WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments

Weiwei Chen
Information Sciences Institute
University of Southern California
Marina del Rey, CA, USA
E-mail: wchen@isi.edu

Ewa Deelman
Information Sciences Institute
University of Southern California
Marina del Rey, CA, USA
E-mail: deelman@isi.edu

Abstract—**Simulation is one of the most popular evaluation methods in scientific workflow studies. However, existing workflow simulators fail to provide a framework that takes into consideration heterogeneous system overheads and failures. They also lack the support for widely used workflow optimization techniques such as task clustering. In this paper, we introduce WorkflowSim, which extends the existing CloudSim simulator by providing a higher layer of workflow management. We also indicate that to ignore system overheads and failures in simulating scientific workflows could cause significant inaccuracies in the predicted workflow runtime. To further validate its value in promoting other research work, we introduce two promising research areas for which WorkflowSim provides a unique and effective evaluation platform.**

Keywords: workflow; clustering; overhead; failure; simulation.

I. INTRODUCTION

Over the years, scientific workflows have emerged as a paradigm for representing complex distributed scientific computations. Scientific workflows can be composed of a large number of tasks and the execution of these tasks may require many complex modules and software. The evaluation of the performance of workflow optimization techniques in real infrastructures is complex and time consuming. As a result, simulation-based studies have become a widely accepted way to evaluate workflow systems. For example, scheduling algorithms, such as HEFT [3], Min-Min [1], Max-Min [2], etc., have used simulators to evaluate their effectiveness. A simulation-based approach reduces the complexity of the experimental setup and saves much effort in workflow execution by enabling the testing of their applications in a repeatable and controlled environment.

However, an accurate simulation framework for scientific workflows is required to generate reasonable results, particularly considering that the overall system overhead [4] plays a significant role in the workflow's runtime. In heterogeneous distributed systems, workflows may experience different types of overheads, which are defined as the time of performing miscellaneous work other than executing users' computational activities. Since the causes of overheads differ, the overheads have diverse distributions and behaviors. For example, the time to run a post-script that checks the return status of a computation is usually a constant. However, queue delays incurred while tasks are waiting in a batch scheduling systems can vary widely. By classifying these workflow overheads in different layers and system components, a simulator can offer a more accurate result than simulators that do not include overheads in their system models.

What's more, many researchers [5][6][7][8][9][10] have emphasized the importance of fault tolerant design and concluded that the failure rates in modern distributed systems should not be neglected. A simulation with support for randomization and layered failures is required to promote such studies.

Finally, progress in workflow research also requires a general-purpose framework that can support widely accepted features of workflows and optimization techniques. Existing simulators such as CloudSim/GridSim[12] fail to provide fine granularity simulations of workflows. For example, they lack the support of task clustering, which is a popular technique that merges small tasks into a large job to reduce task execution overheads. The simulation of task clustering requires two layers of execution model, on both task and job levels. It also requires a workflow-clustering engine that launches algorithms and heuristics to cluster tasks. Other techniques such as workflow partitioning and task retry are also ignored in these simulators.

To the best of our knowledge, none of the current distributed system simulators support these rich-features and techniques. In this paper, we introduce our early work on simulating scientific workflows satisfying these requirements. We evaluate the performance of WorkflowSim with an example of task clustering. We further show that WorkflowSim is promising in providing an evaluation platform for research areas such as fault tolerant clustering and overhead robustness studies.

The contribution of this paper includes an introduction of our WorkflowSim framework and how it distinguishes from other simulators in Section III. We also validate the performance of WorkflowSim with an example of task clustering in Section IV. Finally, in Section V, we discuss the promising usage of this simulation framework. The conclusion and future directions are presented in Section VI.

II. Related Work

Deelman et. al. [13] have provided a survey of popular workflow systems for scientific applications and have classified their components into four categories: composition, mapping, execution, and provenance. Based on this survey, we identified the mandatory functionalities/components and designed the layers in our WorkflowSim. In our design, we add multiple layers on top of the existing workflow scheduling layer of CloudSim, which include the Workflow Mapper, the Workflow Engine, the Clustering Engine, the Failure Generator, the Failure Monitor etc. We will explain the details of these layers in Section III.

CloudSim [12] is a popular framework for modeling and simulating cloud computing infrastructures and services. However, it only supports the execution of single workloads while our work focuses on workflow scheduling and execution. It has a simple model of task execution that does not consider task dependencies or clustering. It also ignores the occurrence of failures and overheads. WorkflowSim extends CloudSim to fulfill these new requirements. Other simulators [14][15] were specifically designed for a few desirable aspects of workflow management such as workflow scheduling, but such simplification does not match the ever-changing world of distributed computing and the evolution of new workflow management techniques. Therefore, rather than focusing on a specific aspect of workflow management, WorkflowSim attempts to extract the common features exposed by various workflow systems and to support widely used workflow management techniques. WorkflowSim supports not only the evaluation of scheduling techniques but also considers diverse task scheduling/execution overheads and failures.

