FLAW: FaiLure-Aware Workflow Scheduling in High Performance Computing Systems

Zhifeng Yu, Chenjia Wang and Weisong Shi

Wayne State University

{zhifeng.yu, chenjia.wang, weisong}@wayne.edu

Abstract— **The goal of workflow application scheduling is to achieve minimal makespan for each workflow. Scheduling workflow applications in high performance computing environments, e.g., clusters, is an NP-Complete problem, and becomes more complicated when potential resource failures are considered. While more research on failure prediction has been witnessed in recent years to improve system availability and reliability, very few of them attack the problem in the context of workflow application scheduling. In this paper, we study how a workflow scheduler benefits from failure prediction and propose FLAW, a failure-aware workflow scheduling algorithm. Furthermore, we propose two important definitions on accuracy, *Application Oblivious Accuracy (AOA)* and *Application Aware Accuracy (AAA)*, from the perspectives of system and scheduling respectively, as we observe that the prediction accuracy defined conventionally imposes different performance implications on different applications and fails to measure how that improves scheduling effectiveness. The comprehensive evaluation results using real failure traces show that *FLAW* performs well with practically achievable prediction accuracy by reducing the average *makespan*, the loss time and the number of job rescheduling.**

I. Introduction

Workflow applications become to prevail in recent years, resulting from stronger demands from scientific computation and more provision of high performance computing systems. Typically, a workflow application is represented as a Directed Acyclic Graph (DAG), where nodes represent individual jobs and edges represent the inter-job dependence. In a DAG, nodes and edges are weighed for computation cost and communication cost respectively. *Makespan*, the time difference between the start and completion of a workflow application, is used to measure the application performance and scheduler efficiency. In the rest of the paper, we use DAG and workflow application interchangeably in this paper.

The goal of scheduling a workflow application is to achieve minimal *makespan*. It was recognized as an NP-Complete one [1]. With the resource failure considered, scheduling a workflow application in a high performance computing system is significantly more difficult and unfortunately few existing algorithms attempt to tackle this. On the other side, the failure tolerance policies applicable to ordinary job scheduling is a reactive approach and deserves another look as job inter-dependencies in workflows complicates the failure handling.

Recent years have seen many analysis on published failure traces of large scale cluster systems [2], [3], [4], [5], [6]. These research efforts contribute to better understanding of the failure characteristics and result in more advanced failure prediction models, and help improve the system reliability and availability. However, not sufficient attention has been paid on how workflow scheduling can benefit from these accomplishments to reduce the impact of failures ion application performance. In particular, we want to answer the following two related questions: One is *what is the right and practical objective of failure prediction in the context of workflow scheduling*? The other is *how does the failure predication accuracy affects workflow scheduling*?

Failures can have a significant impact on job execution under existing scheduling policies that ignore failures [5]. In an error prone computing environment, failure prediction will improve the scheduling efficiency if it can answer queries with a great success rate such as: "*Will a node fail in next 10 hours if the job requires 10 hours to finish on this node?*" The job will be assigned to this node only when the answer is "Yes" with high confidence. The unexpected failure not only causes rescheduling of the failed job and resource waste on the uncompleted job execution, but also affects the subsequent job assignment which is the key for workflow performance. We propose a **F**ai**L**ure **A**ware **W**orkflow scheduling algorithm (**FLAW**) in this paper to schedule workflow applications with resource failure presence.

On the other side, we argue that the conventional definition of failure prediction accuracy does not well reflect how accuracy impacts on scheduling effectiveness. When scheduling a workflow application, the predictor is queried whether a node would fail in a given time window, i.e. the job execution duration. It depends on the capability of the node being assigned to. Typically each individual job has varied computing and communication demands. Moreover, assigned to different nodes the same job may have different execution time decided by node computing capability and data placement. Therefore, a scheduler has to predict potential failure for various nodes for accordingly different time windows. However, the conventional failure prediction result is a set of time periods within which a failure is predicted to happen, different than a scheduler requires. We argue that the conventional approach is not intended for failure aware workflow scheduling and propose two new definitions of failure prediction accuracy: *Application Oblivious Accuracy (AOA)* from a system's perspective and *Application Aware Accuracy (AAA)* from a scheduler's perspective, which we

This work is in part supported by National Science Foundation CAREER grant CCF-0643521.

believe better reflect how failure prediction accuracy impacts on scheduling effectiveness.

The contributions of this paper are three-fold: (1) design FLAW that takes potential resource failures into consideration when scheduling workflow applications and defines the failure prediction requirement in a practical way; (2) propose two new definitions of failure prediction accuracy, which reflects its effect on workflow application scheduling more precisely, in the context of job scheduling in high performance computing environments; and (3) perform comprehensive simulations using real failure traces and find that our proposed FLAW algorithm performs well even with moderate prediction accuracy, which is much easier to achieve using existing algorithms. The number of job resubmission (rescheduling) due to resource failure decreased significantly with trivial *AOA* level with presence of intensive workload.

The rest of the paper is organized as follows. A brief review of related work is given in Section II to motivate our work. The new definitions of failure prediction accuracy are proposed in Section III. Then we describe the *FLAW* algorithm design in Section IV. Section V elaborates the simulation design and analyzes simulation results. Finally, we summarize and lay out our future work in Section VI.

