Wings for Pegasus: Creating Large-Scale Scientific Applications Using Semantic Representations of Computational Workflows

Yolanda Gil, Varun Ratnakar, Ewa Deelman, Gaurang Mehta, Jihie Kim

Information Sciences Institute, University of Southern California
4676 Admiralty Way, Marina del Rey CA 90292, United States
{gil, varunr, deelman, mehta, jihie }@isi.edu

Abstract

Scientific workflows are being developed for many domains as a useful paradigm to manage complex scientific computations. In our work, we are challenged with efficiently generating and validating workflows that contain large amounts (hundreds to thousands) of individual computations to be executed over distributed environments. This paper describes a new approach to workflow creation that uses semantic representations to describe compactly complex scientific applications in a data-independent manner, then automatically generates workflows of computations for given data sets, and finally maps them to available computing resources. The semantic representations are used to automatically generate descriptions for each of the thousands of new data products. We interleave the creation of the workflow with its execution, which allows intermediate execution data products to influence the generation of the following portions of the workflow. We have implemented this approach in Wings, a workflow creation system that combines semantic representations with planning techniques. We have used Wings to create workflows of thousands of computations, which are submitted to the Pegasus mapping system for execution over distributed computing environments. We show results on an earthquake simulation workflow that was automatically created with a total number of 24,135 jobs and that executed for a total of 1.9 CPU years.

Introduction

Scientific workflows are emerging as an effective paradigm to represent and manage complex scientific applications [Deelman and Gil 06; Taylor et al. 06]. Scientific communities are increasingly sharing resources including data repositories, services, instruments, and computing resources. Workflows provide an effective representation that captures how these very heterogeneous resources can be configured and assembled for a wide variety of purposes, and that facilitates the management of their execution in such distributed environments. As sharing of data and resources increases in scientific communities, the creation and management of workflows is central to the future of scientific analysis and computations.

Some scientific applications, notably in bioinformatics, are cast as workflows of Web services in distributed environments [Oinn et al 06; Ludaescher et al. 06]. Many other scientific applications do not use workflows composed of distributed services, but workflows to manage distributed data and computation resources at large scale. Instead, they use *computational workflows* composed of jobs that perform computations on remote hosts in distributed environments through remote job submissions. Computational workflows utilize data that reside in catalogs that are replicated in the execution environment, and create data products that need to be stored back in those repositories. The computations are nodes in the workflow, and the links in the workflow represent the data flow among computations. Data is typically available in files, and each file is described with *metadata* attributes that describe its properties (e.g., its creation date, the instrument used for collection, or the computation that created it, and other domain-specific attributes). Examples of workflow systems of this kind are Pegasus [Deelman et al. 03; Deelman et al. 05; Deelman et al. 06] and Askalon [Wieczorek 05], and more recently Kepler [Ludaescher et al. 06] as well.

Creating and validating these computational workflows is a very challenging enterprise. Some of our prior work has addressed aspects of this problem through intelligent workflow editors that assist users in creating valid workflows [Kim et al 04], using AI planning techniques to generate workflows by searching through the space of possible combinations of computations [Blythe et al. 03], using semantic descriptions of metadata to support the integration of distributed data repositories [Tuchinda et al 04], and propagating metadata through a workflow using semantic representations [Kim et al 06; Kim et al 07]. However, in many scientific applications there is a need to scale up to data sets of thousands of elements. In the coming years, workflows will continue to grow well into petascale data set sizes. Providing assistance in creating and managing these large workflows will be not only desirable but absolutely necessary.

This paper presents recent work to address the creation of computational workflows of thousands of components. Our approach exploits semantic representations of workflows to express repetitive computational structures in

a compact manner, to describe underspecified data collections, and to process these representations to create workflows that can be then mapped to available resources for execution. We have implemented this approach in Wings, and integrated it with Pegasus into an end-to-end workflow creation and execution system. This paper focuses on the process of creating the workflow, and describes the semantic representations that we use and the mechanisms for interleaving workflow creation and execution that are needed in some applications. Additional details regarding workflow validation and metadata propagation that occur during the workflow generation process are described in [Kim et al 06].

We begin with a detailed motivation showing the scale and complexity of creating large workflows of computations. We present our approach and consider issues of incremental workflow generation. We describe the implementation in Wings and Pegasus and show results of generating and running earthquake science workflows in distributed environments such as the grid.

