Pegasus on the Virtual Grid: A Case Study of Workflow Planning over Captive Resources

Yang-Suk Kee
Oracle USA Inc.
yang.seok.ki@oracle.com
Eunkyu Byun
Division of Computer Science, Korea Advanced Institute of Science and Technology
ekbyun@camars.kaist.ac.kr
Ewa Deelman, Karan Vahi
Information Sciences Institute, University of Southern California
{deelman, vahi }@isi.edu
Jin-Soo Kim
School of Information and Communication Engineering, Sungkyunkwan University
jinsookim@skku.edu

Abstract

As scientific applications in a variety of disciplines are being actively developed and computing environments become ever more complicated and dynamic, it is becoming challenging to leverage existing cyber-infrastructures and achieve robust and efficient computing. This paper presents a case study of planning and executing application workflows over provisioned resources. This work integrates the Pegasus workflow framework with the Virtual Grid resource provisioning system. Through this preliminary study, we have identified two key issues that need to be addressed: 1) developing a resource capacity estimate which synthesizes efficient resource descriptions for the Virtual Grid from application workflows and 2) execution site information publication which provides resource configurations for Pegasus through the devirtualization of provisioned resources. In addition, we discuss the challenges and research opportunities that we need to explore in order to exploit the advanced features of the systems.

1. Introduction

The advance of cyber-infrastructures has enabled scientists to explore complicated natural phenomena. Many success stories inspire researchers to solve more complex problems in a variety of disciplines [1-5]. One of key challenges in this exploration is how to transition the scientific knowledge and legacy software to new computing environments. A solution gaining in popularity is the use of high-level application descriptions such as workflows that can specify the structure and overall behavior of applications in a platform-independent manner. Typically, a workflow is represented as a Directed Acyclic Graph (DAG) that consists of nodes, representing tasks, and edges, which denote data/control dependencies between tasks. When an application is specified via a workflow, workflow management systems such as Pegasus [6], Askalon [7], Triana [8], and others manage and execute workflows on distributed resources.

Coordination and management of distributed resources are challenging issues for a number of communities in computer science. The noticeable achievements in distributed computing are the state-of-art resource virtualization technologies such as the Virtual Grid [9], the virtual cluster [10], and compute clouds [11], which take into account a variety of factors such as availability, performance, cost, etc and enable on-demand resource provisioning. Regardless of the base technology that implements virtualization, these technologies commonly encapsulate the complexity of resource management and provide uniform interfaces to resources.

Workflow management systems can potentially benefit from these resource provisioning technologies. First, workflow management systems can be insulated

from the resource management complexity, which subsequently reduces the design complexity of workflow management systems. Second, workflow management systems can take advantage of the benefits of efficient resource management of provisioning systems since the provisioning systems can optimize resource allocations and assure the quality of resources in terms of computing and communication performance, reliability, availability, and so on. Finally, workflow management systems can exploit an extended resource universe with minimal efforts through resource provisioning. At the same time, provisioning systems can have well-defined interfaces to applications via the workflow management systems. In addition, provisioning systems can experience a variety of structural and behavioral characteristics of applications via the workflow management systems which can contribute to the improvement of the provisioning systems themselves. Finally, provisioning systems can extensively evaluate their performance and the quality of provisioning techniques in real settings against the applications already supported by existing workflow management systems.

In this paper we focus on two representative workflow management and resource provisioning systems. Pegasus-WMS [6] is a workflow management system which conducts workflow planning with detailed information about computation and data against a given resource set. The Virtual Grid (VG) [9] is a programmable resource provisioning framework, which enables users to instantiate resources of quality (performance, reliability, availability, etc.) on demand. We believe that the integration of Pegasus with the Virtual Grid can not only deliver the aforementioned benefits but can also have significant synergetic effects on scientific computing. As a result, Pegasus can conduct more efficient workflow planning in terms of cost, performance and quality with the resources provisioned by the Virtual Grid.

As a preliminary study, this paper focuses only on a basic integration and discusses two critical issues to be solved. The paper identifies two core interfaces for the interactions between Pegasus-WMS and the Virtual Grid: resource capacity estimation and execution site information publication. We implement these interfaces in a proxy system named Pegasus-VG proxy which orchestrates Pegasus-WMS and the Virtual Grid and enables workflow execution over provisioned resources. Finally, we present several challenges and the new research opportunities identified through this study.

The rest of this paper is organized as follows. In Section 2, we give a brief overview of Pegasus-WMS and the Virtual Grid. Then, we discuss a simple working scenario and the integration issues in Section 3. We summarize the related studies in Section 4 and finally conclude this paper, discussing the future research directions in Sections 5 and 6.

