

Improving Multisite Workflow Performance using Model-based Scheduling

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

*MCS Division, Argonne National Laboratory
Argonne, IL 60439

†LCF Division, Argonne National Laboratory
Argonne, IL 60439

Abstract—Workflows play an important role in expressing and executing scientific applications. In recent years, a variety of computational sites and resources have emerged, and users often have access to multiple resources that are geographically distributed. These computational 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. 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 naive 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 for a scientist. In this paper, we propose a multi-site workflow scheduling technique that uses performance models to predict the execution time on different resources and dynamic probes to identify the achievable network throughput between sites. We evaluate our approach using real world applications in a distributed environment using the Swift distributed execution framework and show that our approach improves the execution time by up to 60% compared to the default schedule.

I. INTRODUCTION

Large-scale scientific applications involve repetitive communication-, data-, memory- or compute-intensive tasks. These applications are often encoded as workflows in order to improve productivity of a scientist, and deployed on remote computational sites. The workflow engine must then schedule tasks over available resources and manage data movement among the tasks. In recent years, given the increasing prevalence of computational sciences, 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, accelerators, per-node memory, 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. Even in a single large supercomputer, we are witnessing this heterogeneity both at a node-level, as well as at a system level. In the latter case, we now have systems where larger memory footprint nodes are interconnected with compute intensive nodes via a high-performance interconnect.

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 sites, spanning geographical regions, connected through various types of networks. It is often desired, therefore, to deploy a user application over multiple sites in order to best utilize the resources collectively.

Unfortunately, the resource allocation in terms of time for each site may be limited and the system configuration at each site may be suited for some tasks but not others. These constraints must be taken into account while scheduling these workflows. Given these constraints in the environments and dynamic nature of the network connecting these sites, it is not trivial to compute a schedule that will optimally utilize the resources across the various sites to achieve the best time-to-solution for these workflows. An ideal scenario from a user's perspective is:

- 1) User constructs her workflow using a simple and intuitive interface such as Galaxy [1].
- 2) User provides the list of resources that the user has access to.
- 3) Workflow engine executes the workflow in an optimal fashion by spreading the tasks of the workflow into one or more distributed resources (provided by the user in the previous step) without any intervention from the user.

Our work here tackles #3 above and focuses specifically on generating an optimal schedule for a given workflow. We use the Swift framework [2] for distributed workflow execution, an enhanced SKOPE framework [3] for workflow performance modeling, and a network scheduling algorithm for optimizing mapping between tasks and resources. Our system takes the workflow description written in the form of 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 III for details). Using the workflow skeleton, SKOPE builds analytical models about the data transfers between tasks, and empirical models about performance scalability of tasks. It 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 over the last decade on scientific workflow management systems [4]–[8]

and metascheduling systems [9]–[12], optimizing the execution of workflows across heterogeneous resources at multiple geographically distributed sites has not received much attention. Most metascheduling systems run the entire application or workflow at a single site. Systems such as Swift [2] 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 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. In this paper, we focus on optimizing many-task workflows across sites.

Our specific contributions in this work are four-fold:

- Development of the notion of workflow skeletons to capture, explore, analyze and model empirical workflow behavior with regard to dynamics of computation and data movement.
- Formulation of an algorithm to explore and construct an optimized schedule, according to the modeled workflow behavior.
- Integration of the workflow skeleton and the scheduling algorithm into a workflow deployment system.
- Demonstrate the effectiveness of our system using a series of empirical workflow and task patterns and real scientific applications on distributed compute resources.

We show that our system significantly improves (up to 60%) the execution time for various application workflows.

The remainder of the paper is organized as follows. Section II presents an overview of Swift, a workflow expression and execution framework, 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 V presents an evaluation of the proposed approach using real scientific workflows over multiple sites with distinct characteristics. Section VI discusses related work. Conclusions are drawn in Section VII.

II. BACKGROUND

In this section we introduce parallel workflow scripting, resource scheduling, and workload behavior modeling techniques which forms the basis our work. First, we define the terminologies used in this paper. A *workflow* is a well-defined 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.

A. Swift: Parallel Workflow Scripting

Swift is a scripting framework for parallel execution of ordinary programs [13]. The Swift runtime contains a powerful platform for running these user programs on a broad range of computational infrastructures, such as clouds, grids, clusters, and supercomputers out of the box. As the example in Figure 2(a) shows, in Swift, 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 via Swift coasters [13]–[15] 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 number of tasks assigned to each site. However, it does not distinguish the latency in task execution from the overhead in data transfer. Moreover, the number of tasks to be executed on each node remains a global constant; it may starve some CPUs if the number is too low, or thrash the shared cache or memory if the number is too high. Therefore, the resulting schedule is sub-optimal.

