

Model-Driven Multisite Workflow Scheduling Based on Task-Resource Adaptation

Ketan Maheshwari*, Eun-Sung Jung*, Jiayuan Meng*, Venkatram Vishwanath*, Rajkumar Kettimuthu*

*MCS Division, Argonne National Laboratory
Argonne, IL 60439

Abstract—Workflows continue to play an important role in expressing and deploying scientific applications. In recent years, the number of scientific applications adoption to high-end computing has increased significantly. Moreover, a wide variety of computational sites have emerged with shared access to users such that a user often has access to multiple sites with a limited resource allocation at each site. Because of the scarcity and sparsity in the allocated resources, the user may not be able to complete an entire workflow at a single site. It is thus beneficial to run different tasks of a workflow on different sites. For such cases, judicious scheduling strategy is required in order to map tasks in the workflow to resources at multiple sites so that the workload is balanced among sites and the overhead is minimized in data transfer. The key challenges are that the execution time of a task varies across different sites and the data transfer rate varies based on the network capacity and load. In this paper, we propose a multi-site workflow scheduling technique that tackles the multi-site task distribution challenge by using data movement performance modeling. We applied this technique to schedule an earth observation science workflow over three sites. This approach, executed via the Swift parallel scripting paradigm, augments its default schedule and improves the time-to-schedule by up to 52%.

I. INTRODUCTION

Large-scale applications often involve repetitive data- and compute-intensive experiments. These applications are often encoded as workflows and deployed on remote execution sites. The workflow engine must then schedule tasks over available system resources and manage data movement among the tasks. In recent years, execution sites have been significantly expanded, diversified and specialized. They vary widely in system characteristics including computation power, memory bandwidth, file system throughput, and performance of the networks. With such heterogeneity among sites, different tasks within the same workflow may perform better at different sites.

In addition to the issue of resource heterogeneity, users confront logistical constraints in using these systems including allocation time and software compatibility.

These users often subscribe to a multitude of heterogeneous sites, spanning geographical regions, connected through various types of networks. Unfortunately, the resource allocations for each site may be limited. It is often desired, therefore, to deploy a user application over multiple sites in order to best utilize the resources collectively.

This deployment requires a judicious schedule that efficiently maps individual tasks in the workflow to resources at multiple sites, balancing the workload among sites and minimizing the overhead in data transfer. The key challenge is

that the data transfer overhead varies along the communication pattern and network utilization resulted from a schedule. In other words, the time-to-solution of a workflow schedule largely depends on how well tasks in a workflow *adapt* to the designated system resources. The above dynamics resulted by task-resource adaptation makes it difficult to identify an optimal schedule. To address this issue, one needs to study two factors: (a) how computation and data movement may change given a schedule, and (b) how such change would affect the workflow's overall time-to-solution. Such knowledge, however, often remains unknown until the workflow is executed and profiled for a given schedule, making it almost impossible to explore and optimize a large number of scheduling possibilities.

In this paper, we propose a multi-site workflow scheduling technique that tackles these challenges by using data movement performance modeling. We take a user-provided high-level, empirical performance description of the workflow. The description is then used to construct a performance model of the workflow, which projects a given schedule's performance behavior with respect to the dynamics of computation and data movement. Our proposed scheme explore potential schedules and suggest the best performing schedule according to the projected workload behavior. The proposed schedule then is used to deploy the workflow. Our specific contributions are threefold:

- Development of the notion of workflow skeletons to capture, explore, and analyze workflow behavior with regard to dynamics of computation and data movement.
- Formulation of an algorithm to explore and propose an optimized schedule, according to the modeled workflow behavior.
- Integration of the workflow skeleton and the scheduling algorithm into a workflow deployment system.

We demonstrate an implementation of our system using a mock but realistic scientific application. The application is coded as Swift [1] scripts. Experiments are run over multiple sites including large-scale XSEDE [2] infrastructures and relatively medium-scale institutional resources. We show that the proposed workflow schedule using our technique augments Swift's default schedule and saves up to 52% in time-to-solution.

The remainder of the paper is organized as follows. Section II presents an overview of Swift, a workflow scripting language, typical scheduling mechanisms and SKOPE, a

workload modeling framework. Section III presents our optimized scheduling technique based on task-resource adaptation according to workflow skeletons. Section IV describes our experimental setup. Section IV-B evaluates the quality of the proposed schedule by using a real scientific workflow over multiple sites with distinct characteristics. Section VI discusses related work. Conclusions are drawn in Section VII.

II. BACKGROUND

In this section we introduce workflow scripting, resource scheduling, and workload behavior modeling techniques on which our work is based.

A. Swift: Parallel and Distributed Execution System

Swift [1] is an application-level scripting framework designed for composing ordinary programs into parallel applications. Applications encoded in Swift have been shown to execute on multiple computational sites via Swift coasters [3]–[5] mechanism that implements the pilot jobs paradigm. Swift provides a simple reactive resource scheduling wherein, based on an initial “wave” of jobs, it records the per site job completion rate and adjusts the proportionate number of jobs to be sent to these sites. While Swift takes job completion rate per site into account to adjust the score, additional external stimuli can be supplied to Swift based on other factors. A new score can be derived from these factors per site and supplied to Swift such that a favorable schedule emerges. Executing applications with an augmented schedule via Swift is one of goals of this paper. Even though Swift can use GridFTP [6], [7] for high-speed data movement, it does not take data transfer time into account while picking the sites for executing the tasks. Our approach takes data transfer time into account while picking the sites for the tasks in a workflow.