Motivation: Creating Very Large Scientific Workflows

In our work, we distinguish three distinct stages in the creation of workflows [Gil 06]. The first stage is to create *workflow templates*, which specify the high-level structure of the workflow in a data-independent representation. The second stage is to create *workflow instances*, which specify what data is to be used in the computation. Workflow instances are independent of execution resources, that is, they can be mapped to any execution environment by binding tasks to available resources. The third stage is to create an *executable workflow,* which specifies the data replicas to be used and their locations, the hosts where computation will occur, and the appropriate data movements across distributed locations. These three stages enable the management of the complexity of workflow creation by making the process more modular.

The third stage of creation of executable workflows is done by Pegasus [Deelman et al. 03; Deelman et al. 05; Deelman et al. 06]. Pegasus performs automated mapping of workflows to execution resources and the management of their execution in distributed grid environments. Pegasus uses Condor DAGMan [Frey et al 01] as the workflow execution engine. Pegasus and DAGMan are now production-quality workflow mapping and execution engines that are being used in a variety of scientific applications and manage and execute workflows in small, medium, and large size grid environments such as the TeraGrid (www.teragrid.org) and the Open Science Grid (www.opensciencegrid.org). Pegasus manages the mapping of workflows of thousands of interrelated computations whose overall execution often spans weeks. What Pegasus users are currently lacking are general-

purpose mechanisms to create and validate their very large workflows.

Many scientific applications require the processing of dataset elements one at a time with identical computations. Figure 1 shows an example from our collaboration with the Southern California Earthquake Center (SCEC) [Maechlin et al. 05] (left) and an example of a workflow from statistical natural language processing [Knight and Marcu 04] (right). The first workflow conducts the same kind of simulation for all ruptures of all faults being considered. It can be iterated again for several sites. Computation results are merged in later steps to provide a summary view back to the scientist. The second workflow reflects the parallel processing of a large text corpus by breaking it up into smaller chunks. The results from each chunk are merged in later steps. Both examples illustrate the regular structure that appears in many scientific workflows for processing large data sets. Similar kinds of structures have been shown in many other domains, such as workflows for creating image mosaics of astronomical data and for finding clusters of galaxies in the Sloan Digital Sky Survey [Taylor et al. 06].
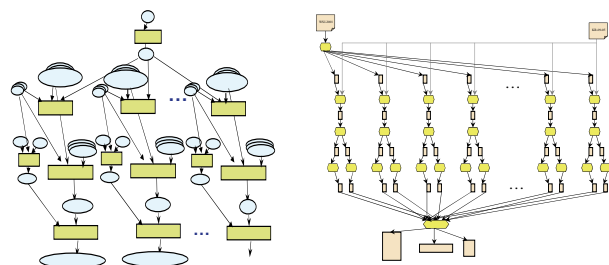

Figure 1. Example sketches of scientific workflows with hundreds or thousands of computations and data products. The figure illustrates the regular structure of two workflows we have implemented: an earthquake simulation (left) and a machine translation application (right).

An on-going problem is that these kinds of workflows are created using ad-hoc scripts that specify each individual job, create meaningful identifiers for all new data products of the workflow, and weave the dataflow connections among the individual jobs. The iterations across data sets are managed implicitly in the scripts. If the workflow needs to be changed, or a new workflow needs to be created with some of the same components, new scripts have to be written. This process is not very practical and is highly prone to errors, and thus it does not scale well as the workflow size increases. The validation of the resulting workflow is a challenge, mostly done by hand. In order to aid validation, there should be for example a way to specify that two separate collections of data used as input to a workflow should have the same cardinality, whatever that cardinality is, in order for the workflow to be valid.

There are important issues regarding the coupling of the workflow creation and the workflow execution. Ideally,

the workflow should be completely specified in terms of the kinds of computations to be performed and the kinds of data to be created, but be independent of the choice of hosts and other resources allocated for execution. This enables reproducibility of results, as well as workflow reuse. Therefore, it is desirable that the same workflow instance can be mapped at execution time to the resources that are available at that time. The Pegasus workflow mapping engine performs this mapping by binding data descriptions to one of many possible replicas, selecting hosts to execute the computations, moving data to where computation will occur, and moving data products to appropriate data repositories. If the entire workflow structure is known ahead of execution, Pegasus can do reservation and provisioning of resources. For this reason, it is useful to generate as complete a description as possible of the anticipated computations and data products before the workflow is executed.

In summary, our goal is to support the creation and execution of scientific workflows that process large collections of data sets and include hundreds or thousands of computational steps. This requires:

1. Creating workflow descriptions that orchestrate large amounts of computations appropriately described in terms of data and computation constraints
2. Handling data sets with many elements and managing the creation of iterative substructures in the workflow that process each of those elements
3. Generating appropriate metadata descriptions for all the new data sets created during execution.