B. On Resource Scheduling

The resource and job scheduling problem is a classic NP-hard problem [16]. It can be formally stated by resource and job definitions, and the algorithms vary depending on characteristics of resources and jobs. For instance, job-shop scheduling [17] is for multiple independent jobs with varying sizes and multiple homogeneous resources.

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 a different set of remote sites. The sites themselves may also have a broad spectrum of system architectures and capacities. In order for scheduling to take into account all these factors, sophisticated algorithms are needed. In this paper, we extend our previous linear programming based scheduling algorithms [18] to incorporate performance models as well as network and compute resources. In general, scheduling algorithms considering several factors (e.g., network and compute resources) at the same time are termed as *joint scheduling* algorithms [19], [20]. Joint scheduling algorithms are advantageous for better performance while they may need more sophisticated mechanisms and may lead to high time complexities. Even though Many previous studies including our work [18] have addressed these issues, many more factors as described above are left unconsidered for reasons such as time complexities. Our scheduling algorithms extended for task performance models for multiple computation sites are unique from the perspective of incorporating all factors such as network and compute resources and site-specific task performance models. In addition, model-driven job graphs will be able to provide rich semantics for flexible and efficient scheduling.

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
XSEDE Trestles	324X32=10368	2.4GHz	63.2 GB	limited	unknown
RCC Midway	160X16=2560	2.60GHz	32.00 GB	limited	Institute-wide access

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 [3]. It asks the user to provide a description, called *code skeleton*, that identifies a workload’s performance behavior including data flow, control flow, and computational 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 structure 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 single-node and MPI applications. In this work, we adopt and extend SKOPE to model workflows with data transfers requirements. 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 job graphs from workload skeletons.

In this section, we describe how we use modeling to schedule a workflow on multiple sites. Figure 1 illustrates the overall steps involved in our technique. User provides the workflow either as a Swift script or using some other interface. In case of latter, an approach such as [1] is used to obtain a Swift script. In the Swift script, a workflow is represented as tasks, each of which consumes or produces a number of files. From the Swift script, we generate a performance property description of the workflow, which includes tasks’ site-specific scaling and the data dependency among them. Such a description is generated in the form of our extended SKOPE language, and is referred to as a *workflow skeleton*, or *skeleton* in short. This process is done manually now but will be automated in the future. User also provides the list of resources available to execute the workflow. The skeleton then automatically generates a *job graph*, where a *job* refers to one or more tasks of the workflow grouped as a scheduling unit. Operating at the task granularity may lead to scheduling algorithm taking a long time to compute an optimal schedule. Given the NP complete nature of the scheduling problem, it is necessary to reduce the problem size and aggregate multiple tasks into one job. The job graph depicts the jobs’ site-specific resource requirements as well as the data flow among them. Such a job graph is then used as input to a scheduling algorithm, which also takes into account the resource graph that describes the underlying properties at multiple sites and the network connecting them. The output of the algorithm is an optimized mapping between the job graph and the resource graph, which the scheduler then uses to dispatch the jobs.

III. MODEL-DRIVEN MULTISITE WORKFLOW SCHEDULING

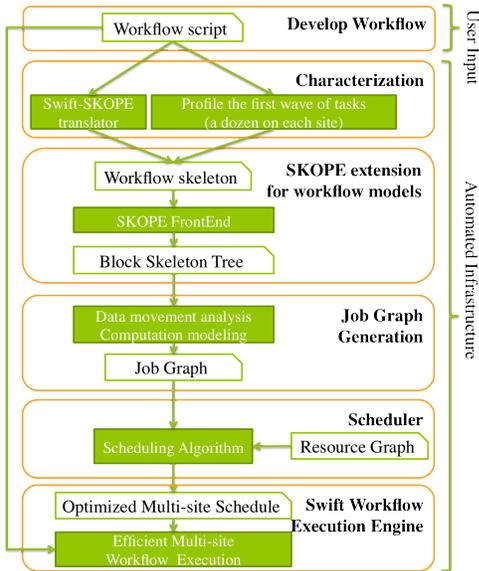


Fig. 1. Our conceptual framework for multi-site workflow scheduling.

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 for loop	for var_range {list_of_statements}
Parallel for 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

A workflow skeleton has two specific purposes: (1) define *tasks*; (2) identify data movements among tasks.

The syntax of a workflow skeleton is summarized in Table II. Figure 2 presents an illustrative workflow from the geoscience area involving *ab initio* simulations via the VASP [21] tool. In

Figure 2(a), shows 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, task definitions, and the control and data flow among the tasks. The size of each type of file are summarized in lines 3-4 of the skeleton.

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 task 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.

