Notice of Violation of IEEE Publication Principles

"Scientific Workflow Partitioning and Data Flow Optimization in Hybrid Clouds"
by Rubing Duan, Rick Siow Mong Goh, Zheng Qin and Young Liu
in Pre-Prints for IEEE Transactions on Cloud Computing

After careful and considered review of the content and authorship of this paper by a duly constituted expert committee, this paper has been found to be in violation of IEEE's Publication Principles.

Due to the nature of this violation, reasonable effort should be made to remove all past references to this paper and it should not be used for research or citation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing

Scientific Workflow Partitioning and Data Flow Optimization in Hybrid Clouds

RUBING DUAN, RICK SIOW MONG GOH, ZHENG QIN, YONG LIU

Institute of High Performance Computing, A*STAR, Singapore

duanr@ihpc.a-star.edu.sg

***Abstract*—The scheduling and execution of workflow applications are complex issues because of the dynamic and heterogeneous nature of distributed computing environments like hybrid clouds. While hybrid cloud computing environments provide good potential for achieving high performance and low economic cost, it also introduces a broad set of unpredictable overheads. This paper describes a novel approach to refine workflow structure and optimize intermediate data transfers without changing the scale of scheduled cloud service for large-scale scientific workflows containing thousands (or even millions) of tasks. These methods include pre- and post-partitioning of workflows, data flow optimization, static and dynamic optimization, as well as virtual single execution environment incorporated into a workflow management system. We demonstrate the effectiveness of our methods on the three real applications, Montage, Broadband, and WIEN2k, executed in a real hybrid cloud.**

I. INTRODUCTION

Scientific workflows [21] are widely being used today to describe complex, multi-step applications that process potentially large amounts of data in bioinformatics, earthquake science, ecology, physics, etc. These workflows often execute in clouds or grids, taking advantage of particular resource characteristics, their availability, the distribution of data sources, and other capabilities. At the same time, applications are becoming increasingly data-intensive, but data management within workflows has received limited attention. Experiments such as those conducted by high-energy physics and gravitational-wave physics [1] are collecting TeraBytes and PetaBytes of data and storing them in distributed archives. The next generation sequencing machines are bringing bioinformatics to the same scale.

Most existing systems consider the allocation of computation resources and distribution of tasks, but do not effectively take into account the refinement of workflow structure, and communication overheads. Many scheduling algorithms can improve partial performance of a workflow, but they cannot optimize communication for the whole workflow, for example, inserting new tasks to refine workflow structure, decreasing the number and size of intermediate files, organizing data transfers in a more efficient way, etc.

Because of the distributed nature of computational and storage resources in clouds, and the fact that data is often hosted in third-party archives like Amazon S3 (Simple Storage Service) , EBS (Elastic Block Store), or Google GCS (Google Cloud Storage), it is sometimes hard to "compute where the data is" - a simple solution to dealing with large-scale data sets. Scientists in search of compute cycles are distributing computations across multiple resources. Thus the issue of efficient data transfers between clouds and efficient data transfer to permanent storage are becoming critical. Whenever possible, data need to be transferred in bulk with as few individual, small data transfers as possible. Data flow management must be flexible and scalable to match the characteristics of the data transfers required by different applications, optimize for network bandwidth, and provide controls to coordinate the transfers across multiple workflow tasks.

In this paper, we present a new approach to optimize workflow structures, and minimize intra- and inter-workflow data transfers while maintaining the scheduled level of cloud service. Our algorithms operate on a workflow scheduled onto resources and use pre- and post-partitioning as a means of identifying data transfer optimization opportunities. We implemented our algorithms within a workflow management system, and demonstrated their effectiveness within the context of three real world applications, one from the astronomy domain, one earthquake science application, and one chemistry application. We evaluated our approach running these applications across the TeraGrid [2] resources and a private cloud.

Our workflow scheduler maps large-scale scientific workflows onto multiple cloud computing sites. Some real-world scientific workflows are used, such as the physics application LIGO [1], the earthquake application Cybershake [7], the astronomy application Montage [5], etc. Condor and DAGMan [22] workflow engine are used to launch workflow tasks and maintain the dependencies between them. In [11], a footprint algorithm minimized the amount of space required by a workflow during executing by removing data files at runtime when they are not required. However, workflow structures and transfers of intermediate data files are not optimized.

In this paper, we aim to devise methods that address and reduce the complexity of scheduled workflow execution plan and the performance overheads related to the workflow applications. The contributions of this paper are both theoretical and experimental. We propose:

- a group of integrated algorithms for pre-partitioning and post-partitioning workflows, which refine workflow structures with data replication and task replication. The objective of most traditional workflow partitioning algorithms is to find a partition that evenly balances the number or execution time of tasks in each sub-workflow whilst minimizing the total number or size of intermediate data files.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing

However, the objective of our workflow pre-partitioning is to find a partition that preserves the parallelism of workflow while decreasing the scheduling complexity and identifying the communication bottlenecks.
- a data flow optimization process for identifying and archiving intermediate data without affecting the parallelism of workflows.
- two-phase control and data flow optimization process for transforming and simplifying the workflow structure to further reduce its execution time.
- a new mechanism and algorithm that significantly simplifies the definition of workflows and reduces the cost of communication for compute-intensive scientific workflows with large numbers of small sized data dependencies.

The paper is organized as follows: Section II describes the conceptual model of workflows and our approach used in a workflow management system, followed by several techniques for optimizing workflow executions in Section III. The experimental results in Section IV show that our approach can effectively improve the performance of workflow execution. Related work is reviewed in Section V, and Section VI summarizes the paper's contributions and outlines the future work.

II. Model and approach

In this section, we define the workflow model and notations that will be used in the following sections. And then, we present the partitioning algorithm for optimizing data flow of workflows.

A. Model

Definition II.1 (Workflow) *Let $\mathcal{W} = (\mathcal{T}, \mathcal{CD}, \mathcal{DD})$ denote a workflow application, where $\mathcal{T} = \{t_1, \ldots, t_n\}$ is the set of tasks, $\mathcal{CD}$ is the set of control-flow dependencies*

$$\mathcal{CD} = \{(t_{src} \succ^c t_{sink}) \mid t_{src}, t_{sink} \in \mathcal{T}\},$$

and $\mathcal{DD}$ is the set of data-flow dependencies

$$\mathcal{DD} = \{(t_{src} \succ^d t_{sink}, \mathcal{D}) \mid t_{src}, t_{sink} \in \mathcal{T}\},$$

where $\mathcal{D}$ denotes the I/O data (i.e. parameters or files) to be transferred between the tasks t_{src} and t_{sink}, and $\succ^c$ and $\succ^d$ denotes the control-flow and data-flow dependency between the tasks.

Although our definition is different from many DAG based works, the separation of control flow and data flow is necessary in many cases cases. For instance, WIEN2K in Fig. 5 contains two parallelFor loops for lapw1 and lapw2, and a while loop for all tasks. Without some control flow constructs such as sequential, parallel, conditional and iterative constructs, it will be very difficult to enable users to define the exact order of tasks. Most existing work commonly suffers from one or several of the following drawbacks: no iterative constructs, no support for processing datasets or portions of datasets, and no flexible dataset-oriented data flow support for iterative constructs. In contrast, our previous work [20] supports a rich

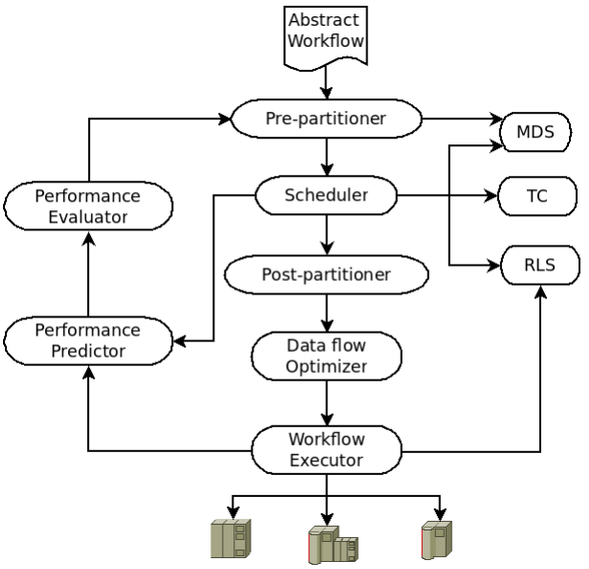


Fig. 1. Workflow execution steps.

set of control flow constructs, including iterative and parallel constructs. In order to manage and optimize the execution of a workflow, we need to define a workflow partition as:

Definition II.2 (Workflow partition) *We define a* workflow partition $\mathcal{W}' = (\mathcal{T}', \mathcal{CD}', \mathcal{DD}')$ *as the sub-workflow of the workflow $\mathcal{W} = (\mathcal{T}, \mathcal{CD}, \mathcal{DD})$ iff*

- $\mathcal{T}' \subseteq \mathcal{T}$, *and*
- $\mathcal{CD}' \subseteq \mathcal{CD} \wedge ((t_1 \succ^c t_2) \in \mathcal{CD}' \rightarrow t_1, t_2 \in \mathcal{CD}')$, *and*
- $\mathcal{DD}' \subseteq \mathcal{DD} \wedge ((t_1 \succ^d t_2) \in \mathcal{DD}' \rightarrow t_1, t_2 \in \mathcal{DD}')$.

A workflow partition $\mathcal{W}'$ also has the following properties:
(1) there must be no control-flow and data-flow dependencies to / from tasks that have predecessors / successors within the partition, which means no cross dependency;
(2) the computation loads of partitions or the number of tasks are approximately even;
(3) the number of dependencies (edges) among partitions is minimized;
(4) the potential data and task replications of partitions are identified and created.