Task clustering [16] is a technique that merges fined-grained tasks into coarse-grained jobs. After task clustering the number of jobs to be executed in a workflow is reduced and the cumulative workflow execution overhead is reduced as well. However, the paper's clustering strategy is static and does not consider dynamic resource characteristics. Also, it does not consider the middleware overhead from diverse grid middleware services, such as the time to query resources, and the time to match jobs with resources, etc. These overheads are included in our model and the values are estimated based on real execution traces.

Failure analysis and modeling [6] present system characteristics such as error and failure distribution and hazard rates. Schroeder et al. [7] have studied the statistics of the execution failure data, including the root cause of failures, the mean time between failures, and the mean time to repair. Sahoo et al. [8] analyzed the empirical and statistical properties of system errors and failures from a network of heterogeneous servers running a diverse workload. Benoit et al. [11] analyzed the impact of transient and fail-stop failures on the complexity of task graph scheduling. Among all the failures, we focus on transient failures because they are expected to be more prevalent than permanent ones. Based on these works, we simulate failures in two layers (task/job) and provide an interface for users to develop fault tolerant algorithms. In section V we will introduce our work in fault tolerant clustering.

III. Models and Features

As Figure 1 shows, there are multiple layers of components involved in preparing and executing a workflow. In this paper, the model of workflow management systems (WMS) is similar to that of Pegasus WMS [17], which contains: a Workflow Mapper to map abstract workflows to concrete workflows that are dependent on execution sites; a Workflow Engine to handle the data dependencies; and a Workflow Scheduler to match jobs to resources. Other components include a Clustering Engine that merges small tasks into a large job, a Provenance collector that tracks the history of task/job execution and a Workflow Partitioner that divides the user workflow into multiple sub-workflows. WorkflowSim has implemented these functionalities as follows.

A. Components

1) Workflow Mapper

We model workflows as Directed Acyclic Graphs (DAGs), where jobs represent users' computation to be executed and directed edges represent data or control flow dependencies between the jobs. Figure 2 (Left) shows an example of the Montage workflow with 9 horizontal levels and 18 tasks together. Montage [20] is an astronomy application used to construct large image mosaics of the sky. All the tasks at a horizontal level of the Montage workflow are invocations of the same routing operating on different input data. For example, tasks at the first level are invocations of mProjectPP, which reprojects an input image to the scale defined by a template header file. The Workflow Mapper is used to import DAG files formatted in XML and other metadata information such as file size. After mapping, the Workflow Mapper creates a list of tasks and assigns these tasks to an execution site. A task is a program/activity that a user would like to execute.

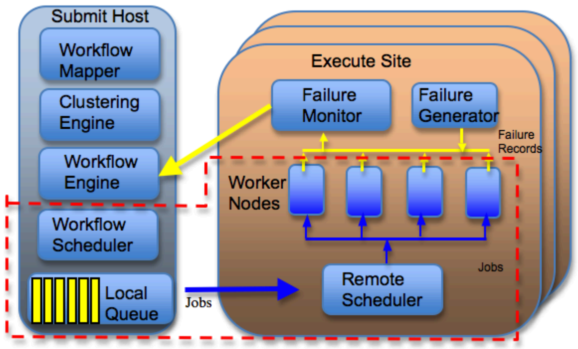

Figure 1 WorkflowSim Overview. The area surrounded by red lines is supported by CloudSim.

2) Clustering Engine

We define a task as a program that a user would like to execute. A job is an atomic unit seen by the execution system, which contains multiple tasks to be executed in sequence or in parallel. The Clustering Engine merges tasks into jobs so as to reduce the scheduling overheads. Figure 2 (Right) shows an example workflow after horizontal clustering, which merges tasks at the same horizontal levels in the original workflow graph. There are other types of clustering strategies, such as for example vertical clustering that merge tasks at the same pipeline vertically.

3) *Workflow Engine*

The Workflow Engine manages jobs based on their dependencies to assure that a job may only be released when all of its parent jobs have completed successfully. The Workflow Engine will only release free jobs to the Scheduler. In the real execution we studied, we use DAGMan [18] as the Workflow Engine.

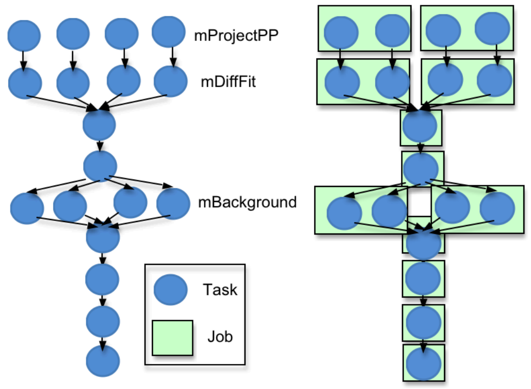

Figure 2 Original Montage Workflow (Left) and the workflow after horizontal clustering (Right).