To describe a task’s distinct behavior over various sites, the user can represent the task’s skeleton as a `switch` statement. Each of its `case` statements describes the task’s performance model for the corresponding site. An example skeleton description of a task is demonstrated by lines 17-35 in Figure 2(b). This performance model is then used by the scheduler to determine how many tasks to assign to a node.

Given the high-level nature of workflows and the structural similarity between workflow script and skeleton, generating the workflow skeleton is straightforward and will be automated in the future by a source-to-source translator. The major effort in writing a workflow skeleton falls on profiling tasks over available systems in order to obtain site-specific performance models. For each task, we measure its single node execution time when it is co-executed with multiple tasks, up to a point where all computing resources (i.e., cores) on the same node are exploited. We then apply quadratic curve fitting to obtain an empirical performance model.

Since a typical workflow is repeatedly executed, such performance information can be obtained from historical data, either by explicit user measurement or by implicit profiling.

B. Procedural Job Graph Generation

The workflow skeleton produces a job graph as input to the scheduling algorithm. A job graph is a DAG describing the performance characteristics of each job and the data movement among them. Figure 3 illustrates the job graph generated from the workflow skeleton in Figure 2(b). In a job graph, nodes refer to jobs and edges refer to data movements. Note that the structure of the job 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 job is a scheduling unit that may refer to a group of tasks. Grouping multiple tasks 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 job with a number of tasks, and the outer loop is sized according to the desired number of jobs, 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 jobs 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.

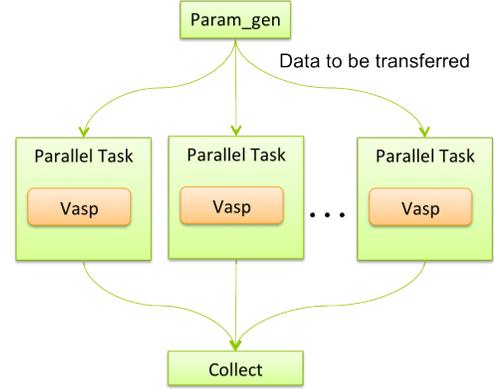


Fig. 3. Job graph for the workflow shown in Figure 2(a).

Generating a job graph from a workflow skeleton involves four major steps. First, we decide the group size according to the number of tasks and the number of sites. Second, we obtain the data footprint for each job by aggregating the data footprints for tasks within a group. Third, we construct the data flow among dependent jobs. Finally, we derive a symbolic expression to express the execution time of a job over different systems; this also involves aggregating the execution time of tasks within a group.

The key to our technique is data movement analysis, for which we apply array section analysis using bounded regular section (BRS) [22]. 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 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 a task accesses $A[r][c]$ in a nested `for` loop with two iterators, r and c . The tile corresponding to the loop is denoted as $\mathbb{T}(r, c) = \{r : \langle r^l : r^u : r^s \rangle; c : \langle c^l : c^u : c^s \rangle\}$, where each of the three components represents the lower bound, upper bound, and stride, respectively. The overall pattern accessed within this loop is denoted as $A[\langle r^l : r^u : r^s \rangle][\langle c^l : c^u : c^s \rangle]$, which is summarized by $\mathbb{P}(A[r][c], \mathbb{T}(r, c))$. Patterns can be intersected and/or unioned.

To obtain the data footprint of a job, 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 job.

We then build the data flow among jobs. First, we scan all BST nodes that correspond to jobs. Pairs of nodes producing and consuming the same array of files become candidate dependent jobs. Next, we perform intersection operations between produced and consumed patterns to determine the exact pattern

