

Combined Fault Tolerance and Scheduling Techniques for Workflow Applications on Computational Grids

Yang Zhang¹, Anirban Mandal², Charles Koebel¹ and Keith Cooper¹

¹Rice University

Department of Computer Science
Houston, TX 77005

²Renaissance Computing Institute
Chapel Hill, NC 27517

E-mail: ¹{yzhang8,chk,keith}@rice.edu ²anirban@renci.org

Abstract

More and more complex scientific workflows are now executed on computational grids. In addition to the challenges of managing and scheduling these workflows, additional reliability challenges arise because of the unreliable nature of large-scale grid infrastructure. Fault tolerance mechanisms like over-provisioning and checkpoint-recovery are used in current grid application management systems to address these reliability challenges. In this work, we propose new approaches that combine these fault tolerance techniques with existing workflow scheduling algorithms. We present a study on the effectiveness of the combined approaches by analyzing their impact on the reliability of workflow execution, workflow performance and resource usage under different reliability models, failure prediction accuracies and workflow application types.

I. Introduction

The grid [4], [5] is a collection of geographically distributed, highly heterogeneous and loosely coupled computing environments. Recent developments in grid infrastructure technologies make it possible to execute large and distributed applications [23], [3], [1] on it. Many of these applications fall in the category of workflow applications, which are often represented as DAGs (Directed Acyclic Graph) with nodes representing computations and edges representing data-movement. Because of its decomposable nature, the workflow paradigm has emerged as the most successful paradigm for applications to be executed on grid

infrastructures. At the same time, the recent growth in size and complexity of the grid infrastructure makes it susceptible to failures at all system levels - power supply, computing hardware, network, operating system, grid middleware etc. For example, the study in [10] shows that the mean time between failures (MTBF) on Grid5000 [6] is only around 12 minutes. Hence, not only is managing and scheduling workflow applications a hard problem and studied in detail [16], [27], [26], additional challenges in providing reliability to workflow executions arise because of the unreliable nature of the underlying hardware and software.

To address the reliability challenges, existing grid systems resort to fault tolerance and recovery mechanisms [19] such as checkpoint-recovery and over-provisioning. Checkpointing-recovery techniques make it possible for the workflow to resume execution from the last checkpoint instead of restarting from the beginning, should a failure occurs. Over-provisioning [11] techniques replicate a task on more than one resources to increase the probability of successful execution. Although these techniques address the reliability challenges to some extent, to the best of our knowledge, no large-scale study has been done on how effective they are when coupled with workflow management and scheduling. In this work, we study the performance, cost and effectiveness of different fault tolerance mechanisms when combined with different scheduling techniques.

The rest of the paper is organized as follows. Section II presents the details of the combined fault tolerance and scheduling techniques that we proposed and implemented. Section III describes our experimental design. Section IV presents our results and evaluation. Section V presents related work and section VI con-

cludes the paper with a summary of our contributions.

II. Scheduling with Fault Tolerance

Fault tolerance and recovery techniques used to mitigate the effects of workflow failures in grid systems fall in two major categories: (a) checkpoint-recovery and (b) over-provisioning/replication. Even though they are different approaches, they can also complement each other because checkpoint-recovery techniques are used mostly during workflow execution, while over-provisioning is used mostly during the scheduling/planning phase. In this section, we describe how we integrated these fault tolerance and recovery techniques during scheduling and execution phases. First, we describe two scheduling algorithms used in the study, which don't consider fault tolerance.

A. Scheduling Algorithms

The first is a list based algorithm called HEFT [8]. It is widely used [26] and Zhang *et al.* [27] have shown that it performs well in a multi-cluster grid environment. HEFT first determines the priorities of each task in the DAG. The priority of a task node i is computed as an upward rank, which is an estimation of the difference between the node's finish time and the whole DAG's finish time. It then schedules the tasks in descending order of their priorities. For each task, HEFT assigns it to the resource that can finish the task the earliest.