B. System overview

Exposing the whole workflow to a scheduler can be a double-edged sword. While the scheduler can allocate all tasks to all available computing resources and perform various optimizations, the workflows that contains thousands or millions of tasks may overwhelm the scheduler, and the scheduled solution may then become too complex to be further optimized at run time, which is very important for workflows on dynamic computing environments like clouds. Even when executing in a stable environment, performance can suffer from unexpected run-time events or resources contention. A lack of flexibility to adapt to unexpected events can be a significant downfall

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing

and is in fact considered to be a major weakness of scheduled solution.

A workflow will be pre-partitioned based on its structure. Our workflow management system (WfMS) is briefly described as follows, as illustrated in Fig. 1. First, Users compose workflows using an XML generator to produce an XML based workflow description that shields users from the system details. WfMS takes this abstract description and information about the available resources and generates an executable workflow that describes the computation, data transfers and data registration tasks and their dependencies. The WfMS scheduler consults various sites information services, such as the Globus Monitoring and Discovery Service (MDS), the Globus Replica Location Service (RLS), Transformation Catalog (TC), and the Metadata Catalog Service (MCS), to determine the available computational resources and data. Given all the information, the scheduler can select resources to execute tasks based on the available resources and their characteristics as well as the location of the required data. Users can select the following algorithms to schedule their workflows: HEFT [23], random, round-robin, min-min [13], game-quick [9], etc. With a scheduled workflow, we can post-partition it and optimize its data flow, and these mechanics are presented in the sequel. WfMS generates an executable workflow, which identifies the resources where the computation will take place, identifies the data movement for staging data in and out of the computation and intermediate data, and registers the newly derived data products in the RLS and MCS.

As depicted in Fig. 1, a workflow can be regularly evaluated and adjusted at run time. Dynamic tuning can be enabled by a variety of means based on the performance prediction and evaluation, including data and task replication, and data flow optimization. Each of these mechanism allows a workflow to be tailored to a particular computing environment.

Replication can be categorized into two classes: data replication and task replication. Data replication stores the same data on multiple storage resources. Task replication executes the same task on multiple computing resources. Data replication and task replication as two different optimization mechanism can transform into each other at run time.

Finally, the executable workflow is given to the workflow executor for execution. The performance predictor and evaluator can gather information to uncover bottlenecks and trigger run-time adaptation. For instance, the performance information can trigger comparisons with expected values and assumptions that were made by the scheduler. Any significant deviation from the expectations can trigger run-time adaptation. Run-time performance and resource contention, as gathered by MDS, will trigger a variety of rescheduling. Reliability anomaly can trigger workflow re-optimization and resource avoidance. WfMS can also leverage run-time information to detect additional opportunities for parallelism, and reschedule certain partitions to execute on resources with less contention.

In summary, exploiting parallelism requires a multi-stage approach. A pre-partitioner can perform the task of maximizing the potential for parallelization assuming unbounded resources and no system load. It will be the task of the scheduler and the post-partitioner to adapt the parallelism to the available computing resources as they changed over time, and also to exploit further parallelization opportunities based on the dynamics of clouds.

Algorithm 1 The workflow pre-partitioning algorithm.

1: **Input:**
   $\mathcal{W} = \{\mathtt{t}_1, ..., \mathtt{t}_n\}$ set of the tasks,
   $\mathcal{CD}$ control-flow dependencies,
   $\mathcal{DD}$ data-flow dependencies
   s the number of available resources
2: **Output:**
   $\mathcal{PS}_{pre}$ pre-partitioned workflow
3: /* Initialization */
4: $\mathtt{t}_i.depth = max(\mathtt{t}_i.parents().depth) + 1$
5: $\mathcal{PS}_{\mathtt{CF}} :=$ partition $\mathcal{W}$ with $\mathcal{CD}$ based on task granularity
6: $\mathcal{PS}_{\mathtt{DF}} :=$ partition $\mathcal{W}$ with $\mathcal{DD}$ based on data granularity
7: **for** every partition in $\mathcal{PS}_{\mathtt{CF}}$ and $\mathcal{PS}_{\mathtt{DF}}$ **do**
8: $\quad \mathcal{PS}_{pre} := \mathcal{PS}_{pre} \cup \{(\mathcal{PS}_{\mathtt{CF}})_i \cap (\mathcal{PS}_{\mathtt{DF}})_j\}$
9: **end for**

C. Workflow partitioning algorithms

Workflow partitioning poses two basic problems: finding an exploitable partition through analysis and partitioning a workflow into a parallel form. Analysis and transformation must work together to find and exploit a specific type of parallelization. Finding workflows that cannot be partitioned is unhelpful.

Data dependencies can greatly degrade the performance of workflows. In clouds, cross-partition dependencies require communication or synchronization to maintain correctness. However, communication and synchronization are undesirable since they are relatively expensive operations. By careful partitioning and scheduling, a workflow management system can minimize the effect of data dependencies. Unfortunately, determining whether or not data dependencies exist between two partitions are nontrivial in general. Based on our experience, the number of partitions can seriously affect the efficiency and effectiveness of scheduling. Excessively large or unbalanced partitions are undesirable, since the rate of execution is limited to at most that of the slowest partition. Excessively short partitions are equally undesirable when the cost of communication between partitions dominates the cost of the partitions themselves. Therefore, an ideal partitioning produces exactly enough partitions to keep all clouds busy, while minimizing communication and latency of the slowest partitions.

Our approach consists of three steps. First, a pre-partition is performed to ensure maximum work load balance. The second step is to schedule workflow based on the partitions. The third step is selectively replicate tasks to increase the opportunities for evenly distributed work, ensure data dependencies are satisfied, and inter-partition communication latency is maximally overlapped with computation.

a) Pre-partition workflows: Workflow pre-partitioning is displayed as pseudo code in Algorithm 1. Partition the original workflow according to three data-flow dependency rules $R_{\mathtt{DF}}$:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing

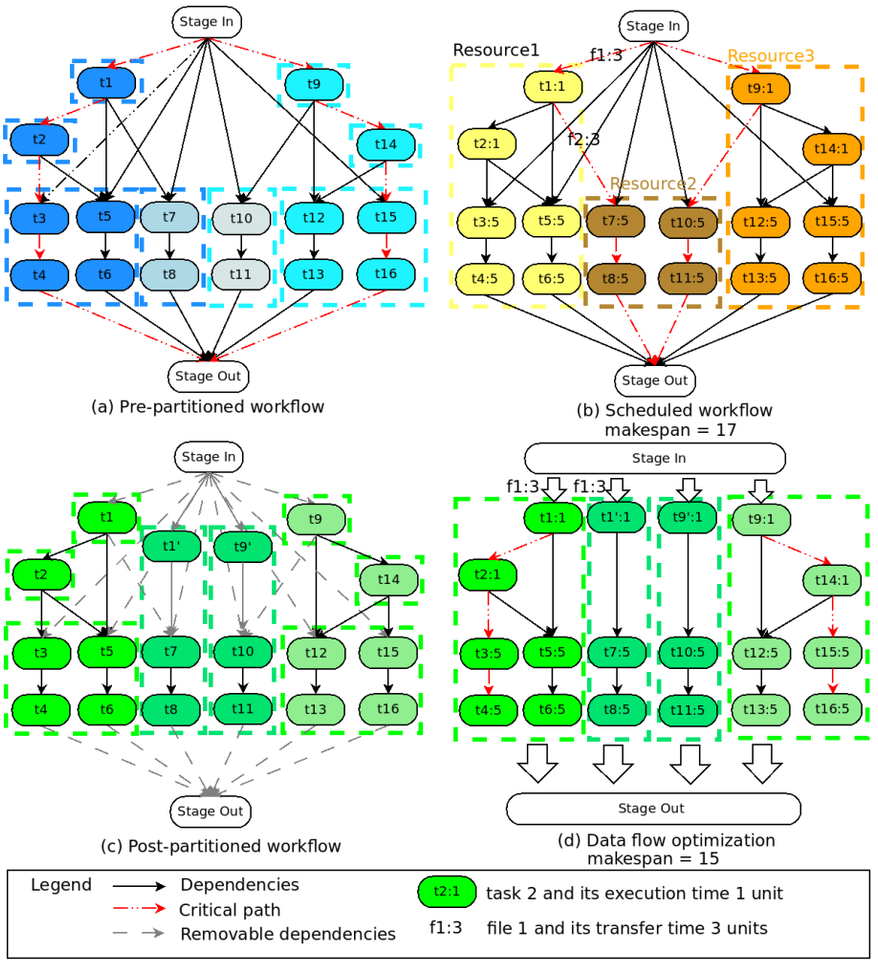


Fig. 2. A Broadband workflow example.

1) Each task of the workflow must belong to exactly one partition: $\forall\ \mathtt{t} \in \mathcal{T},\ \exists\ \mathcal{P} \in \mathcal{PS}_{\mathrm{DF}}\ \wedge\ \mathtt{t} \in \mathcal{P}\ \wedge\ \mathtt{t} \notin \mathcal{P}',\ \forall\ \mathcal{P}' \in \mathcal{PS}_{\mathrm{DF}}$.
2) Tasks scheduled on the same site belong to the same partition (step 5-6, Algorithm 1): $\forall\ \mathtt{t}_1 \in P \in \mathcal{W}_{\mathrm{DF}}\ \wedge\ \forall\ \mathtt{t}_2 \in \mathcal{P}\ \wedge\ \mathit{schedule}(\mathtt{t}_1) = \mathit{schedule}(\mathtt{t}_2)$, where $\mathit{schedule}(\mathtt{t})$ is the schedule of the task $\mathtt{t}$.
3) Eliminate the data dependencies between tasks scheduled on the same site (step 7-9, Algorithm 1): $\mathcal{DD}_{\mathrm{DF}} = \mathcal{DD}(\mathtt{t}_1 \succ^{d} \mathtt{t}_2, \mathcal{D}),\ \forall\ \mathtt{t}_1, \mathtt{t}_2 \in \mathcal{T}\ \wedge\ \mathit{schedule}(\mathtt{t}_1) = \mathit{schedule}(\mathtt{t}_2)$.

