9th IEEE/ACM International Symposium on Cluster Computing and the Grid

Performance under Failures of DAG-based Parallel Computing

Hui Jin, Xian-He Sun, Ziming Zheng, Zhiling Lan and Bing Xie

Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois 60616, USA
{hjin6,sun,zzheng11,lan,bxie3}@iit.edu

***Abstract*— As the scale and complexity of parallel systems continue to grow, failures become more and more an inevitable fact for solving large-scale applications. In this research, we present an analytical study to estimate execution time in the presence of failures of directed acyclic graph (DAG) based Scientific Applications and provide a guideline for performance optimization. The study is four fold. We first introduce a performance model to predict individual subtask computation time under failures. Next, a layered, iterative approach is adopted to transform a DAG into a layered DAG, which reflects full dependencies among all the subtasks. Then, the expected execution time under failures of the DAG is derived based on stochastic analysis. Unlike existing models, this newly proposed performance model provides both the variance and distribution. It is practical and can be put to real use. Finally, based on the model, performance optimization, weak point identification and enhancement are proposed. Intensive simulations are conducted to verify the analytical findings. They show that the newly proposed model and weak point enhancement mechanism work well.**

I. Introduction

Many scientific applications, e.g. workflow based applications, can be represented by directed acyclic graphs (DAG) [29]. The performance of a DAG application mainly depends on its scheduling. In general, a DAG scheduling consists of two parts: assigning the subtasks to resources (nodes), and ordering the execution of subtasks [23]. DAG scheduling is a well-known NP-Complete problem and various heuristic strategies are proposed [11].

DAG scheduling becomes more complex when faults are considered. System failures increase the uncertainty of DAG execution. A failure of a single subtask may block all the subtasks depending on it. The performance degradation might get amplified at each layer of the DAG and become substantial. Failures occur with certain probability and distribution. This uncertainty makes performance scheduling under failures extremely challenging.

In the field of high performance computing, production systems continue to grow in scale. For instance, systems composed of tens-of-thousands to hundreds-of-thousands of nodes are being designed and deployed [25]. For systems of this scale, failures become a major concern. Despite great efforts in the designing of ultra-reliable components, the increase of system size has outpaced the improvement of component reliability. In a parallel system composed of thousands of nodes, system failures may happen several times a day [18]. Conventionally, high performance computing research has mainly focused on performance. However, performance under failures has not received its deserved attention. This situation should be changed by developing fault awareness enabled computing environments for the next generation petaflops computers [20]. Therefore, in this study, we investigate the performance under failures of general DAG based parallel computing.

A DAG is a *scheduled DAG* once the scheduling decision has been made. Our study estimates the performance under failures of a scheduled DAG, identifies the weak points, and the methods of performance improvement.

We conduct this study step by step. We first predict the performance of subtasks based on our previous work [27], in which all subtasks are independent. This prediction provides the prediction of subtasks under one layer of a general DAG.

We next introduce a layered, iterative method to transform a scheduled DAG into a *layered DAG*, which reflects full dependencies among all the subtasks of a scheduled DAG.

Then we apply and cumulate the predicted performance on each layer to form the predicted performance under failures of the DAG. Distinguished from existing reliability modeling, our derived model not only estimates the mean DAG execution time, but also provides the variance and distribution, which are important factors to evaluate the performance.

Finally, we investigate performance optimization under failures. We introduce the concept of *weak points* - the nodes on the DAG that determine the performance, derive an algorithm to find weak points and identify the most unreliable weak point of a scheduled DAG. Improving the performance of the DAG becomes the enhancement of the reliability of the identified weak points, where known methods, such as replacement and duplication exist.

The rest of this paper is organized as follows. In Section II, we present a model to predict the performance of a scheduled DAG under failures. After the introduction of the mathematical model and assumptions of a DAG in II-A, a general failure model based on queuing theory is proposed to derive the subtask computation time in section II-B. In Section II-C, We adopt a layered approach to transform a scheduled DAG to a layered DAG, which reflects full dependencies among the subtasks. Finally in section II-D a DAG execution time estimation algorithm is presented to analyze the impact of failures. For performance improvement, in Section III, we present an algorithm to identify weak points. Simulations and experiments are conducted to verify the correctness of our analytical models in section IV. Related works are discussed in Section V. Finally, we conclude this study and discuss future works in Section VI.

II. DAG Performance under Failures

A. Problem Description

In a DAG, each node (subtask) represents a computation task. Edges between two subtasks represent a precedence constraint. We assume that the system uses space sharing mechanism so that each computing node executes computation subtasks one by one. Throughout this paper, the term subtask represents a computation subtask in a DAG, and node is a computing node in parallel systems.

Assume there are m nodes $M_i, i = 0, 1, ..., m-1$ in a system, and n subtasks $S_j, j = 0, 1, ..., n-1$ in the DAG. To schedule the DAG, each subtask is assigned to a node in the system, and subtasks execution order is determined by the scheduling. We use three random variables to describe the attributes of each subtask S_j: computation time T_j^C, start time T_j^S and finish time T_j^F. Clearly, $T_j^F = T_j^S + T_j^C$. Our goal is to analyze the execution time of a scheduled DAG in the presence of failures. Without loss of generality, we assume that the DAG starts at time zero.