The second is a duplication based algorithm called DSH that was first proposed by Kruatrachue *et al.* [13]. It combines some ideas used in list scheduling with duplication to reduce the makespan. The algorithm considers each task in a descending order of their priorities similar to that used by HEFT. For each task, DSH first calculates the start time of the task on the resource without duplicating any of its predecessors. Then the algorithm attempts to duplicate the parents of the task into a time slot on the same resource until either the slot is used up or the start time of the task does not improve further. This process is repeated for other resources and the task is scheduled to the resource that gives the earliest finish time with the possible duplications.

B. Scheduling Algorithms with Over-provisioning

HEFT and DSH scheduling algorithms do not take into account any fault tolerance. We integrated the following over-provisioning technique with the vanilla

HEFT and DSH scheduling algorithms to come up with a fault-tolerant scheduling scheme.

Fault Tolerance Using Over-provisioning: We use over-provisioning/replication as the primary mechanism for fault tolerance when scheduling workflow tasks onto resources. Over-provisioning is a fault-tolerance mechanism where multiple copies of the workflow task (with the same input data-set) are executed in parallel. The idea is to maximize the probability of success for the workflow task, so that if one copy fails, one of the other copies may succeed.

Each workflow task has performance constraints (expressed through performance models and deadlines) and reliability constraints (expressed through an expected success probability). Our goal is to find the smallest set of resources to replicate the given workflow task to satisfy these constraints. We use an efficient algorithm described by Kandaswamy *et al.* [11] to find the smallest subset of resources that satisfies these constraints. In the cases when it is not possible to satisfy the success probability or deadline constraints, the over-provisioning algorithm returns all possible resource combinations tagged with the success probabilities for each resource set, so that a best-effort replicated set of resources can be chosen.

Integrating Fault Tolerance with HEFT/DSH: Figure 1 describes the algorithm that we use to integrate the over-provisioning algorithm in [11] with HEFT. First, we sort the tasks in the DAG by its upward rank. Then, we assign each task to the resource that has the earliest finish time. After a task is assigned, we check its parent task. If all child tasks of the parent task have been assigned to a resource, we invoke the over-provisioning algorithm to find a set of resources on which the parent task should be replicated so that its deadline and success probability constraints are satisfied. We set the deadline as 30% more than the node's finish time without duplication and the probability constraint as 0.95. We assign the parent task to the resources in the set. If there is no resource set that can statistically guarantee a success probability of 0.95, the over-provisioning algorithm returns a best-effort set of resources and the corresponding success probabilities. In this case, we pick the resource set with the highest reliability. We assign the parent task to those resources in addition to the original one. We also set a limit on the total number of resources one task can be replicated on. We will discuss in section IV how this limit affects the outcome.

We integrated over-provisioning with the DSH scheduling algorithm in a similar fashion. In this case, we duplicated the parent task (for potential improvement of communication performance) and then invoked

```

HEFT-Dup(DAG dag, Resource res, PerfModel pM)
Task[] tasks = dag.sortTask(res)
for each task  $t$  in the tasks
   $t$ .mapResource(res, pM)
   $t$ .assigned = TRUE
  for each  $pTask$  in  $t$ 's parent Task
    Vector dupRes;
    if (  $pTask$ .allChildrenAssigned() == True )
      dupRes = FTR.getDupRes( $pTask$ , res, pM)
    if (dupRes.isEmpty() )
      dupRes = FTR.getAllDup( $pTask$ , res, pM)
      dupRes = dupRes.selectMostReliable()
    for each resource  $r$  in dupRes
      if (  $pTask$  is not assigned to  $r$  already )
        assign  $pTask$  to  $r$ 
        if ( $pTask$  replicated to LIMIT resources)
          break
    end
  end
end

```

Fig. 1. HEFT with Over-provisioning

the over-provisioning algorithm for achieving better reliability (higher success probability).

C. Scheduling Algorithms with Checkpoint-recovery

To mitigate for failures during execution time, we use checkpointing and resubmission of workflow steps. We chose to implement a light-weight checkpointing strategy [19] that saves only the current location of the intermediate data as opposed to a heavy-weight checkpointing strategy [19] that saves the data on a separate system. We made this choice because the performance of the heavy-weight checkpointing mechanism relies heavily on the reliability and performance of the backup system. This could lead to a chicken and egg problem. Hence, we decided to focus on the light-weight checkpointing, which confines our study in one multi-cluster grid.

