Computing the expected makespan of task graphs in the presence of silent errors

Henri Casanova*, Julien Herrmann† and Yves Robert†‡
*Information and Computer Sciences Dept., University of Hawai‘i, USA
Email: henric@hawaii.edu
†École Normale Supérieure de Lyon, France
Email: {julien.herrmann|yves.robert}@ens-lyon.fr
‡University of Knoxville, USA

Abstract—**Applications structured as Directed Acyclic Graphs (DAGs) of tasks correspond to a general model of parallel computation that occurs in many domains, including popular scientific workflows. DAG scheduling has received an enormous amount of attention, and several list-scheduling heuristics have been proposed and shown to be effective in practice. Many of these heuristics make scheduling decisions based on path lengths in the DAG. At large scale, however, compute platforms and thus tasks are subject to various types of failures with no longer negligible probabilities of occurrence. Failures that have recently received increasing attention are "silent errors," which cause a task to produce incorrect results even though it ran to completion. Tolerating silent errors is done by checking the validity of the results and re-executing the task from scratch in case of an invalid result. The execution time of a task then becomes a random variable, and so are path lengths. Unfortunately, computing the expected makespan of a DAG (and equivalently computing expected path lengths in a DAG) is a computationally difficult problem. Consequently, designing effective scheduling heuristics is preconditioned on computing accurate approximations of the expected makespan. In this work we propose an algorithm that computes a first order approximation of the expected makespan of a DAG when tasks are subject to silent errors. We compare our proposed approximation to previously proposed such approximations for three classes of application graphs from the field of numerical linear algebra. Our evaluations quantify approximation error with respect to a ground truth computed via a brute-force Monte Carlo method. We find that our proposed approximation outperforms previously proposed approaches, leading to large reductions in approximation error for low (and realistic) failure rates, while executing much faster.**

I. Introduction

This paper introduces a new algorithm to approximate the expected makespan of a workflow application, i.e., an application structured as a Directed Acyclic Graph (DAG) of tasks, in which tasks can fail and must be re-executed. A key question when executing workflows on parallel platforms is the scheduling of tasks on the available compute resources, or processors. When not considering task failures, list scheduling algorithms are the de-facto standard [?], and tools are available that use such algorithms for scheduling workflow applications onto large-scale platforms in practice [?], [?]. The prevalent list scheduling heuristics prioritize tasks with large bottom levels. The bottom-level of a task is defined as the longest path from that task to the end of the execution, assuming unlimited processors. This heuristic is known as CP-scheduling (*Critical Path* scheduling [?], [?], [?]), and it has been extended to handle heterogeneous environments (see the HEFT algorithm [?]).

It is well-documented that large-scale platforms are increasingly subject to errors that can cause task failures. In particular, the occurrence of "silent errors" (or SDCs, for Silent Data Corruptions) has been recently identified as one of the major challenges for Exascale [?]. Silent errors can be caused by external causes, including cosmic radiations or packaging pollution. In addition, silent errors can occur when using DVFS (Dynamic Voltage Frequency Scaling) to reduce energy consumption. For instance, at low voltage the probability that a task produces incorrect results increases [?], [?].

Regardless of their causes, an effective approach for avoiding propagating results corrupted by silent errors is to use a verification mechanism after executing each task [?]. When an error is detected, the task is then re-executed. The verification mechanism can be general-purpose (e.g., based on replication [?], with re-execution only when the two outputs do not match) or application-specific. Many application-specific error detection methods are available for classical High performance Computing (HPC) applications, such as approximate re-execution for ODE and PDE solvers [?], orthogonality checks for Krylov-based sparse solvers [?], [?]), or Algorithm-Based Fault Tolerance (ABFT) for numerical linear algebra [?].