B. Resource Scheduling

The resource and job scheduling problem is a classic NP-hard problem. It can be formally stated by resource and job definitions, and the algorithms vary depending on characteristics of resources and jobs. For example, job shop scheduling [8] is for multiple independent jobs with varying sizes and multiple homogeneous resources/machines. On the other hand, in the context of distributed computing, jobs may have dependencies among them and take input data from remote sites and send output to multiple remote sites. The sites themselves may also have a broad spectrum of computer architectures and capacities. In scheduling to take into account all these factors, sophisticated algorithms are needed.

C. SKOPE: A Workload Behavior Modeling Framework

SKOPE (SKeleton framewOrk for Performance Exploration) is a framework that helps users describe, model, and explore a workload’s current and potential behavior [9]. It asks the user to provide a description, called *code skeleton*, that identifies a workload’s performance behaviors including data flow, control flow, and computation intensity. These behavioral properties are intrinsic to the application and is agnostic of any system hardware. They are interdependent; the control flow may determine data access patterns, and data values may affect

control flow. Given different input data, they may result in diverse performance outcomes. They also reveal transformation opportunities and help users understand how workloads may interact with and adapt to emerging hardware. According to the semantics and the structures in the code skeleton, the SKOPE back-end explores various transformations, synthesizes performance characteristics of each transformation, and evaluates the transformation with various types of hardware models.

The workloads targeted by SKOPE include computational kernels and MPI applications. In this paper, we adopt and extend SKOPE to model workflows that often have multiple computational tasks with file transfers among them. The SKOPE front-end is extended with syntax and semantics to describe files and computational tasks. The resulting code skeleton is called the *workflow skeleton*. We further add a back-end procedure that constructs task graphs from workload skeletons.

III. SKELETON-BASED MULTISITE WORKFLOW SCHEDULING

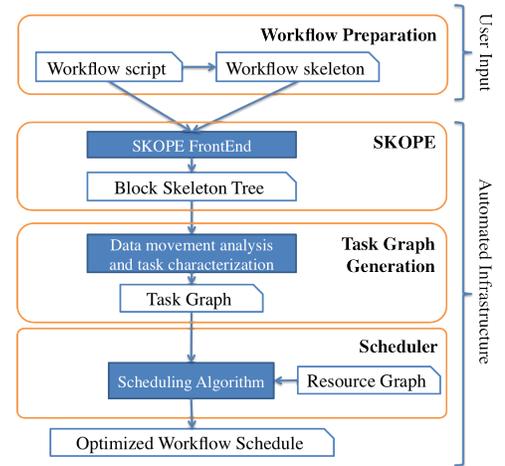


Fig. 1. Skeleton-based workflow scheduling framework

Figure 1 illustrates the overall steps involved in our technique. In the beginning, the user provides a workflow script in Swift, in which a workflow is represented as a sequence of applications, each of which consumes or produces a number of files. In addition to the workflow description, the user needs to provide a workflow skeleton using our extended SKOPE to describe the overall performance properties of the workflow.

The workflow skeleton, or skeleton in short, essentially models the workflow’s behavior, including the application’s computation resource requirements, as well as characteristics of the input and output files. To execute the workflow on multiple sites, the user selects the resources available and submits both the workflow script and the skeleton to the scheduling system. The skeleton then automatically generates a *task graph*, where a *task* refers to one or more applications grouped as a scheduling unit. The task graph depicts the tasks’ site-specific resource requirements as well as the data flow among them. Such a task graph is then used as input to a scheduling algorithm, which also takes into account the resource graph that describes the underlying hardware at

TABLE I. EXECUTION SITES AND THEIR CHARACTERISTICS

Site	CPU Cores	CPU Speed	Usable Memory per Node	Allocation	Remarks
LCRC Blues	310X16=4960	2.60GHz	62.90 GB	unlimited	Early access, 35 jobs cap
XSEDE Stampede	6400X16=102400	2.70GHz	31.31 GB	limited	50 jobs cap
RCC Midway	160X16=2560	2.60GHz	32.00 GB	limited	Institute-wide access

multiple sites and the network connecting them. The output of the algorithm is an optimized mapping between the task graph and the resource graph, which the scheduler then uses to dispatch the tasks.

In this section, we first introduce workflow skeletons and how they generate the task graph. We then describe our multi-site scheduling algorithm.

TABLE II. SYNTAX FOR WORKFLOW SKELETONS

Macros and Data Declarations	
File type and size (in KB)	:MyFile N
Constant definition	:symbol = expr
Array of files	:type array[N][M]
Variable def/assign	var = expr
Variable range	var_name=begin:end(exclusive):stride
Control Flow Statements	
Sequential <code>for</code> loop	for var_range {list_of_statements}
Parallel <code>for</code> loop	forall list_of_var_ranges {list_of_statements}
Branches	if(conditional probability){list_of_statements} else {list_of_statements}
Data Flow Statements	
file input/load	ld array[$expr_i$][$expr_j$]
file output/store	st array[$expr_i$][$expr_j$]
Characteristic Statements	
Run time (in sec.)	comp T
Task description	
Application definition	def app(arg_list){list_of_statements}
Application invocation	call app(arg_list)

A. Workflow Skeleton

Given a workflow, its skeleton summarizes the high-level semantics that relate to its performance behavior. The syntax of a workflow skeleton is summarized in Table II. In Figure 2(a), we show the script for the workflow. Its skeleton is listed in Figure 2(b). The skeleton is structured identically to its original workflow script in terms of file types, application definitions, and the control and data flow among the applications. The size of each type of file are summarized in lines 3-4 of the skeleton.