Since a workflow application consists of multiple tasks, it is natural to do a light-weight checkpointing after each task is finished. If a task failed to finish due to resource unavailability, we migrate the task onto the most reliable resource based on the reliability prediction. However, since we only implement light-weight checkpointing, it is possible that some of the resources that the parent task was running on is also not available. In this case, we will migrate those parent

tasks to restart on a different resource. However, this approach has a potential to cause infinite execution time when a task fails repeatedly while its parent tasks always finish after restart. Therefore, we put a limit on the number of times a task can migrate. In our study, the limit is set to be three, since we did experiments show that a higher limit does not provide better reliability and uses more resources.

Since checkpointing and recovery happens during execution time, we can apply it to the over-provision version of HEFT and DSH directly. So, we have the following combined versions - (a) HEFT with over-provisioning and checkpointing-recovery and (b) DSH with over-provisioning and checkpointing-recovery.

D. Whole DAG Over-provisioning and Migration

The fault tolerance strategy described in section II-B is task based, which means that it can only guarantee the statistical success probability of a task, not the entire DAG. For a workflow application with N tasks, with each task having a success probability of s_n , the success probability of the entire workflow is

$$SuccProb_{overall} = \prod_{i=1}^N s_i$$

The DAG success probability can be very low when N is large. For example, the success probability is only 36% for a workflow application with 100 tasks even with each task having a success probability of 99%. Therefore, in addition to the task based fault tolerance strategy, we also propose a whole DAG over-provision(WDO) mechanism that replicates the whole DAG onto multiple resources.

Figure 2 describes the algorithm that we use for whole DAG over-provisioning. We first estimate the makespan of the entire DAG for each resource according to the performance models of the tasks that constitute the workflow. We then get the failure probability according to each resource's reliability model. After we sort the resource by their failure rate in descending order, we apply a greedy approach that assigns a DAG to the resources with the highest reliabilities until the aggregated success probability is over 0.95. The aggregate success probability is

$$SuccProb_{overall} = 1 - \prod_{i=1}^N f_i$$

where f_i is the failure probability of resource i and N is the number of resources.

We also combine the whole DAG over-provisioning algorithm with the checkpoint-recovery mechanism,

```

DAG-Dup(DAG dag, Resource res, PerfModel pM)
  TreeSet<Entry<Resource, Double>> relSort
  for each resource  $r$  in res
     $t = \text{dag.getEstimateTime}(r, \text{pM})$ 
     $\text{failProb} = r.\text{reliabilityModel.getFailProb}(t)$ 
     $\text{reliabilitySort.put}(r, \text{failProb})$ 
  end
  Sort resources in relSort by the reliability
  for each resource  $r$  in relSort
    calculate the aggregate success prob.
    if overall success prob  $\leq 0.95$ 
      assign  $\text{dag}$  to  $r$ 
      if (  $\text{dag}$  replicated to LIMIT resources)
        break
      else
        break
    end
  end

```

Fig. 2. Whole DAG Over-provisioning

which happens during the execution time. Therefore, we have 10 different scheduling and fault tolerance mechanism combinations in total. We will apply these to our workflow management system and analyse their reliability, performance and resources usages.

III. Experimental Methodology

To study how these fault tolerance and scheduling strategies perform in a multi-cluster grid environment, we implemented a simple workflow management system that schedules and executes a workflow application on a simulated multi-cluster grid. We use this system to schedule and execute workflow applications with different fault tolerance and scheduling techniques. In this section, we will first discuss the resource reliability models we use. Then, we present our experimental design.

A. Resource Reliability Model

Recent studies [20], [10], [22], [18] show that the mean time between failures (MTBF) on modern high performance clusters is best modeled by a Weibull distribution [25]. However, the shape and scale parameters are different for each study. Nurmi *et al.* [18] and Schroeder *et al.* [22] report that the shape parameter is less than 1, which means that the hazard rates (the frequency a system or component fails) decrease with time. Whereas, Iosup *et al.* [10] report that the shape parameter is greater than 1, which indicates an

increasing hazard rate over time. Hence, we wanted to explore both regions for the shape parameter in our study and created two sets of reliability configurations - one set with shape parameter ranging between 0.5 and 0.9 according to [22] and the other set with shape parameter ranging between 10 and 13 according to [10]. Each set comprises of three reliability characteristics based on the quality of the resource (from a reliability standpoint) - stable, normal and shaky.