Approach

To support the creation and validation of very large workflows, we have developed a new approach that uses semantic representations of workflows and data so that:

- **Workflow templates and instances are semantic objects** whose components, data requirements, and data products are represented in ontologies with appropriate constraints among them,
- **Data collections are specified with intensional descriptions in workflow templates and fleshed out to extensional descriptions in workflow instances**, with appropriate relationships as a data set is concretely specified during the creation of a workflow instance from a workflow template,
- **Intensional descriptions of collections of computations that offer appropriate abstractions for the repetitive structure of the workflows** at the template level, and expanded at the instance level once the data sets are specified extensionally.

There are several important benefits of this approach. It is easier to manage the complexity of workflow creation, since many aspects of it can be automated. By making the description of a workflow template very compact through intensional descriptions of data sets and computations, it is easier to create the basic structure of the workflow and validate it with smaller data sets. By specifying declaratively data collections and their constraints and properties we can validate the input data as well as intermediate data products of the workflow.

Execution requirements influence the representation of the workflows. First, workflow representations must support the creation of detailed descriptions of new workflow data products are required before execution so the system can detect pre-existing intermediate data and avoid unnecessary re-computation and optimize execution performance [Kim et al 06; Kim et al 07]. Second, the workflow representations must facilitate execution monitoring and failure recovery. Because many failures can arise when executing workflow components (insufficient memory, full file system, code bugs, etc), the system must manage the recovery from those failures and figure out what remains to be executed in the workflow. Workflows that have control constructs such as conditional branching and iteration are hard to manage because they require the execution system to have some persistent representation of their distributed execution state. For these reasons, the workflows that we have considered in our work to date are structured as directed acyclic graphs (DAGs) without control constructs.

The next section describes our current implementation of this approach and illustrates the main ideas through examples.

Wings for Pegasus

Wings is a workflow creation system that uses semantic representation and planning techniques to support the creation of workflow templates and instances, which are then submitted to Pegasus to create executable workflows. Wings and Pegasus provide a comprehensive workflow creation and execution framework. An overview of the Wings/Pegasus architecture is shown in Figure 2. Note that workflow templates and instances can be created by different users. Experienced scientists can create templates that comply with widely-accepted analyses that reflect valid scientific methodologies. Less experienced users can perform many analyses with different kinds of data by creating and executing workflow instances. A Composition Analysis Tool (CAT) assists users during template creation by checking that the template is valid and making suggestions based on the constraints and definitions in the ontologies [Kim et al. 04].
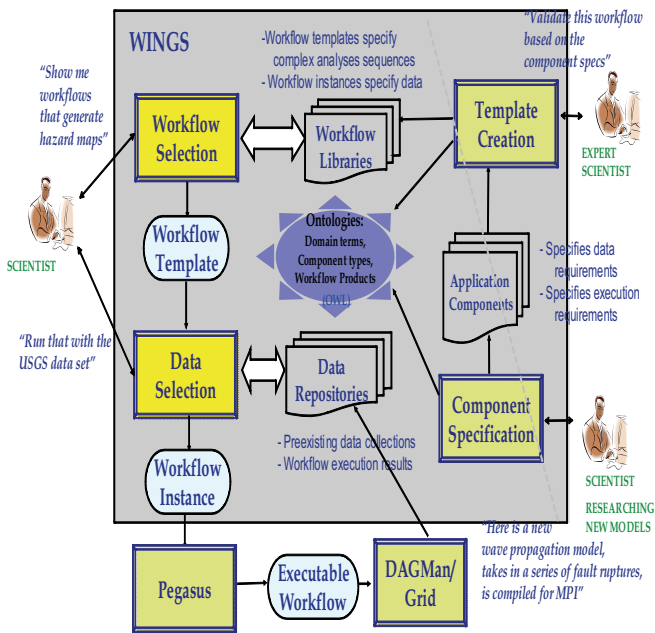
Figure 2. Overview of the Wings/Pegasus Architecture.

In Wings, workflow templates and instances are semantic objects and so are their components (nodes), the links among them, and the data generated by workflows. We use OWL-DL as the representation language, and Jena as the underlying reasoner. More specifically, we use a subset of OWL-DL that includes subClassOf, equivalentClass, and intersectionOf. We use rdf:list to represent ordered collections of files.