II. Related work

Noticeable progress has been made on failure prediction research and practice, following that more failure traces are made public available since 2006 and the failure analysis [2], [3], [4], [5], [6], [7] reveals more failure characteristics in high performance computing systems. Zhang *et al.* evaluate the performance implications of failures in large scale cluster [5]. Fu *et al.* propose both online and offline failure prediction models in coalitions of clusters. Another failure prediction model is proposed by Liang *et al.* [2] based on failure analysis of BlueGene/L system. Recently, Ren et al. [8] develop a resource failure prediction model for fine-grained cycle sharing systems. However, most of them focus on improving the predication accuracy, and few of them provide how to leverage their predication results in practice.

Salfner *et al.* [9] suggest that proactive failure handling provides the potential to improve system availability up to an order of magnitude, and the FT-Pro project [10] and the FARS project [11] demonstrate a significant performance improvement for long-running applications provided by proactive fault tolerance policies. Fault aware job scheduling algorithms are developed for BlueGene/L system and simulation studies show that the use of these new algorithms with even trivial fault prediction confidence or accuracy levels (as low as 10%) can significantly improve the system performance [12].

Failure handling is considered in some workflow management systems but only limited in failure recovery. Grid Workflow [13] presents a failure tolerance framework to address the Grid-unique failure recovery, which allows users to specify the failure recovery policy in the workflow structure definition. Abawajy [14] proposes a fault-tolerant scheduling policy that loosely couples job scheduling with job replication scheme such that applications are reliably executed but with cost of resource efficiency. Other systems such as DAGMan[15] simply ignore the failed jobs and the job will be rescheduled later when it is required for dependant jobs. Dogan *et al.* [16] develop Reliable Dynamic Level Scheduling (RDLS) algorithm to factor resource availability into conventional static scheduling algorithms.

We argue that, however, failure handling can not be practically integrated into existing static scheduling schemes as it is not possible to predict all failures accurately in advance for a long running workflow application. Even for other non-workflow applications, the analysis [4] finds that node placement decision can become ill-suited after about 30 minutes in a shared federated environment such as PlanetLab [17]. Furthermore, another analysis [7] concludes that Time-To-Fail (TTF) and Time-To-Repair (TTR) can not be predicted with reasonable accuracy based on current uptime, downtime, Mean-Time-To-Fail (MMTF) or Mean-Time-To-Repair (MMTR) and a system should not rely on such predictions.

Inspired by these observations, we design the *FLAW* algorithm based on the dynamic scheduling scheme proposed in our previous work [18], redefine the failure prediction requirement and accuracy definition in the context of workflow scheduling. The new failure prediction requirements are better measurable and achievable practically. The extensive simulations using real failure trace from Los Alamos National Laboratory [19] demonstrate that *FLAW* reduces the failure impact on performance significantly with moderate prediction accuracy.

III. Failure prediction accuracy

Reliability based failure prediction techniques use various metrics to measure prediction quality, where *precision* and *recall* are popular ones adopted in the literature [9], [10], [11], [20], where *precision* is the ratio of the number of correctly identified failures to the number of all positive predictions and the *recall* is the ratio of the number of correctly predicted failures to the total number of failures that actually occurred [9]. In other research efforts the accuracy is defined in a statistics context, by measuring how the predicted time between failures is close to actual one [3]. However none of them is intended to be used in job scheduling.

In an error prone high performance computing system, failure prediction is required by scheduler to answer the query before a job is scheduled to the chosen node: *Will this node fail during the job execution?* Intuitively, the quality of failure prediction should be measured how well those queries can be answered. The scheduler's effectiveness will be adversely impacted if either a failure is not predicted, i.e., the job has to be resubmitted later, or the predicted failure does not actually happen, i.e., a preferred resource may be wasted.

The conventional approach defines a failure prediction is a *true positive* if a true failure occurs within the prediction period Δt_p of the failure prediction [9]. As the conventional definitions originally come from information retrieval theory [9], they depend on the size of Δt_p and do not consider the length of failure down time. Even with the same failure prediction results. the prediction accuracy can vary with the size of prediction period Δt_p.

As illustrated in Fig. 1, the failure prediction is rated 100% for both *precision* and *recall* given the prediction period Δt_p. But with a smaller predication period $\Delta t'_p$, both *precision* and *recall* change to 50% for the identical prediction.

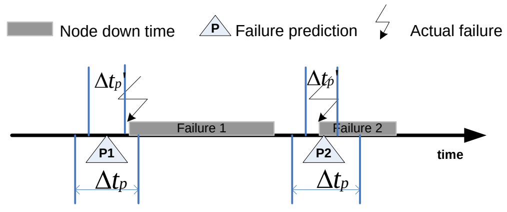


Fig. 1. An example of actual failure trace and associated failure prediction.

We want to know if such definition can be used to measure how much failure prediction can impact job scheduling effectiveness and justify what level of accuracy is good enough. With this in mind, we introduce failure prediction requirements in the context of job scheduling. A failure predictor should be able to predict if a node will fail in the next given time window. The prediction is correct if the node actually goes down in that time window. Before introducing the prediction accuracy, we define three prediction cases as following:

- *True Positive (Hit)*: A failure is predicted and it occurs within the down time of a true failure.
- *False Negative (Fn)*: An actual failure event is not predicted at all.
- *False Positive (Fp)*: A predicted failure does not match any true failure event.

Each failure prediction includes both time and location of predicted failure. By using the same example in Fig. 1, prediction P1 is a false positive as node is actually alive at the time of P1 predicts. P2 is a hit as it predicts the down time. Failure 1 counts as a false negative as it is not predicted.

$$Accuracy_{AOA} = \frac{Hit}{Hit + Fn + Fp} \qquad (1)$$

Finally, we define a so called *Application Oblivious Accuracy (AOA)* in Equation 1. The failure prediction accuracy in above example is rated as 33.3% accordingly. The definition considers failure downtime, penalizes both false negatives and false positives, and it is measured by failure prediction and actual failures only and therefore more objective. Furthermore, we observe that the failure prediction with same level of *AOA* has different impact on job scheduling and the prediction efficiency is application specific, which leads us to define an Application Aware Accuracy (*AAA*) in Section IV-C later.