2. Background

The common design philosophy of Pegasus and the Virtual Grid is that of separation of concerns, which can simplify complex problems and can provide feasible or even better solutions. Specifically, Pegasus assumes that an application can be developed and executed independently of the target execution system using a high-level representation (i.e., application workflow) while the Virtual Grid aims for resource management to be isolated from the application and be virtualized via a resource abstraction (i.e., the Virtual Grid). Since these two systems compliment each other, the integration of the two systems can simplify the overall design complexity of scientific computing, make applications portable, and achieve good performance. In the following subsections, we give a brief overview of Pegasus and the Virtual Grid.

2.1. Pegasus-WMS

Pegasus [6] is a workflow management system which maps abstract workflows onto resources. The abstract workflow describes the logical topology and functionality of the application and executes workflow tasks using Condor's DAGMan [12]. Figure 1 illustrates a typical lifecycle of application workflow in the Pegasus framework. Pegasus takes an abstract workflow, maps it onto the available resources, and invokes DAGMan to execute the workflow. DAGMan then walks through the workflow and releases the workflow tasks in the right order to Condor-G [12]. The latter submits the tasks to the remote resources (via Globus [13]) for execution.

2.1.1. Workflow Creation. The first phase of the workflow lifecycle is to create an abstract workflow for the application. An abstract workflow [6] is a logical representation of control and data flow of the application, independent of resources. An abstract workflow is composed of the tasks described in terms of logical transformations and logical input and output filenames. Depending on their backgrounds, circumstances, expertise in workflow technologies, scientists can create abstract workflows, directly using predefined XML schemas, using the Pegasus Java API, or using the intelligent workflow editor, Wings [14].

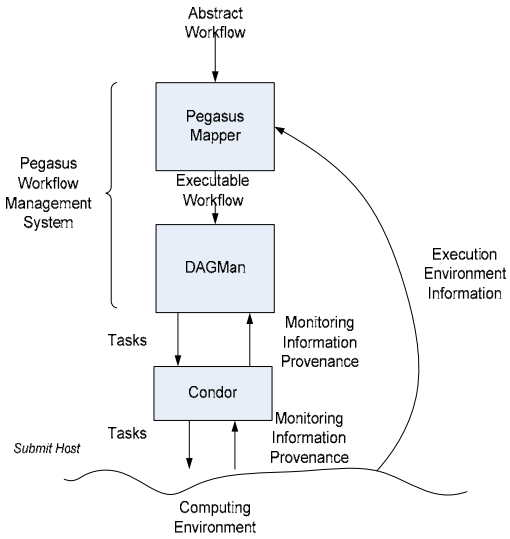


Figure 1. Workflow lifecycle in the Pegasus framework

2.1.2. Workflow Planning. The goal of Pegasus is to find a good mapping of workflow tasks to available execution sites. Pegasus transforms an abstract workflow through a series of refinements to a concrete workflow which can be executed on the resources. Pegasus first indentifies the resources available to the user. Second, Pegasus simplifies the workflow based on the historic computation results that Pegasus keeps track of. If the results of workflow tasks are already available (for example when the data were previously computed and stored), such computations can be replaced with simple data transfers. Pegasus then schedules the tasks by selecting appropriate resources, based on the available resources and their characteristics as well as the location of input data. Pegasus relies on information services such as MDS (Meta-computing Directory Service) [15] and others to retrieve resource characteristics and RLS (Replica Location Service) [16] to locate historic data. For efficient execution, Pegasus can cluster jobs together in cases where a number of small granularity jobs are destined for a same computing resource. Next, Pegasus augments the workflow with tasks that explicitly perform data transfers. The final step is to write out the mapping results in a Condor input file and the associated submit files which can be interpreted by Condor DAGMan [12].

2.1.3. Workflow Execution. Pegasus uses DAGMan for workflow execution. DAGMan is a workflow execution engine which submits jobs to Condor in an order represented by a Condor DAG. DAGMan processes a DAG input file and the associated Condor submit file(s). It is responsible for scheduling, recovery, and reporting on the set of tasks submitted to Condor.