Many computation intensive applications preload data before computing where the data transfer can be counted as part of the workload. Many other computation intensive applications have very low *communication to computation ratios* (CCR) [23], in which the communication cost is negligible[17]. In this paper, we focus on such computation intensive applications and do not consider communication cost explicitly.

B. Modeling of Subtask Computation Time

Suppose subtask S_j is assigned to node M_i, we study the property of T_j^C (i.e., the computation time of subtask S_j), based on the failure characteristics of M_i.

In [27], we have presented a performance model to estimate the mean, variance and distribution of a single sequential task computation time. We adopt this model to estimate the computation time of each subtask in the DAG.

We use an M/G/1 process [7] to describe system failures. Based on the common assumption in the reliability research that failures in a computer system are usually exponentially distributed (or the occurrence of failures is essentially random) [28, 4], we assume that for M_i, the arrival of failures follows a Poisson distribution with λ_f, the failure repair time follows a general distribution with mean μ_f and standard deviation σ_f, which is a generalization on the exponential downtime distribution used in [28, 4]. Notice that λ_f is the reverse of MTBF (Mean Time Between Failure) and μ_f reflects MTTR (Mean Time To Recovery).

Checkpointing has been widely used for fault tolerance in HPC. Hence we focus on modeling the performance influence of checkpointing in this study. Depending on implementation, checkpointing can be performed at either system-level or application-level. Its frequency can be fixed (periodically) or adaptive (adjusted at run time). We assume that the checkpointing/recovery cost follows a general distribution with mean μ_c and standard deviation σ_c. Here the checkpointing/recovery cost refers to the time required by the system to recover the application from the last checkpoint to the failure point. Let w denote the subtask workload, and the application execution is interrupted by failures S times, the computation time of the subtask can be expressed as,

$$T_j^C = X_1 + Y_1 + Z_1 + X_2 + Y_2 + Z_2 + ... + X_S + Y_S + Z_S + L \quad (1)$$

Where $X_i (1 \le i \le S)$ are the computing time consumed by the application, $Y_i (1 \le i \le S)$ are the downtime of system failures, $Z_i (1 \le i \le S)$ are the checkpoint/recover cost after failure interruption and L is the execution time of the last application process that finishes the application.

Please notice that μ_f in general is not the same as $E(Y_i)$. μ_f is the mean time of one single failure repair, whereas $E(Y_i)$ is the mean time of the system downtime due to failures. Failures could overlap each other and in general μ_f and $E(Y_i)$ are different.

Since $w = X_1 + X_2 + ... + X_S + L$, we get

$$T_j^C = w + Y_1 + Y_2 + ... + Y_S + Z_1 + Z_2 + ... + Z_S \quad (2)$$

The mean and variance of T_j^C can be obtained through the following expression:

$$E(T_j^C) = (\frac{1}{1 - \lambda_f \mu_f} + \lambda_f \mu_c) w \quad (3)$$

$$V(T_j^C) = (\frac{\mu_f^2 + \sigma_f^2}{(1 - \lambda \mu)^3} + \mu_c^2 + \sigma_c^2 + 2\frac{\mu_f \mu_c}{1 - \lambda_f \mu_f}) \lambda_f w \quad (4)$$

The cumulative distribution function (CDF) of the application computation time is expressed as:

$$P(T_j^C \le t) = P(T_j^C \le t | S = 0) P(S = 0) + P(T_j^C \le t | S > 0) P(S > 0) \quad (5)$$

Since $P(S = 0) = e^{-\lambda_f w}$, we have

$$P(T_j^C \le t) = \{ \begin{array}{l} e^{-\lambda_f w} + (1 - e^{-\lambda_f w}) P(U(S) \le t - w | S > 0) if\, t \ge w \\ 0, \quad otherwise \end{array} \quad (6)$$

where

$$U(S) = \{ \begin{array}{ll} 0, & if\, S = 0 \\ Y_1 + Z_1 + Y_2 + Z_2 + ... + Y_S + Z_S, & if S > 0 \end{array}$$

The first term $e^{-\lambda_f w}$ on the right-side of Formula (6) is the performance without failure interruption. The second term $(1 - e^{-\lambda_f w}) P(U(S) \le t - w | S > 0)$ is the performance with the failure interruptions. Based on [6], the Gamma distribution is an appropriate distribution to describe $P(U(S) \le u | S > 0)$, which is derivable from its mean and variance. Please refer [27] for detailed description and proof of the formulas.

C. DAG Transformation

To analyze the performance of a scheduled DAG, we need to investigate subtask dependencies in it. In general, there are two types of dependencies, *task dependency* and *resource dependency*. Task dependency reflects the dependency of each subtask given by the task DAG and resource dependency reflects the resource competition of the subtasks,which is determined by the underlying task scheduling. When multiple subtasks are assigned to a node, the subtask that is scheduled to start later has a resource dependency on the one that is scheduled to run earlier. To capture all the dependencies in a scheduled DAG, we transform it into a layered DAG [13].