IV. FAILURE AWARE WORKFLOW SCHEDULING

Inspired by the recent progresses in failure prediction research and increasing popularity of workflow applications, we explore the practical solutions for scheduling a workflow application in an error prone high performance computing environment. In this section, we first discuss the motivation of our research, then describe the solution design and finally illustrate the design by examples.

A. Motivation

There have been extensive research efforts on workflow scheduling and numerous heuristics are proposed as a result. However, they are yet to address the challenges of scheduling workflow applications in a cluster and grid environment: *dynamic work load* and *dynamic resource availability*.

Following our previous work [21] which tackles the resources dynamics, a DAG scheduling algorithm *RANK_HYBD* [18] is developed to handle dynamic workload in clusters and Grid environments. *RANK_HYBD* is a dynamic scheduling approach which schedules the job in the order of predefined priority so that the job with higher priority will get preferred resources. Without considering the potential resource failures, *RANK_HYBD* outperforms the widely used (*FIFO*) algorithm significantly in the case of dynamic work load [18].

Furthermore, the design rationale of *RANK_HYBD* provides it an intrinsic capability to handle resource failures in a proactive way by seamlessly integrating with an online failure predictor. In *RANK_HYBD*, individual jobs are prioritized first to reflect the significance of their respective impact on overall *makespan*, and scheduling decision is made only when a job is ready to execute and resource is available during job execution as predicted. Therefore, failures can be easily handled during scheduling. A workflow typically takes long time to finish, it is very difficult, if not impossible, for a static scheduling approach to plan for all potential failures in advance. However, resource failures can be much better handled at job level as the job execution time is significantly shorter compared with entire workflow and it is practically easier to predict a failure in shorter period. This assumption is well supported by recent research results [10], [11], [12] which propose failure tolerant scheduling schemes for non-workflow jobs.

On the other hand, the advancement in failure prediction techniques based on analysis of real traces of large scale clusters, is not yet to be leveraged by workflow application schedulers. The profound comprehension of failure patterns and characteristics makes a reasonable accurate failure predictor a practically achievable goal, so for the failure aware workflow scheduler.

B. Design

FLAW factors in failure handling by adding an online failure predictor component into the original *RANK_HYBD* design, as Fig. 2 shows. The proposed system consists of four core components: *DAG Planners*, a *Job Pool*, an *Executor* and an *online failure predictor*. The *DAG Planner* assigns each individual job a local priority as defined in [18], manages the job interdependence and job submission to the *Job Pool*, which is an unsorted collection containing all ready to execute jobs from different users. The *Executor* re-prioritizes the jobs in the *Job Pool* and schedules jobs to the available resources in the order of job priority. When making a scheduling decision, the *Executor* will consult the *Failure Predictor* about whether a resource will keep alive for the entire job execution period if the job is assigned to this resource. If a job is terminated due to unpredicted resource failure, the *Executor* will place the job back into *Job Pool* and the job will be rescheduled. When a job finishes successfully, the *Executor* notifies the *DAG Planner* which the job belongs to of the completion status.

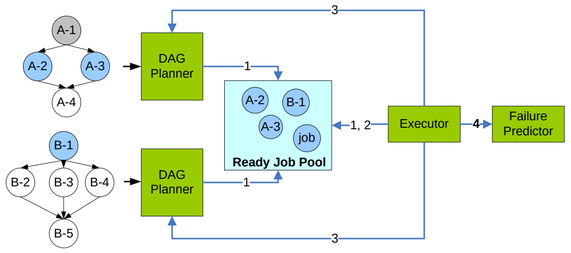


Fig. 2. An overview of *FLAW* design.

The above collaboration among these core components is achieved by the dynamic event driven design illustrated in Fig. 2 and explained as follows:

1) *Job submission*. When a new DAG arrives, it is associated with an instance of *DAG Planner* by the system. After ranking all individual jobs within the DAG locally, the *Planner* submits whichever job is ready to the *Job Pool*. At the beginning, only entry job(s) will be submitted. Afterwards, upon notification by the *Executor* of the completion of a job, the *Planner* will determine if any dependant job(s) become ready and submit them. During the course of workflow execution, the job terminated due to resource failure is put back to the *Job Pool* by the *Executor* to be rescheduled later.
2) *Job scheduling*. Whenever there are resources available and a job is waiting in the *Job Pool*, the *Executor* will repeatedly do:
    a) Re-prioritize all jobs residing in the *Job Pool* based on individual job ranks in a real time fashion.
    b) Remove the job with the highest global priority from *Job Pool* to schedule;
    c) Schedule the job to the resource which allows the earliest finish time and will not fail during job execution. For the chosen job, the available resources are ordered by the estimated finish time starting from the earliest one. If the resource with higher preference, in terms of estimated finish time, is predicted to fail during the job execution, the next resource will be attempted. One may notice that the job execution time varies with resource and so does the failure prediction time window. If none of the resources can keep alive during the period of the chosen job execution, this job will remain in the *Job Pool* and next job will be picked out for scheduling. Otherwise, the job is scheduled and will run on the assigned resource. Fig. 3 describes this scheduling algorithm in more details.
3) *Job completion notification*. When a job finishes successfully, the *Executor* will notify the corresponding *DAG Planner* of job completion status.
4) *Failure Prediction*. The *Failure Predictor* will answer queries coming from the *Executor*: Will the resource X fail in next Y time units? Y is the estimated job execution time if the job is scheduled on resource X. The answer "YES" or "NO" drives the *Executor* make completely different scheduling decisions and therefore impose potentially great impact on the effectiveness of scheduler and overall application performance as well.