Silent errors make it challenging to define efficient list scheduling algorithms to schedule workflow applications. Indeed, after the first execution of a task, the error detector is used to check the result. If the result is correct, the task's execution is marked as successful, and its successor tasks are marked as ready. If the result is incorrect, then the task must be executed (and, given that the first execution has failed, the second execution will typically succeed with very high probability). While this scheme is conceptually simple, it greatly complicates the computation of the bottom-level of a task: one has to account for possible errors (hence task re-executions) along the paths. Yet, computing the expected length of the longest path in a DAG with unlimited processors (or equivalently the expected bottom-level of a task in the DAG), is key to designing silent-error-aware versions of effective list scheduling heuristics (CP-scheduling, HEFT).

It is known that computing the expected length of the longest path in a DAG whose task weights are probabilistic is a difficult problem [?]. Even in the case in which each task is re-executed at most once, i.e., when task weights are random variables taking only two discrete values, the problem remains #P-complete [?] (see Section **??** for a detailed discussion). In this work we develop an algorithm to compute an accurate first-order approximation of the expected length of the longest path in a general DAG in which tasks are subject to silent errors, and whose execution lengths can take two different values, depending upon there is a re-execution or not. More specifically, our contributions are:

- We develop an exact first-order approximation of the expected makespan of a general DAG, which can be computed in polynomial time;
- We compare our approach to two previously proposed approximations for three classes of DAGs from numerical linear algebra computations;
- We quantify approximation errors via comparison to a brute-force Monte Carlo approach; and
- We show that our proposed approximation leads to lower or similar error than previously proposed approximations, and importantly to much lower error when the probability of task failure is low.

This paper is organized as follows. Section **??** reviews related work. Section **??** formalizes the problem that we address and states assumptions. Section **??** describes our proposed approximation. Section **??** presents evaluation results. Section **??** concludes with a brief summary of results and with perspectives on future work.

II. Related Work

In this section we first review previous works that focus on scheduling algorithms for probabilistic DAGs. We then review the relevant literature on silent errors.

A. Expected makespan of probabilistic 2-state DAGs

Computing the expected makespan of a DAG whose task execution times obey arbitrary probability distributions is known to be a difficult problem, even with unlimited processors. When task weights are fixed (deterministic), the makespan is the length of the longest path in the graph (also called critical path). But assume instead a *probabilistic 2-state DAG* where task weights obey a simple 2-state probability distribution: task T_i has weight $a_{i,1}$ with probability p_i and weight $a_{i,2}$ with probability $1 - p_i$. Assume also that all the probability distributions of all tasks are independent. The makespan of the DAG is a random variable. It is known that computing its probability distribution, or even just its expected value, is a #P-complete problem [?]. Recall that the class of #P problems is the class of counting problems corresponding to NP decision problems [?], [?], [?], and that #P-complete problems are at least as hard as NP-complete problems.

An informal explanation of why computing the expected makespan of probabilistic 2-state DAGs is a difficult combinatorial problem, is as follows. The main intuitive reason is that the expected value of the maximum of two random variables is not the maximum of their expectations. As a result, when computing the length of a path in the DAG, one must keep track of all possible values for the starting time of each task, together with their probabilities, and there may be an exponential number of such values [?].

In practice, there are three standard methods to compute the expected makespan of a probabilistic 2-state DAG, as described hereafter.

1) Monte Carlo simulations: The Monte Carlo approach works as follows [?], [?]: For each task in the DAG, a value of its weight is sampled from its probability distribution. Once this is done, the DAG is deterministic and its longest path can be computed as explained in Section **??**. One then repeats this operation for a large number of trials, generating a new value at each trial. The set of these values empirically approaches the actual distribution of the DAG makespan as the number of trials increases.

An interesting question is that of determining the number of trials to obtain a high confidence level in the result. We refer the reader to the relevant discussion in [?]. A key drawback of the Monte Carlo approach is that it is compute-intensive since the necessary number of trials is typically high. In this work, we only use Monte Carlo as a ground truth to assess the accuracy of our and previously proposed algorithms that compute approximations of the expected makespan. Hence, instead of determining a minimal number of trials, we use a very large number of trials so as to guarantee that our ground truth is accurate.