To describe an application’s distinct behaviors over various systems, the user can represent the application’s skeleton as a `switch` statement where cases correspond to different hardware systems. In each case are statements describing the application’s empirically profiled performance characteristics gathered for that corresponding system and the relationship between the execution time and the input data. An example skeleton description of an application is demonstrated by lines 17-35 in Figure 2(b).

A skeleton is parsed by SKOPE into a data structure called the *block skeleton tree* (BST). Figure 2(c) shows the BST corresponding to the skeleton in Figure 2(b). Each node of the BST corresponds to a statement in the skeleton. Statements such as application definitions, loops, or branches may encapsulate other statements, which in turn become the children nodes. The loop boundaries and data access patterns can be determined later by propagating input values.

Given the high-level nature of workflows and the structural similarity between workflow scripts and skeletons, generating

the workflow skeleton can be straightforward and may be automated in the future by a source-to-source translator. The major effort in writing the workflow skeleton falls on profiling the application over available systems in order to obtain performance characteristics. Since a typical workflow is repeatedly executed, such performance information can be obtained from historical data, either by explicit user measurement or by implicit system profiling.

B. Task Graph

A task graph is a directed, acyclic graph describing the performance characteristics of each task and the data movement among them. Figure 3 illustrates the task graph generated from the workflow skeleton in Figure 2(b). In a task graph, nodes refer to tasks and edges refer to data movements. Note that the structure of the task graph is independent of the hardware resources. Moreover, a node is annotated with the amount of computation resources, or the execution time, needed by the corresponding task for each available system. An edge is annotated by the amount of data that is transferred from the source node to the sink node.

Note that a task is a scheduling unit that may refer to a group of application invocations. Since often only a handful of sites are available and a workflow may contain parallel `for` loops that can easily spawn hundreds or thousands of homogeneous tasks, it is necessary to group multiple application invocations into one scheduling unit in order to improve the efficiency of the scheduling algorithm. Grouping can be achieved by simply transforming nested parallel `for` loops in the workflow skeleton into a two level nested loop, where the inner loop defines a task with a number of application invocations, and the outer loop is sized according to the desired number of tasks, which is a predefined constant according to the available number of sites. In this work, we adopt the heuristic where iterations of a parallel `for` loop are grouped into a number of tasks no more than 10 times the number of sites. Such a granularity enables the scheduler to balance the workloads among sites, and at the same time does not lead to a significant overhead in probing a large number of possibilities.

C. Procedural Task Graph Generation

Generating a task graph from a workflow skeleton involves three major steps. First, we obtain the data footprint for each task. Second, we construct the data flow among dependent tasks. Third, we derive the symbolic expression to express the execution time of a task over different systems.

The key of our technique is data movement analysis, for which we apply array section analysis using bounded regular section (BRS) [10]. BRS has been conventionally used to study stencil computation’s data access patterns within loops. It is adopted in our study to analyze data access patterns over arrays of files. In BRS, an array access can be regarded as a function that maps a range of loop iterators to a set of elements over the

```

1. type file;
2. # Files
3. int N = 165;
4. file inputs[N]
5. file outputs[N]
6. param_gen(inputs)
7. foreach n in 0:N
8. {
9.   outputs[n] = process(inputs[n]);
10. }
11. collect(outputs)
12.
13. // function definitions are omitted

```

(a) Original workflow script

```

1. :site = 'stampede'
2. :N = 165
3. :InFile = 431 // file size in KB
4. :OutFile = 1600 // file size in KB
5. :InFile inputs[N]
6. :OutFile outputs[N]
7. def main()
8. {
9.   // produces input files
10.  call param_gen(inputs)
11.  forall n = 0:N
12.  { // execute parallel tasks
13.    call process(outputs[n], inputs[n])
14.  }
15.  call collect(outputs) // collect results
16. }

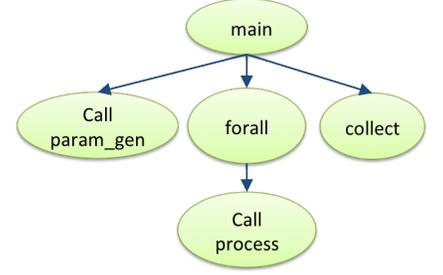
```

(b) Workflow skeleton

```

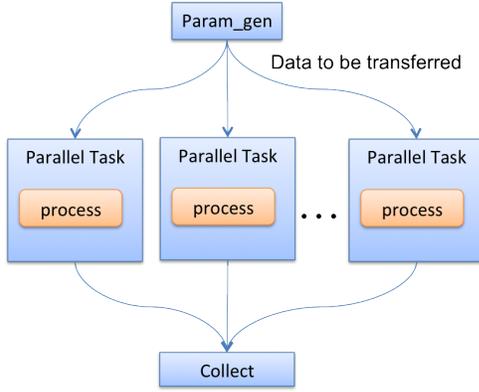
17. def process(output, input)
18. {
19.   // read from input file
20.   ld input
21.   // run time (sec)
22.   switch(site)
23.   {
24.     case 'stampede'
25.     {
26.       comp 9972
27.     }
28.     break
29.     case 'blues'
30.     {
31.       comp 12650
32.     }
33.   }
34.   // write to output file
35.   st output

```



(c) Block Skeleton Tree for the main skeleton function

Fig. 2. Workflow script (a), skeleton (b), and the corresponding block skeleton tree (c) for a pedagogical workflow.