As each design comes with predefined objectives, the design of *FLAW* is to:

- *Reduce the loss time.* Accurate failure prediction will help the scheduler avoid placing jobs on a resource to fail in the middle of job execution. The abnormally terminated execution contributes to system resource waste, i.e., loss time caused by failures, including time spending on both unfinished data transmission and computation.
- *Reduce the number of job rescheduling.* Our system design does not utilize checkpoint and job migration techniques, as these two are arguably costly in practice. A failed job will be rescheduled later and start over. We envision that this metric will be of great interest to domain experts who are using HPC.
- *Reduce the makespan.* The *makespan* is the overall performance indicator for workflow applications and the effectiveness measure of a workflow scheduler.

```
T: a set of jobs in ready job pool
R: a set of free resources
rank: ranking values for all jobs

procedure schedule (T, R) {
  sort T as an array L so that:
        for any $i<j$, $L[i]\in T$ and $L[j]\in T$, $rank(L[i])\geq rank(L[j])$
  FOR i=1 TO size of L
    calculate the earliest finish time of L[i] on each resource $r\in T$
      if L[i] is assigned to r, and sort R as an array of N in increasing
      order of the estimate earliest finish time
    FOR j = 1 TO size of N
      predict if the N[j] will fail when job L[i] runs on N[j]
      IF YES
        INCREMENT j
      ELSE
        schedule job L[i] on resource N[j]
        $T = T - \{L[i]\}$
        $R = R - \{N[j]\}$
      END IF
    INCREMENT i
  END FOR
}
```

Fig. 3. The scheduling algorithm in *FLAW*.

C. *Application Aware Accuracy* (AAA)

The key success factor to the *FLAW* design is the accuracy of failure prediction, which is measured by how effectively the

Failure Predictor can answer the query: "Will the resource X fail in next Y time units?" The effectiveness of failure prediction can be quantified by the ratio of correct answers to total queries in context of job scheduling.

Different than the *AOA* defined earlier, the above ratio is application and prediction timing specific. For example, assuming that the present time is at time unit 0 , a node will go down between 100 and 120 time units and a job can be completed on this node by 140 time units if starting from now. It is further assumed that the *Failure Predictor* forecasts a failure will occur at time unit of 130, which is actually a false positive. In this case, the *Failure Predictor* can still give a correct answer to the query "if the node will be down in next 140 time units?" by telling "Yes".

We referred to this ratio as *Application Aware Accuracy* (*AAA*) and use it to measure the failure prediction effectiveness. Even though a higher *AOA* helps improve *AAA*, however, the *AAA* highly depends on the application behaviors and how and when the query is made. In an extreme example, if a resource is highly error prone and none of the failures on this resource is predicted, the *AAA* can still be very high if this resource is never a preferred one and no query is made about it. This sounds strange but can be very true in workflow scheduling. For instance, in a resource rich environment a node with very low capability can not produce completive earliest finish time for any job and is hardly considered in scheduling. And for a data intensive workflow application, in order to reduce cost on data movement the scheduler may narrow the resource choices to certain nodes which have executed many jobs and retain the data for next dependant jobs.

D. An Example

In this section, we illustrate the *FLAW* design by using examples of a workflow application and a failure trace, as shown in Fig.4. It is assumed that the sample DAG will run in a environment consisted of three nodes. The nodes encounter some failures as defined in the box of the figure.

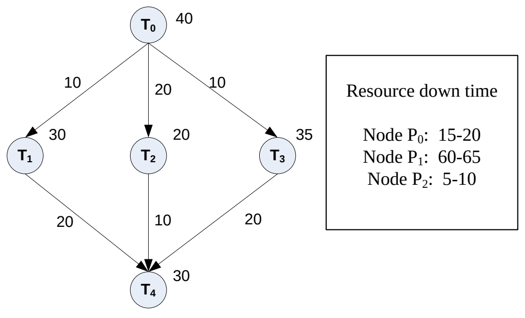


Fig. 4. Example of a DAG and failure trace.

We further assume that a *Failure Predictor* makes the following failure prediction: node P_0 fails at 18, node P_2 fails at 8 and 90 respectively, as shown by Fig. 5. This prediction achieves *AOA* accuracy of 50%, which includes 2 hits, 1 false positive and 1 false negative.

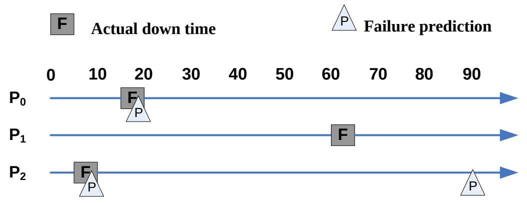


Fig. 5. Failure prediction with 50% of *AOA*.