Merge the two sets $\mathcal{PS}_{\mathrm{CF}}$ and $\mathcal{PS}_{\mathrm{DF}}$ of control-flow and data-flow based partitions computed in the previous two steps into one partition set, as follows:

$$\mathcal{PS}_{pre} = \bigcup_{\substack{\forall \mathcal{PS}_i \in \mathcal{PS}_{\mathrm{CF}} \\ \forall \mathcal{PS}_j \in \mathcal{PS}_{\mathrm{DF}}}} \{\mathcal{PS}_i \cap \mathcal{PS}_j\},$$

while preserving the control-flow and data-flow dependencies and the partitioning goals described in the beginning.

Fig. 2(a) displays the result of the workflow pre-partitioning, the tasks with the same color are in the same partitions:

$$\mathcal{PS}_{pre} = \{\{\mathtt{t}_1\}, \{\mathtt{t}_2\}, \{\mathtt{t}_3, \ldots, \mathtt{t}_6\}, \{\mathtt{t}_7, \mathtt{t}_8\}, \{\mathtt{t}_9\}, \{\mathtt{t}_{10}, \mathtt{t}_{11}\}, \{\mathtt{t}_{14}\}, \{\mathtt{t}_{12}, \mathtt{t}_{13}, \mathtt{t}_{15}, \mathtt{t}_{16}\}\}.$$

Based on this partition, large-scale workflows can be easily processed by scheduling algorithms since the granularity of workflows is decreased.

b) Post-partition workflows.: After the workflow is scheduled by scheduling algorithms, such as HEFT [23], Max-min and Min-min [13], we can identify the communication bottlenecks between partitions. Fig. 2(b) displays the result of the schedule:

$$\mathcal{PS}_{schedule} = \{\{\mathtt{t}_1, \ldots, \mathtt{t}_6\}, \{\mathtt{t}_7, \mathtt{t}_8, \mathtt{t}_{10}, \mathtt{t}_{11}\}, \{\mathtt{t}_9, \mathtt{t}_{12}, \ldots, \mathtt{t}_{16}\}\}.$$

As shown in Algorithm 2, the post-partitioning of a workflow helps WfMS selectively replicate tasks to increase the opportunities for evenly distributed work. For instance, some fork and join tasks can be replicated if their child or parent tasks are scheduled on different sites, and the communication time is larger than the execution time of these fork and join tasks. More importantly, these tasks are on the critical path and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing

directly impact the workflow makespan. In Fig. 2(b), tasks t_1, t_7, and t_8 comprise the critical path, and the communication overheads impact significantly on the length of this path.

Since we need to identify some replicated tasks (step 4-15, Algorithm 2), we need to merge them with the partitions connected through control-flow dependencies : Merge the partitions that are connected through control-flow dependencies but have no data-flow dependencies (i.e. they are scheduled on the same site. step 22-28, Algorithm 2):

$$\mathcal{PS}_{post} = \bigcup_{\forall\ \mathcal{P}_i \neq \mathcal{P}_j \in \mathcal{W}'} \{\{\mathcal{P}_i \cup \mathcal{P}_j\} - \{\mathcal{P}_i\} - \{\mathcal{P}_j\} \mid ((\nexists x : x \neq i, j \wedge (\mathcal{P}_i \succ^c \mathcal{P}_j)) \wedge (\neg \mathcal{P}_i \succ^d \mathcal{P}_j)\};$$

In the final partition, there must be no control-flow and data-flow dependencies to / from tasks that have predecessors / successors within the partitions. This is achieved by iteratively applying the following formula until nothing changes anymore and the largest partitions with the largest number of tasks are achieved.

Fig. 2(c) displays the result of the workflow post-partitioning. Eight partitions are created, and the partition with one task is identified as the fork partition and can be replicated:

$$\mathcal{PS}_{post} = \{\{\mathtt{t}_1\}, \{\mathtt{t}_2\}, \{\mathtt{t}_3, \ldots, \mathtt{t}_6\}, \{\mathtt{t}'_1, \mathtt{t}_7, \mathtt{t}_8\}, \{\mathtt{t}_9\}, \{\mathtt{t}'_9, \mathtt{t}_{10}, \mathtt{t}_{11}\}, \{\mathtt{t}_{14}\}, \{\mathtt{t}_{12}, \mathtt{t}_{13}, \mathtt{t}_{15}, \mathtt{t}_{16}\}\}.$$

In Fig. 2(c), we can find that the task replication has been created and the replicated task is merged into its successor partition.

D. Data flow optimization

While managing workflows is important for clouds, inefficient data flow can cause communication bottlenecks which dominate workflow execution. With proper partitions, we can easily identify the data that need to be transferred to the successive partition. However, in order to transfer archived data, we still need to add one archiving job and one unarchiving job to the workflow for each dependency between partitions. Therefore, while reducing and eliminating some sources of overhead our method introduces some new overheads. There is still more potential to reduce the number of data dependencies between partitions. The communication is optimized and thus performance will increase.

To handle this case, we propose a data-flow optimization that can minimize the number of dependencies between partitions. Algorithm 3 shows the pseudo-code of data-flow optimization algorithm. In this algorithm, we find and remove unnecessary dependencies between partitions by scanning the dependencies between partitions (step 5-14, Algorithm 3). As shown in Fig. 2(c) and (d), 16 dependencies between tasks (the dash lines) can be replaced by partition dependencies or removed because there is no data transfer needed on the same cluster. Fig. 2(d) displays the final result of the two-stage partition and optimization:

$$\mathcal{PS}_{final} = \{\{\mathtt{t}_1, \ldots, \mathtt{t}_6\}, \{\mathtt{t}'_1, \mathtt{t}_7, \mathtt{t}_8\}, \{\mathtt{t}'_9, \mathtt{t}_{10}, \mathtt{t}_{11}\}, \{\mathtt{t}_9, \mathtt{t}_{12}, \ldots, \mathtt{t}_{16}\}\}.$$

The aim of our partitioning algorithms is to make efficient use of resources, achieve low data transfer overheads, and maximize parallelism. For example, the final four partitions can be executed independently and concurrently, and unnecessary data dependencies are eliminated by introducing the new tasks $\mathtt{t}'_1$ and $\mathtt{t}'_9$. The makespan of the scheduled workflow in Fig. 2(b) are reduced from 17 units to 15 units in Fig. 2(d).

After all previous optimizations, we still found that performance was not significantly improved, since data is not transferred in the background concurrently with workflow execution. If we handle all intermediate data after the completion of the whole partition, the archiving, compressing, transferring and uncompressing work will be executed sequentially with the partitions. Hence, we propose a simple

Algorithm 2 The workflow post-partitioning algorithm.

1: **Input:**
   $\mathcal{PS}_{pre} = \{\mathcal{P}_1, ..., \mathcal{P}_n\}$ set of the partitions,
   $\mathcal{PS}_{schedule}$ scheduled workflow,
2: **Output:**
   $\mathcal{PS}_{post}$ post-partitioned workflow
3: /* Split and create new partitions*/
4: **for** d:=1 to maxDepth **do**
5: **for** m:= 1 to $\#\mathcal{PS}_{pre}$ **do**
6: **for** n:= 1 to $\#\mathcal{PS}_{pre}$ **do**
7: **if** $(\exists \mathtt{t}_i : (\mathtt{t}_i.depth = d) \wedge (\mathtt{t}_i \in \mathcal{P}_m)) \wedge \exists(\mathtt{t}_j : (\mathtt{t}_j.depth = d) \wedge (\mathtt{t}_j \in \mathcal{P}_n)) \wedge (\mathcal{P}'_m \succ^d \mathcal{P}'_n) \wedge (\mathcal{P}'_n \succ^d \mathcal{P}'_m)$ **then**
8: $\mathcal{P}_m = \mathcal{P}_m - \{\mathtt{t}_i : \mathtt{t}_i.depth = d \wedge \mathtt{t}_i \in \mathcal{P}_m\}$
9: $\mathcal{PS} = \mathcal{PS} \cup \{\{\mathtt{t}_i : \mathtt{t}_i.depth = d \wedge \mathtt{t}_i \in \mathcal{P}_m\}\}$
10: $\mathcal{P}_n = \mathcal{P}_n - \{\mathtt{t}_j : \mathtt{t}_j.depth = d \wedge \mathtt{t}_j \in \mathcal{P}_n\}$
11: $\mathcal{PS} = \mathcal{PS} \cup \{\{\mathtt{t}_j : \mathtt{t}_j.depth = d \wedge \mathtt{t}_j \in \mathcal{P}_m\}\}$
12: **end if**
13: **end for**
14: **end for**
15: **end for**
16: /* Replicate partitions */
17: **if** a partition is a fork or join partition, evaluate if a replication of this partition can improve performance and minimize cost **then**
18: /* Make a partition replication */
19: $\mathcal{PS} = \mathcal{PS} \cup \mathcal{P}_i.replicate(cluster_x)$
20: **end if**
21: /* Merge partitions */
22: **for** i:=1 to $\#\mathcal{PS}$ **do**
23: **for** j:=1 to $\#\mathcal{PS}$ **do**
24: **if** $(\mathcal{P}_i \succ^c \mathcal{P}_j) \wedge ((\nexists x : x \neq i, j \wedge (\mathcal{P}_i \succ^c \mathcal{P}_x)) \wedge (\neg \mathcal{P}_i \succ^d \mathcal{P}_j)$ **then**
25: $\mathcal{PS}_{post} := \mathcal{PS}_{post} + \{\mathcal{P}_i \cup \mathcal{P}_j\} - \{\mathcal{P}_i\} - \{\mathcal{P}_j\}$
26: **end if**
27: **end for**
28: **end for**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing

Algorithm 3 The data flow optimization algorithm.