(c) Task graph generated from the skeleton

Fig. 3. Task graph for the workflow shown in Figure 2(a).

array. In this paper, we refer to the range of loop iterators as a *tile* (\mathbb{T}) and the set of accessed array elements as a *pattern* (\mathbb{P}). For example, suppose A is a 2-D array of files and an application accesses $A[r][c]$ in a nested `for` loop with two iterators, r and c . The tile corresponding to the loop is denoted $\mathbb{T}(r, c) = \{r : \langle r^l : r^u : r^s \rangle; c : \langle c^l : c^u : c^s \rangle\}$, where each of the three components represents the lower bound, upper bound, and stride, respectively. The overall pattern accessed within this loop is denoted $A[\langle r^l : r^u : r^s \rangle][\langle c^l : c^u : c^s \rangle]$, which is summarized by $\mathbb{P}(A[r][c], \mathbb{T}(r, c))$. Patterns can be intersected or unioned.

To obtain the data footprint of a task, we identify its corresponding node in the BST and obtain the tile \mathbb{T} corresponding to one iteration of all loops in its ancestor nodes (i.e., the outer loops) and all iterations of its child nodes (i.e., the inner loops). Given an access to a file array, \mathbb{A} , we apply \mathbb{T} to obtain a pattern, $\mathbb{P}(\mathbb{A}, \mathbb{T})$, which symbolically depicts the data footprint of the task.

We then build the data flow among tasks. First, we scan all BST nodes that correspond to tasks. Pairs of nodes producing

and consuming the same array of files become candidate dependent tasks. Next, we perform intersection operations between produced and consumed patterns to determine the exact pattern that caused the dependency. The size of the dependent patterns is the amount of data movement associated with an edge in the task graph.

Then, we derive the execution time of each task for different systems. We simply traverse the BST of the code skeleton once for each site; in each traversal, the switch statement in each application is evaluated, and its execution time for that particular site is obtained. We then aggregate the per application execution time into the per task execution time by multiplying it with the number of applications within a task.

The resulting task graph is output in the form of an adjacency list. It is then passed to the scheduler algorithm to generate an optimized mapping among the tasks and resources.

D. Multisite Scheduling

We use a joint scheduling algorithm [11] that takes into account both compute resources and network paths. In distributed workflow scheduling, data movement among sites are not trivial, especially when the data size is big and network resources are not abundant. That means independent scheduling of compute resource and network paths may not give a near optimal schedule. Our scheduling algorithm [11] considers both resources holistically by converting a scheduling problem into a network flow problem. In order to schedule parallel jobs in a workflow while considering concurrently runnable jobs at a compute resource, a new notion of task-resource affinity has been devised by taking into account the number of concurrently runnable jobs at computation sites. Task clustering based on our algorithm is discussed later in this section.

In addition to the task graph of a workflow, our scheduling algorithm also needs a resource graph which describes the underlying hardware architecture to generate an optimized schedule. Figure 4 (a) illustrates an example of the resource graph. Nodes and edges denote compute resources and network paths among those resources. Even though a network path can

span multiple physical network links, we use only one logical link between two sites because we cannot setup paths at our discretion in these experiments. However, if we have control over network path setup in connection-oriented networks, a physical network topology can be used as a resource graph so that our algorithm finds appropriate network paths for data transfers.

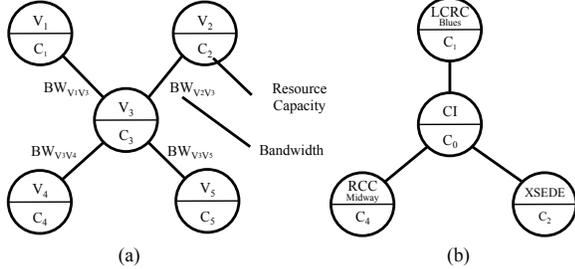


Fig. 4. (a) Resource graph model (b) Resource graph in our experiments.

Table III shows network bandwidth among our execution sites. The network bandwidths among them is measured by *iperf* benchmark tool. Figure 4 (b) is the resource graph corresponding to Table I. The bandwidth values in Table III are associated with edges in Figure 4.