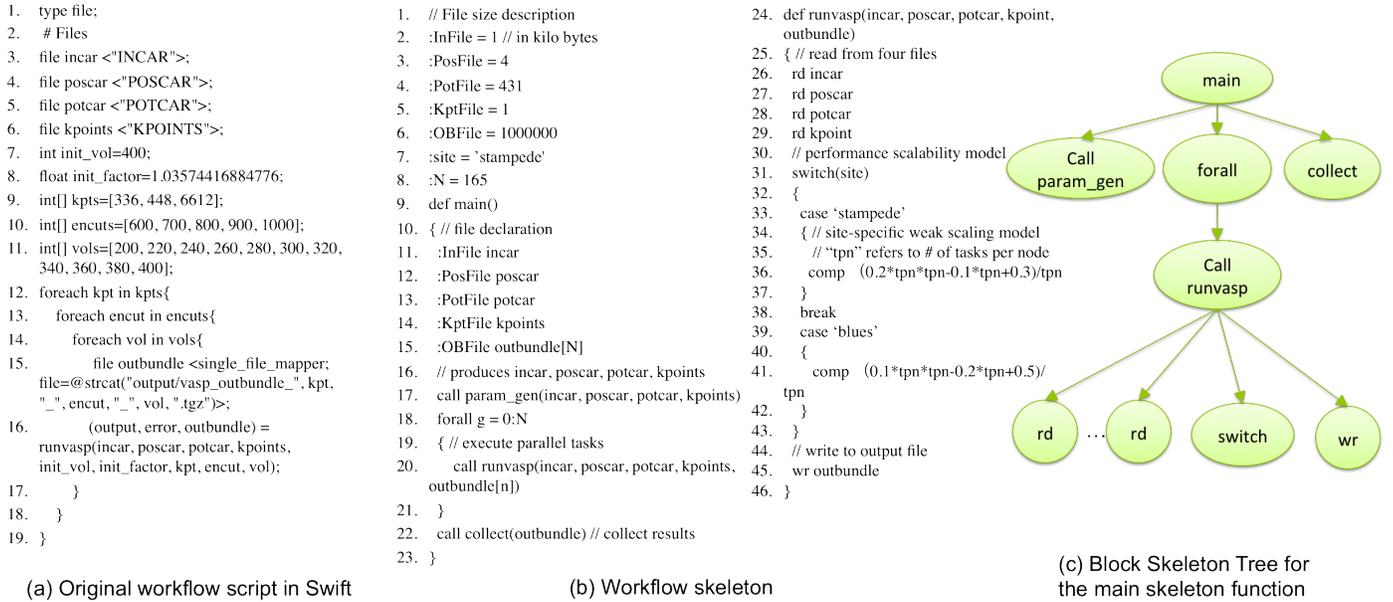


Fig. 2. Illustrative workflow script (a), skeleton (b), and the corresponding block skeleton tree (BST) (c) for a pedagogical workflow related to a geoscience application.

that caused the dependency. The size of the dependent patterns is the amount of data movement associated with an edge in the job graph.

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

The resulting job graph is output in the form of an adjacency list. In addition, each node has two attributes, one is the number of independent tasks within this job, and the other is the performance modeling which estimates the execution time of each job given the number of assigned cores. It is then passed to the scheduler algorithm to generate an optimized mapping among the jobs and resources.

C. Multisite Scheduling

The job graph provides a description about the requirement and behavior of a workflow. In order for a scheduler to recommend an optimized schedule, it requires knowledge about the distributed compute resources where a workflow is deployed. In particular, the knowledge includes available compute nodes at each site, the number of cores per node, and the network bandwidth amongst computation sites. We use a resource graph to describe such information about underlying distributed systems. Fig. 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 (b) is the resource graph corresponding to Table I. We ran disk-to-disk transfer probes to identify the achievable throughput among our computation sites.

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

In this paper, we extend our joint scheduling algorithm [18] that takes into account both compute resources and network paths in order to incorporate site specific performance models. Our previous work based on linear programming considers network resources together with compute resources. In distributed workflow scheduling, data movement among sites are not trivial, especially when the network resources are not abundant. That means independent scheduling of compute resource and network paths may not give a near optimal schedule. Our previous work schedules workflows by converting a scheduling problem into a network flow formulation, for which there are well-known linear programming approaches. However, these formulations lack task performance models for heterogeneous compute resources. In order to schedule workflows among multisite compute resources, a new notion of task-resource affinity is proposed and incorporated into our previous workflow scheduling algorithms. This is novel in a sense that all factors including network and compute resources and task performance models per site are incorporated into one formulation.

We set the resource capacities, C_n , which represents computation power at site n , associated with nodes in Fig. 4. Note that this capacity is used for a bandwidth of a link representing a compute resource in our network flow formulation [18]. 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 . This is analogous to a situation where d_i amount of water flows through a water pipe with capacity C_s . C_s and d_i are relative values. To describe that task i takes 1 second 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 $\frac{t_s^i}{E(CE_s)} \cdot \frac{t_s^i}{E(CE_s)}$ 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 task'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 resource to the edges of the auxiliary graph. [18]

On the other hand, our scheduler combined with intelligent SKOPE job graph generation can achieve better performance. A job in a job graph that SKOPE provides to schedulers may correspond to multiple parallel tasks and is a basic scheduling unit. Previously, partitioning techniques for multiple same-level tasks, called *task clustering* [23], 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. The SKOPE job graph generator may provide as many parallel jobs in a job graph as the number of available computation sites. In this way, our scheduler can maximize parallel execution of jobs according to the current resource environment while static task clustering as in [23] would not utilize idle resources.

IV. EXPERIMENTAL SETUP

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

A. Application Characteristics

a) Synthetic Workflow: We use a synthetic application consisting of various computation and memory intensive tasks. The computation intensive task is matrix multiplication while the memory intensive tasks are array summation, scale, and triad operations from the stream benchmarks [24]. The graph shown in Figure 5 shows the overall data flow of synthetic workflow. The application consists of a total of 255 tasks.