Input:
$\mathcal{PS}_{post} = \{\mathcal{P}_1, ..., \mathcal{P}_n\}$ post-partitioned workflow
Output:
$\mathcal{PS}_{final}$ optimized workflow
/* Identify removable dependencies */
S_R – set of removable dependencies
for i:=1 to $\#\mathcal{PS}_{post}$ **do**
  for j:=1 to $\#\mathcal{PS}_{post}$ **do**
    if $(\exists \mathcal{P}_i \succ^d \mathcal{P}'_j)) \wedge (\mathcal{P}_i.scheduledSite = \mathcal{P}_j.scheduledSite)$ **then**
      $S_R = S_R \cup \{\mathcal{P}_i \succ^d \mathcal{P}_j\}$
    else if $(\exists \mathcal{P}_i \succ^d \mathcal{P}'_j)) \wedge (\mathcal{P}_i.depth - \mathcal{P}_j.depth > 1))$ **then**
      find the shortest path from $\mathcal{P}_i$ to $\mathcal{P}_j$, and combine communication belongs to this dependency
      $S_R = S_R \cup \{\mathcal{P}_i \succ^d \mathcal{P}_j\}$
    end if
  end for
end for
replace or remove the dependencies in S_R and merge partitions again to generate $\mathcal{PS}_{final}$

but effective method, "two-phase data transfer", to solve this problem. This method splits one group of files that need to be transferred between two partitions into two groups: The first group includes the files that do not belong to the tasks with largest depth in their partition or the last few tasks in this partition; The second group has the rest of files that are transferred after the workflow. As a result, the first group contains most files that need to be transferred, and they are transferred concurrently with the execution of last few tasks.

Our algorithms deal with the data sets that workflows operate on are too many and small to move efficiently. Our data flow optimization algorithms take this into account when deciding how and which intermediate data to be transferred for the computation. A challenge for the algorithms is to figure out if the costs involved in archiving and unarchiving data are worthwhile. To conquer this challenge, we need to collect the information on file sizes, network bandwidth, and expected execution time of tasks.

III. Workflow Optimization

The workflow executor focuses on further simplifying the sub-workflows produced in the partitioning phase to achieve an easier and faster execution on the clouds. There are two phases in the optimization process: static optimization, performed before sending the partitions to the queues of clouds, and dynamic optimization, performed during the execution of a workflow.

A. Static Optimization

According to our previous experiences [15], one remote job submission to a cloud site with a batch queuing system yields about $10 - 20$ seconds of overhead, mainly due to mutual authentication latency and polling for status change.

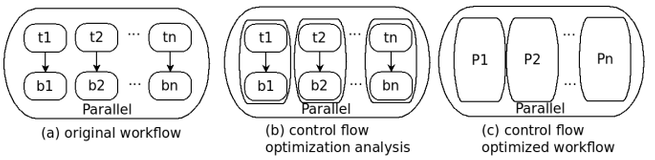


Fig. 3. Static control flow optimization.

This overhead might be significantly larger if the access to cloud sites is performed through batch queuing systems and becomes critical for large workflows comprising hundreds to thousands of activities, which is the case of our real-world workflows. Similarly, the file transfer latency is about one second, which is rather critical for the large numbers of small files that are produced by the applications.

The objective of the static optimization is to simplify and to reduce the workflow structure and size by merging atomic activities and data dependencies to reduce the network latencies and other sources of execution overheads.

1) Static Control Flow Optimization: This optimization phase aims to wrap the atomic activities defined by the user into composite activities that can be executed as one single remote job submission on a cloud site; this optimization reduces the overall latency and also decreases the complexity of large workflows. The static workflow optimizer receives a workflow partition $\mathcal{P}$ and performs a transformation that produces a new partition $\mathcal{P}'$ that merges the activities, linked through control flow dependencies but with no data dependencies (i.e. since they are scheduled on the same cloud site) into composite activities that are executed as an atomic unit of work (i.e., remote GRAM job submission): $\mathcal{P}' = \{\mathcal{P}'_1, \ldots, \mathcal{P}'_n\}$, where $\mathcal{P}'_i = \{\mathtt{t}\} \vee (\forall\ \mathtt{t}_1 \in \mathcal{P}'_i,\ \exists\ \mathtt{t}_2 \in \mathcal{P}'_i \wedge ((\mathtt{t}_1\ \delta^c\ \mathtt{t}_2) \in \mathcal{CD}_{\mathcal{P}}) \vee (\mathtt{t}_2\ \delta^c\ \mathtt{t}_1) \in \mathcal{CD}_{\mathcal{P}}) \wedge \nexists\ \mathtt{t}_3 \in \mathcal{P}'_i \wedge ((\mathtt{t}_1\ \delta^d\ \mathtt{t}_3) \in \mathcal{DD}_{\mathcal{P}} \vee (\mathtt{t}_3\ \delta^d\ \mathtt{t}_1) \in \mathcal{DD}_{\mathcal{P}})\},\ \forall\ i \in [1, n]$.

Fig. 3(a) illustrates one typical static control flow optimization in a workflow consisting of activities $\mathtt{t}_1, \ldots, \mathtt{t}_n$ and $\mathtt{b}_1, \ldots, \mathtt{b}_n$, where $\mathtt{t}_i$ and $\mathtt{b}_i$ are linked through a direct control flow dependency and have been scheduled to the same cloud site, which means that any eventual data dependency has been eliminated in the second step of the partitioning algorithm. Fig. 3(b) displays the analysis of the control flow optimization, which groups activities $\mathtt{t}_i$ and $\mathtt{b}_i$ in one single composite activity that simplifies the workflow and thus reduces the job submission latencies to half in this particular example (see Fig. 3(c)).

2) Static Data Flow Optimization: After the optimization of the control flow, the static data flow optimizer reorganizes first the input and output ports of the partitions and composite activities. Thereafter, it analyzes the data dependencies between all activities, groups them according to all dependencies involving the same source and destination cloud sites, and generates a file-transfer activity of a single, compressed archive whenever the source and destination sites are different:

$$\mathcal{DD}_{\mathcal{P}} = \bigcup_{\forall \mathcal{P}_1, \mathcal{P}_2 \in \mathcal{PS}_{\mathcal{P}}} \{(\mathcal{P}_1, \mathcal{P}_2, Archive)\},$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing

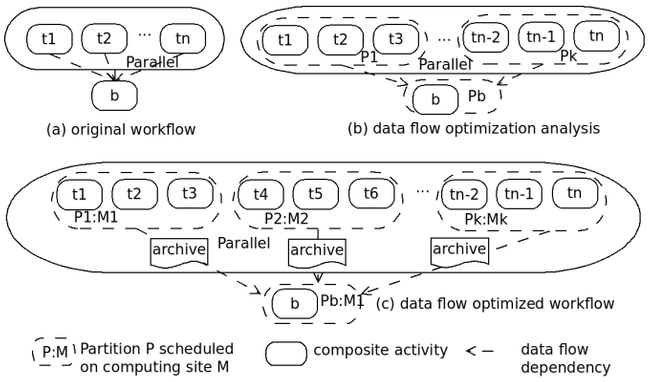
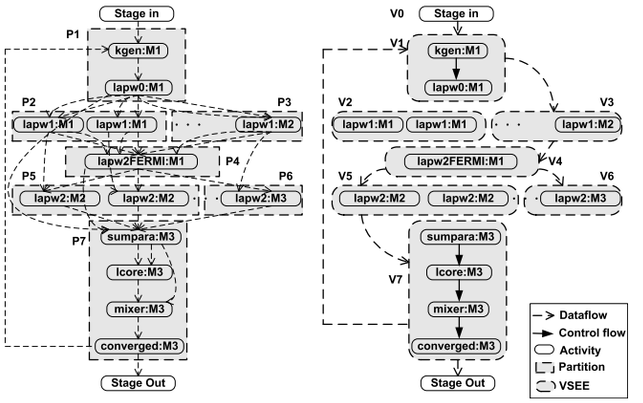


Fig. 4. Static data flow optimization.



(a) Partitioned workflow and original data flow.

(b) VSEE optimized data flow.

Fig. 5. VSEE example.

where

$$Archive = \bigcup_{\forall(\mathtt{t}_1,\mathtt{t}_2,\mathcal{D})\in\mathcal{DD}\wedge\mathtt{t}_1\in\mathcal{P}_1\wedge\mathtt{t}_2\in\mathcal{P}_2} \{\mathcal{D}\}$$

is a compressed archive of all data dependencies between partitions $\mathcal{P}_1$ and $\mathcal{P}_2$.

Fig. 4(a) presents a typical example in which activity $\mathtt{b}$ collects data from a large number of parallel activities $\mathtt{t}_1, \ldots, \mathtt{t}_n$. First, the data flow analysis aggregates the output files of all activities belonging to the same partition (i.e. scheduled on the same site – see Fig. 4(b)). Afterwards, one single file-transfer activity is generated between the partitions that are scheduled on different sites; this step reduces the number of file-transfers from n to one in this example (see Fig. 4(c)).

B. Dynamic Optimization

Many external factors during the execution of large workflows that no longer follows the plan computed by the Scheduler. Such unpredictable factors might include unexpected queuing times, external load on processors (e.g. on cloud sites that also serve as student workstation labs, such as in our real cloud environment, unpredictable availability of processors on workstation networks, jobs belonging to other users on parallel machines, congested networks, or simply inaccurate prediction information). In addition, we often encountered sites that offer a limited capacity for certain resources, for example, an I/O site that allows only a limited number of concurrent file-transfers. The dynamic optimizer handles the runtime steering of the workflow execution and aims to minimize the losses due to such unpredictable factors that violate the optimized static schedule computed by the Scheduler and using HEFT [23].