Algorithm 1 DAG Transform Algorithm

```
Definition: Ready-to-Layer pool rPool is the collection of subtasks that all of whose immediate predecessors have been layered.
Objective: Transform a scheduled DAG to layered DAG
Begin
For each subtask $S \in DAG$
    Compute the immediate predecessors set $\Psi(S)$
    If $\Psi(S) = \emptyset$ then $Layer(S) = 1$ Else $Layer(S) = 0$ End If
End For
Repeat
    Update the ready-to-layer pool rPool.
    For each $S \in rPool$
        $Layer(S) = \max_{S' \in \Psi(S)}(Layer(S')) + 1$
        Add edges for non-redundant dependencies on $S$.
    End For
Until $Layer(S) > 0 \quad \forall S \in DAG$
End
```

The DAG transformation pseudocode is illustrated in Algorithm 1. After initializing the first layer, we repeatedly place subtasks in the Ready-to-Layer pool into the corresponding layer till all the subtasks are layered.

D. Modeling of DAG Execution Time

In this section we present the model of DAG execution time under failures. We first study the relationship among the three random variables introduced by Section II-A (i.e.T_j^C, T_j^S and T_j^F), followed by presenting an algorithm to get the DAG execution time under failures.

In a layered DAG, the start time of S_j is the maximum finish time of its immediate predecessors. We have

$$T_j^S = \begin{cases} 0 & if\, \Psi'(S_j) = \emptyset \\ max_{S_k \in \Psi'(S_j)}\{T_k^F\} & otherwise \end{cases} \tag{7}$$

Where $\Psi'(S_j)$ is the immediate predecessors set of subtask S_j.

Based on the probability theory, if $\Psi'(S_j) \neq \emptyset$, the CDF of T_j^S can be expressed as:

$$P(T_j^S \leq t) = \prod_{S_k \in \Psi'(S_j)} P(T_k^F \leq t) \tag{8}$$

The CDFs of T_j^C and T_j^S can be derived according to Equation (4) and (8). Given that $T_j^F = T_j^S + T_j^C$, according to probability theory, the probability distribution of the sum of two independent random variables is the convolution of each of their distributions. The formula is $f_{T_j^F}(\cdot) = f_{T_j^S}(\cdot) * f_{T_j^C}(\cdot)$, which means

$$f_{T_j^F}(y) = \int_0^y f_{T_j^S}(y-t) f_{T_j^C}(t)dt \tag{9}$$

Here $f_{T_j^F}(\cdot)$ is the density function of T_j^F. $f_{T_j^S}(\cdot)$ and $f_{T_j^C}(\cdot)$ are the density functions of T_j^S and T_j^C, respectively.

The pseudocode to estimate the distribution of DAG execution time under failures is shown in Algorithm 2. Basically, in the outer loop we propagate the execution time of each layer of a layered DAG, and in the inner loop we derive the distribution of three random variables for each subtask in the layer.

Algorithm 2 Execution Time Estimation Algorithm for a Layered DAG

```
Definition: $T_{dag}$ denotes the DAG execution time. $\Omega_{dag}$ is the set of exit subtasks for the DAG.
Objective: Estimate the distribution of execution time of a scheduled DAG
Begin
Transform a scheduld DAG to layered DAG.
For $i = 0$ to depth(layered DAG)
    For each subtask $S_j$ in layer $i$
        Calculate the start time $T_j^S$ CDF in this layer based on formula (8).
        Calculate the computation time $T_j^C$ CDF based on formula (6).
        Calculate the density function of $T_j^S$ and $T_j^C$ based on their CDFs.
        Calculate the density function of finish time $T_j^F$ based on formula (9).
        Calculate the CDF of $T_j^F$ from its density function
    End For
End For
Overall DAG execution time is the result of multiplication on finish times of the all the exit subtasks,
$P(T_{dag} \leq t) = \prod_{S_k \in \Omega_{dag}} P(T_k^F \leq t)$
End
```

III. Identify Weak Point

A *critical path* (CP) of a DAG is a longest path in the DAG from one entry subtask to one exit subtask [11]. A weak point is defined as a node on a critical path of a scheduled DAG that is weak in reliability.

Variance is an important factor of performance. We use $E(T^F)(1 + Coe(T^F))$ to measure a subtask finish time, not

$E(T^F)$. Similarly, we use $E(T^C)(1 + Coe(T^C)) - workload$ to measure the reliability of a subtask, and its corresponding node. Here $E(T^F)$, $Coe(T^F)$ and $E(T^C)$, $Coe(T^C)$ represent the mean and the coefficient of variance of the finish time and computation time respectively for one subtask. We propose an efficient algorithm to identify a candidate weak point for possible performance improvement.

The algorithm consists of two parts, finding the critical path and identifying a candidate weak point. For a collection of subtasks (normally parallel subtasks), we denote *critical subtask* as the subtask with the maximum finish time. According to Section II , the execution time of a DAG is determined by critical exit subtask. If a subtask is located at layer 2 or higher, its start time is decided by its critical immediate predecessor (the start time of subtask in layer one is zero). Starting at a critical exit subtask, we can find its critical immediate predecessor subtask, this process can repeat on these predecessors until an entry subtask (subtask located at layer 1) is reached. The critical exit subtask and its critical predecessor recursively form the critical paths.

