Future Generation Computer Systems 54 (2016) 206–218

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Workflow performance improvement using model-based scheduling over multiple clusters and clouds☆

Ketan Maheshwari [a,*], Eun-Sung Jung [a], Jiayuan Meng [b], Vitali Morozov [b], Venkatram Vishwanath [b], Rajkumar Kettimuthu [a]

[a] Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 60439, USA
[b] Leadership Computing Facility Division, Argonne National Laboratory, Argonne, IL, 60439, USA

ARTICLE INFO

Article history:
Received 16 October 2014
Received in revised form
20 March 2015
Accepted 23 March 2015
Available online 31 March 2015

Keywords:
System modeling
Workflow
Optimization
Swift
Clouds

ABSTRACT

In recent years, a variety of computational sites and resources have emerged, and users often have access to multiple resources that are distributed. These sites are heterogeneous in nature and performance of different tasks in a workflow varies from one site to another. Additionally, users typically have a limited resource allocation at each site capped by administrative policies. In such cases, judicious scheduling strategy is required in order to map tasks in the workflow to resources so that the workload is balanced among sites and the overhead is minimized in data transfer. Most existing systems either run the entire workflow in a single site or use naïve approaches to distribute the tasks across sites or leave it to the user to optimize the allocation of tasks to distributed resources. This results in a significant loss in productivity. We propose a multi-site workflow scheduling technique that uses performance models to predict the execution time on resources and dynamic probes to identify the achievable network throughput between sites. We evaluate our approach using real world applications using the Swift parallel and distributed execution framework. We use two distinct computational environments-geographically distributed multiple clusters and multiple clouds. We show that our approach improves the resource utilization and reduces execution time when compared to the default schedule.

1. Introduction

A significant proliferation of the cloud computing paradigm is seen in recent years. Many independent organizations have started deploying their own clouds and are offering users to use these clouds to run their applications. Modern applications often involve repetitive communication-, data-, memory- or compute-intensive tasks. These applications are often programmed as workflows in order to improve productivity, and are deployed over remote computational sites such as clouds. Given the increasing prevalence of computation, these sites have been significantly grown in number and size and have diversified in terms of their underlying architecture. They vary widely in system characteristics including raw compute power, per-node memory, file system throughput, and performance of the network. With such heterogeneity, different tasks within the same workflow may perform better at different sites. Such observation is also true for multiple clouds. In the latter case, we now have systems where larger memory footprint nodes are interconnected with compute intensive nodes via a high-performance interconnect. However, the emerging clouds have altered the course of computational models in the recent years which must be taken into account to exploit them efficiently. In summary, the existing computational models are still not well-aligned with the cloud model of computation.

In addition to the resource heterogeneity, users confront logistical constraints in using these systems including allocation time and software compatibility. Users often subscribe to a multitude of clouds, spanning geographical regions, connected through various types of networks. It is often desired to deploy an application over multiple sites in order to best utilize the resources collectively.

The resource allocation at each site may be limited by the system administrators and the system configuration may be suited for

☆ The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). The US Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

* Corresponding author.
E-mail addresses: ketan@anl.gov (K. Maheshwari), esjung@mcs.anl.gov (E.-S. Jung), meng.jiayuan@gmail.com (J. Meng), morozov@anl.gov (V. Morozov), venkatv@mcs.anl.gov (V. Vishwanath), kettimut@mcs.anl.gov (R. Kettimuthu).

http://dx.doi.org/10.1016/j.future.2015.03.017
0167-739X/

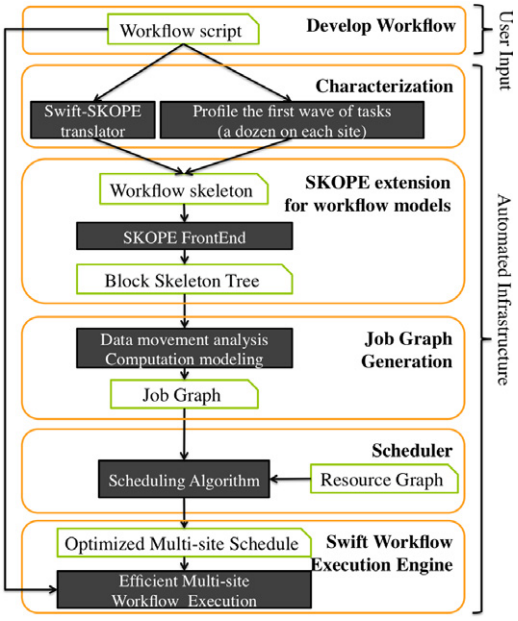

Fig. 1. A Conceptual framework for multi-site workflow scheduling.

some tasks but not others. Such resource-task affinity constraints must be taken into account while scheduling these workflows. Given these constraints and the dynamic nature of the network connecting these sites, it is non-trivial to compute a schedule that will optimally utilize the resources across sites to achieve the best time-to-solution. An ideal scenario from a user's perspective is:

1. Construct a workflow [1].
2. Provide the list of resources.
3. Execute the workflow by spreading the tasks to distributed resources (provided by the user in the previous step) without any intervention from the user.