4) *Workflow Scheduler and Job Execution*

The Workflow Scheduler is used to match jobs to worker nodes based on the criteria selected by users (MaxMin [2], MinMin [1], and many other heuristics). While CloudSim has already supported static scheduling algorithms, we added the support of dynamic workflow algorithms. For static algorithms, jobs are assigned to a worker node at the workflow planning stage. When the job reaches the remote scheduler, it will just wait until the assigned worker node is free. For dynamic algorithms, jobs are matched to a worker node in the remote scheduler whenever a worker node becomes idle. WorkflowSim relies on CloudSim to provide an accurate and reliable job-level execution model, such as time-shared model and space-shared model. However, WorkflowSim has introduced different layers of overheads and failures, which improves the accuracy of simulation.

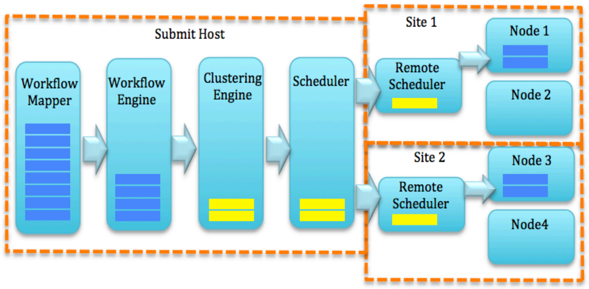

Figure 3 Interaction between components.

To associate and coordinate these layers, we adopted an event-based approach where each component maintains a message queue. Figure 3 shows a simple configuration with two execution sites, of which each has two nodes. Each component maintains its own message queue and iteratively checks whether it can process one message. For example, at each iteration, the Clustering Engine checks whether it has received new tasks from the Workflow Engine and whether it should release new jobs to the Scheduler. When none of these components have any more messages in queue, the simulation is completed.

B. *Layered Overhead*

Based on our prior studies on workflow overheads, we add layered overhead to the workflow simulation. We have classified workflow overheads into five categories as follows.

- *Workflow Engine Delay* measures the time between when the last parent job of a job completes and the time when the job gets submitted to the local queue. In case of retries the value of the last retry is used for the calculation. Since we use a DAG model to represent workflows, the completion time of the last parent job means this job is released to the ready queue and is waiting for resources to be assigned to it. The workflow engine delay reflects the efficiency of a workflow engine (in our case DAGMan).
- *Queue Delay* is defined as the time between the submission of a job by the workflow engine to the local queue and the time the local scheduler sees the job running (potentially on remote resources). This overhead reflects the efficiency of the workflow scheduler (e.g., Condor [19]) to execute a job and the availability of resources for the execution of the job. In case of retries the value is the cumulative of all the retries.
- *Postscript Delay* and *Prescript Delay* is the time taken to execute a lightweight script under some execution systems before and after the execution of a job. Prescripts are usually used to create directories for job execution. Postscripts examine the exit code of a job after the computational part of the job is done.
- *Data Transfer Delay* happens when data is transferred between nodes. It includes three different types of processes: staging data in, cleaning up, and staging data out. Stage-in jobs transfer input files from source sites to execution sites before the computation starts. Cleanup jobs delete intermediate data that is no longer needed by the remainder of the workflow. Stage-out jobs transfer workflow output data to archiving sites for storage and analysis.
- *Clustering Delay* measures the difference between the sum of the actual task runtime and the job runtime seen by the Workflow Scheduler. The cause of Clustering Delay is usually the use a job wrapper used to execute a clustered job. The wrapper takes some time to extract the list of tasks and to launch them.

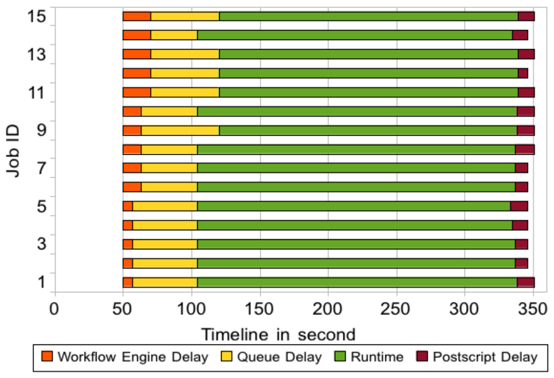

Figure 4 Workflow overhead and runtime. Clustering Delay and Data Transfer Delay are not shown (included in Runtime).

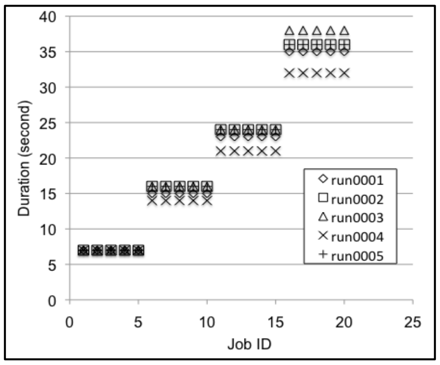

Figure 5 Workflow Engine Delay of mProjectPP.