B. Experimental Setup

We use a multi-cluster simulated grid environment with nine clusters that have the same processor configuration as nine sites in the TeraGrid [24]. Correspondingly, there is a Weibull distribution for each cluster, corresponding to the current shape and scale parameter. The nine distributions comprise the reliability model for the resources for the current configuration of shape/scale parameters. Although the scale parameters are different for each individual distribution, they are close to the mean value of the model. For a given range of shape parameter, we generate three reliability models based on three mean values of the scale parameter - three days (for shaky), one week (for normal) and three weeks (for stable). So, with two different ranges of shape parameters, we explore 6 different reliability models in this study. The resource failures are randomly generated following the Weibull distributions in the reliability model.

We generated three types of DAGs corresponding to three different applications - Montage [23], Fast Fourier Transform and Gaussian elimination. We also generated two types of DAGs that represent common parallel programming models - master slave and MPMD. For each type of DAG, we generated over 100 DAGs with configurations differing in the total number of tasks, the average size of the task and the computation to communication ratio. We use historical performance models generated from the performance data we collected from our previous experiments on production clusters. The estimated running time of those DAGs range from a few hours to a month. The estimated success probabilities of those DAGs range from almost zero to almost one.

In total, we used 635 different DAGs and 6 different reliability models. Since the failures are randomly generated, we run 10 times for each DAG and reliability model combination. For each run, we use all 10 different scheduling and fault tolerance mechanism combinations so that each approach sees the same resource failures. Therefore, we collect 381,000 different executions' results for each batch of experiments. In

total, we ran five batches of experiments that use different parameters we described in section II. We will discuss the experiments and results in the following section.

IV. Results

We present our experimental results for the algorithms we described in section II. We have two basic scheduling algorithms, HEFT and DSH and a duplication based whole DAG over-provision(WDO) algorithm. We denote the over-provisioning versions of them with an “O” at the end and the checkpointing-restart version with a “C” at the end. An “OC” in the end means both fault tolerance mechanisms are applied. Figure 3 shows the overall percentage of workflow applications that successfully finished after using one of the ten scheduling and fault tolerance technique combinations. From the graph, we observe that the over-provisioning mechanism can increase the DAG success probability by around 25% while light-weight checkpointing-restart can increase the success probability by around 12%.

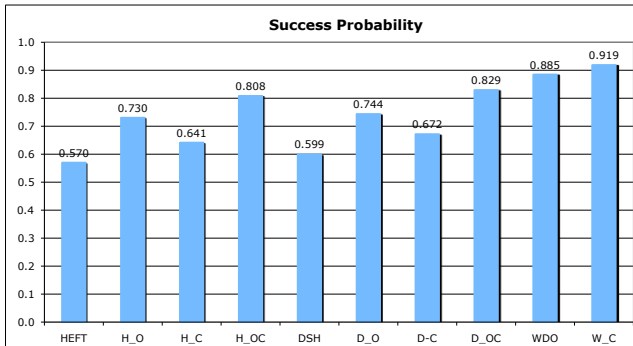


Fig. 3. Overall Success Probability

Figure 4 shows the workflow performance for each approach. Since the different approaches do not have the same success rate, we observed that an approach that has higher success rate finishes more large size DAGs. In this case, an average value of turn-around time would clearly favor HEFT and DSH since they finish more small DAGs. Therefore, we use the standard length ratio (SLR), which is defined as $SLR = \text{turn-around time} / CPIC$, as a measure of performance. The turn-around-time is the application’s running time we measured in our simulation environment. We approximate the *Critical Path Including Communication (CPIC)* with the b-level [14] value of the root node in the DAG. Lower SLR indicates better performance. We can see that over-provisioning only increases the SLR

by at most 5% while checkpointing-restart increases SLR by at most 6%. The relatively small performance penalty is because the makespan for the schedule for the over-provisioning version is same as that of the original schedule unless the fast resource is down. In that case, the penalty is just the completion time difference between the next fastest resource and the fastest one. Whole DAG over-provision approach also performs better than HEFT. This is because WDO eliminates communication time unless there are task failures since each task’s parent is already duplicated on the same resource.