2) Approximation by a series-parallel graph: Basic probability theory tells us how to compute the probability distribution of the sum of two random variables (by a convolution) and of the maximum of two random variables (by taking the product of their cumulative density functions). This simple consideration leads to an exact method to compute the expected makespan when the DAG is series-parallel (see [?] for a definition of series-parallel graphs). However, the problem with probabilistic 2-state series-parallel graph remains NP-complete in the weak sense and admits a pseudo-polynomial solution [?].

When the DAG is not series-parallel, one approach is to approximate it by a series-parallel graph, which is constructed iteratively, first by a sequence of reductions and then by duplicating some vertices. Dodin's method [?] constructs such an approximated series-parallel graph, whose expected makespan is used to estimate that of the original DAG. See [?], [?] for a detailed description of Dodin's method. We include this method in our quantitative experiments in Section ??.

3) Approximation with normality assumption: The central-limit theorem states that the sum of independent random variables tends to be normally distributed as the number of variables increases. The expected makespan of the DAG is a combination of sums and maximums of the original task weights, so a popular approach proposed by Sculli [?] is based on the *normality assumption*:

- Approximate the distribution of each task by a normal distribution of same mean and variance. This step has constant cost per task for probabilistic 2-state DAGs.
- Use Clarke's formulas in [?] to compute the mean and variance of the sum and maximum of two (correlated) normal distributions, and then assuming that they also follow normal distributions.
- Traverse the original DAG and compute the mean and variance of the makespan.

See [?] for a full description of Sculli's method, which we we evaluate experimentally in Section ??.

B. Silent errors

Considerable efforts have been directed at verification techniques to handle silent errors. A guaranteed, general-purpose verification is only achievable with expensive techniques, such as process replication [?], [?] or redundancy [?], [?]. However, application-specific information can be exploited to decrease the verification cost. Algorithm-based fault tolerance (ABFT) [?], [?], [?] is a well-known technique to detect errors in linear algebra kernels using checksums. Various techniques have been proposed in other application domains. Benson et al. [?] compare a higher-order scheme with a lower-order one to detect errors in the numerical analysis of ODEs. Sao and Vuduc [?] investigate self-stabilizing corrections after error detection in the conjugate gradient method. Heroux and Hoemmen [?] design a fault-tolerant GMRES capable of converging despite silent errors. Bronevetsky and de Supinski [?] provide a comparative study of detection costs for iterative methods.

Recently, detectors based on data analytics have been proposed to serve as partial verifications [?], [?], [?]. These detectors use interpolation techniques, such as time series prediction and spatial multivariate interpolation, on scientific dataset to offer large error coverage at the expense of a negligible overhead. Although not perfect, the accuracy-to-cost ratios of these techniques tend to be very high, which makes them attractive alternatives at large scale.

As mentioned in Section ??, lowering the voltage/frequency is believed to have an adverse effect on system reliability [?], [?]. In particular, many papers (e.g., [?], [?], [?], [?]) have assumed the following exponential error rate model:

$$\lambda(s) = \lambda_0 \cdot 10^{\frac{d(s_{max}-s)}{s_{max}-s_{min}}} , \qquad (1)$$

where λ_0 denotes the average error rate at the maximum speed s_{max}, $d > 0$ is a constant indicating the sensitivity of error rate to voltage/frequency scaling, and s_{min} is the minimum speed. This model suggests that the error rate increases exponentially with decreased processing speed, which is a result of decreasing the voltage/frequency and hence lowering the circuit's critical charge (i.e., the minimum charge required to cause an error in the circuit). This is bad news for resilience, because these studies suggest that minimizing energy via DVFS techniques will also lead to an increased number of silent errors.

After detecting a silent error via some verification mechanism, the task is re-executed a second time. Because silent errors are not detected when their occur, but only at the end of the execution of the task (by the verification mechanism), the task must be fully re-executed, and its weight thus doubles. This is the reason why we consider probabilistic 2-state DAGs where the weight of a task T is its initial cost a if there is no error, and $2a$ otherwise.

