A Methodology and a Tool for Execution of Workflows in Grids

Jimmy Ortega, Sergio Naranjo and Emilio Hernández

Universidad Simón Bolívar, Departamento de Computación,

Apartado 89000, Caracas 1080-A, Venezuela

{jimmy.ortega,sergio.naranjo}@ldc.usb.ve, emilio@usb.ve

Abstract

This paper presents a working methodology and a complementary interactive tool, called JobDag, for execution of commands on files selected from a working area. Workflows based on DAGs (Directed Acyclic Graphs) can be specified using JobDag, for deferred execution on remote platforms such as clusters or grids. We have used the package SU (Seismic Unix) for testing JobDag, and we defined the object types this package accepts and the commands that apply on these objects, normally contained in files

Keywords: Grids, Workflows, Interfaces

1 INTRODUCTION

Interaction with remote computing platforms, for instance multiprocessors or computer clusters accessible via grids, can be done through a variety of tools. One of the most basic forms is submission of jobs to a remote queue using shell commands, which allows users to send jobs to a SLURM [14] or Torque [4] queue subsystem installed in a cluster, or to a queue implemented within a Grid environment such as gLite [9] or Globus Toolkit 4 [7]

In this article, we describe the combined use of a virtual machine platform, such as Xen [5] or VirtualBox [13], configured for easy connection to a grid and an interactive tool, called JobDag, installed in that virtual machine. This tool is oriented to the execution of workflows defined as a DAG (Directed Acyclic Graph) of commands, both for clusters and grids. Its main goal is to execute the DAG commands in a deferred way in order to enable the access to shared computing platforms which are typically used through job queues.

The execution of multiple commands under a precedence relationship scheme has been commonly used in other contexts, for example through the command “make”, available for all Unix-like computing environments. The sequence of commands under a workflow discipline usually follows a DAG pattern. There are many proposals and tools about this kind of workflows, such as tools for defining the workflows [10, 11, 2, 1], languages for specification of workflows [6, 12] and execution taxonomies under this scheme [15, 8], among many specific techniques that are very useful for grid environments in particular.

Most workflow specification tools allow users to build a graph in which the nodes represent the commands to be executed. In the case of JobDag, the nodes are the objects to be manipulated and the edges represent the actions or commands to be executed. In other words, we work with the dual graph. The objects are contained in files. In the current JobDag version, the object type is identified by the file extension, even though we plan to identify the objects from the file contents and structure in the future.

This approach for command DAG specification allows users to interact with a more familiar interface, because they can use an interface similar to that of file explorer tools, such as Konqueror or Galeon. In these interfaces, commands can be defined on the objects with an action similar to the standard “Open with...”, normally activated by a right click with the mouse. Additionally, JobDag allows users to define different execution environments, such as different kinds of clusters or grids, which may differ not only in the submission interfaces but also in the kind of software support that may be installed in each platform. These environments are handled through execution plugins that can be installed in JobDag. Another important feature is that the commands available in a particular computing platform can be easily added to the associated JobDag execution environment.

The rest of this article is organized as follows. Section 2 describes the methodology and the environment in which JobDag can be used. Section 3 describes the overall design of JobDag. Section 4 contains a more detailed description of the JobDag components. Section 5 shows the results we have obtained so far, as well as some use examples. Finally, Section 6 contains our conclusions and some suggestions as future work.

2 INTERACTION METHODOLOGY WITH JOBDAG

Just like many other similar tools, JobDag was designed for a specific execution platform. This platform consists of a virtual machine such as Xen or VirtualBox connected to a computing platform such as a cluster, a gLite grid or a GT4 grid, in such a way that job submission commands can be executed locally in the virtual machine. This can be accomplished by connecting the virtual machine to the grid through a secure tunnel in such a way that the virtual machine becomes a trusted User Interface of the grid.

In this context, the application interfaces can be developed directly on the virtual machine, like any desktop application, for instance, using window interfaces such as Qt or GTK. This contrasts with the use of a Web Portal. The connection scheme is represented in Figure 1.

Figure 1: User Interface Proxy

Most applications designed for workflow specification must be executed in a grid work in interactive mode but the tasks defined in the workflow will be executed in a batch mode afterwards. In this sense, JobDag is not an exception, but the main difference with previous proposals is that JobDag interaction is very similar to that of file explorers such as konqueror (KDE) or galeon (GNOME). The similarity lies in the fact that the right button of the mouse is used for executing the action “Open with...” and only the commands present in the selected execution environment that can be applied on the specific object will appear as options.