Data is represented as individual files that can be grouped into file collections. Nested collections are also supported. All the items within a collection must have a common type. The core definitions of the file ontology are:

- **File**: Represents the basic File class.
- **DataCollection** is used to represent a collection of objects (either files or other collections). These are the subtypes of DataCollection:
    - **CollOfDataCollection**: A collection of data collections.
    - **FileCollection**: A collection of Files

Computations (codes) are represented as workflow *components*. They can process several inputs and several outputs (each with its own unique id). A given input or output can take an entire data set. Components are organized in hierarchies of component types. The core definitions of the component ontology are:

- **ComponentType**: This is the top-level class of component types. A **Component** is an instance of this class and corresponds to an actual code that can be run.

- **ComponentCollection**: This represents a collection of components. It uses the property *hasComponentType* to specify the type of components in this collection.

ComponentCollection is used in nodes to indicate iterations over file collections.

Nodes in a workflow represent the component to be executed. A node in a workflow template can contain a single component or a component collection. A component collection is an intensional set of components, and will be expanded with concrete jobs when workflow instances are created. The definitions of nodes, and components are as follows:

- **Node**: Represents a node in the workflow. Uses a property *hasComponent* to specify the component that the node contains. Its range can be a any subclass of ComponentType or a ComponentCollection

A link in a workflow template carries data, and the type of data being carried must be consistent with the output data type of the origin node and the input data type of the destination node. Consequently, a link can carry single files or file collections. Links are defined as follows:

- **Link**: Represents a generic link in the workflow. It uses the properties *hasDestinationNode* and *hasOriginNode* to identify the destination and origin nodes respectively of the link. It also uses the properties *hasDestinationFileDescription* and *hasOriginFileDescription* to indicate the specific input/output for the components in the origin/destination nodes that this link connects. It has the following subclasses:
    - **InputLink**: These links do not have an origin node
    - **InOutLink**: These links must have an origin node and a destination node.
    - **OutputLink**: These links do not have a destination node.

Workflow templates are defined as including nodes that can be collections and links that can carry collections as well. This is done using the property *hasFile* of a link. Because we need to assert properties (metadata values) of these data collections, we need to represent them in the a-box as instances. Therefore, the data collections carried in the links are represented with *Skolem* instances, that is, instances that stand in for the actual data to be used in the instance. Properties and constraints can be asserted of these Skolem instances, which is important to validate the workflow. This requires that the entire workflow template is described with instances. Figure 3 shows an example of a workflow template, which corresponds to the following description:

```
<wflns:WorkflowTemplate rdf:ID="WT3">
    <wflns:hasLink
rdf:resource="#InputFCSG_to_Cone"/>
```

```
    <wflns:hasLink
rdf:resource="#InputFCSK_to_Cone"/>
    <wflns:hasLink
rdf:resource="#Inout_from_Cone_to_Cmany"/>
    <wflns:hasLink
rdf:resource="#InputFSY_to_Cmany"/>
    <wflns:hasNode rdf:resource="#Cone"/>
    <wflns:hasNode rdf:resource="#Cmany"/>
</wflns:WorkflowTemplate>
<wflns:InputLink rdf:ID="InputFCSG_to_Cone">
    <wflns:hasDestinationNode
rdf:resource="#Cone"/>
    <wflns:hasDestinationFileDescription
rdf:resource="&clib;#D1"/>
    <wflns:hasFile><FileCollection
rdf:ID="#FCSG"/></wflns:hasFile>
</wflns:InputLink>
<wflns:InputLink rdf:ID="InputFCSK_to_Cone">
    <wflns:hasDestinationNode
rdf:resource="#Cone"/>
    <wflns:hasDestinationFileDescription
rdf:resource="&clib;#D2"/>
    <wflns:hasFile><FileCollection
rdf:ID="#FCSK"/></wflns:hasFile>
</wflns:InputLink>
<wflns:InOutLink
rdf:ID="Inout_from_Cone_to_Cmany">
    <wflns:hasOriginNode rdf:resource="#Cone"/>
    <wflns:hasDestinationNode
rdf:resource="#Cmany"/>
    <wflns:hasOriginFileDescription
rdf:resource="&clib;#D3"/>
    <wflns:hasDestinationFileDescription
rdf:resource="&clib;#DC11"/>
    <wflns:hasFile><FileCollection
rdf:ID="#FCSZ"/></wflns:hasFile>
</wflns:InOutLink>
<wflns:InputLink rdf:ID="InputFSY_to_Cmany">
    <wflns:hasDestinationNode
rdf:resource="#Cmany"/>
    <wflns:hasDestinationFileDescription
rdf:resource="#D12"/>
    <wflns:hasFile><File
rdf:ID="#FSY"/></wflns:hasFile>
</wflns:InputLink>
<wflns:Node rdf:ID="Cmany">
    <wflns:hasComponent
rdf:resource="&clib;Cmany"/>
</wflns:Node>
<wflns:Node rdf:ID="Cone">
    <wflns:hasComponent>
       <clns:ComponentCollection>
          <clns:hasComponentType
rdf:resource="&clib;Cone"/>
       </clns:ComponentCollection>
    </wflns:hasComponent>
</wflns:Node>
```