In our prior work [4], we have introduced the distribution of overheads and the relationship between them. Following this, we indicate the necessity to consider the distribution of overheads rather than simply adding a constant delay after job execution. We use Workflow Engine Delay as an example to show the necessity to model overheads appropriately. Figure 4 shows a real trace of overheads and runtime in Montage 8 degree workflow (for scaling issues, we only show the first 15 jobs in the mProjectPP level). We can see that the Workflow Engine Delay increases steadily after every five jobs. For example, the Workflow Engine Delay of jobs with ID from 6 to 10 is approximately twice of that of jobs ranging from ID1 to ID5. Figure 5 further shows the distribution of Workflow Engine Delay of mProjectPP level in Montage workflow that was run five times. After every five jobs, the Workflow Engine Delay increases by 8 seconds approximately. We call this special feature of workflow overhead as cyclic increase. The reason is that the Workflow Engine (in this trace it is DAGMan) releases five jobs by default in every working cycle. Therefore, simply adding a constant delay after every job execution has ignored its potential influence on the performance. For this reason we adopt a message queue based approach and iteratively check the message queues in WorkflowSim.

Figure 6 shows the average value of Clustering Delay of mProjectPP, mDiffFit, and mBackground. It is clear that with the increase of k (the maximum number of jobs per horizontal level), there are less and less tasks in a clustered job, and thus the Clustering Delay for each job decreases. For simplicity, we use an inverse proportion model in Eq. (1) to describe this trend of Clustering Delay with k. Intuitively we assume that the average delay per task in a clustered job is constant (n is the number of tasks in a horizontal level). An inverse proportion model can estimate the delay when k=i directly if we have known the delay when k=j. Therefore we can predict all the clustering cases as long as we have gathered one clustering case. This capability is required in our validation in Section IV. Other complex models may require multiple clustering cases to predict all other clustering cases.

$$\frac{Clustering\ Delay|_{k=i}}{Clustering\ Delay|_{k=j}} = \frac{n/_i}{n/_j} = \frac{j}{i} \quad (1)$$

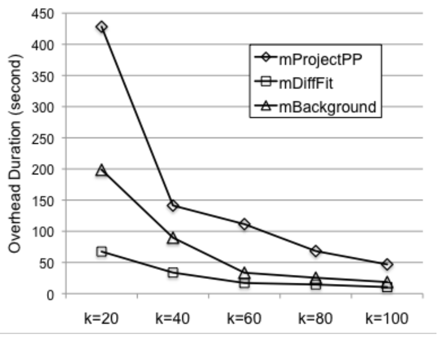

Figure 6 Clustering Delay of mProjectPP, mDiffFit, and mBackground

C. Layered Failures and Job Retry

Failures can occur at different times during the workflow execution. Consistent with the definition of tasks and job, we divide transient failures into two categories: task failure and job failure. If the transient failure affects the computation of a task (task failure), other tasks within the job do not necessarily fail. If the transient failure affects the clustered job (job failure), all of its tasks fail. We have added two components in response to the simulation of failures:

- *Failure Generator* component is introduced to inject task/job failures at each execution site. After the execution of each job, Failure Generator randomly generates task/job failures based on the distribution and average failure rate that a user has specified.
- *Failure Monitor* collects failure records (e.g., resource id, job id, task id) and returns them to the workflow management system so that it can adjust the scheduling strategies dynamically.

We also modified other components to support fault tolerant optimization. In a failure-prone environment, there are several options to improve workflow performance. First, one can simply retry the entire job or only the failed part of this job when its computation is not successful. This functionality is added to the Workflow Scheduler, which checks the status of a job and takes action based on the strategies that a user selects. Furthermore, Reclustering is a technique that we have proposed [1] that attempts to adjust the task clustering strategy based on the detected failure rate.

This functionality is added to the Workflow Engine. Details of this method are explained in Section V.A.

IV. VALIDATION

We use task clustering as an example to illustrate the necessity of introducing overheads into workflow simulation. The goal was to compare the simulated overall runtime of workflows in case the information of job runtime and system overheads are known and extracted from prior traces.

In this example, we collected real traces generated by the Pegasus Workflow Management System while executing workflows on FutureGrid [21]. We built an execution site with 20 worker nodes and we executed the Montage workflow five times in every single configuration of k, which is the maximum number of clustered jobs in a horizontal level. These five traces of workflow execution with the same k is a training set or a validation set. Partly illustrated by Figure 5, the results are stable enough to be used as a training set. We ran the Montage workflow with a size of 8-degree squares of sky. The workflow has 10,422 tasks and 57GB of overall data. We tried different k from 20 to 100, leaving us 5 groups of data sets with each group having 5 workflow traces.