The scientific workflows that we use to validate our work are typically characterized by a large number of independent activities to be executed in parallel, which we model in XML as a composite activity using a *parallel loop* syntax. Moreover, such workflows may have a dynamic structure that is known only at runtime; for example, the number of activities to be executed in parallel might depend on the value taken by some output port of a preceding activity. The dynamic optimizer handles this situation by building the new structure of the workflow as soon as it is known and by sending it back to the scheduler for a new optimized mapping onto the cloud.

In addition, executing such large numbers of parallel activities often yields a load-imbalance (due to, for example, some unexpected external jobs submitted to the queue) that leaves some of the cloud sites idle, while others are overloaded with activities waiting in the queue. To handle this situation, the workflow executor regularly checks the load of available cloud sites based on the number of activities queued. We implemented the job queue with an open source message broker, RabbitMQ, to encapsulate a task as a message and send it to the queue. If an uneven distribution is detected (using execution time information provided by the Performance Prediction service), it migrates some of the queued activities or replicates them to the less loaded sites (e.g. with free processors). Additionally, the executor must also replicate the necessary input files as part of a data flow optimization process.

C. Virtual Single Execution Environment (VSEE)

Certain scientific workflow applications are characterized by a large (hundreds to thousands) number of activities with complex data dependencies instantiated by a large number (hundreds to thousands) of small files (several kilobytes each). In such cases, the overhead of communication is dominated by latencies caused by sending individual small files where the effective data transfer is small. This overhead can be easily modeled through a commonly used analytical formula that models the predicted communication time T_{comm} for sending n files of aggregated size $size$:

$$T_{comm} = n * Latency + \frac{size}{Bandwidth},$$

where the latency for mutual authentication to the file transfer server (orders of seconds) clearly dominates the communication time if n is large.

To handle this case, we propose a data flow optimization called *Virtual Single Execution Environment* (VSEE) that replaces the data dependencies between activities with the full

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing

Algorithm 4 The VSEE dataflow optimization algorithm.

Input:
$\mathcal{W}_\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_m\}$ - partitioned workflow,
$R_\mathcal{P}$ – partition data dependency
$VSEE = \{V_1, V_2, ..., V_m, V_{in}, V_{out}\}$ - set of VSEE
R_V – VSEE partition relationship
Output:
$E_\mathcal{P}$ – environment transfer set
/* **Phase 1** : Initialize $E_\mathcal{P}$*/
for every partition $\mathcal{P}_i$ in $\mathcal{W}$ **do**
  for every partition data dependency R_{Pj} in $R_\mathcal{P}$ **do**
    if R_{Pj} satisfies $\mathcal{P}_x\ \delta^d\ \mathcal{P}_i$ and $\mathcal{P}_x \in \mathcal{W}$ **then**
      $E_{Pi} := E_{Pi} \cup \{V_x\}$
    end if
  end for
end for
/***Phase 2** : Optimize $E_\mathcal{P}$*/
for every set E_{Pi} in $E_\mathcal{P}$ **do**
  $Tmp := \phi$
  for every set V_j in E_{Pi} **do**
    /*subset() gets all subsets of V_j with R_V*/
    $Tmp := Tmp \cup E_{Pj} \cup subset(V_j)$
    if $V_i = V_j$ **then**
      $Tmp := Tmp \cup V_j$
    end if
  end for
  $E_{Pi} := E_{Pi} - Tmp$
end for

I/O *data environment*, defined for a partition $\mathcal{P}$ as follows:

$$V_\mathcal{P} = \bigcup_{\forall(\mathcal{P}',\mathcal{P},\mathcal{D})\in\mathcal{DD}_\mathcal{P}} V_{\mathcal{P}'} \bigcup_{\forall(\mathcal{P},\mathcal{P}'',\mathcal{D})\in\mathcal{DD}_\mathcal{P}} \{\mathcal{D}\}.$$

This equation indicates that a virtual execution environment $V_\mathcal{P}$ consists of data from its preceding environment and the activities in the partition $\mathcal{P}$. Clearly, the following property holds:

$$\exists\ (\mathcal{P}', \mathcal{P}, \mathcal{D}) \in \mathcal{DD}_\mathcal{P} \iff V_{\mathcal{P}'} \subset V_\mathcal{P}. \quad (1)$$

Another benefit of using VSEE is the fact that specifying many data input and output ports for activities are time-consuming and error-prone. With this technique, users can assume that activities have one, single, aggregated data dependency to its predecessors' activities, which eliminates the need to explicitly specify all fine grained, logical data ports. This technique therefore shields users from the complexity of the workflow definition and gives users a friendly interface to the clouds. The VSEE mechanism can also reduce the overhead of activity migration defined in the dynamic optimization process.

Upon executing a workflow partition on a cloud site, each slave engine automatically creates and removes one working directory that represents its execution environment. The VSEE mechanism transforms the complex data dependencies between activities to one environment dependency between partitions. At runtime, the engine packs the data environment that can be transferred by one, single file-transfer. VSEE, therefore, noticeably reduces the latency and the number of intermediate data transfers for compute-intensive applications that have large amounts of small sized data dependencies.

TABLE I
VSEE RELATIONSHIPS.

R_V	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_{out}
V_{in}	$\subset$	$\subset$	$\subset$	$\subset$	$\subset$	$\subset$		
V_1	–	$\subset$	$\subset$	$\subset$	$\subset$	$\subset$	$\subset$	
V_2		–		$\subset$	$\subset$	$\subset$		
V_3			–	$\subset$	$\subset$	$\subset$		
V_4				–	$\subset$	$\subset$	$\subset$	
V_5					–		$\subset$	
V_6						–	$\subset$	
V_7	$\subset$						–	$\supset$

Fig. 5 illustrates one real-world workflow – WIEN2k [14] – to be executed on three sites $\{M_1, M_2, M_3\}$. WIEN2k is a program package for performing electronic structure calculations of solids using density functional theory, based on the full-potential (linearised) augmented plane-wave ((L)APW) and local orbitals (lo) method. The various programs that compose the WIEN2k package are organized in a workflow. The *lapw1* and *lapw2* tasks can be solved in parallel by a fixed number of so-called *k-points*. A final task applied on several output files tests whether the problem convergence criterion is fulfilled. The number of recursive loops is unknown at initialization.

First, the WIEN2k workflow is split into seven partitions $\mathcal{PS}_\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_7\}$ (see Fig. 5(a)) based on the algorithm presented in Section II-C and II-D. Thereafter, the data flow between partitions is optimized according to the VSEE-based relationships depicted in Table I. Therefore, we can minimizing the data transfers between partitions by aggregating data dependencies:

1) $\mathcal{P}_1$: $V_1 = V_{in} \Rightarrow V_{in}$
2) $\mathcal{P}_2$: $V_2 = (V_{in} \cup V_1) \wedge (V_1 = V_2) \Rightarrow No\ Transfer$
3) $\mathcal{P}_3$: $V_3 = (V_{in} \cup V_1) \wedge (V_{in} \subset V_1) \Rightarrow V_1$
4) $\mathcal{P}_4$: $V_4 = (V_{in} \cup V_1 \cup V_2 \cup V_3) \wedge (V_{in} \subset V_1 \subset V_2 \subset V_3) \Rightarrow V_3$
5) $\mathcal{P}_5$: $V_5 = (V_{in} \cup V_1 \cup V_2 \cup V_3 \cup V_4) \wedge (V_{in} \subset V_1 \subset V_2 \subset V_3 \subset V_4) \Rightarrow V_4$
6) $\mathcal{P}_6$: $V_6 = (V_{in} \cup V_1 \cup V_2 \cup V_3 \cup V_4) \wedge (V_{in} \subset V_1 \subset V_2 \subset V_3 \subset V_4) \Rightarrow V_4$
7) $\mathcal{P}_7$: $V_7 = (V_1 \cup V_4 \cup V_5 \cup V_6) \wedge (V_1 \subset V_4 \subset V_5) \wedge (V_6 = V_7) \Rightarrow V_5$

For example, transferring data between partitions only according to the data flow dependencies requires $\mathcal{P}_6$ receive data from $V_{in} \cup V_1 \cup V_2 \cup V_3 \cup V_4 = V_4$, since $V_{in} \subset V_1 \subset V_2 \subset V_3 \subset V_4$. Table II displays the final result of this VSEE data flow optimization process that was automatically performed by Workflow Executor. Fig. 5(b) shows the optimized data flow and only 6 data transfers required for this case.

Algorithm 4 shows the pseudo-code of the VSEE data flow optimization algorithm. In Phase 1 (step 4-10, Algorithm 4), we initialize the environment transfer set $E_\mathcal{P}$, and all the data needed to be transferred will be included in this set. In Phase 2 (step 12-22, Algorithm 4), $E_\mathcal{P}$ is minimized based on the data dependencies and the partition relations that follows the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing

TABLE II
MINIMUM VSEE TRANSFER SET.

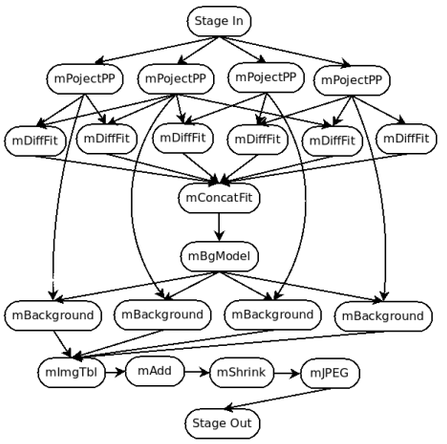
Transfer	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$	*Output*
V_{in}	✓							
V_1			✓					
V_2								
V_3				✓				
V_4					✓	✓		
V_5							✓	
V_6								
V_7	✓							✓

TABLE III
THE HYBRID CLOUD TESTBED.