The algorithm to identify a candidate weak point for reliability/performance enhancement is given below. We first find critical path of a layered DAG recursively by finding the predecessor of the critical exit subtask. The finish time of a critical exit subtask is the execution time of the DAG (assuming the DAG starts at time zero) and it is also the weight of the critical path which ends at the critical exit subtask. Similarly, for a subtask S_j, the finish time of its critical immediate predecessor reflects the weight of the longest path from one entry subtask to S_j. The performance of the nodes in a critical path directly determines the performance of the DAG. All the nodes on a critical path are important. But from the reliability and fault tolerant point of view, only those nodes not reliable are weak. We find a critical subtask with the maximum $E(T^C)(1+Coe(T^C))-workload$ and identify its corresponding node as the candidate weak point for possible performance enhancement. Please notice that for every subtask, the mean, coefficient of variance for the computation time and finish time can be derived in the process of DAG performance estimation with no extra cost. The cost of the algorithm is very low. The algorithm can be applied repeatedly to enhance weak points one by one.

Algorithm 3 Algorithm to Find a Candidate Weak Point

```
Definition: Critical Path Stack $Stack_{cp}$ is the stack used to store the subtasks in the critical path.
Objective: Identify the weakpoint in a layered DAG
Begin
Find the critical exit subtask from all the exit subtasks in a layered DAG, push it to $Stack_{cp}$
$S_c = top(Stack_{cp})$
While $layer(S_c) \neq 1$
    Find the critical immediate predecessor of $S_c$ and push it to $Stack_{cp}$
    $S_c = top(Stack_{cp})$
End While
$max = -1$    $weakP = NULL$
While $Stack_{cp} \neq \emptyset$
    $S_j = pop(Stack_{cp})$
    $w = E(T_j^C)(1 + Coe(T_j^c)) - workload_j$
    $if \quad w > max \quad then$
        $max = w$
        $weakP = S_j$
    $end \quad if$
End While
Return $WeakP$
```

IV. Experiments

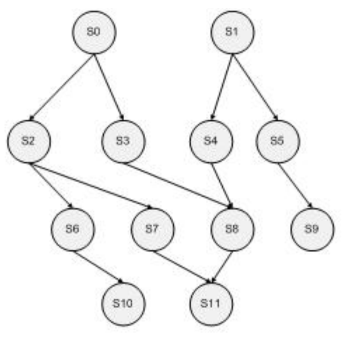
We use two DAGs to evaluate our model: one is a synthetic DAG with considerable complexity, and the other is a DAG from the LQCD application [14]. We also study the model sensitivity with respect to different workloads and number of subtasks.

The first DAG studied in our simulation is shown in Figure 1.a. We have 12 subtasks in the task DAG. The workload of each subtask is a multipliable to test the accuracy and sensitivity for different workloads. The second DAG is shown in Figure 1.b, which is a prototype of LQCD workflow, two-point (2-pt) analysis. LQCD (Lattice QCD) is the numerical simulation of QCD[16]. Its calculations allow us to understand the results of particle and nuclear physics experiments in terms of QCD [16]. In our experiment the number of subtasks in level 3 ($S_9, S_{10}, ..., S_{k-1}$) is a variable. By changing the number of subtasks, we can test the effectiveness of the model for different number of subtasks.

We simulate a 256-node system. For each node, the mean failure arrival rate ranges from 1.5×10^{-3} to 2.5×10^{-3} per hour and the mean failure downtime ranges from 2 hours to 4 hours. Fault recovery overhead in our simulation is set to be 2 hours. The overhead includes both checkpointing cost and the rollback cost. The standard deviations of both downtime and recovery cost are 2. The parameters on each node are randomly generated within their corresponding ranges so that the failure rate and downtime on each machine are independent. Failure downtime and recovery overhead follow lognormal distributions. The choice of parameter values and distributions are based on Los Alamos data [12, 18].

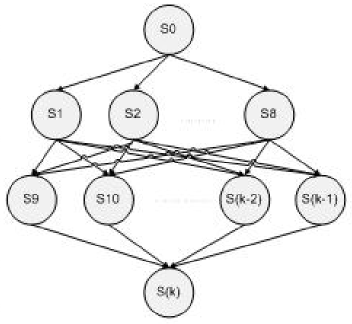
The scheduling algorithms employed in our experiments include list schedules such as HLEFT, HLFNET, SCFNET, etc [11]. The experimental results prove that the accuracy of the proposed model is independent of scheduling algorithms.