III. Problem statement

We consider a general model of computation in which an application is structured as a Directed Acyclic Graph, in which vertices represent tasks and edges represent task precedence. More formally, let $G = (V, E)$ be a DAG, with V a set of tasks, and $E \subset V \times V$ a set of edges. For each task i, let a_i be its weight, i.e., its failure-free execution time. For each task $i \in V$, let $Pred(i) = \{j \in V | (j, i) \in E\}$ and $Succ(i) = \{j \in V | (i, j) \in E\}$, i.e., the set of predecessors and of successors of i, respectively. For each task i let $tl(i)$ denote the *top-level* of task i: if $Pred(i) = \emptyset$, then $tl(i) = 0$, otherwise

$$tl(i) = \max_{j \in Pred(i)} \{tl(j)\} .$$

Similarly, for each task i let $bl(i)$ denote its bottom-level: if $Succ(i) = \emptyset$, then $bl(i) = 0$, otherwise

$$bl(i) = \max_{j \in Succ(i)} \{a_i + bl(j)\} .$$

Let $pa(i, j)$ denote the length (as a sum of task weights) of the longest path from task i to task j, if such a path exists, otherwise let $pa(i, j) = -\infty$. The longest path length in G is then defined as $d(G) = \max_{i,j \in V}\{pa(i, j)\}$. Let us call $d(G)$ the *failure-free makespan* of G. Because G is acyclic, we can compute $d(G)$ in $O(|V| + |E|)$ time as follows: add two zero-weight vertices v_1 and v_2 to G, where v_1 represents a unique source task and v_2 a unique sink task. Also add an edge from v_1 to any entry task in G (a task without predecessor), and an edge from any exit task in G (a task without successor) to v_2. Then $d(G) = pa(s_1, s_2)$ can be computed in time $O(|V| + |E|)$ [**?**, Section 24.2].

We consider that tasks fail independently and task failure arrival times are exponentially distributed with Mean Time Between Failure (MTBF) $1/\lambda$. Therefore, the probability that task i fails during its first execution attempt is $1 - e^{-\lambda a_i}$, in which case the task must be re-executed from scratch.

Our objective is to compute an approximation of the *expected makespan* of G, i.e., the longest path length in G taking into accounts that tasks can fail and must be re-executed. The failure-free makespan defined above is a clear lower bound on the expected makespan.

IV. Approximating the Expected Makespan

In this section we compute a *first-order approximation* of the expected makespan of a DAG G, which we denote as $\mathcal{E}(G)$. Our approximation relies on the fact that in practice λ is close to zero, which allows us to neglect $O(\lambda^2)$ terms.

The probability that the first execution attempt of task i succeeds is:

$$p_i = e^{-\lambda a_i} = 1 - \lambda a_i + O(\lambda^2) .$$

The probability that the first execution attempt of task i fails but that its second execution attempt succeeds is :

$$(1 - e^{-\lambda a_i})e^{-\lambda a_i} = \lambda a_i + O(\lambda^2) = 1 - p_i + O(\lambda^2) .$$

Neglecting the $O(\lambda^2)$ terms leads to the approximation that a task either takes time a_i, with probability $1 - \lambda a_i$, or time $2a_i$, with probability λa_i. In other terms, our first-order approximation consists in assuming that a task never fails more than once. Hereafter when we say that a task fails, we mean that its first execution attempt fails and that its second execution attempt succeeds.

For any $S \subset V$, let $P(S)$ denote the probability that all tasks in S fail and that no task in $V \setminus S$ fails. Let also $L(S)$ denote the length of the longest path in G when all tasks in S fail and no task in $V \setminus S$ fails. $\mathcal{E}(G)$ is thus defined as:

$$\mathcal{E}(G) = \sum_{S \subset V} P(S) \times L(S) .$$

We note that:

$$P(\emptyset) = \prod_{i \in V}(1 - \lambda a_i + O(\lambda^2)) = 1 - \sum_{i \in V} \lambda a_i + O(\lambda^2) ,$$
$$P(\{i\}) = (\lambda a_i + O(\lambda^2)) \times \prod_{j \in V \setminus \{i\}} (1 - \lambda a_j + O(\lambda^2))$$
$$= \lambda a_i + O(\lambda^2) \text{ , and}$$
$$P(S) = O(\lambda^2) \text{ if } |S| > 1 .$$