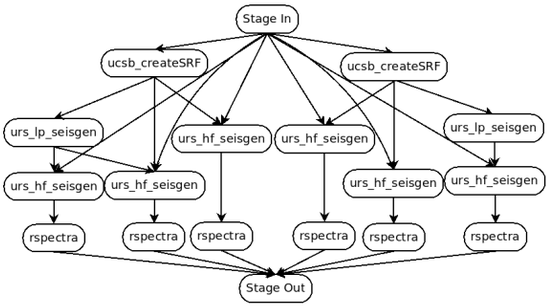
Site	*#node, #CPU*	*GHz*	*CPU*	*Manager*	*Memory(Total)*
Mercury	887,1774	1.3, 1.5	Itanium 2	PBS	4-12GB (4.5TB)
Abe	1200, 9600	2.33	Intel 64	PBS	1GB (9.6TB)
Private cloud	16	2.4	Xeon 2	PBS	2GB (8GB)

property defined in Eq. 1.

For certain compute-intensive applications characterized by large numbers of small data dependencies, the VSEE mechanism can drastically decrease the number of file-transfers – up to several orders of magnitude – while increasing the data size by several factors. Considering fixed latencies of about 1 second for every individual file-transfer and the improved bandwidth utilization for transferring large data archives, this mechanism can significantly improve the communication time in scientific workflows.



(a) Montage.



(b) Broadband.

Fig. 6. Real world workflows.

IV. EXPERIMENTAL RESULTS

In this section, we evaluate our proposed methods using Montage and Broadband. We executed these applications on a subset testbed of the TeraGrid infrastructure consisting of a set of parallel computers and workstation networks accessible through the Globus toolkit and local job queuing systems as separate sites. For the sake of clarity, our experimental testbed consists of three sites: Mercury, Abe, and a private cloud. We used 8 processors on each site. The characteristics of the machines are given in Table III.

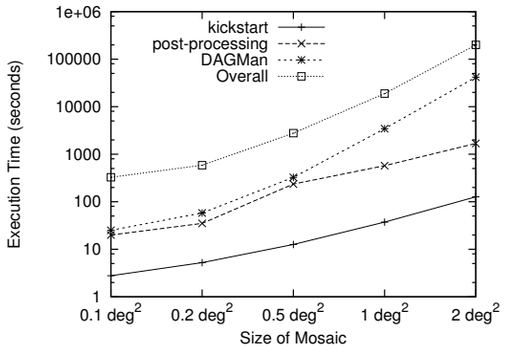
We used a resource provisioning tool called Corral in WfMS to start Glideins [24] and to help improve the performance of workflow applications. Glidein is a mechanism by which Condor workers (the program condor glidein) are submitted as user jobs to a remote cluster and used to add a compute node to a local Condor pool. The glideins are configured to contact the local Condor pool controlled by the user where they can be used to execute the user's jobs on the remote resources. This Condor pool can contain resources from multiple sites. The use of glideins can significantly improve the performance by reusing the resources once they are allocated, and by eliminating many of middleware overheads that occur when jobs are submitted using traditional grid mechanisms.

Montage [5] was created by the NASA/IPAC Infrared Science Archive as an open source toolkit that can be used to generate custom mosaics of the sky using input images in the Flexible Image Transport System (FITS) format. During the production of the final mosaic, the geometry of the output is calculated from the geometry of the input images. The inputs are reprojected by mProjectPP to be of the same spatial scale and rotation. The background emissions in the images are then corrected to be of the same level in all images. The re-projected, corrected images are co-added to form the final mosaic. Fig. 6(a) displays a relative small Montage workflow of the mosaic size of 0.1 deg^2.

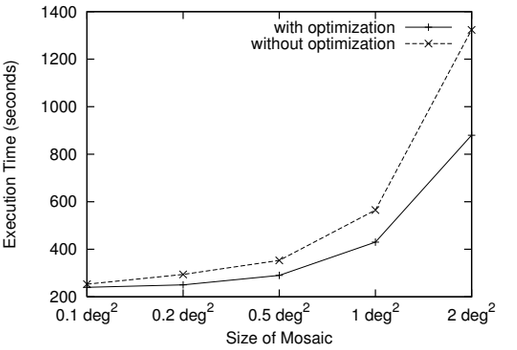
Broadband is an application developed by the Southern California Earthquake Center (SCEC) to produce more accurate seismic hazard maps. These maps, generated as part of the SCEC CyberShake project, indicate the maximum amount of shaking expected at a particular geographic location over a certain period of time. The hazard maps are used by civil engineers to determine building design tolerances. Fig. 6(b) shows a Broadband workflow for a single source, stations and velocity file.

We conducted the Montage experiments using 5 different size mosaics: 0.1 deg^2, 0.2 deg^2, 0.5 deg^2, 1 deg^2, and 2 deg^2. Fig. 7(a) displays the accumulated execution time used by kickstart, post-processing and DAGMan. As expected, the overheads increase as the number of tasks increases. As shown in Fig. 7, the sum of all these overheads is about 10% of the overall execution time (or makespan) for all problem size, which shows that it has great potential for further improvement. The overheads can be minimized if we can decrease the number of submission jobs and the complexity of workflow structure. Fig. 7(b) shows the workflow makespans

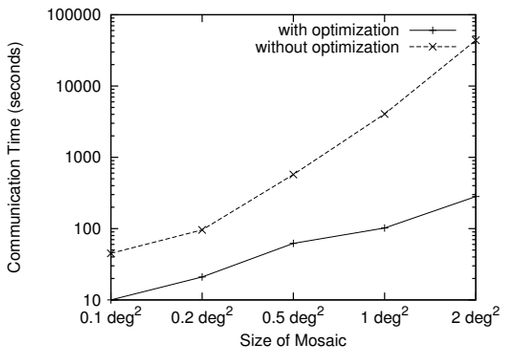
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing



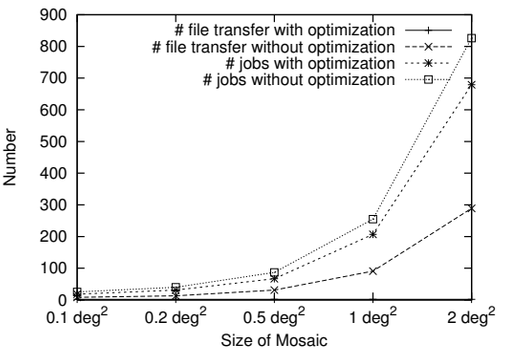
(a) Montage Accumulated Overhead.



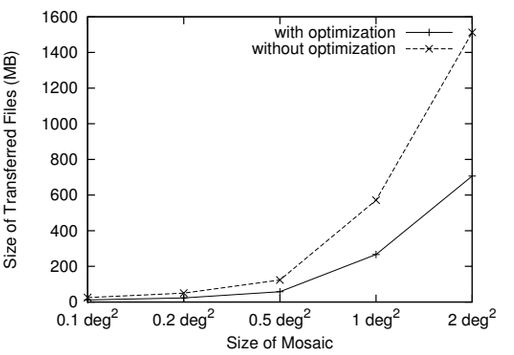
(b) Montage Makespan.



(c) Montage Data Transfer Overhead.



(d) Montage Data Transfer Number.



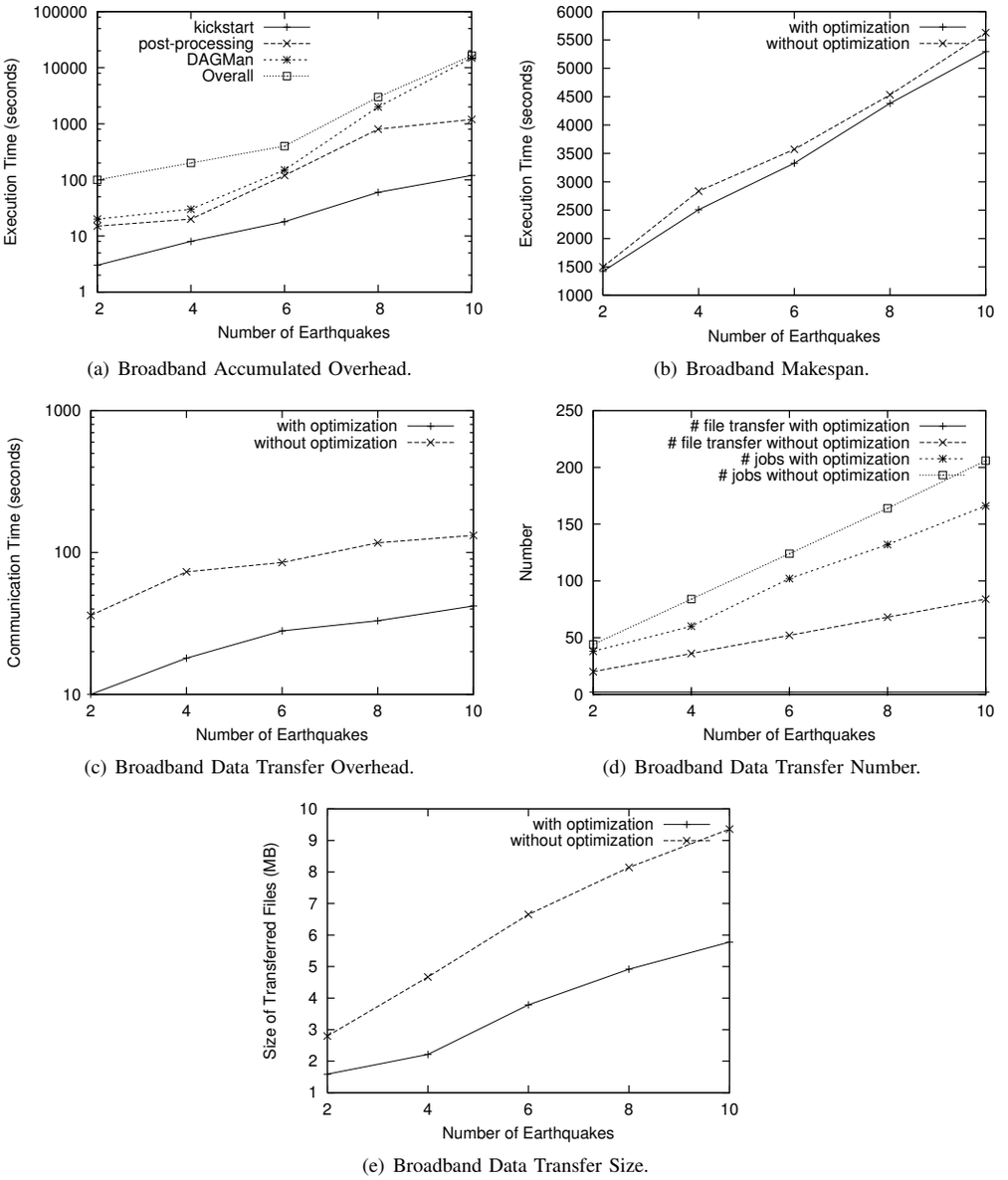
(e) Montage Data Transfer Size.