For each scheduled DAG, we issue 20,000 runs, then we get a sample set of size 20,000 to be used as our simulation results. We calculate the mean μ, standard deviation σ and the CDF of the simulation results. The model is evaluated in three aspects. For a sample set of normal distribution with mean μ and standard deviation σ, almost all (99.7%) of the values in a normal distribution sample set lie within a range of $[\mu \pm 3\sigma]$[8]. To test whether our model predicts an accurate range of DAG execution time, first we plot error bars within the range $[\mu - 3\sigma, \mu + 3\sigma]$for both the prediction results and



Subtask	Workload
s_0	10
s_1	20
s_2	25
s_3	30
s_4	20
s_5	35
s_6	20
s_7	30
s_8	20
s_9	30
s_{10}	5
s_{11}	20

(a) A Synthetic DAG and Base Workloads (Hours)



Subtask	Workload
s_0	2
s_1	32
$\cdots$	32
s_8	32
s_9	128
$\cdots$	128
s_{k-1}	128
s_k	4

(b) 2-pt DAG and Workloads (Hours)

Fig. 1: DAGs in The Experiments

the simulation results. We introduce *average residual error* (or *average residual* for short) to show the goodness of the predicted performance distribution. Residual is defined as the sum of vertical distances between the simulated points and the predicted model curve [8]. Suppose $Q_1, Q_2, ..., Q_n$are n identical observations of DAG execution times in the simulation, we have $AverageResidual = \sum_{j=1\,to\,n} \frac{|Predicted(P(T_j))-Simulated(P(T_j))|}{n}$. Smaller average residual indicates better fitness of our model with the simulation. Through this section, the term "predicted" refers to the results derived from the model, the term "simulated" means the results gained from the simulation. In the simulation, we also study the prediction mean error, which is defined as $\frac{|Predicted\,Mean-Simulated\,Mean|}{Simulated\,Mean}$ [26].

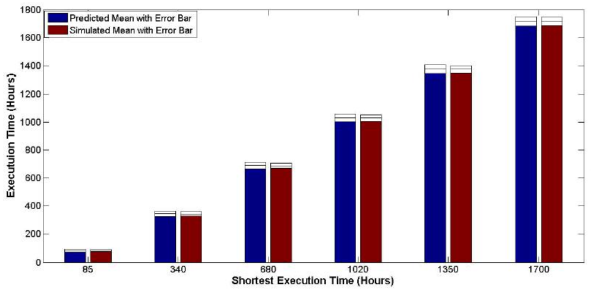
The simulation results for the synthetic DAG of Figure 1.a are plotted in Figure 2. In Figure 2.a we demonstrate the means with error bars for different workloads. The workloads are multiplied by 1, 4, 8, 12, 16 and 20 times. The corresponding *shortest execution time*, the DAG execution time without failure interruption, is marked in x-axis. The colored bars represent $[0, \mu - 3\sigma]$ for prediction and observation results, while error bars of $[\mu - 3\sigma, \mu + 3\sigma]$ are indicated by the uncolored bars on the top of colored bars. The first observation is that all the simulated error bars are covered by the predicted error bars, which implies that almost all the simulated results can be predicted by the model. Further, for each identical workload, the predicted and simulated error bars are quite close to each other, which convinces that we can provide a precise prediction of the performance range.

The average residual of Figure 2.b indicates the accuracy of CDF predicted by the model. The average residuals are kept less than 4% through our simulation, which means that the average prediction error for the CDF is better than satisfactory. Also, we do not see the loss of accuracy with the increase of workload: the average residual even decreases for an expected execution time of more than 1700 hours. In real applications we rarely have a task that will run as long as 8500 hours, we just list this data to prove the stability of our model with the increase of workload. Mean error bars are also plotted in Figure 2.b and we can find that all the observed mean error are kept less than 0.5%.

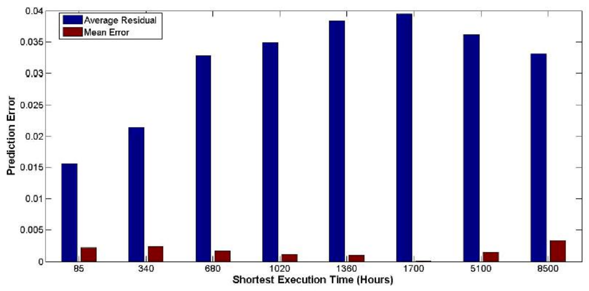
In the simulation of 2-pt analysis DAG shown in Figure 3, we set the number of subtask in level 3 as a variable ranged from 32 to 1024. The more number of subtasks implies more parallelism of the application.

In Figure 3.a, we notice that with the increase of subtask numbers, the portion of simulated error bars that cannot be covered by the predicted error bars is remained as stable as 15%, which means that we can predict the range of DAG execution time with 85% accuracy or more. Another encouraging observation of Figure 3.a is that when the number of subtasks in level 3 is more than 128, the standard deviation prediction error will be remained below 5%. This indicates that the difference between predicted error bars and simulated error bars is mostly due to the prediction mean error for more parallel applications. Standard deviation prediction error contributes little to the error bar difference. For example, for the DAG with 1024 subtask at level 3, the predicted standard deviation is 10.0972, compared with 10.5492 of simulated standard deviation. So the mismatching of error bars is mostly due to the mean difference: 594.158 of predicted mean compared with 583.33 of simulated mean. This fact inspires us to work toward more accurate mean prediction to improve the model in the future.