Therefore:

$$\mathcal{E}(G) = P(\emptyset) \times L(\emptyset) + \sum_{i \in V} P(\{i\}) \times L(\{i\}) + O(\lambda^2) .$$

By definition, $L(\emptyset) = d(G)$ (G's failure-free makespan). Similarly, $L(\{i\}) = d(G_i)$, where G_i a DAG identical to

G but such that task i has weight $2a_i$ instead of weight a_i. We thus obtain:

$$\begin{aligned}\mathcal{E}(G) &= (1 - \sum_{i \in V} \lambda a_i) \times d(G) + \sum_{i \in V} (\lambda a_i) * d(G_i) + O(\lambda^2) \\ &= d(G) + \lambda \sum_{i \in V} a_i (d(G_i) - d(G)) + O(\lambda^2) \; .\end{aligned}$$

For a DAG $G = (V, E)$, $d(G)$ can be computed in $O(|V| + |E|)$ time. Therefore, the above approximation can be computed in $O(|V|^2 + |V|.|E|)$ time. Lower complexity can be achieved by exploiting the fact that G and the G_i's differ in only the weight of one task.

V. Evaluation

A. *Makespan Approximation Techniques*

In this section we evaluate three expected makespan approximations techniques:

1) **First Order** – The approximation described in Section **??**;
2) **Dodin** – The bound in [**?**], which is computed by transforming any general DAG into an approximately equivalent series-parallel graph and then computing the exact expected makespan of this graph using the approach explained in Section **??**.
3) **Normal** – The approach that consists in approximating the discrete execution time of each task (i.e., a_i with probability p_i and $2a_i$ with probability $1-p_i$), by a continuous Normal distribution of same mean and standard deviation. The overall expected makespan in the approximated as explained in Section **??**.

B. *Application DAGs*

We measure the error of the three makespan approximation techniques using 3 classes of DAGs used in numerical linear algebra computations. More specifically, we consider 3 classical factorizations of a $k \times k$ tiled matrix: Cholesky, LU, and QR factorization. Each tile has size $b \times b$, where b is a platform-dependent parameter. Hence the actual size of a $k \times k$ tiled matrix is $N \times N$, where $N = kb$. For each factorization, the number of vertices in the DAG depends on k as follows: the Cholesky DAG has $\frac{1}{3}k^3 + O(k^2)$ tasks, while the LU and QR DAGs have $\frac{2}{3}k^3 + O(k^2)$ tasks (but the tasks in QR entail, on average, twice as many floating-point operations as in LU).

Figures **??**, **??**, and **??** show examples for $k = 5$. The tasks in these DAGs are labeled by the corresponding BLAS kernels [**?**], and their weights are based on actual

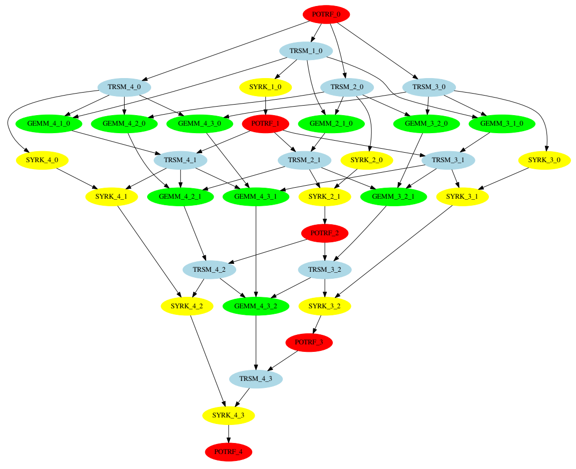

Figure 1: DAG of a Cholesky factorization on a 5×5 tiled matrix

kernel execution times as reported in [**?**] for an execution on Nvidia Tesla M2070 GPUs with tiles of size $b = 960$.

For simplicity, in all that follows we call k the DAG size (i.e., the larger k the more tasks). For each DAG class we perform experiments with $k = 4, 6, 8, 10, 12$, for a total of $3 \times 5 = 15$ DAGs with up to 650 tasks.