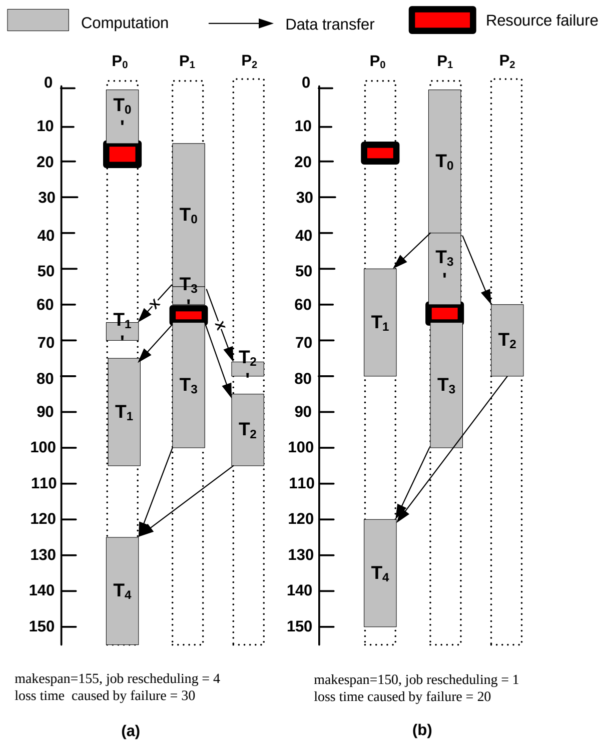


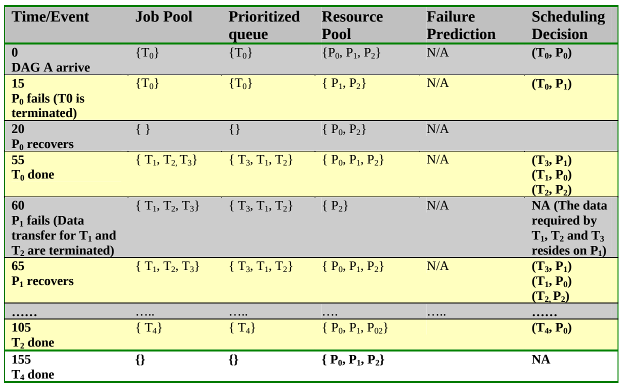
Fig. 6. Scheduling results: (a) *RANK_HYBD* without failure prediction; (b) *FLAW* with failure prediction of 50% of AOA.

Fig. 6 gives the scheduling result by the *RANK_HYBD* algorithm without failure prediction and the *FLAW* algorithm with presence of failures defined. It shows that the *FLAW* finishes the sample application with 150 time units *makespan*, 20 time units loss time and 1 job rescheduling. *FLAW* outperforms *RANK_HYBD* in all areas which completes with *makespan* of 155, loss time of 30 and 4 job rescheduling.

A detailed trace which records how each scheduling decision is made is illustrated in Fig. 7 and Fig. 8 for *RANK_HYBD* and *FLAW* respectively. *FLAW* finishes with 90% of *AAA* accuracy, as there are totally 10 queries and only one false negative prediction is made.

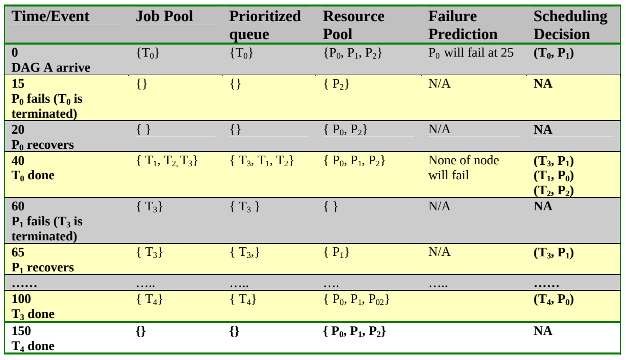
V. Performance Evaluation and Analysis

To verify the design of *FLAW* and study how failure prediction accuracy affects the scheduling effectiveness, we present the simulation design and result analysis in this section.



Time/Event	Job Pool	Prioritized queue	Resource Pool	Failure Prediction	Scheduling Decision
0 DAG A arrive	$\{T_0\}$	$\{T_0\}$	$\{P_0, P_1, P_2\}$	N/A	**(T_0, P_0)**
15 P_0 fails (T0 is terminated)	$\{T_0\}$	$\{T_0\}$	$\{ P_1, P_2\}$	N/A	**(T_0, P_1)**
20 P_0 recovers	{ }	{}	$\{ P_0, P_2\}$	N/A	
55 T_0 done	$\{ T_1, T_2, T_3\}$	$\{ T_3, T_1, T_2\}$	$\{ P_0, P_1, P_2\}$	N/A	**(T_3, P_1) (T_1, P_0) (T_2, P_2)**
60 P_1 fails (Data transfer for T_1 and T_2 are terminated)	$\{ T_1, T_2, T_3\}$	$\{ T_3, T_1, T_2\}$	$\{ P_2\}$	N/A	**NA (The data required by T_1, T_2 and T_3 resides on P_1)**
65 P_1 recovers	$\{ T_1, T_2, T_3\}$	$\{ T_3, T_1, T_2\}$	$\{ P_0, P_1, P_2\}$	N/A	**(T_3, P_1) (T_1, P_0) (T_2, P_2)**
……	…..	…..	….	…..	……
105 T_2 done	$\{ T_4\}$	$\{ T_4\}$	$\{ P_0, P_1, P_{02}\}$		**(T_4, P_0)**
155 T_4 done	**{}**	**{}**	**$\{ P_0, P_1, P_2\}$**		**NA**

Fig. 7. *RANK_HYBD* scheduling trace.



