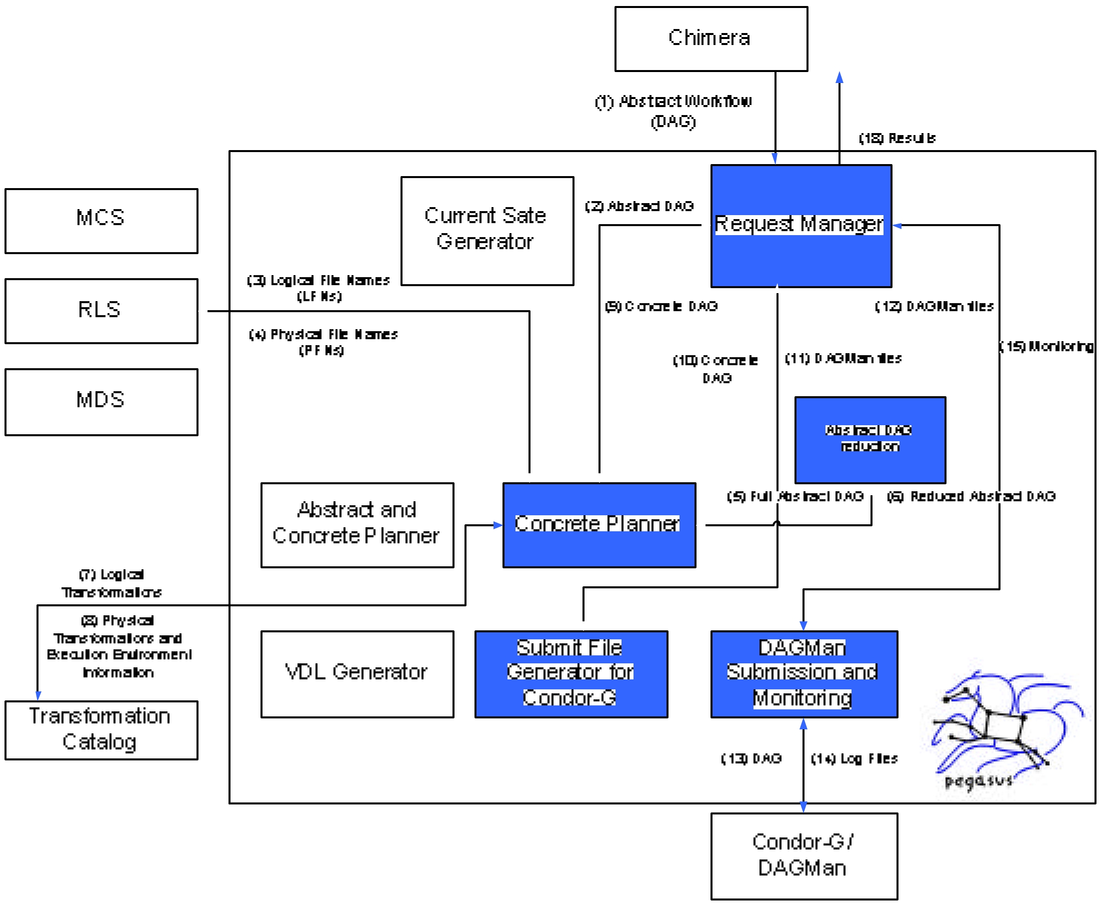
## 3 Pegasus Overview

**Pegasus**, which stands for Planning for Execution in Grids, was developed at ISI as part of the GriPhyN and SCEC/IT projects. 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 scheduling and deferred-scheduling.

Pegasus was first integrated with the GriPhyN Chimera system [2]. In that configuration (see Figure 2), Pegasus receives an abstract workflow (AW) description from Chimera, produces a concrete workflow (CW), and submits it to DAGMan for execution. The workflows are represented as Directed Acyclic Graphs (DAGs). AW describes the transformations and data in terms of their logical names. CW, which specifies the location of the data and the execution platforms, is optimized by Pegasus from the point of view of Virtual Data. If data products described within AW are found to be already materialized (via queries to the Globus Replica Location Service (RLS)), Pegasus reuses them and thus reduces the complexity of CW. This optimization is performed in the "abstract DAG reduction" component. The "Concrete planner" component then consults the Transformation Catalog [3] to determine the locations where the computation can be executed. If there are more than one possible location, a location is chosen randomly. The Concrete Planner also adds transfer and registration nodes. The transfer nodes are used to stage data in or out. Registration nodes are used to publish the resulting data products in the Replica Location Service. They are added if the user requested that all the data be published and sent to a particular storage location.