TABLE III. DISK-TO-DISK BANDWIDTH BETWEEN SUBMIT AND EXECUTION SITES

Site	LCRC Blues	XSEDE Stampede	RCC Midway
Submit Host	896 Mbit/s	592.88 Mbit/s	430 Mbit/s

We next set the resource capacities, C_n , which represents computation power at site n , associated with nodes in Figure 4. d_i denotes the amount of compute resource that task i demands and d_i is associated with task i . So $\frac{d_i}{C_s}$ represents how fast a task demand, d_i , can be processed by compute resources at site s , C_s . C_s and d_i are relative values. To describe that task i takes 1 sec at compute resource site s , we can assign either 100 or 10 to both of C_s and d_i . We have execution time of a task i on site s , t_s^i through performance modeling. Equation 1 is task-resource affinity equation where CE_s is a random variable of the number of concurrently runnable tasks at site s . We assume that we do not have a fixed reservation at computation site and we know the probability of job execution from job queues. We define task-resource affinity as $\frac{t_s^i}{E(CE_s)}$. $\frac{t_s^i}{E(CE_s)}$ is the expected run time per task if multiple tasks are run at computation site s . For example, if 10 same parallel tasks are run at a site that can run 10 tasks at the same time, the expected run time per task is one tenth of the tasks's run time.

$$\frac{t_s^i}{E(CE_s)} = \frac{d_i}{C_s} \quad (1)$$

Equation 1 means task-resource affinity equals $\frac{d_i}{C_s}$, which is the runtime of task i at site s . We can thus set C_s for a computation resource site with fewest computation resource to 100. Then for each task, we can get d_i and assign this to the corresponding task in the workflow. To compute C_n , when

$n \neq s$, we can use Equation 2, where T is a set of tasks. Since C_n can be arbitrary values relative to d_i according to Equation 1, we should normalize C_n regarding the base case by Equation 2.

$$C_n = 100 \times \frac{1}{|T|} \sum_{i \in T} \frac{t_s^i}{t_n^i} \cdot \frac{E(CE_n)}{E(CE_s)} \quad (2)$$

Equation 2 averages affinities of tasks to resources. We can easily extend our model such that resource affinity per task is considered. For instance, while t^1 can be executed two times faster on site 1 than on site 2, t^2 may have similar execution times regardless of sites. We can define d_s^i representing the demand of task i at site s so that we can assign different demands of tasks per each resource to the edges of the auxiliary graph. [11]

Now we are ready to run the scheduling algorithms and getting the schedule from it. However, the task graph in Figure 2 (c) has 634 parallel tasks, which could result in much higher execution time of the scheduling algorithms. In this paper, we partition the parallel tasks into 10 groups with same number of tasks, 63–64, and use this reduced task graph for scheduling. If the resulting schedule maps different groups to the same resource we can group them into one group again. In this way, we can find the better partitions for parallel tasks in part if a large size of task graph is given. Previously, partitioning techniques for multiple same-level tasks, called *task clustering* [12], groups all the tasks into fixed number of partitions (e.g., 2 or 4) with the same number of tasks to reduce scheduling overhead or task queue wait time. With our algorithm or combined with other workflow scheduling heuristics, better task clustering will lead to improved makespans of workflows. For example, after HEFT schedules the whole workflow by grouping all same-level tasks into one logical task, our algorithm can do online task clustering/scheduling for proper partitions when the workflow management system is ready to dispatch the logical task.

IV. EXPERIMENTAL SETUP

In this section we introduce the application and computation sites used in our experiments.

A. Application Characteristics

We use a mock application—MODIS (*modis.gsfc.nasa.gov*), derived from NASA's MODIS (Moderate Resolution Imaging Spectroradiometer) instrument aboard the Terra and Aqua satellites. The workflow model for this application is depicted Figure 5 shows the workflow model with data and job numbers for each computational stage. The application involves reading satellite data, processing it for land-usage classification and colorization. The two dominating tasks are 'analyzelanduse' and 'colormodis'. The execution time of instances of each of these tasks are homogeneous, as shown in figure 6. For the current work, we use a set of 317 image tiles but the application could be scaled up with a larger number of image tiles and multiple operations on them.

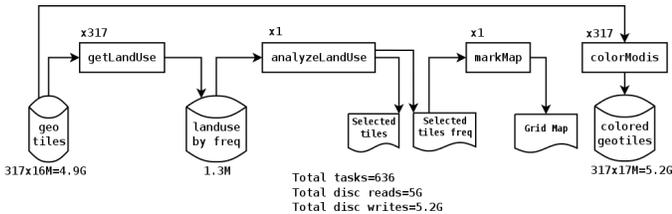


Fig. 5. MODIS application workflow depiction.

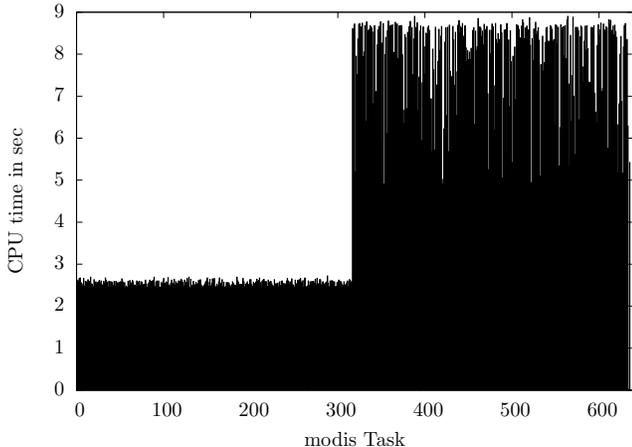


Fig. 6. MODIS application execution time distribution for different tasks. The two tasks dominating MODIS workflow are ‘getlanduse’ and ‘colormodis’. As seen in the plot, ‘getlanduse’ tasks are shorter with about 2.6 seconds whereas the ‘colormodis’ tasks are longer with 8.8 seconds execution time.