Time/Event	Job Pool	Prioritized queue	Resource Pool	Failure Prediction	Scheduling Decision
0 DAG A arrive	$\{T_0\}$	$\{T_0\}$	$\{P_0, P_1, P_2\}$	P_0 will fail at 25	**(T_0, P_1)**
15 P_0 fails (T_0 is terminated)	{}	{}	$\{ P_2\}$	N/A	**NA**
20 P_0 recovers	{ }	{}	$\{ P_0, P_2\}$	N/A	**NA**
40 T_0 done	$\{ T_1, T_2, T_3\}$	$\{ T_3, T_1, T_2\}$	$\{ P_0, P_1, P_2\}$	None of node will fail	**(T_3, P_1) (T_1, P_0) (T_2, P_2)**
60 P_1 fails (T_3 is terminated)	$\{ T_3\}$	$\{ T_3 \}$	{ }	N/A	**NA**
65 P_1 recovers	$\{ T_3\}$	$\{ T_3,\}$	$\{ P_1\}$	N/A	**(T_3, P_1)**
……	…..	…..	….	…..	……
100 T_3 done	$\{ T_4\}$	$\{ T_4\}$	$\{ P_0, P_1, P_{02}\}$		**(T_4, P_0)**
150 T_4 done	**{}**	**{}**	**$\{ P_0, P_1, P_2\}$**		**NA**

Fig. 8. *FLAW* scheduling trace.

A. Workload simulation

The published test bench [22] for workflow applications is used in the simulation. The test bench consists of randomly generated DAGs and is structured according to the following DAG graph properties:

- *DAG Size*: the total number of jobs in a DAG. As our goal is to evaluate the algorithm performance with intensive workloads, we only use the DAG group with the most jobs, i.e. the DAG consists of 175 to 249 jobs.
- *Meshing degree*: the extent to which the nodes are connected with each other.
- *Edge-length*: the distance between the connected nodes.
- *Node- and Edge-weight*. These two parameters describe the time required for a jobs computation and communication cost and are related to *CCR*, the communication to computation ratio.

As our simulation focuses on how to handle failures in scheduling, the DAGs we choose for this simulation are those being random on all properties of meshing degree, edge-length and node-weigh and edge-weigh. In order to utilize the real failure trace with granularity of minutes, we treat one time unit in DAG as 5 minutes.

The test bench [22] provides DAGs for test targeting environment of different scale measured by the total number of Target Processing Elements (TPE). A TPE can safely represent a node in a cluster system. We choose the DAGs designed for 32 TPEs in this simulation as this is a popular cluster scale in practice.

B. Failure traces and prediction accuracy

Studies in [2], [3], [4], [5], [6] recognize the temporal and spatial correlation of failures in large scale cluster systems. In order to mimic these failure characteristics in the simulations, we choose to use the failure traces published by Los Alamos National Laboratory [19].

In the simulation we extract 10 two-month failure traces from the real failure trace [19] by randomly picking up 32 nodes out of 49 nodes in the Cluster 2 and randomly choose two-month period for these 32 nodes between calendar year 2001 and 2002.

For each real failure trace, we randomly generate an associated prediction trace which is planed with random mixture of hits, false-positives and false-negatives to simulate different levels of *AOA*, i.e., 50%, 60%, 70%, 80% and 90% of *AOA* respectively. Five prediction traces are generated for each real failure trace at a defined *AOA* level. The simulation uses 10 actual failure traces and 250 generated failure prediction traces. Finally, the level of *AOA* is simulated by that the *Predictor* looks up prediction traces to answer the query.

C. Performance metrics

The evaluation is designed to study what is the right objective of failure prediction and how failure prediction affects the scheduling effectiveness, and the following metrics are measured against different levels of *AOA*:

- *Makespan*, which is the total execution time for a workflow application from start to finish. It is used to measure the performance of a scheduling algorithm from the perspective of workflow applications.
- *The loss time*, which is defined as the total time of partial execution including both data transmission and computation. It measures the system resource waste caused by resource failures.
- *The number of rescheduling jobs*, which is defined as the total number of job rescheduling which is caused by resource failures. If a node fails in the middle of job execution, the job is terminated and placed back to job pool for rescheduling.
- *Corresponding AAA*, which measures the effectiveness of failure prediction.

D. Simulation results and analysis

Our previous work [18] demonstrates that *RANK_HYBD* outperforms *FIFO* without resource failure presence, the simulation result in Fig. 9 further proves that *RANK_HYBD* based failure aware scheduler, i.e., *FLAW*, outperforms *FIFO* based one in terms of *makespan* when 10 concurrent DAGs are running in the system. As our interest is studying the impact of failure prediction, we do not further evaluate *FIFO*.

Most of the analysis below is to evaluate performance metrics against different levels of *AOA*. The simulation also

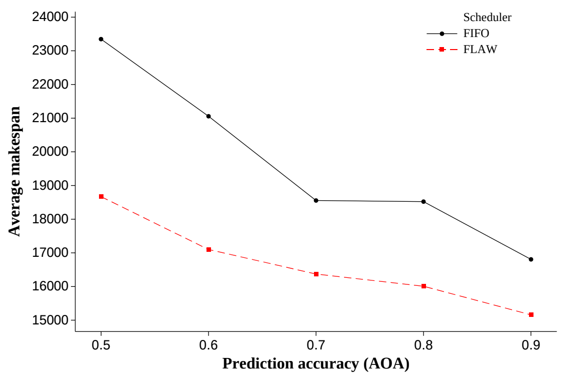


Fig. 9. *FIFO* vs *FLAW* with 10 concurrent DAGs.

includes two extreme situations: 1) *AOA* is 0, which means the scheduler is failure blind and does not predict failure at all (i.e., the basic *RANK_HYBD*); 2) *AOA* is 1.0, which means the scheduler knows exactly failure happens by looking up the actual failure trace.