C. *Experimental Methodology*

For each DAG, and for a given failure rate λ (see Section **??**), we compute the First Order, Dodin, and Normal approximations of the expected makespan. To compute approximation errors one would ideally use the exact expected makespan (the computation of which is a #P-complete problem). Instead, we resort to the brute-force Monte Carlo approach described in Section **??**. We use 300,000 random trials. For each task in a trial, the task succeeds or fails as determined by sampling a random time-to-next-failure value from an Exponential distribution with parameter λ. We then approximate the expected makespan as the average makespan over the 300,000 samples. This method is prohibitively expensive in practice, but provides us with a reasonable ground truth in our experiments. In all results hereafter we report on the relative error between the approximations and this ground truth.

To allow for consistent comparisons of results across different DAGs (with different number of tasks and different task weights), in our experiments we simply fix the probability that a task of average weight fails, p_{fail}, to 0.01, 0.001, or 0.0001. More precisely, for a given DAG $G = (V, E)$ and a given p_{fail} value, we compute

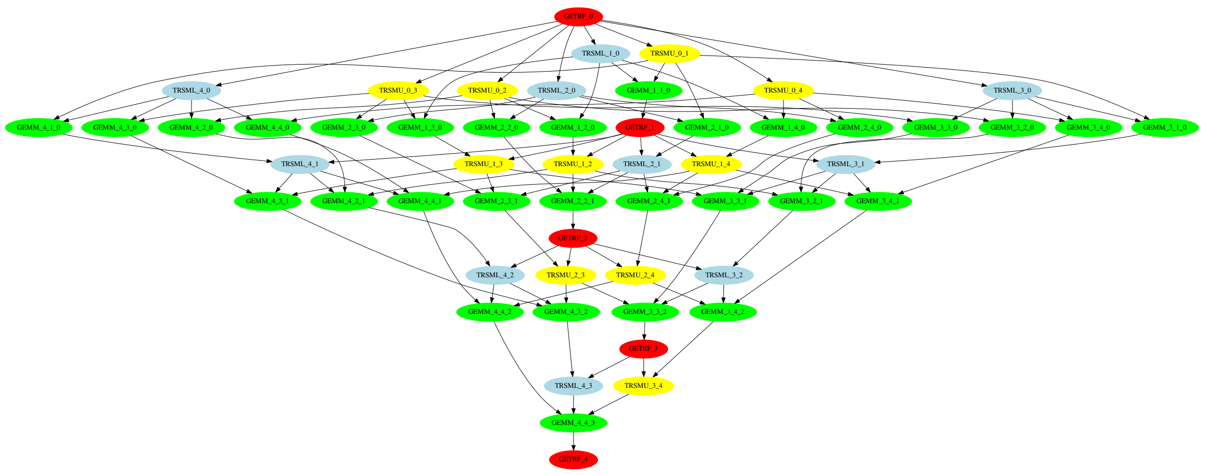

Figure 2: DAG of a LU factorization on a 5×5 tiled matrix

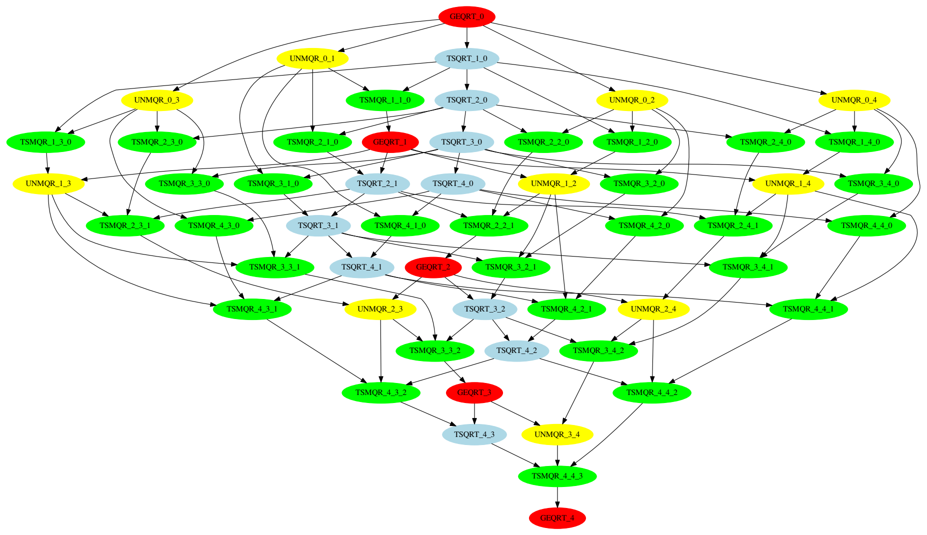

Figure 3: DAG of a QR factorization on a 5×5 tiled matrix

the average task weight as $\bar{a} = \sum_{i \in V} a_i/|V|$ and pick the failure rate λ such that

$$p_{\text{fail}} = 1 - e^{-\lambda \bar{a}} .$$

While the above values of p_{fail} are chosen to evaluate the performance or our proposed approximation and to compare it to competitors, we point out that these values are quite pessimistic. In other words, they lead to error rates much higher than those observed or expected on current and future large-scale computing platforms. The average execution time of a task in our experiments is $\bar{a} = 0.15$ seconds, which leads to an error rate $\lambda = 0.067$ with $p_{\text{fail}} = 0.01$. Equivalently, the MTBF is $\mu = 1/\lambda = 14.9$ seconds. For a platform with $100,000$ processors, this corresponds to an individual MTBF (per processor) of 17.27 days, quite an unrealistically low value. The intermediate value of p_{fail}, namely 0.001, leads to an individual MTBF of 174 days. The smallest value of p_{fail}, namely 0.0001, leads to an individual MTBF of 4.7 years. As seen in the results in the next section, the lower p_{fail}, the lower the error incurred by our proposed approximation (and importantly much lower than that of its competitors). In other words, our selected relatively high p_{fail} values put our algorithm at a disadvantage with respect to its competitors.