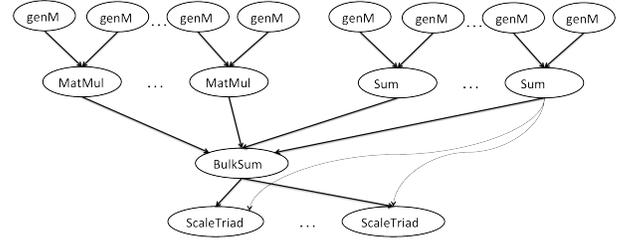


Fig. 5. The synthetic application workflow representation

b) Image Reconstruction: The Advanced Photon Source (APS) at Argonne National Laboratory provides the Western Hemisphere's most brilliant X-ray beams for research. Typically, during the experiments, the data generated at the beamlines is typically moved to a local HPC cluster for quasi real-time processing through a LAN. At the end of the experiments, raw and processed data is moved to the users home institution typically over a WAN. The science workflows at APS is changing in nature rapidly as a result of double exponential growth in both detector resolution and experiment repetition rate. As data volumes increase, we see increased need to use remote computation facilities to process and analyze the data. For example, a near-real-time analysis of a few terabytes of APS data at a remote compute cluster at Pacific Northwest National Laboratory was done recently. In our experiments, we use a downsized APS application consisting of two tasks, a raw image reconstruction task and an image analysis task, which are both compute-intensive tasks and can be distributed among remote computation sites. Fig. 6 shows an APS 3D volume which can be obtained through 2D slice images reconstructed from raw tomography data at computation sites in our experiments. We also assume a workflow in which ten datasets are generated by ten experiments performed at X-ray beamlines (note that there are more than 60 beamlines at APS) and will be processed using remote computation sites.

c) PowerGrid: The electrical power prices in a region are a result of combination of many stochastic and temporal factors, including variation in supply and demand due to market, social, and environmental factors. Evaluating the feasibility of future generation power grid networks and renewable energy sources requires modeling and simulation of this complex system. In particular, the power grid application described here is used to statistically infer the changes in the unit commitment prices with respect to small variations in random factors. The application involves running a stochastic model for a large number of elements generated via a three-level nested

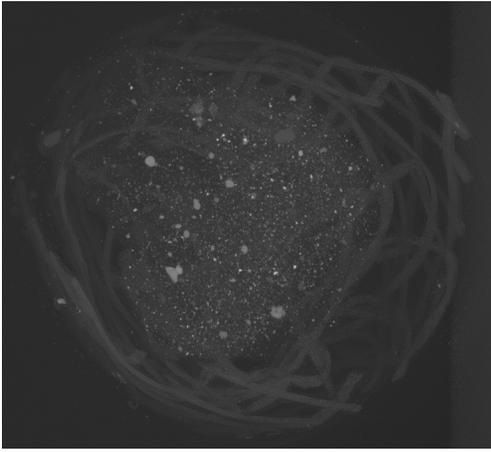


Fig. 6. APS 3D volume image reconstructed from raw tomography data.

foreach loop. A numerical algorithm is run to compute lower and upper bounds, which converge for large enough samples. A moderate sample size of five samples can generate hundreds of thousands of tasks. Each task makes call to the Python-implemented sample generation and AMPL models making it an interlanguage implementation spanning Python and AMPL interpreters, as depicted in Fig. 7.

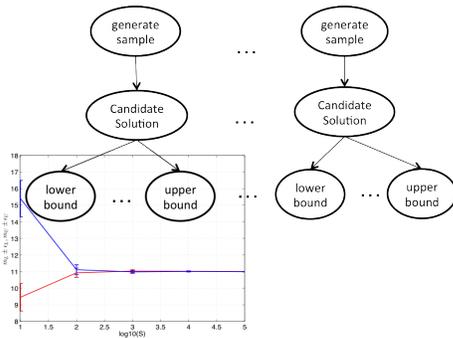


Fig. 7. Electrical power price analysis application components: tasks and a plot showing convergence of upper and lower bounds for large sample sizes.

B. Computation Sites

We used four execution sites—XSEDE’s Stampede and Trestles clusters, Argonne’s Laboratory Computing Resource Center (LCRC) cluster ‘Blues’ and University of Chicago Research Computing Center (RCC) cluster ‘Midway’ to evaluate our approach. A summarized characterization of these four 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. Similarly, Trestles is another supercomputing environment offered by XSEDE. It consists of 324 compute nodes and over 100 TFlop/s of peak performance. It employs a PBS based resource manager.