It can be easily seen that with *AOA* increasing, workflow applications perform better in terms of *makespan*, as shown in Fig. 10. As the work load intensity increases measured by the number of concurrent DAGs, the performance improvement is even bigger.

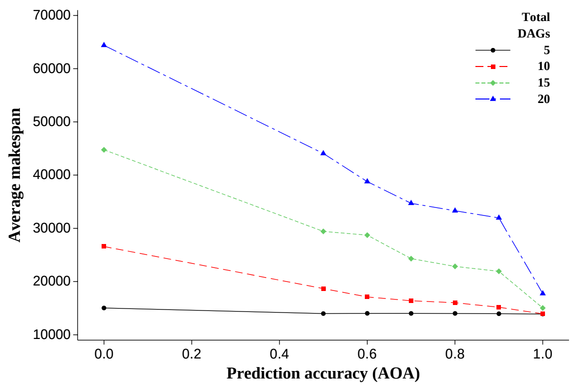


Fig. 10. Average *makespan* vs prediction accuracy (*AOA*).

Fig. 11 shows that, on average, higher level of AOA helps *FLAW* reduces the loss time considerably. Similarly, the number of rescheduling jobs is improved with more accurate failure prediction as shown in Fig. 12. Figure 13 reports the impact of predication accuracy on *makespan* of different workloads. We can see that the advantage of failure predication increases as the workload increases.

We also study whether false positives or false negative has more significant impact on scheduler effectiveness. In order to do that, for each one of 10 real failure traces, additional 10 prediction traces are generated with full spectrum of possible mixtures of Fp and Fn. The simulation is performed against total 200 generated traces with *AOA* levels of 50% and 60%. The Fig. 14 does not tell any correlation of average *makespan* and the percentage of $F_p/(F_p + F_n)$.

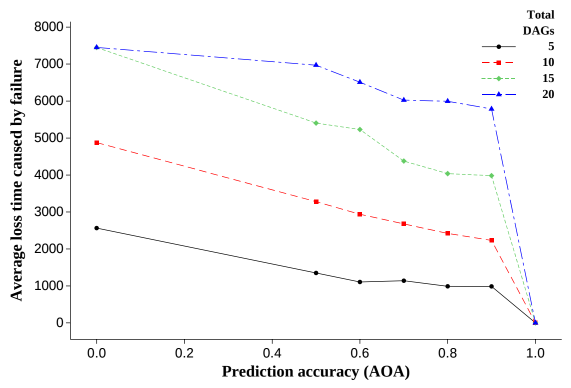


Fig. 11. Average loss time vs. prediction accuracy (*AOA*).

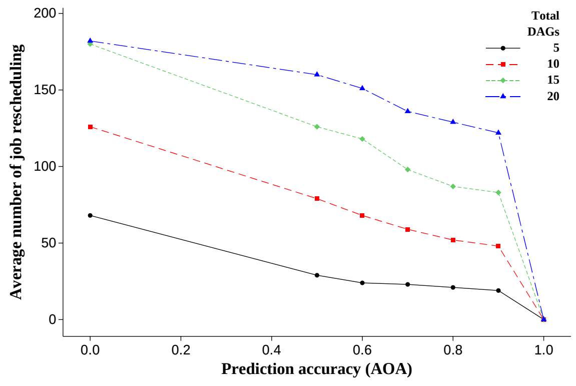


Fig. 12. Number of job rescheduling vs. prediction accuracy (*AOA*).

Finally we try to understand what is the right and practically achievable objective of failure prediction accuracy. Fig. 15 shows that the effectiveness measured by *AAA* is about 96% when *AOA* is as low as 50%, which indicates that a high *AAA* can be achieved with moderate *AOA*. As work load gets more intensive, a failure blind scheduler (i.e., *AOA*=0.0) can accomplish closer to 50% of *AAA*, and the *AAA* rate is more stable and increases steadily as *AOA* improves and *FLAW* performs well even with trivial *AOA*s.

VI. Summary and Future Work

In this paper, we present *FLAW*, and propose two new definitions of failure prediction accuracy in the context of workflow scheduling. Proactive failure handling is introduced into a dynamic scheduling scheme, *HYBD_RANK*, which significantly reduces the impact of failures on application performance in terms of *makespan*, loss time and the number of job rescheduling. A comprehensive simulation is conducted using real failure trace of a large scale cluster and a published workflow application test bench. The evaluation results not only demonstrate that *FLAW* can effectively improve the application performance in a failure prone high performance computing system but also shows that the *FLAW* can be very effective even with moderate failure prediction accuracy in a system with intensive workloads.

The paper presents a preliminary study of what is the

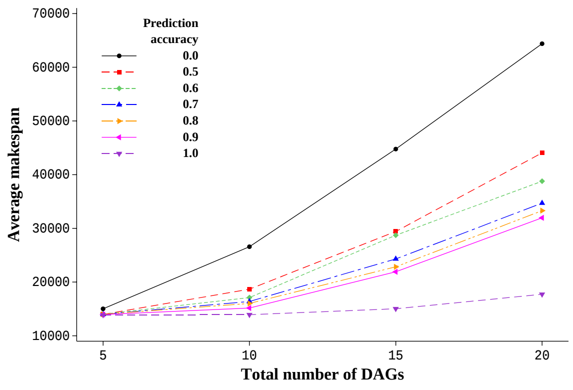


Fig. 13. Average *makespan* vs. the total number of DAGs.