Note that the number of elements in the collection is not specified since it is different for every instance to be created and depends on the size of the data set to be processed. Other constraints and properties of the set can be specified at the template level.

Workflow instances are specified by binding the inputs of a workflow template to specific datasets.

Wings can validate the creation of this instance by the user. Both collections provided as inputs for links L1 and L2 must have the same number of elements. They must also comply with any constraints defined in the template.

Wings takes the descriptions of the input data and propagates them to create descriptions for all the data products of the workflow. For example, the collection in link L3 in the example is now known to have the same number of elements as the one in the link L1.
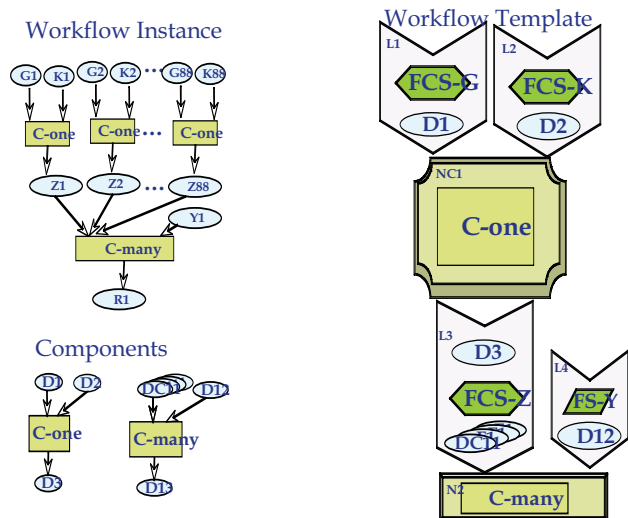

Figure 3. An illustrative example of a workflow template in Wings, shown on the right. A workflow instance from this template is shown on the top left. Also shown is a sketch of the components in the bottom left.

Note that this is an abbreviated specification of a workflow instance. Wings must then expand it to specify all the nodes that are to be executed.

Throughout the creation of the workflow instances, Wings propagates metadata information for all new data products. The metadata handling aspects of Wings are described in [Kim et al. 06; Kim et al. 07], showing this same algorithm in terms of how new metadata is generated for all new workflow data products.

Pegasus takes a very specific format for workflow instances. It is called a DAX (Directed Acyclic Graph in XML), and it is a directed acyclic graph of jobs where each job consists of code and file names for the inputs and outputs of the job. It also takes specifications of which data must be registered, since some intermediate data may be of temporary utility only but others may be useful to the user. These are specified as defaults in the workflow template. Wings generates a workflow in DAX format. Below is an excerpt of a very small DAX generated by Wings as an example:

```
<!-- part 1: list of all files used -->
  <filename file="file.f.a" link="input"/>
  <filename file="file.f.b1" link="inout"/>
  <filename file="file.f.b2" link="output"/>
<!-- part 2: definition of all jobs (at least
one) -->
  <job id="ID000001" name="removeDups"
version="1.0" level="3">
    <argument>-a top -T60  -i <filename
file="file.f.a"/>  -o <filename
file="file.f.b1"/> </argument>
    <uses file="file.f.a" link="input"
dontRegister="false" dontTransfer="false"/>
    <uses file="file.f.b1" link="output"
dontRegister="true" dontTransfer="true"
temporaryHint="true"/>
  </job>
```

```
  <job id="ID000002" name="countWords"
version="1.0" level="2">
    <argument>-a left -T60  -i <filename
file="file.f.b1"/> -o <filename
file="file.f.b2"/>  -p 0.5</argument>
    <uses file="file.f.b1" link="input"
dontRegister="false" dontTransfer="false"
temporaryHint="true"/>
    <uses file="file.f.b2" link="output"
dontRegister="true" dontTransfer="true"
temporaryHint="true"/>
  </job>
<!-- part 3: control-flow dependencies (empty for
single jobs) -->
  <child ref="ID000002">
    <parent ref="ID000001"/>
  </child>
```