In Figure 3.b we plot the average residuals and mean errors for DAGs with different number of subtasks. The average residual are kept lower than 18%, or less than 15% when the size of level 3 is more than 128. The mean prediction error is stabled at less than 8% or less than 6% when the size of level

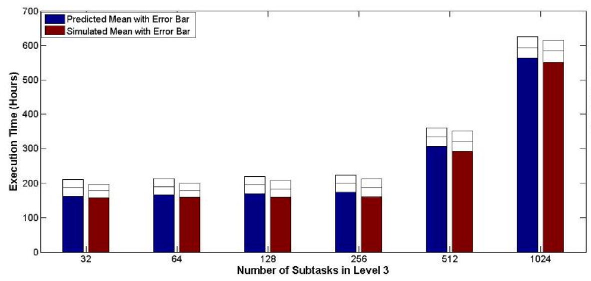


(a) Mean with Error Bars for the Synthetic DAG

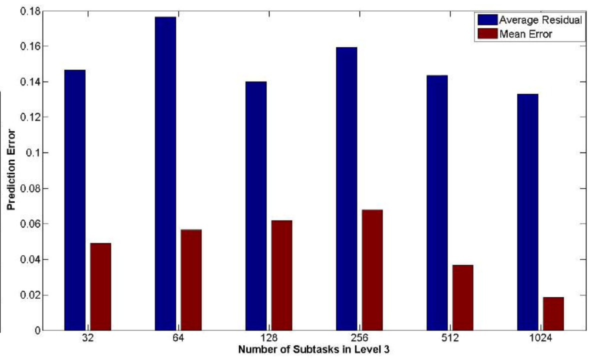


(b) Average Residual and Mean Error for The Synthetic DAG

Fig. 2: Simulation Results for The Synthetic DAG



(a) Mean with Error Bars for 2-pt DAG



(b) Average Residual and Mean Error for 2-pt DAG

Fig. 3: Simulation Results for 2-pt DAG

3 is more than 256. Though both average residual and mean error is bigger than those in Figure 2.b, they are acceptable due to the high workload. For example, compared with the shortest execution time of 1024 subtasks at level 3 of 550 hours, the prediction error of 10 hours is trivial. We conclude the model is scalable, based on the observation that both average residual and mean error have the trend of decreasing with the increase of the number of subtasks.

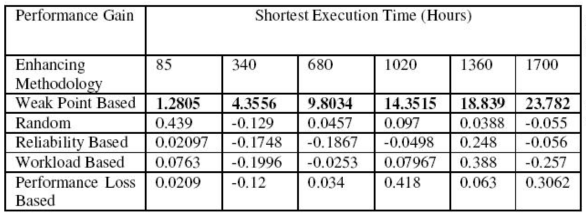
Simulations are conducted to evaluate the efficiency of the proposed weak point identification algorithm. We assume a failure-free node is used to replace the identified weak point to enhance the DAG performance. We compare several performance enhancing technologies:

- Weak Point Based: we enhance the weak point identified by the proposed algorithm with a failure-free node.
- Random: nodes are randomly selected to be enhanced by the failure-free node.
- Reliability Based: The least reliable node of the system is replaced. The reliability of a node is approximated by $\lambda_f \times (\mu_f + \sigma_f + \mu_c + \sigma_c)$. The symbols share the same meaning with those in Section II-B.
- Workload Based: we simply enhance the subtask with highest workload.
- Performance Loss Based: The node with maximum performance loss is selected for enhancement. The performanceloss of subtask S_j is defined as $E(T_j^C)(1 + Coe(T_j^C)) - workload_j$.

The failure data used in the experiment are from 32 representative computing nodes of [12]. We introduce *Performance Gain*, the time saved by issuing the enhancing technology, as the metric to evaluate different methodologies.

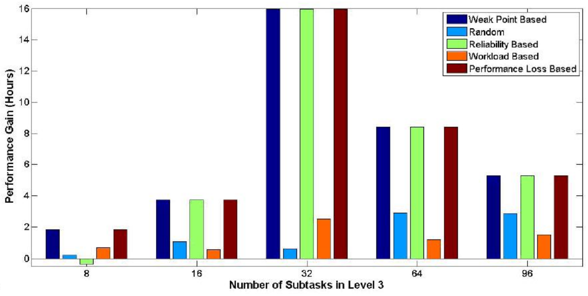
In Figure 4.a we illustrate the performance gains for the synthetic DAG of Figure 1.a with different enhancing technologies and workloads. The advantage of weak point based enhancing technology is obvious. We can improve the DAG performance effectively by replacing the node of weak point with an ultra reliable node. Other enhancing approaches, with performance gains less than 1, rarely improve the DAG performance compared with the optimized weak point solution. The negative gains are generated by the random runs of each DAG and can be considered as zero.