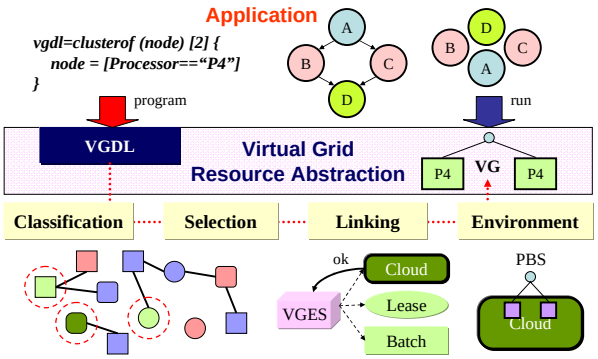


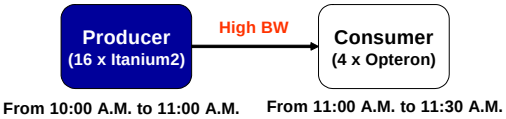
Figure 2. Scientific Computing via Virtual Grid

In the case of distributed resources such as Grids, DAGMan submits jobs to Condor-G. Condor-G [12] is a job management system which locates resources and submits, cancels, and monitors jobs on the behalf of the users. Condor-G provides a uniform resource interface over heterogeneous resources via a Condor pool and enables users to transparently access resources and manage jobs. For remote execution, Condor-G relies on the Globus Toolkit [13] which enables secure accesses to resources, uniform accesses to a variety of batch systems, file staging, status monitoring, and so on.

2.2. The Virtual Grid

The Virtual Grid (VG) [9, 17] is a high-level resource abstraction enabling virtualized distributed computing across heterogeneous resources. Separated by the VG abstraction layer, a user specifies the characteristics of the desired execution target while the Virtual Grid execution system (VGES) [17, 18] renders the abstraction. VGES manages the uncertainty and dynamics associated with queuing delays, failures, and contention effects underneath the abstraction. As such, VG can isolate application development, scheduling, and optimization procedures from the complexity of managing resources.

Figure 2 illustrates a computing scenario in the context of VG. First, users abstract their resource requirements and program the structure and properties of desired execution target in the resource description language named VGDL [17]. VGES then analyzes the VGDL description and compiles a collection of quality resources into a VG instance through resource selection, binding, and environment setup. Once a VG is created, users can operate on the VG to retrieve re-



(a) Resource requirement

```
Producer-consumer =
  Producer = ClusterOf (nd1) [16]
                    <12/12/2006@10:0:0[EXEL], 1:0:0[EXEL]>
          { nd1 = [Processor == "Itanium 2"] }
  HighBW
  Consumer = ClusterOf (nd2) [4]
                    <12/12/2006@11:0:0[EXEL], 0:30:0[EXEL]>
          { nd2 = [Processor == "Opteron"] }
```

(b) Resource specification

Figure 3. An example of resource specification in VGDL
(***ClusterOf*** denotes a set of homogeneous resources; ***HighBW*** represents a high bandwidth network; *<xx/xx/xx@xx:xx:xx[avail_prob], xx:xx:xx[rel_prob]>* describes the start time and the availability of resource arrival and the duration and the reliability of resource allocation.)

source information, to execute the application tasks, and to manage the resource collection. In the following sections, we highlight three components that implement the Virtual Grid concept.

2.2.1. Resource description language. VGDL (Virtual Grid Description Language) [17] is a resource description language, which provides constructs for expressing constraints on the attributes associated with computing resources. Users can describe resource requirements in terms of desired values of attributes (e.g., process type, memory capacity). The key advances of VGDL are the capability of hierarchical qualitative specification of resource aggregates and network proximity. The qualitative approach enables applications to construct simple and robust specifications regardless of technology advances.

Users can capture simple widely-used resource abstractions to achieve the portability in design and to manage the complexity of resource environments via VG resource aggregates. Moreover, VG connectivity operators express the coarse notions of network proximity between aggregates in terms of latency and bandwidth. Resource aggregates and network connectivity operators enable users to compose individual resources into an arbitrary structure. At the same time users can specify application-specific resource quality in a user-defined resource ranking function. Users can specify the temporal resource availability for time-constrained applications [9]. Moreover, users can specify the availability probability of resources at a certain time (i.e., start time) and the reliability probability of resources for a certain duration.

Figure 3 illustrates how a program that consists of a pair of a producer and a consumer can be described in VGDL. In this example, the user needs two tightly coupled clusters in a temporal order, each of which has 16 Itanium processors and 4 Opteron processors, respectively. In addition, the user requires a high level of confidence that the resources will be allocated in a timely manner and should be highly reliable while the application is running. Two ClusterOf aggregates are used to represent clusters and a HighBW connectivity operator is used to tightly couple two clusters.

2.2.2. Resource compilation. Compiling a VG instance from a VGDL specification is the process of configuring a network of resources for the specification [9]. A VG compilation consists of selection and binding. Selection is used to identify the possible resources satisfying the specification in the resource universe while binding it to secure the resource allocation with a certain confidence.