First of all, we adopt a simple approach that picks up a training set to train WorkflowSim and then use the same training set as validation set to compare the predicted overall runtime and the real overall runtime in the traces. We define accuracy in this section as the ratio between the predicted overall runtime and the real overall runtime:

$$Accuracy = \frac{Predicted\ Overall\ Runtime}{Real\ Overall\ Runtime} \qquad (2)$$

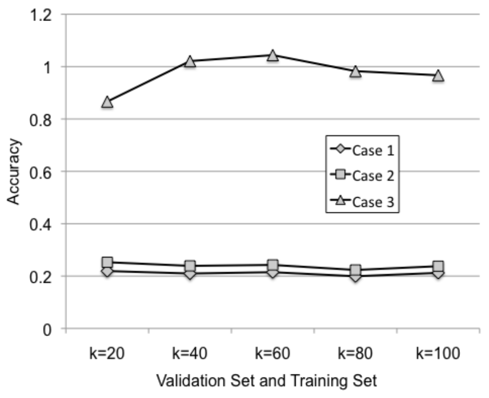

Figure 7 Performance of WorkflowSim with different support levels.

To train WorkflowSim, from the traces of workflow execution (training sets), we extracted information about job runtime and overheads, such as average/distribution and, for example, whether it has a cyclic increase shown in Figure 5. We then added these parameters into the generation of system overheads and simulated them as close as possible to the real cases. Here, we do not discuss the randomization or distribution of job runtime since we rely on CloudSim to provide a convincing model of job execution.

To present an explicit comparison, we simulated the cases using WorkflowSim that has no consideration of workflow dependencies or overheads (Case 1), WorkflowSim with Workflow Engine that has considered the influence of dependencies but ignored overheads (Case 2), and WorkflowSim, that has covered both aspects (Case 3). Intuitively speaking, we expect that the order of the accuracy of them should be Case 3 > Case 2 > Case 1.

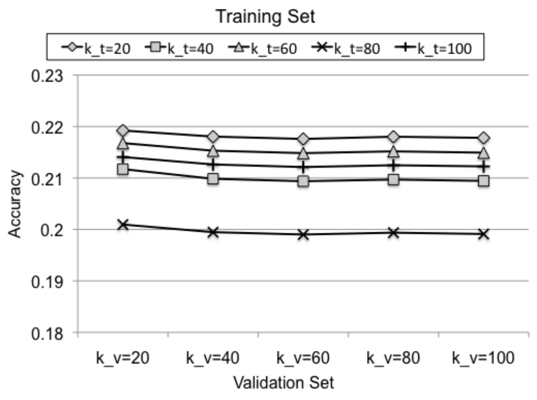

Figure 8 Performance of WorkflowSim of Case 1 (No workflow engine, or overhead support).

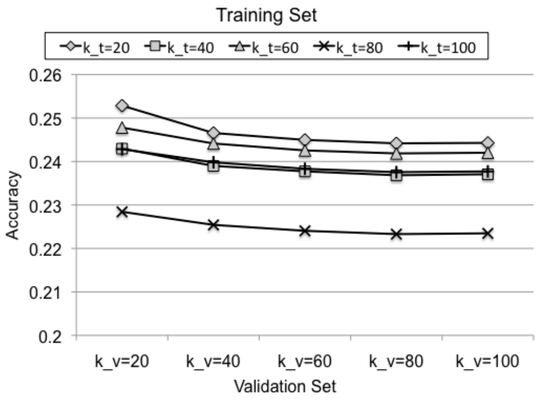

Figure 9 Performance of WorkflowSim of Case 2 (No overhead support)

Figure 7 shows the performance of WorkflowSim with different support levels is consistent to our expectation. The accuracy of Case 3 is quite close to but not equal to 1.0 in most points. The reason is that to simulate workflows, WorkflowSim has to simplify models with a few parameters, such as the average value and the distribution type. It is not efficient to recur every overhead as is present in the real traces. It is also impossible to do since the traces within the same training set may have much variance. Figure 7 also shows that the accuracy of both Case 1 and Case 2 are much lower than Case 3. The reason why Case 1 does not give an exact result is that it ignores both dependencies and multiple layers of overheads. By ignoring data dependencies, it releases tasks that are not supposed to run since their parents have not completed (a real workflow system should never do that) and thereby reducing the overall runtime. At the same time, it executes jobs/tasks irrespective of the actual overheads, which further reduces the simulated overall runtime. In Case 2, with the help of Workflow Engine, WorkflowSim is able to control the release of tasks and thereby the simulated overall runtime is closer to the real traces. However, since it has ignored most overheads, jobs are completed and returned earlier than that in real traces. The low accuracy of Case 1 and Case 2 confirms the necessity of introducing overhead design into our simulator.

To further evaluate our task/job model and the performance of WorkflowSim, we adopted a cross-validation approach in which we picked up one group of data set (e.g., *k_t*=20) as input traces/training sets and simulated all the validation sets with *k_v*=20 to 100. To make it clear, we use *k_t* to indicate the *k* for a training set and *k_v* for a validation set. Then we compare the accuracy in Figure 8, Figure 9 and Figure 10 respectively.