Fig. 7. Montage Experimental Results.

with and without data transfer clustering optimization. The makepans include the overheads presented in Fig. 7(a). We gain more and more performance as the problem size increased. In other words, the gaps between makespans with and without data flow optimization increased as the number of tasks increased. When the Montage problem size is 2 deg^2, the makespan considerably decreases from 1300 seconds without optimization to 880 seconds with optimization. Fig. 7(c) compares the data transfer overheads with and without data flow optimization mechanism we proposed. Since there are more new data transfers when the problem size increases, it is necessary to analyze and simplify the data flow. Figures 7(d) and 7(e) show that the number and size of file transfers are considerably reduced when data flow optimization is applied which gives more details about the optimization on this application. Not only is the number of data transfers reduced to a very small number, but the number of jobs is reduced. For compute intensive applications like Montage characterized by large numbers of small data dependencies, our methods can drastically decrease the number of data transfers up to several orders of magnitude while decrease the data size by several factors. After we archive and compress the files, the size of transfers is reduced to half of its original value.

The results help us understand the effectiveness of optimization. We define the workflow data transfer time: Time = Number * Overhead + Size / Bandwidth, where Overhead is the overheads from GridFTP service and network, and Bandwidth is average network bandwidth. As the bandwidth growing up and system overheads increasing due to middleware growing complex, the number of data transfers will dominate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing



(a) Broadband Accumulated Overhead.

(b) Broadband Makespan.

(c) Broadband Data Transfer Overhead.

(d) Broadband Data Transfer Number.

(e) Broadband Data Transfer Size.

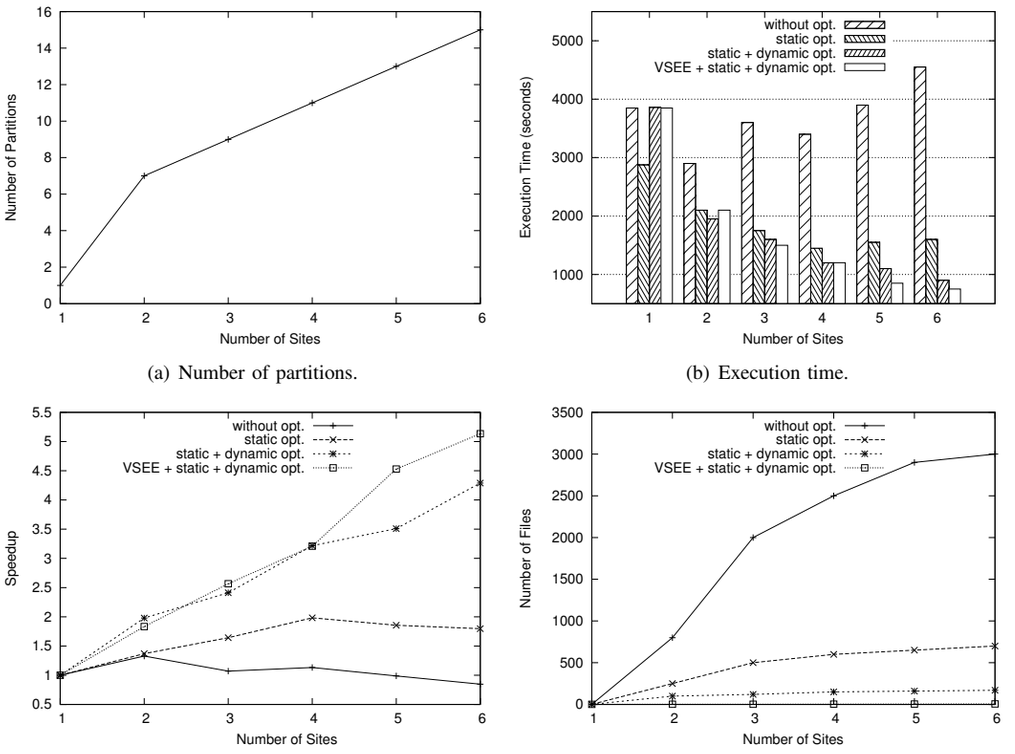
Fig. 8. Broadband Experimental Results.

the communication time of intermediate data. The data flow optimization for compute-intensive application becomes more important. In practice, intensive invokes of GridFTP service formulate denial-of-service attack to some weak sites. When these problems happen, the data transfer tasks keep receiving FAILED events from GridFTP services, and the workflow execution is stuck.

Moreover, there is a tradeoff between the number of data transfers and the size of compressed files. The number-size tradeoff is a situation where WfMS can reduce the number of data transfers at the cost of less parallelism, or can reduce the size of compressed files at the cost of more communication overhead. Obviously, it is beneficial to reduce the number of files for compute-intensive applications like Montage and Broadband, and reduce the data size for data intensive applications. Lots of data transfers are the main source of Montage's overhead. From Fig. 7, we know that the more tasks Montage has, the more benefits we can gain from the optimization.

However, for Broadband, the results obtained from our data flow optimization are not as good as Montage. We run Broadband workflows that involve 2's to 10's of sources and 2's of stations. There are several reasons behind why we cannot gain as much performance from this application. Compared to accumulated overheads shown in Fig. 8(a), the overheads are still about 10%. Although Broadband does not benefit notably from the data flow optimization, other overheads are reduced as shown in Fig. 8(b). Fig. 8(c) compares the data transfer overheads with and without optimization mechanism we proposed. The number of Broadband intermediate files is rather small, as shown in Fig. 8(d). The results displayed in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing



(a) Number of partitions.

(b) Execution time.

(c) Speedup.

(d) Number of file transfers.

Fig. 9. WIEN2k Experimental results.

Fig. 8(e) show that the total sizes of intermediate data are less than 10 MB, which shows that the communication overhead of this application is much smaller than Montage.

Fig. 9(a) presents the number of WIEN2k partitions computed by the partitioning algorithm for each site. The number of partitions depends on the workflow structure and the original schedule. This number is proportional to the number of sites. Fig. 9(b) shows the execution times for running the same WIEN2k problem on different size configurations ranging from one to six aggregated sites. Similarly, Fig. 9(c) displays the speedup computed as the ratio between execution time on multiple distributed sites and the execution time on the *fastest* local site available. First, without any optimization the performance and the speedup decrease with the increase in the number of sites used for scheduling and running the workflow. With static and dynamic optimization, the WIEN2k execution time improves because of the simplified data-flow and balanced execution of the *lapw1* and *lapw2* parallel activities. We exhibit a slow down from five to six sites using static optimization because of the increased communication time across six distributed sites. Figure 9(d) shows that the number of file transfers are considerably reduced when optimization is applied, which explains the performance results obtained.

V. Related Work

Workflow management system is concerned with the automation of processes in which tasks are structured based on their control and data dependency. There are some existing workflow systems, such as Condor/DAGMan/Stork set [22], GridFlow [6], Gridbus [4], Triana [24], Pegasus [3], ASKALON [15], etc. Most workflow management systems support DAG-based structure.

The Directed Acyclic Graph Manager (DAGMan) developed by the Condor project allows scheduling of workflow applications using opportunistic techniques such as matchmaking based on resource offers, resource requests, and cycle stealing without support for advanced optimization heuristics. Condor was designed to harvest CPU cycles on idle machines, but running data intensive workflows with DAGMan is very inefficient. However, our new data flow optimization is more fine-grained and only transfers the intermediate data needed by unexecuted tasks. Pegasus [11] optimizes disk usage and runtime performance by removing data files when they're no longer required. It pursues different purpose from us. UNICORE [17] provides manual scheduling support by allowing the user to allocate resources and to specify data transfer tasks through a graphical user interface. Our approach can automatically map the data between its source and sink, and can automatically optimize the data movement. Scheduling in Taverna [16] in ^{my}Grid and Triana [12] is done using just-in-time. Triana provides a visual programming interface with functionality represented by units. Triana clients such as Triana GUI can log into a Triana Controlling Service (TCS) to remotely build and run a workflow, and then visualize the result on their devices. Our system supports the hybrid clouds and is extensible to other public cloud services. The scheduler in Gridbus provides just-in-time mappings using

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2378790, IEEE Transactions on Cloud Computing

Grid economy mechanisms. It makes scheduling decisions on where to place jobs on the Grid and cloud depending on the computational resources characteristics and users' Quality of Service (QoS) requirements. Our system supports both static and dynamic resource allocation. ASKALON is a Grid application development and computing environment developed by the University of Innsbruck, Austria. The main objective of ASKALON is to simplify the development and optimization of most workflow applications that can harness the power of distributed computing. ASKALON provides a new hybrid approach for scheduling workflow applications through dynamic monitoring and steering combined with a static optimization. The main difference between ASKALON and our system is that they used a different scheduling and optimization method.