The Virtual Grid reformulates the resource selection problem through a resource classification and an online search for efficient selection and implements the fast resource selection by exploiting the relational database technology [18]. On the other hand, the likelihood of binding success is contingent upon the resource management policy of the resource manager. VG exploits the compositional structure of resource specification and identifies the components that can be allocated independently. The Virtual Grid probabilistically guarantees the success of resource binding by identifying multiple solutions for each component [18].

2.2.3. Personal cluster. A personal cluster is a virtual cluster instantiated on demand from physical resources. This gives users an illusion that the instant cluster is dedicated to the user for a certain time period [20]. A personal cluster reserves a partition of resources and enables a uniform, cost-effective use of batch resources. The user has a dedicated cluster under the control of a private resource manager. As such a personal cluster can provide a uniform job/resource management environment over heterogeneous resources regardless of system-level resource management paradigms.

The current implementation is based on the WS-based Globus Toolkit [22] and a PBS [21] installation. The Personal cluster uses the similar mechanism to Condor glidein [12]. Once a system-level resource

manager allocates a partition of resources, a user-level PBS scheduled on the resources holds the resources for a user-specified time and a user-level WS-GRAM (configured at runtime for the partition) accepts jobs from the user and relays them to the user-level PBS. As a result, users can bypass the system-level resource manager and benefit from the low scheduling overhead with the private scheduler.

3. Pegasus-WMS on the Virtual Grid

The intuition behind integrating Pegasus-WMS with the Virtual Grid is that a workflow planning system can benefit from the advanced resource management services of the provisioning system and consequently enable robust and efficient computing. However, the scope of this paper is limited to a simple scenario of interactions between Pegasus and the Virtual Grid. To minimize impacts on both systems and enable independent development, we propose a proxy system named Pegasus-VG proxy which implements the interfaces and the services required for integration, insulating the systems from each other. In this section, we present our computing scenario and detail the issues of this study.

3.1. Computing Scenario

As discussed in Section 2.1, a workflow basically goes through three phases in the Pegasus framework: creation, planning, and execution. Workflow creation is a totally application-specific phase, independent of target resources. Pegasus only interacts with the Virtual Grid at the planning and execution phases. An issue here is that the resources in VG are presented in a virtualized manner. However, Pegasus needs concrete resource information such as hostname, port number, directory name, and so on. Therefore, the Pegasus-VG proxy devirtualizes the provisioned resources and allows Pegasus to follow the normal planning and execution processes without modifying the Pegasus internals.

Figure 4 illustrates how Pegasus interacts with the Virtual Grid in a simple devirtualization scenario. First, the user specifies application-specific knowledge about resource requirements (e.g., processor type, memory capacity) and the application-level information (e.g., locations of executable, data, and replica) needed to run his/her application in the Pegasus framework. When Pegasus' planning is invoked with this abstract workflow, a wrapper program for the Pegasus planning command intercepts the resource information before the ordinary planning of Pegasus takes place and contacts the Pegasus-VG proxy. The proxy then synthesizes a vgDL description through a resource capacity estimate and instantiates a VG on behalf of Pegasus. The proxy devirtualizes the VG instance and generates a new *site catalog* which is a formal input of Pegasus describing the information of the VG-provisioned resources. The site catalog is then sent back to the wrapper and finally the wrapper invokes the ordinary Pegasus planner with the site catalog. Pegasus now continues its normal planning process. Since a site catalog contains the detailed information about how to access resources such as hostname, GRAM port number, scheduling adaptor, and so on, Pegasus can run applications directly on the resources via DAGMan as usual.

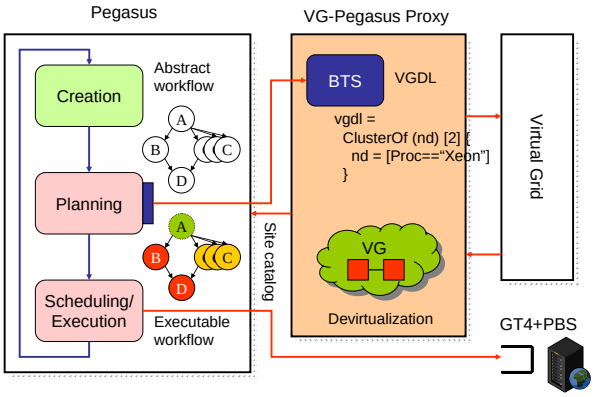
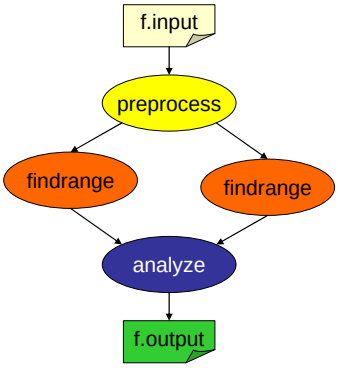