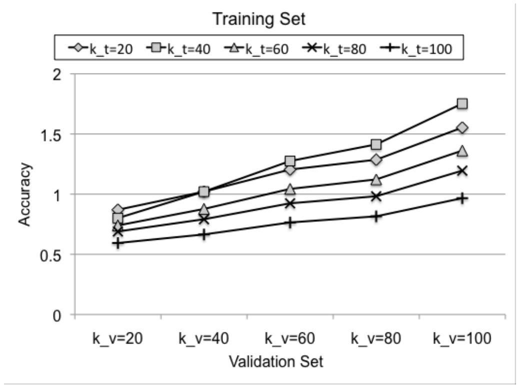

Figure 10 Performance of WorkflowSim of Case 3 (all features are enabled)

Figure 8 and Figure 9 show similar conclusion as in Figure 7 and the accuracy of Case 2 and Case 1 are not sensitive to the task clustering. The reason is that Case 1 has no support of data dependencies, where jobs are all submitted at the beginning of workflow execution. Case 2 has no support of system overhead and thereby task clustering does not improve the overall runtime much.

Figure 10 shows the simulated results of WorkflowSim, which has considered both layered overhead and data dependencies. Although the accuracy is closer to 1.0, it still does not guarantee a 100% accuracy in some cases. Particularly when we use a training set with a smaller *k* (e.g., *k_t*=20) to simulate the case with larger *k* (e.g., *k_v*=100), the accuracy suffers (accuracy=1.8). The reason is that the average Clustering Delay in the case of *k*=20 is much larger than that of other cases (as shown in Figure 6), and thereby it is still larger than the predicted one using an inverse proportion function. Using such a large Clustering Delay to simulate the case with many clustered jobs (*k_v* is large) would extend the predicted overall runtime of workflow. Our model has simplified and classified the distribution of overheads based on the horizontal level of tasks but we still need to further study the overhead distribution in accordance to different clustering strategies. However, a complex model may limit its general usage.

V. APPLICATIONS

With the features introduced in last section, we are able to carry out research studies as follows.

A. Fault Tolerant Clustering

Task clustering has been proven to be an effective method to reduce execution overhead and increase the computational granularity of workflow tasks executing on distributed resources. However, a job composed of multiple tasks may have a greater risk of suffering from failures than a job composed of a single task. In this section we use WorkflowSim to indicate that such failures can have a significant impact on the runtime performance of workflows under existing clustering policies that ignore failures. We therefore propose three dynamic methods to adjust the *clusters.size* (the maximum number of tasks in a clustered job) based on the measured failure rate: specifically, WorkflowSim provides the modeling and simulation of the layered overhead, job retry and failure generation. The three methods are described as follows. Details of the implementation are described in [1]. We use the same Montage workflow and execution environment as that in Section IV except that this is a failure-prone environment. We compare the three methods with a default method that has no optimization for faulty environments.

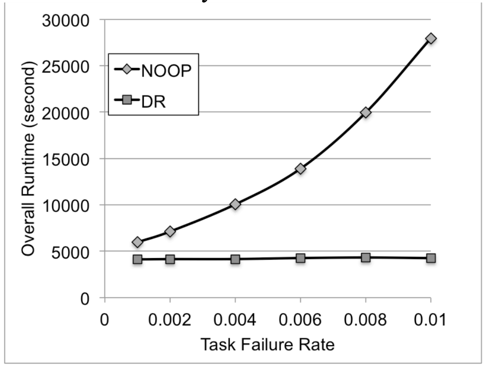

Figure 11 Performance of Fault Tolerant Clustering. NOOP uses the original job retry without further optimization. DR uses dynamic reclustering to improve the performance.

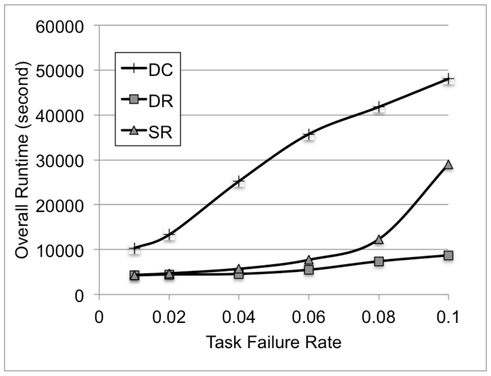

Figure 12 Performance of the three dynamic methods

- *Dynamic Clustering* (DC) decreases the *clusters.size* if the measured job failure rate is high.

- *Selective Reclustering* (SR) selects the failed tasks in a job and merges them into a new job for retry.
- *Dynamic Reclustering* (DR) selects the failed tasks in a job and also adjusts the *clusters.size* if the measured job failure rate is high. It is a combination of DC and SR.
- *No Optimization* (NOOP) retries the failed jobs without identifying whether there are successful tasks in it.

Figure 11 shows that without any fault tolerant optimization, the performance degrades significantly especially when the task failure rate is high. Figure 12 compares the three methods that we proposed and it shows that Dynamic Reclustering outperforms the other two because it derives strengths from both. In reality, it is difficult to simulate failures with precise failure rates while WorkflowSim provides a unique platform to evaluate fault tolerant designs.