We used Wings to create workflows for several applications including language translation and earthquake science. The template for the seismic hazard analysis, which we used to obtain the results reported in the next section, is shown in Figure 4. This template corresponds to the structure of the example workflow shown on the left of Figure 1. From a workflow template consisting of two dozen component and file types, we created workflow instances of more than 8,000 nodes that are then mapped by Pegasus and submitted for execution. This is described in detail in the results section below.
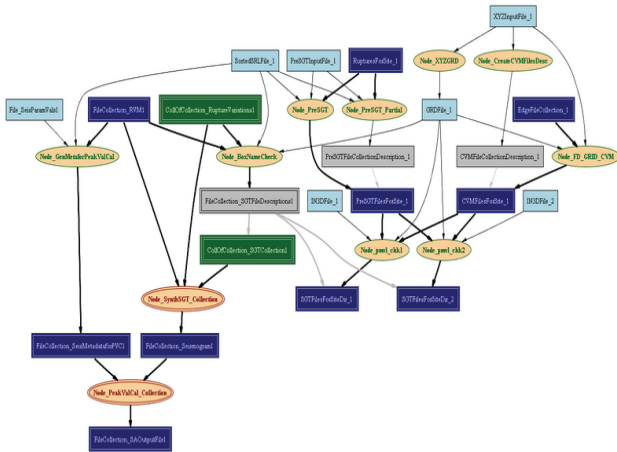

Figure 4. A template created in Wings for seismic hazard analysis in the CyberShake application. Double lines indicate collections of files or nodes.

Interleaving Workflow Generation and Execution

The seismic hazard analysis application presented in the previous section raises an additional and important challenge. The workflow instance cannot be fully generated by Wings and then submitted to Pegasus, because the descriptions of the input data (number of files in a collection for example) in the latter portions of the workflow are not known until some initial portions of the

workflow have been executed. Thus the workflow generation and execution processes need to be interleaved.
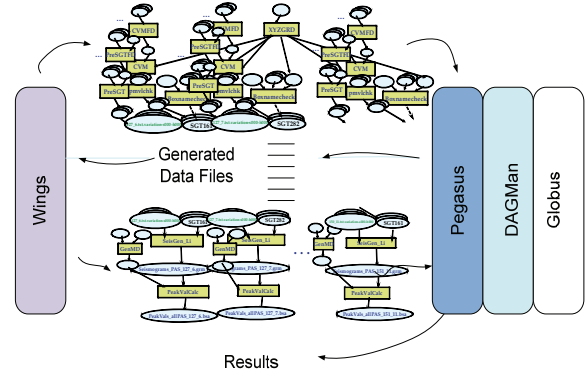

Figure 5. Interleaving workflow generation and mapping.

Figure 5 illustrates the interactions between Wings, Pegasus and the underlying execution services. First, Wings instantiates portions of the template that can be generated from the metadata available about the initial data. The resulting partial workflow instance is sent to Pegasus. Once the mapping is performed, the partial executable workflow is given to DAGMan for execution. Wings then uses the actual results and their corresponding metadata to continue the generation of the rest of the workflow instance. In the case of the seismic hazard workflow, two iterations of the workflow instance generation and mapping are needed.

Results

We show the results of applying our combined Wings/Pegasus workflow generation and mapping system for the seismic hazard analysis workflow in Fig 4, called CyberShake [Deelman et al. 06]. This workflow contains physics-based simulations that are designed to generate far more accurate hazard maps than it was ever possible with previous statistically-based estimates. SCEC scientists are validating and fine-tuning codes and parameters based on the results to date. When the validation process is finished, the refined workflow would be run for each of the thousands of sites in a seismic hazard map.

Wings creates the initial partial workflow instance for Pegasus. This partial workflow is small relative to the whole workflow instance, and can be run on a single local machine within 11 minutes. The partial workflow consists of 1 data stage-in job, 5 application domain jobs, 5 data stage-out jobs and 5 data registration jobs. The data obtained by running the partial workflow is then kept in a Wings-accessible location. These new datasets are used by Wings in creating the remainder of the workflow instance. The final workflow instance contains all the analysis required to generate hazard curves for a single site, taking as input thousands of possible fault ruptures each specified

in an input file for a total of 110,000. Curves from several hundreds of such sites are required to complete the hazard analysis for a region by constructing a hazard map.