This work tackles the aforementioned step 3. In particular, our work addresses the following key challenge:

> *Efficiently schedule and run data and compute-intensive workflows over multiple, heterogeneous, and geographically dispersed computational sites.*

We use the Swift framework [2] for workflow execution, SKOPE framework [3] for workflow performance modeling, and a network scheduling algorithm for optimizing mapping between tasks and resources. A scheme of our framework with steps and their interconnections is shown in Fig. 1. The framework takes the workflow description encoded as a Swift script and profiles different tasks in the workflow on available resources to generate a workflow skeleton in the format required by SKOPE (see Section 3 for details). Using the workflow skeleton, SKOPE builds analytical models about the data transfers between tasks, and empirical models about performance scalability of tasks. SKOPE then constructs a job graph describing the estimated computation and data transfer according to the models. The job graph is used as inputs to the scheduling algorithm, which generates an optimized schedule by taking into account the performance scalability of tasks and network condition between the relevant sites. Eventually, the Swift framework executes the workflow using the recommended schedule.

Although considerable work has been done in the past on scientific workflow management systems [4–8] and metascheduling systems [9–11], optimizing the execution of workflows across heterogeneous resources at multiple geographically distributed sites has not received much attention. This is due in part to the lack of access to multiple independent resources and in part to the lack of workflow enactment capabilities. Most metascheduling systems run the entire application or workflow at a single site. Systems such as Swift enables the execution of various tasks of a workflow at different sites but they do not have sophisticated scheduling algorithms to optimize the execution of workflow across different sites. A good scheduling algorithm must take into account not only the heterogeneous nature of the compute infrastructure at various sites but also the network connectivity and load between the computational sites and the data source(s) and sink(s). Our goal is to develop better schedules for workflows across geographically distributed resources.

Our specific contributions in this work is as follows:

- Development of the notion of *workflow skeletons framework to capture, explore, analyze and model empirical workflow behavior* with regard to dynamics of computation and data movement.
- An algorithm to *construct an optimized schedule*, according to the modeled workflow behavior.
- *Integration of the workflow skeleton and the scheduling algorithm into a deployment system.*
- Demonstration of the effectiveness of our approach over two distinct distributed environments: a collection of traditional clusters and multiple clouds. We use the *Amazon AWS, the Google Compute Engine and the Microsoft Azure cloud platforms* in this work.

The remainder of the paper is organized as follows. Section 2 presents an overview of Swift, a workflow expression and execution framework; typical scheduling mechanisms; and SKOPE, a workload modeling framework. Section 3 presents our optimized scheduling technique. Section 4 describes our experimental setup. Section 5 presents an evaluation of the proposed approach using real scientific workflows over multiple sites with distinct characteristics. Section 6 discusses related works. Conclusions are given in Section 7.

2. Background

We introduce parallel workflow scripting, resource scheduling, and workload behavior modeling techniques in this section, which forms the basis of our work. The following terminology is used. A *workflow* is a process that involves the execution of many programs. The invocation of an individual program is referred to as a *task*. These tasks may be dependent on each other or can run in parallel. Tasks are dispatched to various sites in groups of scheduling units, or *jobs*. Jobs define the granularity in which the schedule maps tasks to resources. A job consists of one or multiple tasks that correspond to the same program but different input data.

2.1. Swift: parallel workflow scripting

Swift is a parallel workflow scripting language for execution of ordinary programs [12]. The Swift runtime contains a powerful platform for running user programs on a broad range of computational infrastructures, such as clouds, grids, clusters, and supercomputers out of the box. A user can define an array of files, and use `foreach` loops to create a large number of implicitly parallel tasks. Swift will then analyze the script and execute tasks based on their dataflow; a task is executed only after any of its dependent tasks finish.

Applications encoded as Swift scripts have been shown to execute on multiple computational sites (clusters, clouds, supercomputers) via Swift coasters [13,12,14] mechanism which implements the pilot jobs paradigm. The pilot job paradigm dispatches a pilot task to each of the sites and measures the task completion rate. The task completion rate for the corresponding task–site combination then serves as an indicator to increase or decrease the