Other research in performance-oriented distributed computing focused on partitioning algorithms for system-level load balancing or task scheduling [8] that aims to maximize the overall throughput or to minimize response time of the system. However, our new partition and optimization algorithms are designed for large-scale workflows and use data and task replication mechanism, which can obviously improve performance and require more analysis of data flow.

In terms of multiple objective scheduling algorithms, an interesting work has been presented in [10], [25]. Ramakrishnan et al. solved this problem by a two-fold approach: 1) minimize the amount of space that a workflow requires during execution by removing data files at runtime, and 2) schedule workflows in a way that assures the amount of data required and generated by the workflows fits into individual resources. They did not consider replication mechanism as well. The algorithm in [25] introduces economic cost as a part of the objective function for data and computation scheduling, but it does not consider communication optimization problem.

Dryad [18], [26] is a distributed execution engine for coarse-grain data-parallel applications, which shows the benefits of allowing applications to perform automatic dynamic refinement of graphs/workflows. Some approaches used in Dryad and our work are similar, However, Dryad does not have a comprehensive partitioning algorithm that allows more flexible and stable execution in a distributed computing environment. Dryad is used for a wide variety of applications like relational queries, matrix computations, and text processing tasks. Our work mainly focus on large-scale scientific workflows.

CIEL [19] provides a distributed execution engine that supports dynamic control flow, fault tolerance, locality-based scheduling, etc. However, CIEL's scheduler only uses a multiple-queue-based method to schedule tasks. Our partitioning and scheduling algorithms are more complicated and suitable for large-scale applications with more complex data dependencies.

Workflow partitioning is a typical sub-graph problem in graph theory. For different objectives, we can partition a workflow with different criteria. Most other existing algorithms can simply split or merge a workflow for one purpose. There are two differences with our approach. First, we consider a two-stage partitioning on workflows, which improves both parallelism and performance. Second, we successfully solved the cross dependencies and performance optimization problem with big data and task replication, which are not solved by other algorithms.

VI. Conclusions and Future Work

The execution of workflow applications in distributed computing environments like clouds, grids or hybrid clouds becomes more and more data centric. The complexity of scientific workflows is expected to continue growing. Workflow scheduling and optimization face new and difficult challenges.

In this paper, we proposed a two-stage partitioning algorithm to refine workflow structure that offers higher performance and lower communication overheads. Intermediate data transfers can be one of the most severe overheads in heterogeneous computing environments, which can be reduced through grouping and simplifying transfers. Our approach improves the execution of large-scale workflows on multiple, geographically distributed sites through data flow optimization. This optimization simplifies workflow structure prior to the workflow execution by reorganizing data flow dependencies, significantly reduces data transfer latencies exhibited in computing environments, and reduces the overall communication overheads. The workflow optimization handles peculiar scientific workflows that change their graph-based structure at runtime. Our approach replaces large numbers of data dependencies between individual tasks with two-phase data movement between partitions, which significantly decreases data transfer complexity of compute-intensive workflow applications. We have validated our methods for three real-world workflow applications executed in a hybrid cloud testbed.

The static optimization simplifies the workflow structure prior to the workflow execution by archiving and compressing multiple files and by merging multiple atomic activities scheduled on the same site. This optimization significantly reduces the huge service-latencies exhibited in hybrid clouds. The dynamic workflow-optimization handles peculiar scientific workflows that dynamically change their graph-based structure at runtime; it also eliminates eventual load-imbalance by using idle and redundant resources during executions that do not follow the original plan computed by the scheduler. In addition, we proposed a new optimization approach, called VSEE, that replaces a large number of data dependencies between individual activities within the entire I/O data environment. This optimization significantly reduces data transfer complexity of compute-intensive workflow applications.

In the future, we plan to explore new methods to dynamically tune the performance with data replication and task replication at run time. Performance can always suffer from unexpected events or resource contention in distributed environment. Moreover, task replications can be replaced with data replications in some cases and vice versa. Also in this work, we assume that users have unlimited budget and the goal is minimize the makespan while maintaining the scale of scheduled cloud service. Our next step is to optimize workflow performance under a budget constraint.

References

[1] LIGO Data Grid. http://www.lsc-group.phys.uwm.edu/lscdatagrid.

[2] The TeraGrid Project, 2006. http://www.teragrid.org.

[3] Jim Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, and Karan Vahi. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 2005.

[4] Rajkumar Buyya and Srikumar Venugopal. The gridbus toolkit for service oriented grid and utility computing: An overview and status report. In *Proceedings of the First IEEE International Workshop on Grid Economics and Business Models*, pages 19–36, New Jersey, USA, April 23 2004. IEEE Press.

[5] CalTech. Montage: An astronomical image mosaic engine. http://montage.ipac.caltech.edu.

[6] Junwei Cao, Stephen A. Jarvis, Subhash Saini, and Graham R. Nudd. GridFlow: workflow management for grid computing. In *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid 2003))*, Tokyo, Japan, May 2003. IEEE Computer Sociery Press.

[7] Southern California Earthquake Center. Cybershake project. http://scec.usc.edu/research/cme/groups/cybershake.

[8] Sumir Chandra and Manish Parashar. Towards autonomic application-sensitive partitioning for samr applications. *J. Parallel Distrib. Comput.*, 65(4):519–531, April 2005.

[9] Rubing Duan, Radu Prodan, and Thomas Fahringer. Performance and cost optimization for multiple large-scale grid workflow applications. In *SC'07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2007. ACM Press.

[10] Arun Ramakrishnan et al. Scheduling data-intensive workflows onto storage-constrained distributed resources. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid 2007 (CCGrid 2007)*, 2007.

[11] Gurmeet Singh et al. Optimizing workflow data footprint. *Scientific Programming*, 15(4):249–268, December 2007.

[12] Ian Taylor et al. Triana applications within grid computing and peer to peer environments. *Journal of Grid Computing*, 1(2):199–217, 2003.

[13] Muthucumaru Maheswaran et al. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.

[14] Peter Blaha et al. *WIEN2k: An augmented plane wave plus local orbitals program for calculating crystal properties*. Institute of Physical and Theoretical Chemistry, Vienna University of Technology, Vienna., 2001.

[15] Thomas Fahringer et al. ASKALON: a grid application development and computing environment. In *6th International Workshop on Grid Computing (Grid 2005)*, Seattle, Washington, USA, November 2005.

[16] Tom Oinn et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[17] Unicore Forum. UNICORE: Uniform Interface to Computing Resources. http://www.unicore.eu.

[18] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007.

[19] Derek Gordon Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy, and Steven Hand. Ciel: A universal execution engine for distributed data-flow computing. In *NSDI*, volume 11, pages 9–23, 2011.

[20] Jun Qin and Thomas Fahringer. Advanced Data Flow Support for Scientific Grid Workflow Applications. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (Supercomputing 2007, SC—07)*, Reno, NV, USA, November 2007. IEEE Computer Society.

[21] Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields. *Workflows for e-Science: scientific workflows for grids*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[22] The Condor Team. Dagman (directed acyclic graph manager). http://www.cs.wisc.edu/condor/dagman/.

[23] Haluk Topcuouglu, Salim Hariri, and Min you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274, 2002.

[24] The Triana project. http://www.trianacode.org.

[25] Srikumar Venugopal and Rajkumar Buyya. A Deadline and Budget Constrained Scheduling Algorithm for e-Science Applications on Data Grids. In *Proceedings of the 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2005)*, volume 3719, Melbourne, Australia., October 2005.

[26] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI*, volume 8, pages 1–14, 2008.

Rubing Duan received his Ph.D. in 2008 from the Institute of Computer Science, University of Innsbruck, Austria. Duan is currently a scientist at the A*STAR Institute of High Performance Computing (IHPC), Singapore. He is interested in cloud computing, big data analytics, distributed software architectures, performance analysis and optimization, and scheduling for parallel and cloud computing. Duan participated in several national and Singapore projects. He is the author of over 20 journal and conference papers.

Rick Siow Mong Goh is the Director of the Computing Science Department at the A*STAR Institute of High Performance Computing (IHPC). At IHPC, he leads a team of more than 50 scientists in performing world-leading scientific research, developing technologies to commercialization, and engaging and collaborating with industry. The research focus areas include high performance computing (HPC), distributed computing, big data analytics, intuitive interaction technologies, and complex systems. His expertise is in discrete event simulation, parallel and distributed computing, and performance optimization and tuning of applications on large-scale computing platforms. Dr. Goh received his Ph.D. in Electrical and Computer Engineering from the National University of Singapore. More details on Rick can be found at: http://ihpc.a-star.edu.sg/gohsm.php?display=1

Zheng Qin is the Manager of the Distributed Computing Capability Group under the Computing Science Department at the A*STAR Institute of High Performance Computing (IHPC). His research interests include cloud computing, big data analytics, and smart cities. He received his Ph.D. in Electrical and Computer Engineering from the National University of Singapore and his B.Eng. in Information Engineering from the Xian Jiaotong University. More details on him can be found at: http://www.ihpc.a-star.edu.sg/qinz.php?display=1

Yong Liu received his B.Eng degree from Hunan University, China in 1997, M.Eng. from Hunan University, China in 2000, and PhD degree from National University of Singapore (NUS) in 2006. He has worked as a Research Fellow in National University of Singapore (NUS). He is currently working as Scientist at A*STAR, Institute of High Performance Computing (IHPC). His research interests include distributed computing, big data analytics, machine learning, network calculus, and traffic engineering in computer networks. He has co-authored two books in the area of computer networks.