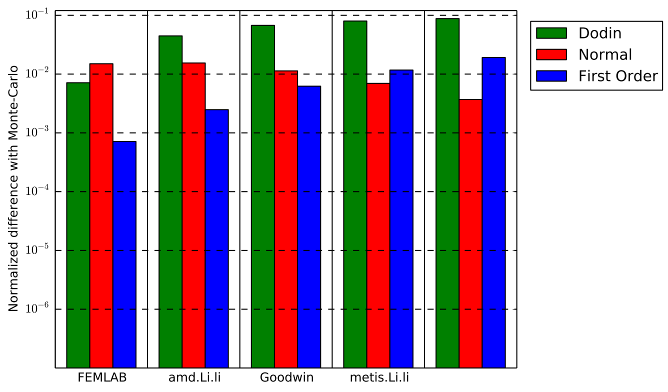

Figure 4: Cholesky, $p_{\text{fail}} = 0.01$

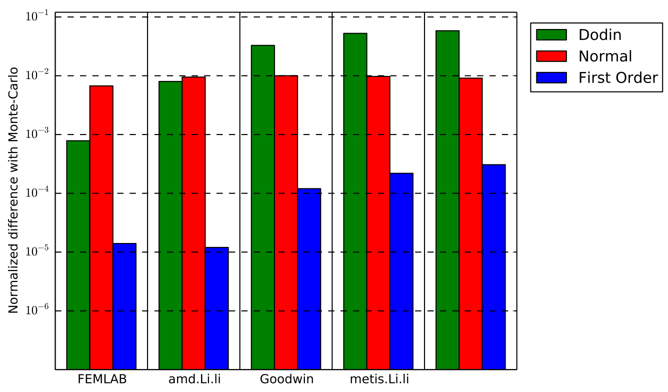

Figure 5: Cholesky, $p_{\text{fail}} = 0.001$

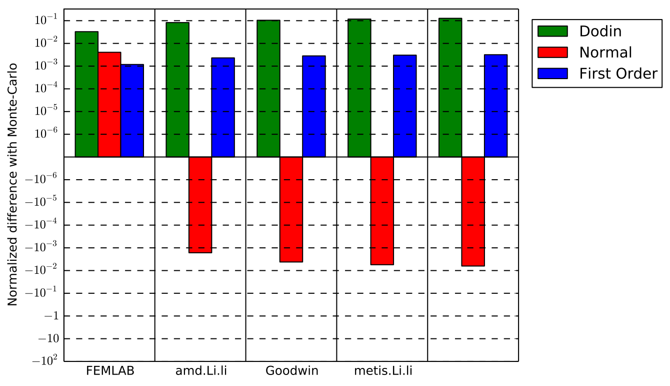

Figure 7: LU, $p_{\text{fail}} = 0.01$

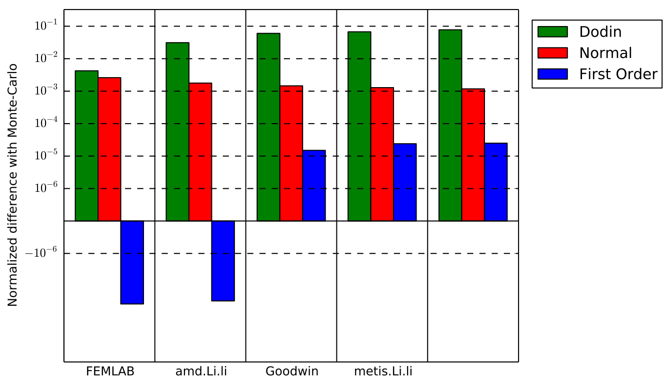

Figure 8: LU, $p_{\text{fail}} = 0.001$

D. Approximation Error Results

In this section we show relative error results, relative to expected makespans computed using the Monte Carlo method. Negative values denote an underestimation, while positive values denote an overestimation. All figures use a logarithmic scale on the vertical axis.

Figures **??**, **??**, and **??** show relative error vs. graph size for Cholesky graphs and for $p_{\text{fail}} = 0.01, 0.001$, and 0.0001, respectively. For $p_{\text{fail}} = 0.01$, we see that First Order leads to the lowest relative error for graphs sizes below 10, and that Normal leads to the lowest relative error for larger graphs. Dodin leads to the largest relative error, but