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 recently put in production mode. 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 ‘synthetic’, ‘powergrid’ and ‘image reconstruction’ application workflows encoded in Swift. We submit the application workflows to four sites (Blues, Midway, Stampede and Trestles) from a remote machine (located outside of the network domains of the clusters), where all input data resides. We first gather data needed to build empirical performance models. To do so, we conduct a pilot run, where we allocate one node for each site and execute each task on each site, with different number of tasks per node. This measures the execution time of a task when there are other tasks running on other cores of the same node. Figure 8 shows one such set of measurements on each site for the powergrid workflow. This provides the node-level weak scaling trend for each task. We use quadratic curve fitting and generate a scaling model, which is incorporated into the workflow skeleton.

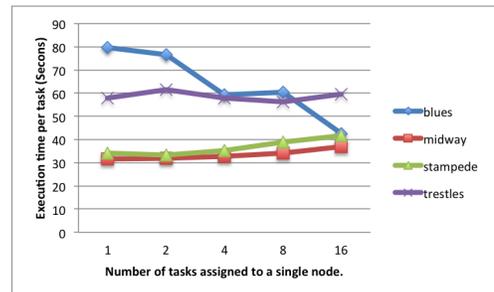


Fig. 8. Profiled single-node weak scaling for the PowerGrid workflow on different sites.

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.

In the second set of experiments, we alter the Swift configuration and distribute the jobs according to a schedule proposed by our scheme. We plot the execution trace log in order to generate a cumulative task completion plot as shown in figures 9, 10, 11, 12. The plot labeled ‘baseline’ and ‘enhanced’ show cumulative task completion with default and proposed schemes respectively. We notice an initial poor performance (especially figure 10) as the schedule starts acquiring resources via local resource managers which rapidly improves as the resources become available on remote sites. We note a sharp and steady increase in task completion for short tasks as seen in

most of the synthetic workflow. On the other hand, a steps-like plot is seen in the case of compute intensive APS workflow of figure 12. Similarly the poor performance of default scheme can be attributed to ramping up of jobs at sites with poor affinity for those jobs and/or a lower bandwidth to carry data to the site once the bandwidths are saturated by initial rampup.

It can be noted from the results in figures 9, 10, 11, 12, we achieve a shorter makespan with an informed schedule and save the effort for the users to fine tune the performance in the default Swift schedule. In complex and large real workflows interfaced with multiple remote sites, such fine tuning will consume a lot of time or even impossible. Our scheme achieves up to 60% improvement in makespan over the default scheme. This is possible because of the following key decisions taken by our schedule:

- 1) Accounts for both computational and data movement characteristics of tasks and capacities of resources.
- 2) Steers computation according to a proactive plan based on task-resource affinity.
- 3) Groups tasks into chunks to send out to sites ensuring load-balancing from the beginning of execution.

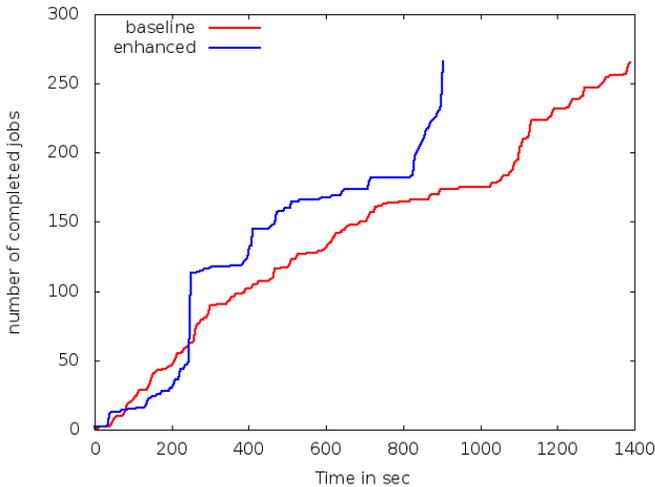


Fig. 9. Synthetic workflow performance with a load of 255 tasks.

VI. RELATED WORK

Large scale applications have been shown to benefit significantly on heterogeneous systems [25] for data-intensive science [26] and under multiple sites infrastructure [27], [28]. 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.

