Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 108C (2017) 705–714

Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

Facilitating the Reproducibility of Scientific Workflows with Execution Environment Specifications

Haiyan Meng and Douglas Thain

Department of Computer Science and Engineering, University of Notre Dame
Notre Dame, Indiana, USA
`hmeng@nd.edu and dthain@nd.edu`

Abstract

Scientific workflows are designed to solve complex scientific problems and accelerate scientific progress. Ideally, scientific workflows should improve the reproducibility of scientific applications by making it easier to share and reuse workflows between scientists. However, scientists often find it difficult to reuse others' workflows, which is known as *workflow decay*. In this paper, we explore the challenges in reproducing scientific workflows, and propose a framework for facilitating the reproducibility of scientific workflows at the task level by giving scientists complete control over the execution environments of the tasks in their workflows and integrating execution environment specifications into scientific workflow systems. Our framework allows dependencies to be archived in basic units of OS image, software and data instead of gigantic all-in-one images. We implement a prototype of our framework by integrating *Umbrella*, an execution environment creator, into *Makeflow*, a scientific workflow system.

To evaluate our framework, we use it to run two bioinformatics scientific workflows, *BLAST* and *BWA*. The execution environment of the tasks in each workflow is specified as an Umbrella specification file, and sent to execution nodes where *Umbrella* is used to create the specified environment for running the tasks. For each workflow we evaluate the size of the Umbrella specification file, the time and space overheads of creating execution environments using *Umbrella*, and the heterogeneity of execution nodes contributing to each workflow. The evaluation results show that our framework improves the utilization of heterogeneous computing resources, and improves the portability and reproducibility of scientific workflows.

Keywords: reproducible research, scientific workflows, execution environment specifications

1 Introduction

The reproducibility of scientific applications has become increasingly important for the progress of computational science because it allows the original author and others to reproduce, verify, and further extend the original applications [10]. Different solutions have been proposed to

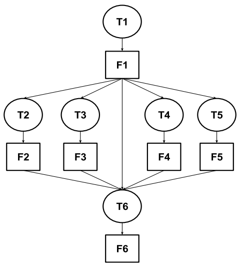
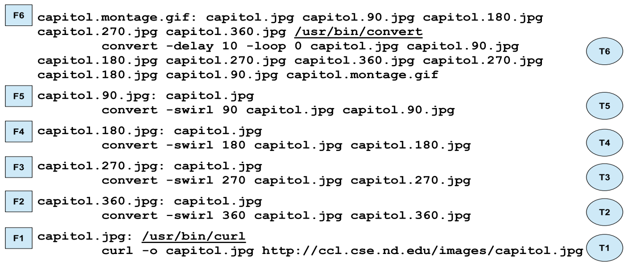


Figure 1: An Example Makeflow in DAG


F6
T6
F5
T5
F4
T4
F3
T3
F2
T2
F1
T1

```
capitol.montage.gif: capitol.jpg capitol.90.jpg capitol.180.jpg
capitol.270.jpg capitol.360.jpg /usr/bin/convert
        convert -delay 10 -loop 0 capitol.jpg capitol.90.jpg
capitol.180.jpg capitol.270.jpg capitol.360.jpg capitol.270.jpg
capitol.180.jpg capitol.90.jpg capitol.montage.gif

capitol.90.jpg: capitol.jpg
        convert -swirl 90 capitol.jpg capitol.90.jpg

capitol.180.jpg: capitol.jpg
        convert -swirl 180 capitol.jpg capitol.180.jpg

capitol.270.jpg: capitol.jpg
        convert -swirl 270 capitol.jpg capitol.270.jpg

capitol.360.jpg: capitol.jpg
        convert -swirl 360 capitol.jpg capitol.360.jpg

capitol.jpg: /usr/bin/curl
        curl -o capitol.jpg http://ccl.cse.nd.edu/images/capitol.jpg
```

Figure 2: An Example Makefile: Image Rotation

reproduce single-machine scientific applications. Some popular solutions include virtual machines [9], Linux Containers (e.g., Docker [14]), and user-space ptrace-based tools (e.g., CDE [7] and Parrot-packaging tool [13]). In spite of their differences, these solutions all emphasize the importance of preserving the complete software stack (i.e., execution environment) of scientific applications for conducting reproducible research [12].

However, many scientific applications are too big to be solved on a single machine, due to their huge computing and storage requirements. To solve this, *scientific workflows* [16] were designed to disseminate complex data transformations and analysis procedures into a set of smaller and possibly independent tasks, which allows computing resources from clusters, grids and clouds to be utilized. The tasks involved in a scientific workflow are often organized into a directed acyclic graph (DAG), where nodes represents tasks and files, and edges represent data flow and dependency relationship. Figure 1 shows a DAG including six tasks, which represents the simple workflow example in Figure 2, written in the *Makefile* language [1]. A real scientific workflow is usually more complex in both task number and task dependencies.

To make it easy for scientists to compose and execute scientific workflows, a variety of *scientific workflow systems* have been developed [18], such as Taverna [15], Pegasus [4] and Makeflow [1]. The end-users of these workflow systems only need to specify a DAG of tasks. The workflow systems respond to communicate with execution engines, schedule tasks to the underlying computing resources, manage data sets and deliver fault tolerance.

Ideally, scientific workflows should improve the reproducibility of scientific applications by making it easier to share and reuse workflows between scientists. However, scientists often find it difficult to reuse others' workflows, which is known as *workflow decay* [8]. For example, a study in 2012 of Taverna workflows on myExperiment [6], a social website allowing scientists to share their workflows, shows that 80% of the workflows on the site cannot be reproduced [19].

Among the causes of workflow decay, the incompatible execution environments on execution nodes is a recurring significant problem [3, 8, 5, 2]. This work aims to improve the reproducibility of scientific workflows by bringing the incompatible execution environments to a minimum.

Depending on the scientific workflow system used, scientists have different levels of control over the underlying execution environments on execution nodes. Pegasus [4] allows scientists to compose *abstract workflows* without worrying about the details of the underlying execution environments, which means sysadmins must respond to the cumbersome job of configuring computing resources to meet all the requirements of different workflows. Makeflow [1] allows executables to be specified in workflow specifications and delivered to execution nodes, such as `/usr/bin/convert` in Figure 2. This is simple but not always correct, because executables may be sent to execution nodes with incompatible execution environments. To fix this, Makeflow allows scientists to specify a Docker image [14] containing the required execution environment, and delivers the image to execution nodes [20]. This gives scientists more control over the execution environments, but ends up with gigantic images which are expensive to store.