The experimental results for 2-pt DAG are plotted in Figure 4.b. Unlike Figure 4.a, the weak points in Figure 4.b are always the nodes with least reliability and performance loss. This is due to that 2-pt DAG is more symmetrical than the synthetic DAG. The workloads for the subtasks in the same level are identical, which introduce many "homogeneous" candidates for the critical path. As a result, the most unreliable node with most performance loss will outperform their peers and be identified as the weak point. This observation suggests that



Performance Gain	Shortest Execution Time (Hours)					
Enhancing Methodology	85	340	680	1020	1360	1700
Weak Point Based	**1.2805**	**4.3556**	**9.8034**	**14.3515**	**18.839**	**23.782**
Random	0.439	-0.129	0.0457	0.097	0.0388	-0.055
Reliability Based	0.02097	-0.1748	-0.1867	-0.0498	0.248	-0.056
Workload Based	0.0763	-0.1996	-0.0253	0.07967	0.388	-0.257
Performance Loss Based	0.0209	-0.12	0.034	0.418	0.063	0.3062

(a) Performance Gains for The Synthetic DAG



(b) Performance Gains for 2-pt DAG

Fig. 4: Performance Gains for Different Enhancement Techniques

more effective algorithms can be conducted for DAGs with featured structure, e.g., fork-join DAG.

Figure 4.b also differs from Figure 4.a in that the optimal performance gains do not grow proportionally with the increase of subtask number. They initially increase with the number of subtask grows, and diminishes after arriving at the peak when the number is 32. The performance gain grows at first because failures have more impact on more parallel DAG and the enhancement can remedy the loss. However, in the experiment we only have one ultra reliable node for use, which is not enough to compensate the performance loss when the DAG scales beyond some threshold. More reliable nodes are needed to keep the performance gain grow proportionally. The identification of multiple weak points for replacement can be achieved by applying our proposed algorithm repeatedly. The optimal number of weak point replacement for DAG of a specific size is an interesting topic and will be the task of future work.

V. Related Work

In [27], we have proposed a queuing theory-based approach to estimate the CDF, as well as mean and variance of sequential task execution time. The models separate the influence of failure rate, failure repair time, checkpointing period and cost. The parallel task addressed in [27] is assumed to be "completely parallel", which means that there is no dependency among the subtasks. In this study, we present a general model for any application that can be represented by a DAG, where dependency is considered. The models proposed in this study can be applied to general HPC environment to predict the application performance. In addition, we have introduced the concept of weak point to enhance performance under failures for a general DAG based workflow system.

In probability theory, mean is a way to describe the location of a distribution, while the variance is a way to capture its scale and degree of distribution. The combination of mean and variance provides a comprehensive description of a random variable [9, 10]. Most existing models only provide the mean of execution time. One advantage of our model is that it provides both mean of the DAG execution time and the variance and distribution. The latter is vital for layered DAG performance analysis where the distribution of variance can be amplified through layers and lead to an execution time far from the mean.

With the assistance of distribution and variance, we provide the range of DAG execution time, the percentile that the DAG can be done within some time. Unlike existing studies [28, 4, 5], our model provides the CDF of job execution time which can be used to derive the percentile. Mean expected execution time alone is not a good metric for choosing an appropriate scheduling; the distribution of variance must be considered in practice. That makes our model unique. A user can use the estimated execution time and its distribution to choose an appropriate scheduling.

In [21,19], DAG scheduling algorithms are designed to maximize the reliability. Their goal is to minimize the number of failures during the DAG execution. Heuristic methods, e.g. high workload tasks are allocated to more reliable nodes, are devised to schedule a DAG. However, in their papers they did not consider the impact of other parameters such as repair time and checkpointing, which may significantly degrade the application performance [27]. Our study provides a more comprehensive solution in that we focus on DAG execution time under the impact of almost all the failure related parameters, i.e., MTBF, MTTR, checkpointing cost and their standard deviations. The scheduling is made not only based on the mean or medians, the coefficient of variance is also considered.

VI. Conclusion and Future Work

In this study, we have proposed performance models to estimate the execution time of DAG in the presence of failures. We have utilized a layered approach to capture both resource dependency and task dependency among the subtasks in a DAG and then derived the distribution of execution time of the layered DAG. We have also introduced the concept of weak points to enhance performance under failures.

Experiments have shown that the newly proposed models work well with an accuracy of more than 85%. The identification of weak point can also be used effectively to improve the DAG performance. This research of performance under failures is aimed to large-scale parallel computing and has a real potential for the petaflops high-end computing era.

In the future, we plan to further study the practical part of the proposed models, their applications in task scheduling and reliability enhancement. We're currently working on the development of a fault aware computing environment for parallel computing based on these models. We are also interested in integrating the proposed failure models with existing job management middleware, e.g. PBS, Condor.

Acknowledgement

We would like to thank our colleagues at the LQCD workflow group at the Fermi National Accelerator Laboratory for their valuable inputs on the LQCD workflow test cases. This work was supported in part by National Science Foundation under NSF grants CNS0834514, CNS0751200, CNS0720549, CCF0702737, CNS0406328, EIA0224377, and by Department of Energy SciDAC program under the contract No. DOE, DE-FC02-06 ER41442.

References

[1] T.L. Adam, K.M. Chandy, and J.R. Dickson. "A comparison of list scheduling for parallel processing systems," *Commun.*ACM 17, pp. 685-690, Dec. 1974.

[2] E. G. Coffman, and P.J. Denning, *Operating Systems Theory*. Prentice-Hall, Englewood Cliffs, N.J. 1974.