A. Workflow Management

Some of the well-known workflow management systems include Condor DAGMan [29], Pegasus [5], Taverna [4] and makeflow [30]. Condor DAGMan provides a 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

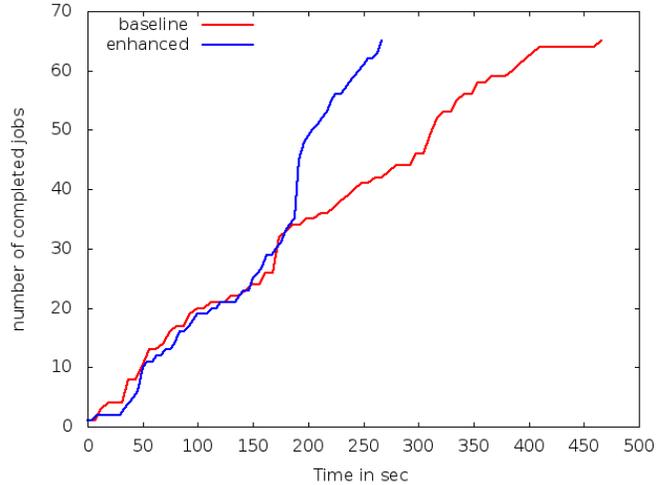


Fig. 10. PowerGrid workflow performance for a small sample size resulting in 65 tasks.

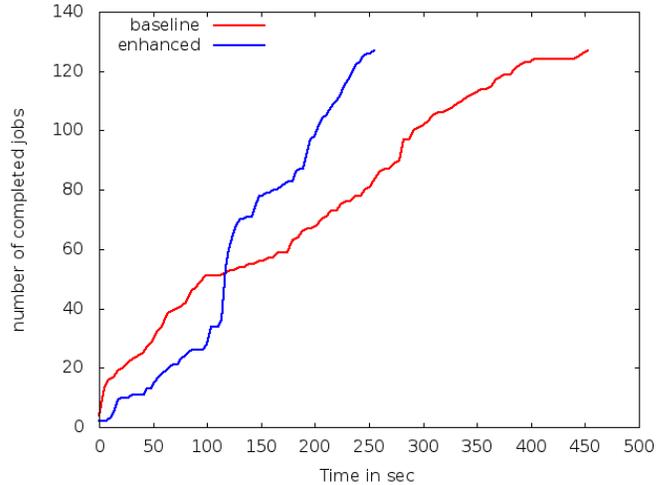


Fig. 11. PowerGrid workflow performance for a large sample size resulting in 127 tasks.

by a DAG. Pegasus requires task execution time information related to each task in a workflow. Condor DAGMan is 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 resource affinity among the tasks.

HEFT or other heuristics use averaged execution times of a task over every possible resource 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,

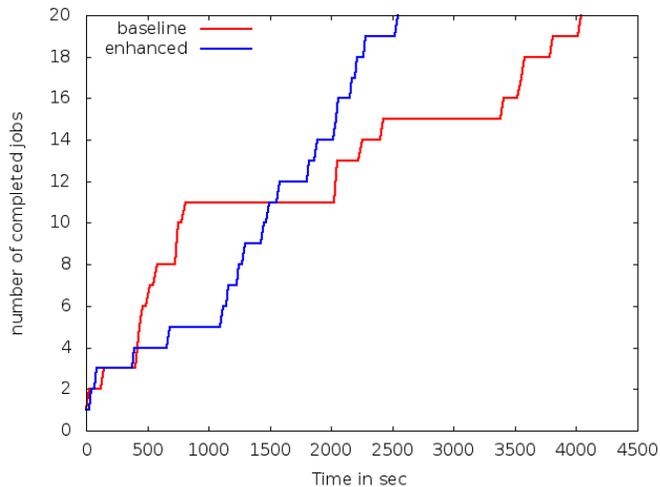


Fig. 12. APS application performance with a load of 20 tasks.

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 [31] on generic wide-scale environments and more recently on European Grids [32]. While they provide detailed analysis of workflow deployment, simulations often take a significant amount of time to emerge with an accurate picture of real-world scenarios and tend to lag behind in mapping the new architectures.

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 parallel and distributed processing 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 [33], [34], where applications are usually run at a small scale to obtain knowledge about the execution overhead and their performance scaling. Snavely et al. developed a general modeling frameworks [35] 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 [36]–[39]. All the above techniques target computational kernels and parallel applications.

SKOPE [3] provides a generic framework to model workload behavior. It has been used to explore code transformations when porting computational kernels to emerging parallel hardware [40], [41]. 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 propose a multi-site scheduling approach for scientific workflows using performance modeling. We introduce the notion of workflow skeletons and extended the SKOPE framework to capture, analyze and model the computational and data movement characteristics of workflows. We develop a resource and job aware scheduling algorithm that utilizes the job graph generated using the workflow skeleton and the resource graph generated using the resource description. We incorporate our approach into Swift, a script-based parallel workflow execution framework. We evaluate our approach using real-world applications in image reconstruction for an experimental light-source and for modeling of power-grid in a multi-site environment. Our approach improves the total execution time of the workflows by as much as 60%.