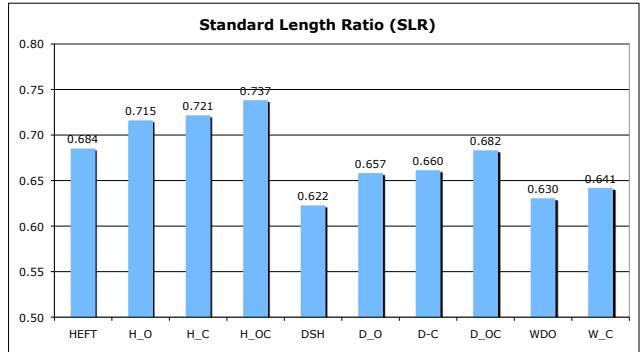


Fig. 4. Overall Standard Length Ratio (SLR)

Figure 5 shows the resource usage for each approach in terms of total cpu hours used. However, since each approach finishes different number of DAGs, we divided the total usage into three parts in order to further analyze the result. We use “used” resource time to denote the total cpu hour consumed by the completed tasks in the DAGs that successfully finished. The “wasted” resource time is the total cpu hour consumed by the completed tasks in the DAGs that failed to finish. The “failed” resource time is the total cpu hour consumed by the failed tasks no matter whether the DAG finishes or not. The solid stack bar in figure 5 shows the aggregated cpu hour that a workflow used including all three usage types. We can see that over-provision uses around 2.5 times more resources than HEFT while checkpoint restart uses 1.5 times more resources than HEFT. Besides that, we also calculated the total cpu hour that a failed DAG may need to complete successfully, when there are no more resource failures. We call it the “potential” resource usage and plot it as a transparent bar on top of the solid bar. We can see that since HEFT and DSH have a lower completion rate, the “potential” resource usage is higher.

Figure 6 illustrates how resource reliability affects the success probability of our approaches. For each

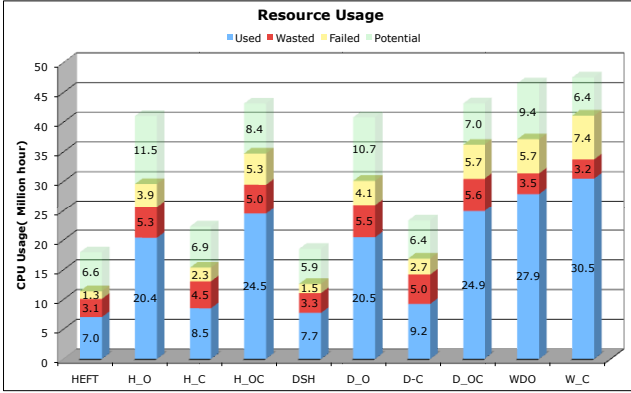


Fig. 5. Overall Cpu Time Usage

shape parameter, we categorize our reliability models into three groups. The results show that, using HEFT only, the average success probability of workflows is 32% when they are executed on the most unreliable resources, referred to as the shaky resources. The average success probability of workflows using HEFT only is 83% when they are executed on the most reliable resources, referred to as the stable resources. We refer to the third group as the normal resources, where reliability is somewhere between the two. We can see that the more fault tolerant techniques we use, the less is the dependence of the workflow application’s success probability on the underlying resource reliability. The success probability of HEFT on the stable resources is over 150% more than that on the shaky resources. The algorithms with over-provisioning alone has a better success probability by about 80%. Whereas, the whole DAG over-provisioning with checkpointing-restart has only about 20% difference in success rate. Also, the less reliable the resource is, the more effective are the fault tolerance techniques. The WDO algorithm improves reliability by over 200% than HEFT on shaky resources, while only by 17% on stable resources.