The second portion of the workflow instance provided by Wings to Pegasus contains a total of 8043 application jobs, 3 of the application jobs are parallel codes that use MPI (Message Passing Interface) for inter-processor communications and run on hundreds of processors. Two of these MPI jobs generate the majority of the input data required for the analysis. This collection of data is a set of Strain Green Tensors. Sub collections of these tensors are consumed by 4017 Seismogram processing jobs that do the bulk of the application analysis to generate individual seismograms. Following each seismogram step there is a post-processing job that extracts the peak spectral acceleration value out of the generated seismogram. The total time to generate the workflow instance in Wings was 22 mins.

The complete workflow was run on the HPCC (High Performance Computing & Communications) cluster at USC, the second fastest in academia with 1,830 dual processor nodes and 10.75 Teraflops. Not all of the resources are available in the general queue, so at any given time the resources available for execution of our application were never more than 144 nodes.

Pegasus first tries to reduce the workflow if any output data are already available elsewhere (this did not happen in this case). The jobs are then mapped to execute on particular sites, and auxiliary jobs such as data staging (input as well as output) are added along with output data registration jobs that register the data products generated into data repositories.

The executable workflow had a total of 24,135 jobs, with 15 data stage-in jobs, 8,040 application jobs, 8,040 stage-out jobs and 8,040 data registration jobs. Pegasus took less than15 minutes to plan this workflow.

The MPI jobs ran on 144 nodes dual processors with the first MPI job (FD_GRID_CVM) which fills a given mesh with a velocity model taking about 33mins, 36secs. The two main MPI jobs, pmvl3d1 and pmvl3d2, which generate the Strain Green Tensors, ran for 25 hours, 30 minutes and 29 hours, 43 minutes respectively.

As seen in Figure 6, the time taken by a single seismogram job varies widely from a few minutes to a few days. This is because each seismogram operates on different amounts as well as different regions of the dataset. In contrast to this the post processing jobs (PeakSA jobs) as seen in Figure 7 finish in less than one minute. The runtime of the entire workflow was approximately 16562 CPU hours (approximately 1.9 CPU years).
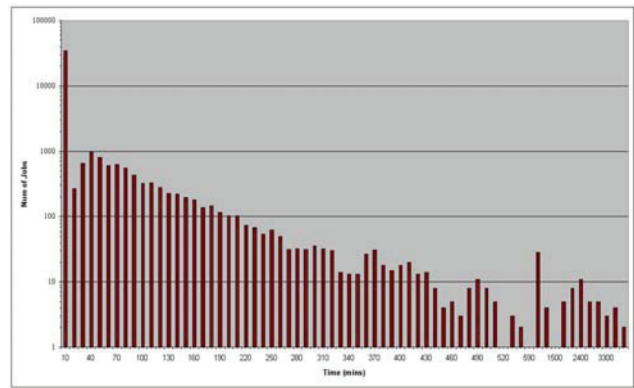

Figure 6. Histogram of the number of seismogram jobs vs. the time each job took to complete.
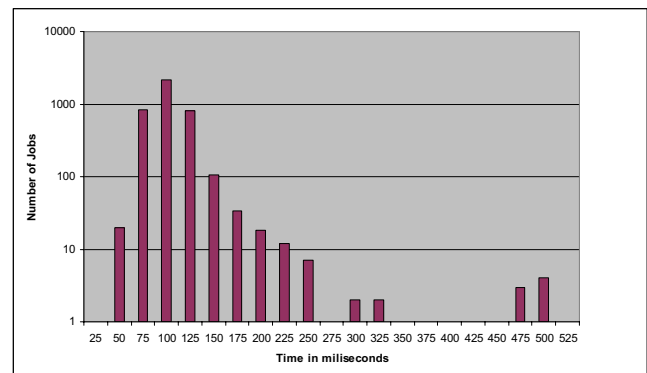

Figure 7. Histogram of the number of PeakSA jobs vs the time each jobs took to complete.

The workflow stopped 7 times and was restarted successfully each time based on information generated by Condor DAGMan, our workflow execution engine called "rescue DAGs"). The workflows stopped due to various failures including the unavailability/failure of the execution nodes, exceeding the wall clock time limit on the remote system by the remote scheduler, and other such typical failures.