[3] G. Decanda, D. Hastonrun, M. Jampani, G. Kakulapati., etc. "Dynamo: Amazon's Highly Available Key-value Store," *Proceedings of SOSP'07*. Oct, 2007.

[4] A. Duda, "The Effects of Checkpointing on Program Execution Time," *Information Processing Letters*, vol. 16, pp. 221-229, June 1983.

[5] S. Garg, Y. Huang, C. Kintala, K.S. Trivedi, "Minimizing Completion Time of a Program by Checkpointing and Rejuvenation," *Proceedings of 1996 ACM SIGMETRICS Conference*, pp. 252-261, Philadelphia, PA, May 1996.

[6] L. Gong, X.-H. Sun, and E. Waston, "Performance Modeling and Prediction of Non-Dedicated Network Computing," *IEEE Trans. on Computers*, Vol 51, No 9, pp. 1041-1055, Sep., 2002.

[7] D. Gross, C. M. Harris, *Fundamentals of Queuing Theory*, 3rd Edition, John Wiley Sons, 1998.

[8] R. Jain., *The art of Computer Systems Performance Analysis*, John Wiley Sons, 1991,

[9] A. Kamthe and S.Y. Lee, "A Stochastic Approach to Estimating Earliest Start Times of Nodes for Scheduling DAGs on Heterogeneous Distributed Computing Systems," *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*

[10] A. Kagan, L. A. Shepp, "Why the variance," *Statistics Probability Letters.* Vol 38, Issue 4, pp. 329-333, July 1998.

[11] Y.K. Kwok, I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys (CSUR)*, Vol 41, Issue 4, pp. 406-471, Dec. 1999

[12] Los Alamos National Laboratory, Operational Data to Support and Enable Computer Science Research, *http://institute.lanl.gov/data/lanldata.shtml*

[13] Y.A.Li and J.K.Antonio, "Estimating the execution time distribution for a task graph in a heterogeneous computing system," *proceedings of the 6th Heterogeneous Computing Workshop (HCW'97)*, p.172, April 1997.

[14] Lattice QCD Project, *http://www.usqcd.org/*

[15] C.D. Lu, "Scalable Diskless Checkpointing. for Large Parallel Systems," Ph.D dissertation, Department of Computer Science. University of Illinois at Urbana-Champaign, 2005.

[16] L. Piccoli, X.-H. Sun, J. N. Simone, D. J. Holmgren, etc. "The LQCD Workflow Experience: What Have We Learned," *Posters of ACM/IEEE SuperComputing Conference. 2007 (SC'07)*, Nov. 2007.

[17] P. Ratn, F. Mueller, M. Schulz and B. de Supinski, "Preserving Time in Large-Scale Communication Traces," *Proceedings of International Conference on Supercomputing*, Jun 2008, pages 46-55.

[18] B. Schroeder, G. A. Gibson, "A large-scale study of failures in high-performance computing systems," *Proceedings of the 2006 International Conference on Dependable Systems and Networks*, Philadelphia, PA, June 2006.

[19] S. Shatz, J. Wang, and M. Goto, "Task Allocation for Maximizing Reliability of Distributed Computer Systems," *IEEE Trans. on Computers*, Vol 41(9), 1992,pp. 1156 - 1168

[20] X.-H. Sun, Z. Lan, Y. Li, H. Jin, and Z. Zheng, "Towards a Fault-aware Computing Environment," *Proceedings of the High Availability and Performance Computing Workshop (HAPCW)*, Mar. 2008.

[21] S. Srinivasan, and N.K. Jha, "Safety and Reliability Driven Task Allocation in Distributed Systems," *IEEE Trans. Parallel and Distributed Systems*, Vol 10(3), 1999, pp. 238-251

[22] A.B. Tayyab and J.G.Kuhl, "Stochastic Performance Models of Parallel Task Systems," *proceedings of 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp.284-285, May 1994.

[23] H. Topcuoglu, S. Hariri and M.Y Wu, "Performance Effective and low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. on Parallel and Distributed Systems*, Vol13, NO. 3, pp.260-274, March 2002.

[24] H. Topcuoglu, S. Hariri, and M.Y. Wu, "Task scheduling algorithms for heterogeneous processors," *Proceedings of the Eighth Heterogeneous Computing Workshop, 1999(HCW'99)*, pp. 3-14, April 1999.

[25] Top 500 Supercomputing Website. *http://www.top500.org*

[26] R. Wolski, "Dynamically forecasting network performance using the network weather service," *Cluster Computing*, Vol 1, pp. 119-132, 1998.

[27] M. Wu, X.-H. Sun and H. Jin, "Performance under Failure of High-End Computing," *Proceedings of the ACM/IEEE SuperComputing Conference. 2007 (SC'07)*, Nov. 2007.

[28] J. W. Young, "A First Order Approximation to the Optimal Checkpoint Interval," *Comm. ACM*, Vol 17, No 9, pp. 530-531, 1974.

[29] J. Yu and R. Buyya, "A Taxonomy of Scientific Workflow Systems for Grid Computing," *Special Issue on Scientific Workflows, ACM SIGMOD Record*, Vol34, NO 3, ACM Press, Sept. 2005.