B. Computation Sites

We selected three execution sites—XSEDE Stampede, LCRC (Laboratory Computing Resource Center) Blues and RCC (Research Computing Center) Midway to demonstrate our approach. We identify their characteristics and utility. A summarized characterization of these sites is tabulated in Table I.

XSEDE (www.xsede.org) is an NSF-funded, national cyberinfrastructure comprising multiple large-scale computation systems on sites across the United States. Stampede is one of the supercomputing systems offered by XSEDE. Stampede runs the SLURM scheduler for submitting user jobs.

LCRC Blues (www.lcrc.anl.gov/about/blues) is a recently-acquired cluster available to science users at the Argonne National Laboratory. The Blues cluster is a new system not in production mode as of this writing; the system is available for early access. Blues runs the PBS scheduler.

RCC Midway (rcc.uchicago.edu) is the University of Chicago Research Computing Center cluster supporting University wide high-end computational needs. The cluster has multiple resource partitionings dedicated to specialized computing such as HPC, HTC and GPU computing and runs a SLURM batch queue scheduler.

V. EVALUATION

In this section we present an evaluation of our approach. We use the MODIS application workflow encoded in Swift for

these evaluations.

We chose 16 cpu-cores on each of the three clusters (Blues, Midway and Stampede) resulting in a capability of running 48 application jobs in parallel across these clusters. We use a remote machine for application submissions. Application input data is stored on the disk on this remote machine.

We ran the application on individual sites to identify the makespan time on each site. This involves the total amount of time spent in data movement, execution, site-scheduler overhead and Swift’s startup and shutdown overhead. Figure 7 shows the application makespan for each site. The least execution time on Blues can be attributed to a higher bandwidth from submit host and the lightly loaded queues (Blues is an early access system and is not opened for production use yet).

We then evaluate the workflow performance over multiple sites. In the first set of experiments, we use the default scheduler in Swift and merely tune a configuration parameter, “throttle”, which controls the number of parallel jobs to send to sites and hence the number of parallel data transfers to sites. The default scheduler distributes an equal number of jobs to each of the execution sites. It can be observed from figure 8 that higher “throttle” results in larger makespan. This is because higher “throttle” value results in more parallel jobs and thus more input data files in each batch sent to the execution sites. As the number of files increases, the transfer time increases. Also, since these files are transferred concurrently, it can cause network congestion and disk I/O contention resulting in increased transfer time. Makespan decreases as we tune the “throttle” to match the number of available cpu cores.

In the second set of experiments, we alter the Swift script and distribute the jobs according to a schedule proposed by our scheme. The first such schedule, shown by the bar labeled ‘sched1’ takes the data movement into account assigns 256, 124 and 256 jobs to Stampede, Midway and Blues respectively. The second proposed schedule takes the difference in job execution time of ‘landuse’ and ‘colormodis’ into account and assigns 124, 256 and 256 jobs to Stampede, Midway and Blues respectively. Based on the resource description, our scheme automatically picks the optimal “throttle” value.

It can be noted from the results in figure 8, we achieve a minimum makespan with an informed schedule and saving the effort of fine tuning with throttle changes. Our scheme achieves a 52% improvement in makespan over the default scheme (‘th:123’ in the Figure) and a 10% improvement over the best performance obtained with manual tuning (‘th:48’ in the Figure). In complex and large real workflows interfaced with multiple remote sites, such fine tuning might be expensive or impossible. This makes our scheme valuable by making informed decision on how the jobs will be distributed across execution sites and relieving the user from manual tuning.

VI. RELATED WORK

Large scale applications have been shown to benefit significantly on heterogeneous systems [13] for data-intensive science [14] and under multiple sites infrastructure [15], [16]. We demonstrate the value of these arguments in a realistic scenario. There has also been much prior work on workflow management and performance modeling, which we discuss below.

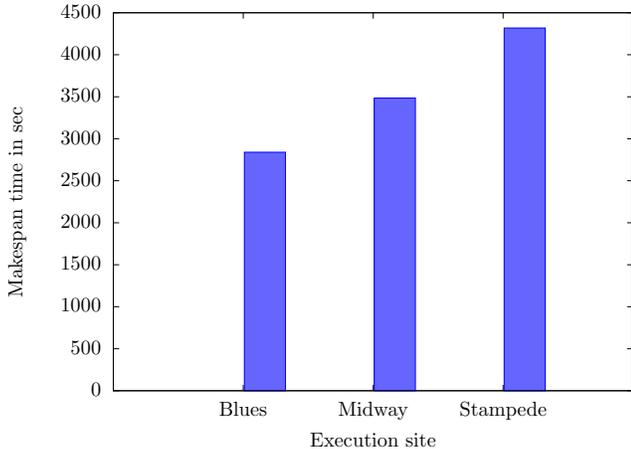


Fig. 7. Makespan time of application execution on individual resources

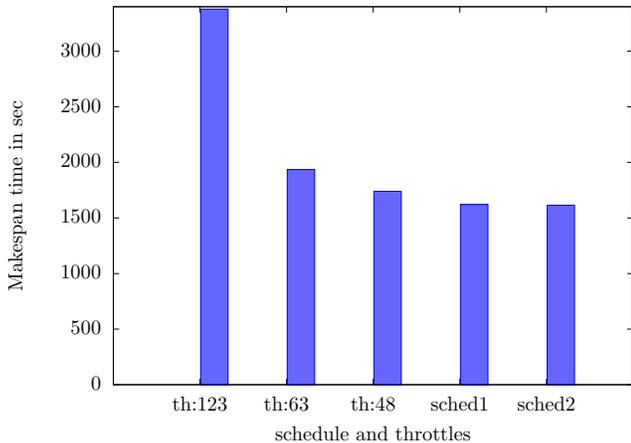


Fig. 8. Performance comparison of application execution on combined resources with default and heuristics based schedule