Conclusions

We have described a new approach to creating and validating very large scientific workflows that process data sets through computational steps that are executed in distributed grid environments. Our approach uses semantic descriptions of workflow templates and workflow instances where all their constituents are semantic objects that are described with properties and workflow level constraints. Once a workflow template is created and validated by an experienced user, it is easy for more junior scientists to create sophisticated analyses simply by specifying input data for pre-defined templates. The system ensures that the input data specified is appropriate given the definitions in the workflow template, and automatically generates a workflow instance that can be mapped to execution resources. We have implemented this

approach in the Wings system, and it is fully integrated with the Pegasus workflow mapping and execution system. We have also addressed the issue of incrementally generating workflow instances, interleaving workflow generation with execution. We have demonstrated our approach on a seismic hazard analysis application, CyberShake. The workflow took approximately 1.9 CPU years to execute on high-performance resources.

Augmenting workflows with semantic descriptions has additional benefits for scientists. They enable searches of previous workflow instances or templates in cases where a "similar" analysis is sought. A scientist may want to find out if someone else has come across a particular problem or used a particular methodology. The explicit representations of the workflow templates and instances also support result reproducibility. Templates provide a means of systematically and diligently describing the high-level analytical steps involved. Workflow instances created from those templates are valid since they follow established methodology (as described in the template) and the data complies with the constraints expressed in the workflow.

References

Blythe, J., Deelman, E., Gil, Y., and C. Kesselman. "Transparent Grid Computing: A Knowledge-Based Approach". Proceedings of the 15th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), Acapulco, Mexico, August 12-14, 2003.

Deelman, E. et al. " Mapping Abstract Workflows onto Grid Environments", *Journal of Grid Computing*, Vol. 1, No. 1, 2003.

Deelman, E. et al. "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems". *Scientific Programming Journal*, Vol 13(3), 2005.

Deelman, E. et al., "Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance tracking: The CyberShake Example," e-Science 2006, Amsterdam, December 4-6, 2006. Best paper award.

Deelman, E. and Gil, Y. (Eds.) *Final Report of the NSF Workshop on Challenges of Scientific Workflows*, National Science Foundation, Arlington, VA, May 1-2, 2006. At www.nsf.gov/events/event_summ.jsp?cntn_id=108411&org=IIS.

Frey, J., Tannenbaum, T., Foster, I., Livny, M., Tuecke, S., "Condor-G: A Computation Management Agent for Multi-Institutional Grids". In 10th International IEEE Symposium on High Performance Distributed Computing (HPDC), 2001.

Gil, Y. "Workflow Composition: Semantic Representations for Flexible Automation", in "Workflows for e-Science", Deelman, E., Gannon, D. Shields, M., and Taylor, I. (Eds), Springer Verlag, 2006.

Kim, J., Spraragen, M., and Y. Gil. "An Intelligent Assistant for Interactive Workflow Composition", Proceedings of the 2004 International Conference on Intelligent User Interfaces (IUI), Madeira Islands, Portugal, January 2004.

Kim, J., Gil, Y., and Ratnakar, V. "Semantic Metadata Generation for Large Scientific Workflows", Proceedings of the Fifth International Semantic Web Conference (ISWC-06), Athens, GA, November 5-9, 2006.

Kim, J., Deelman, E., Gil, Y., Mehta, G., and Ratnakar, V. "Provenance Trails in the Wings/Pegasus Workflow System", Concurrency and Computation: Practice and Experience, Special Issue on the First Provenance Challenge, 2007.

Knight, K. and D. Marcu. Machine Translation in the Year 2004. Proceedings of the 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2005.

Ludäscher, B., et al. "Scientific Workflow Management and the Kepler System". *Concurrency and Computation: Practice and Experience* Special Issue on Workflow in Grid Systems, August 2006.

Maechling, P. et al. "Simplifying Construction of Complex Workflows for Non-Expert Users of the Southern California Earthquake Center Community Modeling Environment". In ACM SIGMOD Record, Special issue on Scientific Workflows, 2005.

Oinn, T. et al. "Taverna: Lessons in creating a workflow environment for the life sciences". *Concurrency and Computation: Practice and Experience* Special Issue on Workflow in Grid Systems, August 2006.

Taylor, I.J.; Deelman, E.; Gannon, D.B.; Shields, M. (Eds.) *Workflows for e-Science: Scientific Workflows for Grids*, Springer Verlag, 2006.

Tuchinda, R., Thakkar, S., Gil, Y. and E. Deelman. "Artemis: Integrating Scientific Data on the Grid". Proceedings of the 16th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), San Jose, CA, July 25-29, 2004.

Wieczorek, M., Prodan, R. and T. Fahringer. "Scheduling of Scientific Workflows in the ASKALON Grid Environment." *ACM SIGMOD Record*, special issue on Scientific Workflows, 2005.