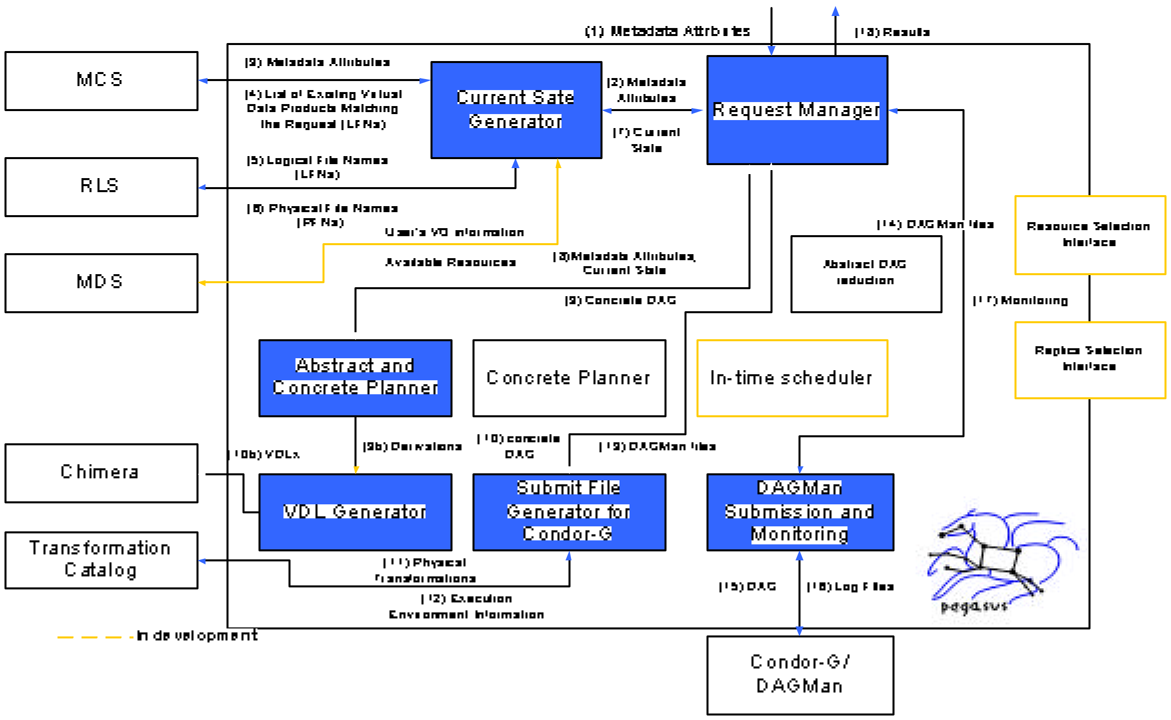
Once the resources are identified for each task, Pegasus generates the submit file for Condor-G. The resulting concrete DAG is sent to DAGMan for execution.

In that configuration, Pegasus has been shown to be successful in mapping workflows for very complex applications such as the Sloan Digital Sky Survey [4] and the Compact Muon Source [5].



Pegasus uses the Metadata Catalog Service (MCS) [Chervenak, 2002 #1853], newly developed at ISI, to perform the mapping between application-specific attributes and logical file names of existing data products. AI-based planning technologies are used to construct both the abstract and concrete workflows. The abstract and concrete planner models the application components along with data transfer and data registration as operators. Each operator's parameters 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 Replica Location Service. 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.

Pegasus also contains a Virtual Data Language generator that can populate the Chimera catalog with newly constructed derivations. In this configuration, Pegasus similarly generates the necessary submit files and sends the concrete workflow to DAGMan for execution. We have configured Pegasus to support the LIGO pulsar search. Details about the search can be found in [6].



ahead to in-time scheduling etc. We also envisage providing the capability to configure the system to interface to user-defined resource selection and replica selection modules.

Finally, we want to research interactions between Pegasus and resource brokers. In such cases Pegasus would plan out in broad strokes and the brokers would refine the plans.

**Acknowledgments:**

The Pegasus system and the Metadata Catalog Service have been developed at ISI by Amit Agrawal, James Blythe, Gaurang Mehta, Gurmeet Singh and Karan Vahi.

## References

[1] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," presented at 10th International Symposium on High Performance Distributed Computing, 2001.

[2] 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.

[3] E. Deelman, C. Kesselman, and G. Mehta, "Transformation Catalog Design for GriPhyN," Technical Report GriPhyN-2001-17, 2001.

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

[5] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, A. Arbree, and R. Cavanaugh, "Mapping Abstract Complex Workflows onto Grid Environments," *in submission*.

[6] 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," presented at 11th Intl Symposium on High Performance Distributed Computing, 2002.