Figure 7 shows the expected resource usage for each approach under different reliability models. We use the expected value instead of the actual value because algorithms using no or less fault tolerance techniques complete less number of DAGs. In order to better compare their total resource usage, we incorporate the success rate of the approach to normalize the resource usage. We view the repeated run of a DAG as a Bernoulli process and the success probability is the success probability p of the approach it uses. Therefore, we know that the expected number of trials before one sees a success is $1/p$. We then calculate the expected resource usage for algorithm $algo$ as the

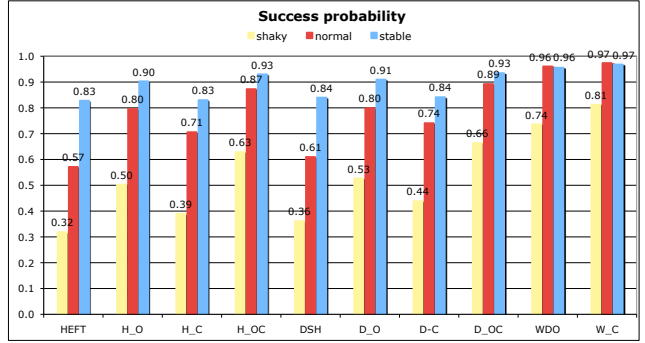


Fig. 6. Success Probability with Different Reliability Models

$cpu_hour_{algo} \times 1/p$. That is the expected resource one approach uses to get a successful execution. We can see that the fault tolerant versions of the algorithms have over 100% more expected resource usage than the vanilla scheduling algorithm on shaky resources, while uses 50% more on stable resources. We notice that WDO uses the lowest total expected resources among all over-provisioning based approaches while providing the best reliability. We also measured the performance in terms of SLR. The reliability does not affect the performance much. Each approach’s SLR differences on different reliability models are within 10% and the difference between different approaches is similar to the data in figure 4.

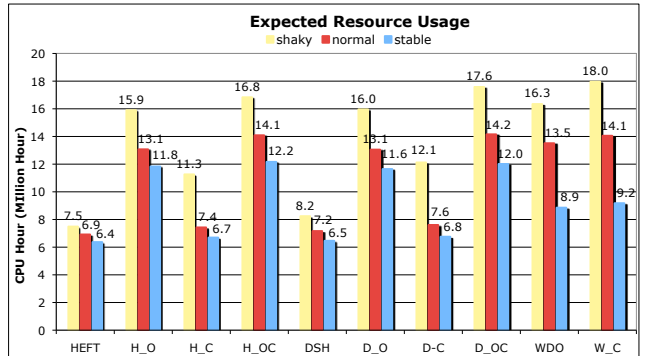


Fig. 7. Resource Usage with Different Reliability Models

Since perfect failure probability prediction is hard, we tested the robustness of our fault tolerance approaches with respect to failure prediction accuracies. Figure 8 shows the workflow application execution success probabilities with different failure prediction accuracies. The accurate prediction is the failure probability that we get from the Weibull distribution’s

cumulative distribution function. For the optimistic prediction, we multiply the accurate failure probability by a random number evenly distributed between 0 and 1. Therefore, the expected failure probability is half of the accurate one. For the pessimistic prediction, we divide the accurate failure probability by a random number. We also take the lesser of the outcome and 0.9999 so that the failure probability stays under 1. From figure 8 we can see that although the accurate prediction always leads to best success probabilities, all fault tolerance mechanisms are pretty robust under inaccurate failure predictions. We also have results showing that the approaches' performance is very minimally affected by the failure probability prediction. Figure 9 shows the expected resource usages under