B. *Overhead Robustness of DAG Scheduling Heuristics*

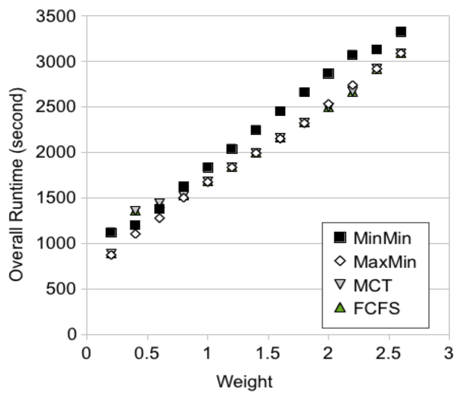

Figure 13 Influence of Queue Delay. The duration of overheads are multipled by the weights.

With the emergence of distributed heterogeneous systems, such as grids and clouds, and applications such as large scale of workflows with complex data dependencies, significant overheads can be incurred during workflow execution. Most of the existing DAG scheduling heuristics underestimate or even ignore the influence of workflow overheads. In such a distributed environment, a carefully crafted schedule based on deterministic and static information may fail to provide a sufficient solution. In this study, we analyze the overhead robustness of multiple static and dynamic DAG scheduling heuristics. Overhead robustness describes the influence of overheads on the workflow runtime. We investigate whether the dynamic change in workflow overheads influences the overall runtime of workflows. The reason why we are interested in this study is that in reality, system overheads are difficult to estimate or track. Existing heuristics and algorithms may have different sensitivity to the dynamic change of system overhead or the inaccurate estimation of them. Analyzing their performance in terms of the change of overheads can offer us a unique aspect of their robustness in real systems and suggest the direction of designing new heuristics or algorithms.

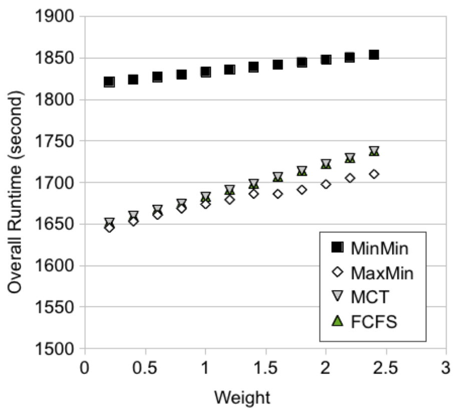

Figure 14 Influence of Workflow Engine Delay.

In this experiment, we doubled the computation capabilities of half of the available resources so as to create an environment where heuristics and algorithms can select their allocated resources to execute workflow jobs. We varied the duration of overheads by multiplying them with a weight that ranges from 0.2 to 2.5 in our experiment. The original workflow has the weight is 1.0. We evaluated the performance of four heuristics with the same Montage workflow used in Section VI:

- **FCFS***:* First Come First Serve is the basic version of scheduling algorithm used in our simulator. It assigns each job, in the arriving order to the next available resources, regardless of the jobs' expected completion time on that worker node. If there are multiple resources available, it randomly chooses one as the candidate.
- **MCT**: Minimum Completion Time [2] assigns each job in an arbitrary order to the available resource with the best expected completion time of that job.
- **MinMin**: The MinMin [1] heuristic begins with a set of all free jobs and then sorts them by the order of completion time. The job with the minimum completion time is selected and assigned to the corresponding resource. Then, the newly mapped job is submitted to the queue and the process repeats until all free jobs are scheduled. The intuition of MinMin is to create a local optimal path so as to reduce the overall runtime.
- **MaxMin**: Similar to MinMin, but MaxMin [2] picks up the job with the maximum completion time and assigns it to its best available resource. The intuition of MaxMin is to avoid penalty from long running jobs.

Experiments show that overheads have significant influence on the overall runtime and they have shown different behaviors. Figure 13 and Figure 14 show the influence of Queue Delay and Workflow Engine Delay respectively. Consistent with our expectation, MinMin performs worst compared to the other three methods since it assigns the best resources to small jobs while longer jobs have to wait and suffer overhead. MaxMin performs better than MCT and FCFS slightly because it tends to assign longer jobs to better resources and thereby reduces the overall runtime. Figure 15 shows that when the weight of Clustering Delay is lower than 1.0, MCT and FCFS perform better than MinMin. However, when the weight of Clustering Delay is larger than 2, MinMin performs better than the other two. The reason is probably because Clustering Delay only occurs to clustered jobs and in Montage these levels have better parallelism than other levels that have only non-clustered jobs. Increasing Clustering Delay thereby offers MinMin a chance to enhance its influence on the overall workflow execution. Therefore, in such an environment, the selection of heuristics is not sensitive to the estimation error of the Queue Delay or Workflow Engine Delay because the overall runtime increases at the same speed. However, the estimation error of the Clustering Delay can change the heuristics' relative performance.