Once a command is selected, one or more nodes are added to the DAG with their respective edges coming out from the previous node. The number of new nodes depends on the output specification of the selected command. These new nodes represent “future objects”, because they will come into existence only after the DAG has been submitted and executed in the computing platform.

The DAG is built with nodes representing objects not yet actually generated, and the actions are represented by the edges of the DAG. Figure 2 shows how a JobDag DAG looks like while it is being built. Future objects appear in a transparent form, which indicates that the DAG has not been executed.

Figure 2: Interaction with JobDAG

The construction of the DAG starts by importing an object into the JobDag workspace, for instance, from the local file system. The initial objects can be also generated by an application that creates it.

3 DESIGN AND IMPLEMENTATION OF JOB DAG

The design of JobDag is oriented to the possibility of using different computing platforms, in particular grid platforms. Its main module executes the interaction with users and generates the data structures that represent the DAG specification, including descriptions of the execution environments, descriptions of the functions that can be invoked and, of course, the generated DAGs. These data structures are generated as XML files.

The modules that interpret the DAG and execute the functions associated with its edges are independent of the main module; in fact they are implemented as plugins of the run time module (see Figure 3). These plugins must read and parse the XML files generated by the main module, generate the execution scripts (for instance JDL specifications for Globus-based grids) and invoke the execution in the computing platform, for instance, by executing the *globus-job-submit* command.

The main features of JobDag are:

1. The number of supported computing platforms can be increased by adding appropriate plugins, which are independent of the main module, which only executes the user interaction and generates the DAG.
2. The functions or commands that can be invoked in a computing platform can be loaded in the main module as function lists. These lists define an Execution Environment associated with the computing platform. Users can specify default functions that apply to a specific file type, currently determined by the file name extension. The number and type of the parameters associated to each function can be also defined by the user.
3. User interface is similar to that used in file explorers such as konqueror (KDE) or galeon (GNOME).
4. All file structures follow the XML standard and no database is used, which makes the tool independent of specific database tools.

4 OTHER JOB DAG FEATURES

Some additional features make JobDag an easy-to-use tool. Some of them are mentioned in the following sections:

Figure 3: Design of JobDag

4.1 Templates

In JobDag a generated DAG can be saved as a template in such a way that it can be applied to a different data set. The template is saved as a DAG (its XML representation, actually) but the file names are substituted by variable specifications. The intermediate files, temporarily generated during the DAG interpretation and execution, have names with a prefix specified by the user, which should be different for each data set if the executions will take place in the same execution submission.

4.2 Execution Environments

Execution Environments represent the computing platforms as a number of *function lists*. When JobDag is used for generating a DAG and executing it on a cluster, for example, an Execution Environment associated to this cluster must be selected. An Execution Environment is associated to these components:

1. An execution plugin, which parses the DAG and generates the execution script
2. An output gathering plugin
3. A set of function lists
4. Optionally, visualization plugins

These components are explained below.

4.3 Plugins associated to the Execution Environments

The execution plugins are programs that must be developed independently and added to JobDag for them to be included as additional options for the DAG generation. These plugins are closely related to the way jobs are sent to the job queues in a computing platform, as well as the commands available in such platform. A test plugging for local execution is initially included.

Once the DAG has been built, users can submit it to the computing platform, and at that moment the execution plugin is invoked. The execution plugin can show an interaction window for modifying additional execution parameters, if any. The execution information is mainly extracted from the XML specification of the DAG. The execution plugin makes this extraction, builds the necessary scripts for the execution and finally submits the scripts to the computing platform, saving the necessary information for later output recovery.

An output recovery plugin is used for bringing back the output files, both final and temporal if required. This plugin transfers the files from the computing platform to the local environment. When the files are brought to the local environment, the icons representing the DAG objects change from transparent to solid, meaning the execution is finished.

Visualization plugins can be executed once the output files have been transferred to the local environment, and they can be applied to particular file types, as defined by users during the Execution Environment specification. The visualization programs must be installed locally.

4.4 Function Lists

The Execution Environment specification includes the definition of function lists, which are commands that can be executed in the computing platform associated to the Execution Environment. The commands can be introduced by users, but they must be sure that these commands are actually installed in the computing platform. A function list can be, for example, the commands that belong to a particular numerical package. The commands that will appear with the action “Open with...” during a DAG specification, may vary from one Execution Platform to another, depending on the function lists associated to each Execution Environment.