ACKNOWLEDGMENT

We thank our colleague Victor Zavala for their collaboration on power application. 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] K. Maheshwari, A. Rodriguez, D. Kelly, R. Madduri, J. Wozniak, M. Wilde, and I. Foster, "Extending the galaxy portal with parallel and distributed execution capability," in *SC'13 workshops*, Nov. 2013, accepted for publication.
- [2] 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, pp. 633–652, 2011.
- [3] 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," *ANL Tech. report*, 2012.
- [4] 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>
- [5] 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.
- [6] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, Aug. 2006. [Online]. Available: <http://dx.doi.org/10.1002/cpe.v18:10>
- [7] E. Bartocci, F. Corradini, E. Merelli, and L. Scortichini, "Biomwms: a web-based workflow management system for bioinformatics," *BMC Bioinformatics*, vol. 8, no. S-1, 2007. [Online]. Available: <http://dblp.uni-trier.de/db/journals/bmcbi/bmcbi8S.html>
- [8] P. Mhashikar, Z. Miller, R. Kettimuthu, G. Garzoglio, B. Holzman, C. Weiss, X. Duan, and L. Lacinski, "End-to-end solution for integrated workload and data management using glideinwms and globus online," *Journal of Physics: Conference Series*, vol. 396, no. 3, p. 032076, 2012. [Online]. Available: <http://stacks.iop.org/1742-6596/396/i=3/a=032076>

- [9] G. Sabin, R. Kettimuthu, and A. Rajan, "Scheduling of parallel jobs in a heterogeneous multi-site environment," in *in the Proc. of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes In Computer Science*, 2003, pp. 87–104.
- [10] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan, "Distributed job scheduling on computational grids using multiple simultaneous requests," in *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, ser. HPDC '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 359–. [Online]. Available: <http://dl.acm.org/citation.cfm?id=822086.823345>
- [11] E. Huedo, R. S. Montero, and I. M. Llorente, "A framework for adaptive execution in grids," *Softw. Pract. Exper.*, vol. 34, no. 7, pp. 631–651, Jun. 2004. [Online]. Available: <http://dx.doi.org/10.1002/spe.584>
- [12] D. Sulakhe, A. Rodriguez, M. Wilde, I. Foster, and N. Maltsev, "Using multiple grid resources for bioinformatics applications in gadu," in *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, ser. CCGRID '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 41–. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1134822.1134954>
- [13] 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.
- [14] 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.
- [15] 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. DIDD '12. New York, NY, USA: ACM, 2012, pp. 3–12. [Online]. Available: <http://doi.acm.org/10.1145/2286996.2287000>
- [16] O. Sinnen, *Task scheduling for parallel systems*. Hoboken N.J.: Wiley-Interscience, 2007.
- [17] R. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Tech. Journal*, vol. 45, pp. 1563–1581, 1966.
- [18] 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.
- [19] W. Guo, W. Sun, Y. Jin, W. Hu, and C. Qiao, "Demonstration of joint resource scheduling in an optical network integrated computing environment [topics in optical communications]," *Communications Magazine, IEEE*, vol. 48, no. 5, pp. 76–83, 2010. [Online]. Available: 10.1109/MCOM.2010.5458366
- [20] Y. Wang, Y. Jin, W. Guo, W. Sun, W. Hu, and M.-Y. Wu, "Joint scheduling for optical grid applications," *Journal of Optical Networking*, vol. 6, no. 3, pp. 304–318, Mar. 2007. [Online]. Available: <http://jon.osa.org/abstract.cfm?URI=JON-6-3-304>
- [21] J. P. Perdew and A. Zunger, "Self-interaction correction to density-functional approximations for many-electron systems," *Phys. Rev. B*, vol. 23, p. 5048, 1981.
- [22] P. Havlak and K. Kennedy, "An implementation of interprocedural bounded regular section analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 2, 1991.
- [23] 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>
- [24] J. D. McCalpin, "STREAM: Sustainable memory bandwidth in high performance computers," University of Virginia, Tech. Rep., 1991-2007.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] 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>
- [29] 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/r6un6312103m47t5/>
- [30] 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>
- [31] 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.
- [32] 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.
- [33] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on GPUs," in *PPoPP*, 2010.
- [34] 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.
- [35] A. Snively, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for performance modeling and prediction," in *SC*, 2002.
- [36] 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.
- [37] 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.
- [38] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *ASPLOS-XII*, 2006.
- [39] 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.
- [40] J. Meng, V. A. Morozov, K. Kumaran, V. Vishwanath, and T. D. Uram, "GROPHECY: GPU performance projection from CPU code skeletons," in *SC*, 2011.
- [41] J. Meng, V. A. Morozov, V. Vishwanath, and K. Kumaran, "Dataflow-driven GPU performance projection for multi-kernel transformations," in *SC*, 2012.