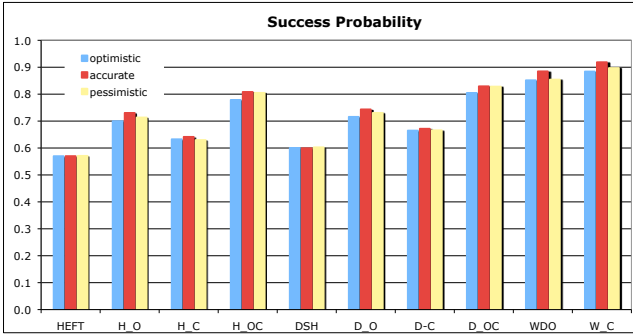


Fig. 8. Success Probability with Different Failure Prediction Accuracies

accurate, optimistic and pessimistic failure probability predictions. We see that the pessimistic prediction can lead to more resource usages by as much as 20% more than in the optimistic prediction. Since optimistic prediction does not lead to significantly less success probability, it shows that it's better to lean toward the optimistic side when an accurate failure prediction is hard to obtain. We believe that one reason why our fault tolerance mechanisms perform well under pessimistic failure predictions is that we have a limit set for how many resources each task can be over-provisioned. The default value is set to 3. Thus, even if the failure prediction is too pessimistic, we are not going to over-provision too much. To show the effect of the resource limit, we also used 5 and 10 as the limit. Results show that it does not affect the application success probability and performance much. However, figure 10 shows how it does affect the expected resource usage. We can see that when the replication limit is over 3 times, it can use almost 100% more resources while providing only less than 3% more success probability.

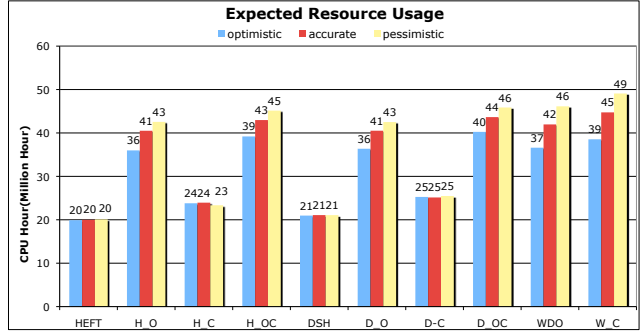


Fig. 9. Resource Usage with Different Failure Prediction Accuracies

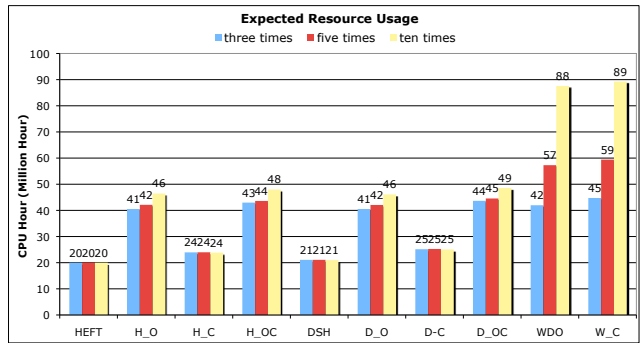


Fig. 10. Resource Usage with Different Replication Limits

V. Related Work

Workflows application scheduling on grids [16], [26], [27] is an active area of research. Workflow scheduling has largely focused on heuristic techniques using performance models to qualitatively select resources and map tasks to the resources that have good performance [12]. Few scheduling algorithms take into account reliability of the grid resources.

One of the most widely implemented fault-tolerance techniques on computational Grid is simple retry [19] which means the application is resubmitted on a resource in case of a failure. In many workflow management frameworks [17], [7], the remaining portion of the workflow is resubmitted in case of a failure.

Hwang *et al.* [9] present a failure detection service (based on notifications) and a flexible framework for handling Grid failures. Limaye *et al.* [15] have developed a checkpoint/restart mechanism that places checkpoints based on system reliability. Ramakrishnan *et al.* [21] use “performability” to capture the degraded performance that might result from varying resource

availability. Dongarra *et al.* [2] use the product of failure rate and unitary instruction execution time to guide the scheduling of independent tasks onto heterogeneous clusters.

VI. Conclusions

This paper presents workflow scheduling and execution mechanisms that incorporate a balanced approach toward reliability and performance. It also presents a new algorithm that replicates the whole DAG (WDO) onto several clusters. We quantitatively evaluate the three way trade-off between reliability, performance and resources usage for each approach in a large-scale simulated environment. From the experiments, we observe that the fault tolerance techniques are effective. They can increase the reliability of workflow executions by as much as 200% and do not affect performance by more than 10%. Among them, the whole DAG over-provisioning approach consistently ranks among the top two in terms of reliability, performance and expected resource usages. We also verified that accuracy of failure probability prediction does not affect the results by more than 10%.