Figure 4. Pegasus on Virtual Grid

3.2. Issues

3.2.1. Resource capacity estimation. A critical capability required for this integration is to synthesize a vgDL specification from application workflow(s). The most dominant attribute from the perspective of high-level workflows is the number of processors required by the application because the resource size is one of the important factors in determining the makespan of workflow application and the cost of resource allocation. If the number of resources is large, the parallel execution of independent tasks can reduce the execution time while too many resources can cause low resource utilization, high scheduling overhead, and high cost. On the other hand, if the number of resources is too small, the execution time of the workflow can increase. Therefore, it is important to estimate the number of resources as small as possible so as to complete a workflow within a given deadline. This problem is different from the conventional workflow scheduling or cost-optimization problems, which aim at minimizing the application's runtime against a fixed set of resources.



(a) Black-diamond workflow

```
<!-- part 1: list of all files used (may be empty) -->
<filename file="f.input" link="input"/>
<filename file="f.intermediate" link="input"/>
<filename file="f.output" link="output"/>
<filename file="keg" link="input">

<!-- part 2: definition of all jobs (at least one) -->
<job id="ID000001" namespace="pegasus" name="preprocess" version="1.0" >
<argument>
   -i <filename file="f.input"/> -o <filename file="f.intermediate"/>
</argument>
<uses file="f.input" link="input" register="false" transfer="true"/>
<uses file="f.intermediate" link="output" register="false" transfer="false">
</job>

<job id="ID000002" namespace="pegasus" name="analyze" version="1.0" >
<argument>
   -i <filename file="f.intermediate"/> -o <filename file="f.output"/>
</argument>
<uses file="f.intermediate" link="input" register="false" transfer="true"/>
<uses file="f.output" link="output" register="true" transfer="true"/>
</job>
…
```

(b) A fragment of Pegasus DAX

```
Diamond = ClusterOf [2] (nd) [, 01:00:00]{ nd = [Processor == "Xeon"] }
```

(c) A synthesized vgDL description requesting a cluster consisting of 2 Xeon processors

Figure 5. vgDL synthesis for a black-diamond application workflow

This issue was already addressed by several studies [23, 24]. In particular, the BTS algorithm estimates the resource capacity very efficiently [24]. The algorithm scales well even with very complex workflows and provides a good estimate of resources needed - close to the optimal for a variety of workflows. Moreover, the resource estimate is abstract and independent of description languages and selection mechanisms so it can be easily integrated with any resource description language and provisioning system.

The Pegasus-VG proxy has a wrapper of BTS which takes the abstract workflow (DAX) from Pegasus and generates a vgDL description. For example, Figure 5 (a) depicts a simple synthetic application. The user can describe the structure and behavior of the application in a DAX as shown in Figure 5 (b). Then, the BTS wrapper extracts the workflow information (e.g. task and link) and invokes BTS. BTS then estimates the number of processors required for the workflow and synthesizes a vgDL description as shown in Figure 5 (c) using the resource requirements (e.g., processor type, clock rate, memory capacity) given by the client. In this example, the workflow needs a cluster consisting of 2 Xeon processors. The processor requirements are embedded into the node definition while the cluster size is determined automatically by BTS.

3.2.2. Site catalog publication. Pegasus conducts workflow planning against resources described in site cataloges. The BTS wrapper generates a complete vgDL specification through the resource capacity estimate and then the proxy acquires resources via the Virtual Grid. Once a VG instance is successfully created, the proxy devirtualizes the Virtual Grid and creates a site catalog describing the provisioned resources.

Figure 6 (a) is a site catalog created for the vgDL specification presented in Figure 5 (c). As discussed in Section 2.3, the Virtual Grid deploys a personal cluster based on Globus Web Services and PBS to the provisioned resources. The key information that the proxy retrieves is the information related to the WS-GRAM service such as the Globus version, service endpoint, batch scheduler type, and so on. In this example, the cluster has 2 processors and provides a GRAM web service for PBS available at https://cat7.kaist.ac.kr: 9000.