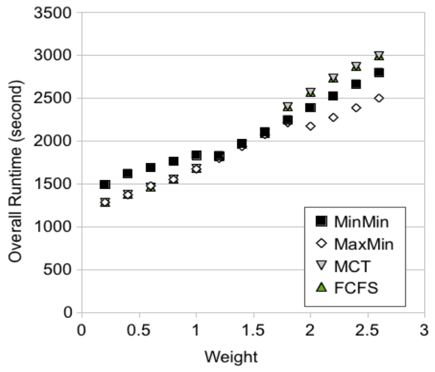

Figure 15 Influence of Clustering Delay.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have introduced a novel workflow simulator WorkflowSim to assist researchers to evaluate their workflow optimization techniques with better accuracy and wider support than existing solutions. By comparing the results of real traces and simulation, we have validated our simulator and concluded that it is necessary to consider multiple layers of overheads and failures.

In the future, we would also define more types of failures, such as the Job Submit Failure that simulates the case when a job is not successfully submitted due to a problem in workflow scheduler or a network issue between it and remote scheduler. We also plan to incorporate more workflow techniques (such as workflow partitioning) into our simulator. We will evaluate the influence of overheads in other workflow metrics besides overall runtime, for example, resource utility.

ACKNOWLEDGMENT

This work is supported by NFS under grant number IIS-0905032 and 0910812. We thank the Pegasus team and the FutureGrid team for their assistance and Gideon Juve for his valuable comments. Any opinions, findings, and conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] J. Blythe,S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task Scheduling Strategies for Workflow-Based Applications in Grids. CCGrid 2005, 2005.

[2] T. D. Braun, H. J. Siegel, N. Beck, et al. A Comparison of Eleven Static Heuristic for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computing, 61, pp. 810-837, 2001.

[3] H. Topcuoglu, S. Hariri, and M. -Y. Wu., Performance-Effectiveand Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Transactions on Parallel and Distributed Systems, 13(3), pp. 260-274, 2002.

[4] Weiwei Chen, Ewa Deelman, Workflow Overhead Analysis and Optimizations, The 6th Workshop on Workflows in Support of Large-Scale Science, Seattle, USA, Nov 2011.

[5] Y. Zhang, etc., Performance Implications of Failures in Large-Scale Cluster Scheduling, In 10th Workshop on Job Scheduling Strategies for Parallel Processing, June 2004.

[6] Dong Tang, et al., Failure Analysis and Modeling of a VAXcluster System, FTCS-20, 1990.

[7] B. Schroeder, et al., A large-scale study of failures in high-performance computing systems, DSN 2006, Philadelphia, PA, USA, Jun 2006.

[8] R. K. Sahoo, et al., Failure Data Analysis of a Large-Scale Heterogeneous Server Environment, DSN 2004, Florence, Italy, Jul 2004.

[9] David Oppenheimer, et al., Why do Internet services fail, and what can be done about it?, USITS'03, Seattle, USA, Mar 2003.

[10] S.R. McConnel, D.P. Siewiorek, and M.M. Tsao, "The Measurement and Analysis of Transient Errors in Digital Compute Systems," Proc. 9th Int. Symp. Fault-Tolerant Computing, pp. 67-70, 1979.

[11] Anne Benoit, et al., On the complexity of task graph scheduling with transient and fail-stop failures, Research report, LIP, Jan 2010

[12] Rodrigo N. Calheiros, et al., CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, Software: Practice and Experience, Volume 41, Number 1, Pages: 23-50, ISSN: 0038-0644, Wiley Press, New York, USA, January 2011.

[13] Deelman, E., et al., Workflows and e-Science: An overview of workflow system features and capabilities, Future Generation Computer Systems, July 10th, 2008.

[14] Hirales-Carbajal, A., et al., A Grid simulation framework to study advance scheduling strategies for complex workflow applications, IPDPSW, April 2010, Atlanta, GA.

[15] Merdan, M., et al., Simulation of Workflow Scheduling Strategies Using the MAST Test Management System, 10th Inrl., Conf. on Control, Automation, Robotics and Vision, Hanoi, Vietnam, Dec 2008.

[16] G. Singh, et al., Workflow Task Clustering for Best Effort Systems with Pegasus, Mardi Gras Conference, Baton Rouge, LA, Jan 2008.

[17] E. Deelman, et al., Pegasus: Mapping scientific workflows onto the Grid. Lecture Notes in Computer Science: Grid Computing, pp. 11–20, 2004

[18] Peter Couvares, Tevik Kosar, Alain Roy, Jeff Weber and Kent Wenger, "Workflow in Condor", in *In Workflows for e-Science*, Editors: I.Taylor, E.Deelman, D.Gannon, M.Shields, Springer Press, January 2007 (ISBN: 1-84628-519-4)

[19] Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed Computing in Practice: The Condor Experience" *Concurrency and Computation: Practice and Experience*, Vol. 17, No. 2-4, pages 323-356, February-April, 2005

[20] G. B. Berriman, etc., "Montage: A Grid Enabled Engine for Delivering Custom Science-Grade Mosaics On Demand," presented at SPIE Conference 5487: Astronomical Telescopes, 2004.

[21] FutureGrid: https://portal.futuregrid.org/

[22] Weiwei Chen, Ewa Deelman, Fault Tolerant Clustering in Scientific Workflows, IEEE Servcies 2012, Honolulu, Hawaii, June 2012.