References

- [1] Charlie Catlett and et al. <http://www.griphyn.org>, 2002.
- [2] Jack J. Dongarra, Emmanuel Jeannot, Erik Saule, and Zhiao Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 280–288, New York, NY, USA, 2007. ACM.
- [3] Linked Environments for Atmospheric Discovery (LEAD) Portal. <https://portal.leadproject.org>.
- [4] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999.
- [5] I. Foster and C. Kesselman. *The Grid 2*. Morgan Kaufmann Publishers, Inc., 2003.
- [6] Tristan Glatard, Johan Montagnat, and Is Cnrs. An experimental comparison of grid5000 clusters and the egee grid. In *In Workshop on Grid*, 2006.
- [7] Ewa Deelman Gurmeet Singh, Carl Kesselman. Optimizing grid-based workflow execution. *Journal of Grid Computing*, 3(3):201–219, 2005.
- [8] H.Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 2(13):260–274, 2002.
- [9] Soonwook Hwang and Carl Kesselman. Gridworkflow: A flexible failure handling framework for the grid. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, page 126, Washington, DC, USA, 2003. IEEE Computer Society.
- [10] A. Iosup, M. Jan, O. Sonmez, and D.H.J. Epema. On the dynamic resource availability in grids. *Grid Computing, 2007 8th IEEE/ACM International Conference on*, pages 26–33, Sept. 2007.
- [11] Gopi Kandaswamy, Anirban Mandal, and Daniel A. Reed. Fault tolerance and recovery of scientific workflows on computational grids. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 777–782, Washington, DC, USA, 2008. IEEE Computer Society.
- [12] K. Kennedy, K. Cooper, F. Berman, A. Chien, I. Foster, C. Kesselman, D. Reed, J. Dongarra, and R. Wolski et al. Toward a framework for preparing and executing adaptive Grid programs. In *Proceedings of NSF Next Generation Systems Program Workshop (International Parallel and Distributed Processing Symposium 2002)*, Fort Lauderdale, FL, April 2002, 2002.
- [13] Boontee Kruatrachue and Ted Lewis. Grain size determination for parallel processing. *IEEE Softw.*, 5(1):23–32, 1988.
- [14] Y. Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.
- [15] K. Limaye, B. Leangsuksun, Yudan Liu, Z. Greenwood, S. L. Scott, R. Libby, and K. Chanchio. Reliability-aware resource management for computational grid/cluster environments. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 211–218, Washington, DC, USA, 2005. IEEE Computer Society.
- [16] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling strategies for mapping application workflows onto the Grid. In *14-th IEEE Symposium on High Performance Distributed Computing (HPDC14)*, pages 125–134, 2005.
- [17] Dagman MetaScheduler. <http://www.cs.wisc.edu/condor/dagman>.
- [18] Daniel Nurmi, John Brevik, and Rich Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *In Euro-Par'05*, pages 432–441. Springer, 2005.
- [19] Kassian Plankensteiner, Radu Prodan, Thomas Fahringer, Attila Kertesz, and Peter K Kacsuk. Fault-tolerant behavior in state-of-the-art grid workflow management systems. Technical report, 2007.
- [20] Narasimha Raju, Yudan Liu, Chokchai Box Leangsuksun, Raja Nassar, and Stephen Scott. Reliability analysis in hpc clusters. In *Proceedings of the High Availability and Performance Computing Workshop*, 2006.
- [21] Lavanya Ramakrishnan and Daniel A. Reed. Performability modeling for scheduling and fault tolerance strategies for scientific workflows. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 23–34, New York, NY, USA, 2008. ACM.
- [22] Bianca Schroeder and Garth A. Gibson. A large-scale study of failures in high-performance computing systems. In *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks*, pages 249–258, Washington, DC, USA, 2006. IEEE Computer Society.
- [23] G. Singh, E. Deelman, and G. Bruce Berriman et al. Montage: a Grid enabled image mosaic service for the National Virtual Observatory. *Astronomical Data Analysis Software and Systems*, 2003.
- [24] TeraGrid. <http://www.teragrid.org/about>.
- [25] Weibull and Waloddi. A statistical distribution function of wide applicability. In *Journal of Applied Mechanics*, 1951.
- [26] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for Grid computing. *SIGMOD Rec.*, 34(3):44–49, 2005.
- [27] Y. Zhang, C. Koelbel, and K. Kennedy. Relative performance of scheduling algorithms in grid environments. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 521–528, Washington, DC, USA, 2007. IEEE Computer Society.