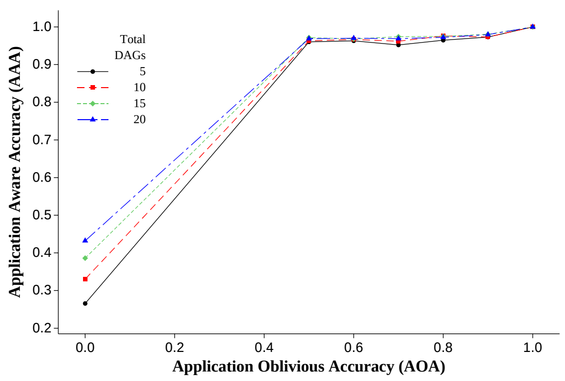


Fig. 15. The comparison between *AOA* and *AAA*.

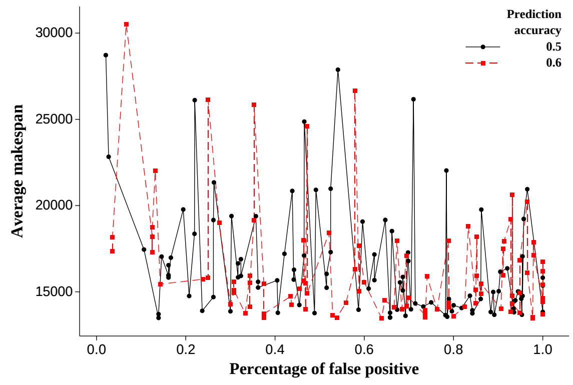


Fig. 14. Average *makespan* vs. the percentage of false positive.

right and practically achievable objective of failure prediction accuracy, further work is required to refine the definition of *AAA* and *AOA* so these metrics can be better measurable in practice. While this paper assumes the availability of an online failure predictor, we will need to implement *FLAW* with an actual one and integrate with popular workflow schedulers, such as DAGMan [15] or Pegasus [23].

References

[1] M. Garey and D. Johnson, *Computers and Intractibility: A guide to the Theory of NP-completeness*. W. H. Freeman, 1979.

[2] Y. Liang, A. Sivasubramaniam, and J. Moreira, "Filtering failure logs for a bluegene/l prototype," in *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 476–485.

[3] S. Fu and C. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'07)*, 2007.

[4] D. Oppenheimer *et al.*, "Service placement in shared wide-area platforms," in *Proceedings of the twentieth ACM symposium on Operating systems principles (SOSP'05)*. New York, NY, USA: ACM, 2005, pp. 1–1.

[5] Y. Zhang *et al.*, "Performance implications of failures in large-scale cluster scheduling," in *Proceedings of 10th International WorkshopJob Scheduling Strategies for Parallel Processing (JSSPP'04)*, 2004, pp. 233–252.

[6] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 249–258.

[7] P. Yalagandula *et al.*, "Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems," in *Proceedings of the Workshop on Real, Large Distributed Systems (WORLDS'04)*, 2004.

[8] X. Ren *et al.*, "Prediction of resource availability in fine-grained cycle sharing systems empirical evaluation," *Journal of Grid Computing,*.

[9] F. Salfner, M. Schieschke, and M. Malek, "Predicting failures of computer systems: a case study for a telecommunication system," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006.

[10] Y. Li and Z. Lan, "Exploit failure prediction for adaptive fault-tolerance in cluster computing," in *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 531–538.

[11] Y. Li *et al.*, "Fault-driven re-scheduling for improving system-level fault resilience," in *Proceedings of the 2007 International Conference on Parallel Processing (ICPP'07)*. Washington, DC, USA: IEEE Computer Society, 2007, p. 39.

[12] A. Oliner *et al.*, "Fault-aware job scheduling for bluegene/l systems," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium(IPDPS'04)*. IEEE Computer Society, 2004.

[13] S. Hwang and C. Kesselman, "Gridworkflow: A flexible failure handling framework for the grid," in *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*. Washington, DC, USA: IEEE Computer Society, 2003, p. 126.

[14] J. H. Abawajy, "Fault-tolerant scheduling policy for grid computing systems," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*. IEEE Computer Society, 2004.

[15] "Dagman." [Online]. Available: http://www.cs.wisc.edu/condor/dagman/

[16] A. Dogan and F. Özgüner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 308–323, 2002.

[17] "Planet lab." [Online]. Available: http://www.planet-lab.org

[18] Z.Yu and W. Shi, "A planner-guided scheduling strategy for multiple grid workflow applications," Wayne State University, Tech. Rep. MIST-TR-2007-006, 2007.

[19] "Los alamos national laboratory. operational data to support and enable computer science research," 2006. [Online]. Available: http://institutes.lanl.gov/data/fdata/

[20] F. Salfner and M. Malek, "Proactive fault handling for system availability enhancement," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 16*. Washington, DC, USA: IEEE Computer Society, 2005, p. 281.1.

[21] Z. Yu and W. Shi, "An adaptive rescheduling strategy for grid workflow applications,," in *Proceeding of 21st International Parallel and Distributed Processing Symposium (IPDPS'07)*, Long Beach, Florida, USA, March 2007.

[22] U. Hönig and W. Schiffmann, "A comprehensive test bench for the evaluation of scheduling heuristics," in *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS'04)*. IEEE, 2004.

[23] "Pegasus (planning for execution in grids)." [Online]. Available: http://pegasus.isi.edu/