Pegasus then generates a Condor input file and the associated Condor submit files. The key information in a submit file is *universe*, *grid_type*, *globusscheduler*, and *jobmanager_type*. *Universe* specifies the Condor execution environment, *grid_type*, the Globus version installed on the remote resource, *globusscheduler*, the

```
<sitecatalog xmlns="http://pegasus.isi.edu/schema/sitecatalog" …>
<site handle="cat7" gridlaunch="/home/globus/pegasus-2.1.0/bin/kickstart" sysinfo="INTEL32::LINUX">
<profile namespace="env" key="PEGASUS_HOME">/home/globus/pegasus-2.1.0</profile>
<profile namespace="env" key="GLOBUS_LOCATION">/usr/local/globus-4.0.7</profile>
<profile namespace="env" key="LD_LIBRARY_PATH">/usr/local/globus-4.0.7/lib</profile>
<profile namespace="env" key="JAVA_HOME">/opt/jdk</profile>
<profile namespace="condor" key="grid_type">gt4</profile>
<profile namespace="condor" key="jobmanager_type">PBS</profile>
<lrc url="rlsn://cat7.kaist.ac.kr" />
<gridftp url="gsiftp://cat7.kaist.ac.kr:2811" storage="/home/globus" major="4" minor="0" patch="7" />
<jobmanager universe="transfer" url="https://cat7.kaist.ac.kr:9000/wsrf/services/ManagedJobFactoryService" major="4" minor="0"
patch="7" total-nodes="2" />
<jobmanager universe="vanilla" url="https://cat7.kaist.ac.kr:9000/wsrf/services/ManagedJobFactoryService" major="4" minor="0" patch="7"
total-nodes="2" />
<workdirectory>$HOME/workdir</workdirectory>
</site>
…
</sitecatalog>
```

(a) A simplified site catalog published for the provisioned cluster consisting of 2 Xeon processors

```
environment = GLOBUS_LOCATION=/usr/local/globus-4.0.7;JAVA_HOME=/opt/jdk;PEGASUS_HOME=/home/globus/pegasus-2.1.0;
LD_LIBRARY_PATH=/usr/local/globus-4.0.7/lib;
arguments = "-n black::preprocess:1.0 -N black::top:1.0 -R cat7 /home/globus/pegasus-2.1.0/…/black-preprocess-1.0 -a top -T60 -i f.a -o f.b1
f.b2"
error = /home/globus/pegasus-2.1.0/temp/dags/globus/pegasus/black-diamond/run0001/preprocess_ID000001.err
executable = /home/globus/pegasus-2.1.0/bin/kickstart
globusrsl = (jobtype=single)
globusscheduler = https://cat7.kaist.ac.kr:9000/wsrf/services/ManagedJobFactoryService
grid_type = gt4
jobmanager_type = PBS
output = /home/globus/pegasus-2.1.0/temp/dags/globus/pegasus/black-diamond/run0001/preprocess_ID000001.out
remote_initialdir = /home/globus/pegasus-2.1.0/temp/$HOME/workdir/pegasusexec/globus/pegasus/black-diamond/run0001
transfer_error = true
transfer_executable = false
transfer_output = true
universe = grid
+pegasus_generator = "Pegasus"
+pegasus_version = "2.1.0"
+pegasus_wf_name = "black-diamond-0"
+pegasus_wf_time = "20080909T182531+0900"
+pegasus_job_id = "preprocess_ID000001"
+pegasus_site = "cat7"
Queue
```

(b) A simplified Condor submit file generated by Pegasus for the DAX against the site catalog

Figure 6. Planning of abstract workflow against provisioned resources

end point information to access the GRAM web service, and *jobmanager_type*, the batch scheduler name of the remote resource manager. Figure 6 (b) illustrates a Condor submit file generated by Pegasus 2.1.0 version. In this example, *universe* is grid, *grid_type* is gt4, *globusscheduler* is https://cat7.kaist.ac.kr:***9000***/wsrf/services/ManageJobFactoryService, and *jobmanager_type* is PBS; this information is extracted from the site catalog presented in Figure 6 (a).

Pegasus can conduct normal planning for the DAX presented in Figure 5 (b) with this site catalog and generate an executable workflow (Condor DAG) as shown in Figure 6 (b). Since Globus Web Services and a PBS job manager are already deployed on the provisioned resources, Condor DAGMan can run the workflow tasks directly on the resources by using the GT4 options of Condor submit command.

4. Related Work

In [25], we discussed workflow planning over provisioning resources across multiple sites or VOs (Virtual Organization) [26]. We used Pegasus as a workflow management framework and Condor-G [12] as a

provisioned framework. Condor-G working with Condor [27] can be regarded as a resource provisioning framework which can support advance reservation [28] and dynamic resource acquisition using the glidein mechanism [12]. However, this integration was done ad-hoc. In contrast to Condor-G, the Virtual Grid is not only a resource provisioning framework but also a resource programming and virtualization framework. In addition, VG supports a variety of resource management paradigms such as best-effort space-sharing, advance reservation, price-based reservation [29], compute cloud [11], and time-sharing resources. Moreover, the Virtual Grid can deploy any user-level job manager including PBS on demand.