A. Workflow Management

Some of the well-known workflow management systems include Condor DAGMan [17], Pegasus [18], Taverna [19] and makeflow [20]. Condor DAGMan provides a minimal set of keywords for directed acyclic graph (DAG)-based workflows. The workflow model of Condor DAGMan does not require additional information other than task precedence requirements given by a DAG. Pegasus requires task execution time information related to each task in a workflow. Swift provides a rich set of keywords for parallel task execution. However, it does not utilize task execution time except job completion rate per site. These differences among workflow management systems result in different scheduling capabilities. Condor DAGMan is only capable of best-efforts batch-mode scheduling while Pegasus deploy heuristics such as heterogeneous-earliest-finish-time (HEFT), considering task execution time and/or data transfer times.

Our work differ from those previous work in two aspects. First, our scheduler takes into account the variance of task's execution time over different sites and data affinity among the tasks. HEFT or other heuristics use averaged execution times

of a task over every possible resources or try earliest/latest completion task first approach. These heuristics do not consider the resource affinity per task effectively, and performance would be worse combined with data transfer requirements. Second, instead of having users manually measure tasks' execution time and data transfer overhead for individual schedules, we model the relationships between a schedule and its resulting task execution time and data transfer overhead. As a result, we can project the overall time-to-solution without executing each possible schedule, which may not be feasible.

Simulation studies on multi-site resources have been done in the past such as Workflowsim [21] on generic wide-scale environments and more recently on European Grids [22]. While they provide detailed analysis of workflow deployment, simulations take a significant amount of time. Our work models the high level behavior of workflows so that the scheduler can suggest an optimized schedule online when deploying a workflow.

Overall, our approach based on workflow skeleton captures the application characteristics while offloading the execution responsibility to Swift which leads to a better division of responsibility. This approach makes our work distinct and a valuable contribution to e-science community.

B. Performance Modeling

Performance modeling has been widely used to analyze and optimize workload performance. Application or hardware specific models have been used in many scenarios to study workload performance and to guide application optimizations [23], [24], where applications are usually run at a small scale to obtain knowledge about the execution overhead and their performance scaling. Snaveley et al. developed a general modeling frameworks [25] that combine hardware signatures and application characteristics to determine the latency and overlapping of computation and data movement. An alternative approach uses black-box regression, where the workload is executed or simulated over systems with different settings, to establish connections between system parameters and run time performance [26]–[29]. All the above techniques target computational kernels and parallel applications.

SKOPE [9] provides a generic framework to model workload behavior. It has been used to explore code transformations when porting computational kernels to emerging parallel hardware [30], [31]. We apply the same principles in modeling kernels and parallel applications and extend SKOPE to model workflows. In particular, we propose workflow skeletons and use that to generate task graphs, which are in turn used to manage workflow.

VII. CONCLUSION

In this paper, we proposed a multi-site scheduling approach for scientific workflows using performance modeling. We introduced the notion of workflow skeletons and extended the SKOPE framework to capture, analyze and model the computational and data movement characteristics of workflows. We developed a resource and task aware scheduling algorithm that utilizes the task graph generated using the workflow skeleton and the resource graph generated using the resource

description. We incorporated our approach into Swift, a script-based workflow framework and showed that our approach can improve the total execution time of the workflows by as much as 52%.

ACKNOWLEDGMENT

We thank our colleague Mainak Mookherjee at Cornell for providing us with valuable estimates in a previous version of this paper. We thank our colleague David Kelly for technical help. We thank John Valdes from systems at Argonne for helping with computational resource access. We thank Gail Pieper of Argonne for proofreading help. This work was supported in part by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