Function lists are also stored as XML files and contain all the information needed for the specification of each command, both for the execution of the command and for the visualization of the command when selected during the interactive DAG specification.

5 RESULTS

We have made preliminary tests with JobDag, submitting jobs to a remote cluster of 48 processors, AMD 2.4 GHz interconnected with Myrinet 10Gbps full duplex. The tests were performed from the local network to which the cluster is attached, which means that the delay time is mainly determined by the queuing time. Several tests with command packages such as SU (Seismic Unix [3]), a set of commands for seismic processing, were made.

Figure 4: Example: DAG specification

As a simple example of the type of execution DAGs used in these tests, Figure 4 shows the specification of a small DAG whose final result is a couple of images. One of these images is shown in Figure 5. This image is generated by a command called *supswigb*. The image is visualized from JobDag using the mouse, once the output file has been transferred from the computing platform (after the DAG execution).

Figure 5: Example: execution and one of the output files

We are not providing execution times of JobDag itself, because they are irrelevant when compared to the queuing time and the execution time of the tasks on the computing platform, once the processors are assigned to the DAG execution. The main advantage of JobDag is the way the workflows are specified, which resembles the way a user interacts with a file explorer in a window.

Currently we have a functional version of JobDag, with plugins for local execution and submission to a gLite type grid, implemented on top of an API provided by the gLite package. We are currently developing a JobDag extension that implements the execution of DAGs under a *parameter sweeping* scheme. This makes JobDag useful for performing computational experiments in which many executions of the same model can be done with a single specification and submission.

6 CONCLUSIONS AND FUTURE WORK

The use of high performance computing platforms, such as computing clusters and grids, requires interfaces that make program execution easier, especially for users that are not computer experts. The combination of (1) a virtual machine that momentarily attaches itself to a grid and (2) applications such as JobDag, which allow for specification and later execution of scripts, makes the interaction with a grid easier, when compared with other tools such as Web Portals.

In particular, JobDag provides the functionality for specifying the different computing platforms as Execution Environments in which a set of commands must be executed. The job is modeled as a DAG of commands or tasks which are submitted to a computing platform. One of the main advantages of JobDag is that interfaces are easy to use because they are familiar to most users, who frequently use file explorers. JobDag allows users to click with the right button on an icon representing an object; in that moment, the message “Open with...” appears and users can select a particular command. Once the DAG has been built, the execution can be invoked in a deferred way.

We are currently working on the integration of this workflow scheme with a parameter sweeping scheme, which will add more functionalities to JobDag allowing users to define a broader range of computational experiments.

REFERENCES

- [1] K. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. Gridant: a client-controllable grid workflow system. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference*, pages 10+, 2004.
- [2] Junwei Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. Gridflow: workflow management for grid computing. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 198–205, 2003.
- [3] Center for Wave Phenomena. CWP/SU: Seismic Unix, 2007. <http://www.cwp.mines.edu/cwpcodes/index.html>.
- [4] Cluster resources inc. TORQUE Resource Manager, 2007. <http://www.clusterresources.com/>.
- [5] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003.
- [6] T. Fahringer, J. Qin, and S. Hainzer. Specification of grid workflow applications with agwl: an abstract grid workflow language. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 676–685 2, 2005.
- [7] Ian T. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computer Science and Technology*, 21(4):513–520, 2006.
- [8] Geoffrey Charles Fox and Dennis Gannon. Special issue: Workflow in grid systems. *Concurrency and Computation: Practice and Experience*, 18(10):1009–1019, 2006.
- [9] gLite. Lightweight Middleware for Grid Computing, 2006. <http://glite.web.cern.ch/glite/>.
- [10] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Res*, 34(Web Server issue), July 2006.
- [11] Soonwook Hwang and C. Kesselman. Grid workflow: a flexible failure handling framework for the grid. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 126–137, 2003.
- [12] The Condor Team. DAGMan (Directed Acyclic Graph Manager), 2007. <http://www.cs.wisc.edu/condor/dagman/>.
- [13] Virtual Box Team. Virtual Box. <http://www.virtualbox.org/>.
- [14] A. Yoo, M. Jette, and M. M. Grondona. Slurm: Simple linux utility for resource management. *Lecture Notes in Computer Science*, 2862:44–60, 2003.
- [15] Jia Yu and Rajkumar Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200, September 2005.