In addition to the system-level integration, there have been several studies on scheduling application workflows on provisioned resources [30-32]. These studies demonstrate the potential of integrating workflow management systems with resource provisioning systems. A common lesson from these studies is that provisioning resources can deliver good and predictable performance to applications, compared to the best effort space-sharing of resources. The Virtual Grid can instantiate resource collections that have specific characteristics across distributed resources. As such it can meet the assumption that the application-level schedulers have.

5. Conclusions & Discussions

This paper presented a case study of workflow planning and execution over provisioned resources through integrating the Pegasus workflow framework with the Virtual Grid resource provisioning system. We identified that the resource capacity estimate and the site catalog publication through resource devirtualization are two key features required for this basic integration. As an implementation, we introduced the Pegasus-VG proxy as a common ground where Pegasus interacts with the Virtual Grid. This proxy-based implementation enables an easy integration without changing the internals of either of the systems. We believe this integration enables scientists to explore their problems more efficiently over distributed resources. Since resource provisioning is opaque to the user, the application development cycle is the same even with more advanced resource allocation.

This study is the first step in understanding the issues of integrating workflow management systems with resource provisioning systems. Through this integration, we also identified several challenges. Pegasus can partition a workflow into multiple subworkflows which can be planned and executed separately over time. Since allocating a large set of resources for a long time is expensive, difficult and exposes applications to resource failures, provisioning resources over time can be cost-efficient and even provide better performance and reliability. For temporal resource provisioning for multiple subworkflows, users can specify time constraints on their resource specifications and let the Virtual Grid allocate resources according to the user-specified schedules. On the other hand, users can allocate resources on-the-fly for each subworkflow whenever Pegasus conducts planning. In either case, the Virtual Grid will optimize resource allocation, taking into account resource characteristics. In the future, we will explore to what extent the temporal resource provisioning can improve application performance as compared to the static resource allocation.

Even though the resource devirtualization makes this integration easy, it sacrifices the advanced features of the Virtual Grid. For instance, DAGMan repeats the same computation in case of computation failures until it reaches to the maximum retries. However, repeating computation on the same resource is not likely to succeed if the failures do not result from transient errors. Unless Pegasus provides multiple plans for a workflow or a dynamic re-planning feature at failures, DAGMan cannot handle non-transient runtime failures. Restarting the failed job on different resources, on the other hand, is more likely to succeed. The Virtual Grid supports a variety of functionalities for fault-tolerance. First of all, VG can provision more reliable resources so it can proactively minimize the likelihood of failures. Moreover, the Virtual Grid can swap resources dynamically after resource failures and restart the failed tasks on the new resources.

Finally, the overall performance of applications is influenced by a variety of factors such as resource quality, resource reliability, data location, etc. We are also exploring how to improve the effective performance, which represents not only the performance of successful executions but also the penalty due to failures, against dynamic resource environments.

Acknowledgements

The authors and research described here were supported by the National Science Foundation under grants: OCI-0722019 (Pegasus) and supported in part by the National Science Foundation under NSF Cooperative Agreement NSF CCR-0331645 (VGrADS).

References

[1] B. Plale, D. Gannon, et al., "CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting," *IEEE Computer*, vol. 39, pp. 56–64, 2006.

[2] W. W. Li, R. W. Byrnes, et al., "The Encyclopedia of Life Project: Grid Software and Deployment," *New Generation Computing*, vol. 22, pp. 127-136, 2004.

[3] W. Chrabakh and R. Wolski, "GridSAT: A Chaff-based Distributed SAT Solver for the Grid," in *IEEE International Conference on High Performance Computing and Communication (SC'03)*: IEEE, 2003.

[4] S. J. Ludtke, P. R. Baldwin, and W. Chiu, "EMAN: Semiautomated Software for High-Resolution Single-Particle Reconstructions," *Journal of Structural Biology*, vol. 128, pp. 82-97, 1999.

[5] G. B. Berriman, E. Deelman, et al., "Montage: a Grid Enabled Engine for Delivering Custom Science-Grade Image Mosaics on Demand," in *SPIE Conference on Astronomical Telescopes and Instrumentation*, vol. 5493: SPIE, 2004.

[6] E. Deelman, G. Singh,et al. , "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming Journal*, vol. 13, pp. 219-237, 2005.

[7] T. Fahringer, A. Jugravu, et al., "ASKALON: a Tool Set for Cluster and Grid Computing," *Concurrency and Computation: Practice and Experience*, vol. 17, pp. 143-169, 2005.