REFERENCES

- [1] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37 (In Press, published online), pp. 633–652, 2011.
- [2] "Extreme science and engineering discovery environment (xsede)," <http://www.xsede.org/>.
- [3] M. Hategan, J. Wozniak, and K. Maheshwari, "Coasters: uniform resource provisioning and access for scientific computing on clouds and grids," in *Proc. Utility and Cloud Computing*, 2011.
- [4] K. Maheshwari, K. Birman, J. Wozniak, and D. V. Zandt, "Evaluating cloud computing techniques for smart power grid design using parallel scripting," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, 2013.
- [5] K. Maheshwari, A. Espinosa, D. S. Katz, M. Wilde, Z. Zhang, I. Foster, S. Callaghan, and P. Maechling, "Job and data clustering for aggregate use of multiple production cyberinfrastructures," in *Proceedings of the fifth international workshop on Data-Intensive Distributed Computing Date*, ser. DDC '12. New York, NY, USA: ACM, 2012, pp. 3–12. [Online]. Available: <http://doi.acm.org/10.1145/2286996.2287000>
- [6] B. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus striped GridFTP framework and server," in *SC'2005*, 2005.
- [7] R. Kettimuthu, L. Laciniski, M. Link, K. Pickett, S. Tuecke, and I. T. Foster, "Instant gridftp," in *IPDPS Workshops*. IEEE Computer Society, 2012, pp. 1104–1112.
- [8] R. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Tech. Journal*, vol. 45, pp. 1563–1581, 1966.
- [9] J. Meng, X. Wu, V. A. Morozov, V. Vishwanath, K. Kumaran, V. Taylor, and C.-W. Lee, "SKOPE: A Framework for Modeling and Exploring Workload Behavior," 2012.
- [10] P. Havlak and K. Kennedy, "An implementation of interprocedural bounded regular section analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 2, 1991.
- [11] E.-S. Jung, S. Ranka, and S. Sahni, "Workflow scheduling in e-science networks," in *Computers and Communications (ISCC), 2011 IEEE Symposium on*, 2011, pp. 432–437.
- [12] G. Singh, M.-H. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, and G. Mehta, "Workflow task clustering for best effort systems with pegasus," in *Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities*, ser. MG '08. New York, NY, USA: ACM, 2008, pp. 9:1–9:8. [Online]. Available: <http://doi.acm.org/10.1145/1341811.1341822>
- [13] R. Ramos-Pollan, F. Gonzalez, J. Caicedo, A. Cruz-Roa, J. Camargo, J. Vanegas, S. Perez, J. Bermeo, J. Ojalora, P. Roza, and J. Arevalo, "Bigs: A framework for large-scale image processing and analysis over distributed and heterogeneous computing resources," in *E-Science (e-Science), 2012 IEEE 8th International Conference on*, 2012, pp. 1–8.
- [14] F. De Carlo, X. Xiao, K. Fezzaa, S. Wang, N. Schwarz, C. Jacobsen, N. Chawla, and F. Fusses, "Data intensive science at synchrotron based 3d x-ray imaging facilities," in *E-Science (e-Science), 2012 IEEE 8th International Conference on*, 2012, pp. 1–3.
- [15] M. Silberstein, "Building an online domain-specific computing service over non-dedicated grid and cloud resources: The superlink-online experience," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, 2011, pp. 174–183.
- [16] H. Yang, Z. Luan, W. Li, and D. Qian, "Mapreduce workload modeling with statistical approach," *J. Grid Comput.*, vol. 10, no. 2, pp. 279–310, Jun. 2012. [Online]. Available: <http://dx.doi.org/10.1007/s10723-011-9201-4>
- [17] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger, "Workflow management in condor," in *Workflows for e-Science*, I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, Eds. London: Springer London, 2007, pp. 357–375. [Online]. Available: <http://www.springerlink.com/content/t6un6312103m47t5/>
- [18] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, "Task scheduling strategies for workflow-based applications in grids," in *IEEE International Symposium on Cluster Computing and the Grid, 2005. CCGrid 2005*, vol. 2, 2005, pp. 759–767 Vol. 2.
- [19] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, Nov. 2004. [Online]. Available: <http://dx.doi.org/10.1093/bioinformatics/bth361>
- [20] M. Albrecht, P. Donnelly, P. Bui, and D. Thain, "Makeflow: a portable abstraction for data intensive computing on clusters, clouds, and grids," in *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, ser. SWEET '12. New York, NY, USA: ACM, 2012, pp. 1:1–1:13. [Online]. Available: <http://doi.acm.org/10.1145/2443416.2443417>
- [21] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *E-Science (e-Science), 2012 IEEE 8th International Conference on*, 2012, pp. 1–8.
- [22] S. Pop, T. Glatard, and H. Benoit-Cattin, "Simulating application workflows and services deployed on the european grid infrastructure," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, 2013.
- [23] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on GPUs," in *PPoPP*, 2010.
- [24] H. Gahvari, A. H. Baker, M. Schulz, U. M. Yang, K. E. Jordan, and W. Gropp, "Modeling the performance of an algebraic multigrid cycle on HPC platforms," in *ICS*, 2011.
- [25] A. Snaveley, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for performance modeling and prediction," in *SC*, 2002.
- [26] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz, "A regression-based approach to scalability prediction," in *ICS*, 2008.
- [27] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, "Methods of inference and learning for performance modeling of parallel applications," in *PPoPP*, 2007.
- [28] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *ASPLOS-XII*, 2006.
- [29] V. Taylor, X. Wu, and R. Stevens, "Prophecy: an infrastructure for performance analysis and modeling of parallel and grid applications," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 4, pp. 13–18, Mar. 2003.
- [30] J. Meng, V. A. Morozov, K. Kumaran, V. Vishwanath, and T. D. Uram, "GROPHECY: GPU performance projection from CPU code skeletons," in *SC*, 2011.
- [31] J. Meng, V. A. Morozov, V. Vishwanath, and K. Kumaran, "Dataflow-driven GPU performance projection for multi-kernel transformations," in *SC*, 2012.

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract DE-AC02-06CH11357 with the U.S. Department of Energy. The U.S. 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.