[8] D. Churches, G. Gombas, et al., "Programming Scientific and Distributed Workflow with Triana Services," *Concurrency and Computation: Practice and Experience*, vol. 18, pp. 1021-1037, 2006.

[9] Y.-S. Kee and C. Kesselman, "Grid Resource Abstraction, Virtualization, and Provisioning for Time-targeted Applications," in *ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGRID'08)*: IEEE, 2008.

[10] D. Irwin, J. Chase, et al., "Sharing Networked Resources with Brokered Leases," in *USENIX Annual Technical Conference (USENIX)*: Usenix, 2006.

[11] "Amazon Elastic Compute Cloud."

[12] J. Frey, T. Tannenbaum, et al., "Condor-G: A Computation Management Agent for Multi-Institutional Grids," in *IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*: IEEE, 2001, pp. 55-63.

[13] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, vol. 11, pp. 115-128, 1997.

[14] Y. Gil, V. Ratnakar, et al., " Wings for Pegasus: Creating Large-Scale Scientific Applications Using Semantic Representations of Computational Workflows," in *The 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI)*. Vancouver, British Columbia, Canada, 2007.

[15] S. Fitzgerald, I. Foster, et al., "A directory service for configuring high-performance distributed computations," in *International Symposium on High Performance Distributed Computing (HPDC '97)*. IEEE Computer Society Press, 1997, pp. 365-375.

[16] A. L. Chervenak, N. Palavalli,et al., "Performance and Scalability of a Replica Location Service," in *Proceedings of the International IEEE Symposium on High Performance Distributed Computing (HPDC-13)*: IEEE, 2004.

[17] Y.-S. Kee, D. Logothetis,et al., "Efficient Resource Description and High Quality Selection for Virtual Grids," in *ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGRID'05)*. Cardiff, United Kingdom: IEEE, 2005, pp. 598-606.

[18] Y.-S. Kee, K. Yocum, et al., "Improving Grid Resource Allocation via Integrated Selection and Binding," in *ACM/IEEE International Conference on High Performance Computing and Communication (SC'06)*. Tampa, United States: IEEE, 2006.

[19] C. Liu and I. Foster, "A Constraint Language Approach to Matchmaking," in *IEEE International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04)*: IEEE, 2004, pp. 7-14.

[20] Y.-S. Kee, C. Kesselman, et al., "Enabling Personal Clusters on Demand for Batch Resources Using Commodity Software," in *International Heterogeneity Computing Workshop (HCW'08) in conjunction with IEEE IPDPS'08*, 2007.

[21] R. L. Henderson, "Job Scheduling Under the Portable Batch System," in *Lecture Notes in Computer Science*, vol. 949, *IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*: Springer, 1995, pp. 279-294.

[22] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," in *Lecture Notes in Computer Science*, vol. 3779, *IFIP International Conference on Network and Parallel Computing*: Springer, 2005, pp. 2-13.

[23] R. Huang, H. Casanova, and A. A. Chien, "Automatic Resource Specification Generation for Resource Selection," in *ACM/IEEE conference on Supercomputing (SC'07)*: IEEE, 2007.

[24] E.-K. Byun, Y.-S. Kee, et al., "Efficient Resource Capacity Estimate of Workflow Applications for Provisioning Resources," in *IEEE International Conference on e-Science (e-Science08)*. Indiana: IEEE, 2008.

[25] E. Deelman, S. Callaghan, et al., "Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance tracking: The CyberShake Example," in *IEEE International Conference on e-Science and Grid Computing (e-Science'06)*: IEEE, 2006.

[26] 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.

[27] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," in *IEEE International Conference on Distributed Computing Systems (ICDCS-8)*: IEEE, 1988, pp. 104-111.

[28] L. C. Wolf and R. Steinmetz, "Concepts for Resource Reservation in Advance," *Multimedia Tools and Applications*, vol. 4, pp. 255 - 278, 1997.

[29] G. Singh, C. Kesselman, and E. Deelman, "Adaptive Pricing for Resource Reservations," in *IEEE/ACM International Conference on Grid Computing (Grid 2007)*. Austin, Texas, 2007.

[30] G. Singh, C. Kesselman, and E. Deelman, "Performance Impact of Resource Provisioning on Workflows," University of Southern California, Technical Report CS05-850, 2005.

[31] R. Huang, H. Casanova, and A. A. Chien, "Using Virtual Grids to Simplify Application Scheduling," in *IEEE International Parallel & Distributed Processing Symposium (IPDPS'06)*: IEEE, 2006.

[32] Y. Zhang, A. Mandal, et al., "Scalable Grid Application Scheduling via Decoupled Resource Selection and Scheduling," in *ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*. Singapore: IEEE, 2006.