for the smallest graph size. Considering the largest graph size, First Order has 1.9% relative error while Normal has 0.3% relative error and Dodin has 8.7% relative error. For $p_{\text{fail}} = 0.001$, First Order leads to dramatically lower error than its competitors (at least one order of magnitude lower). For instance, for the largest graph, First Order has 0.03% relative error, compared to 5.8% for Dodin and 0.9% for Normal. These results are even more striking for $p_{\text{fail}} = 0.0001$. In this case, still for the largest graph, First Order has -0.0006% relative error, compared to 3.7% for Dodin and 0.4% for Normal. In this case, First Order is an underestimation of the expected makespan.

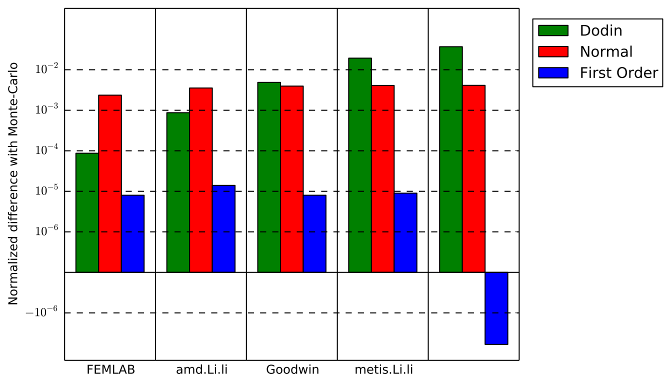

Figure 6: Cholesky, $p_{\text{fail}} = 0.0001$

Figures **??**, **??**, and **??** show similar results for LU DAGs. The overall message from these results is the same, meaning that First Order leads to drastically lower error than its competitors as p_{fail} decreases. Here again

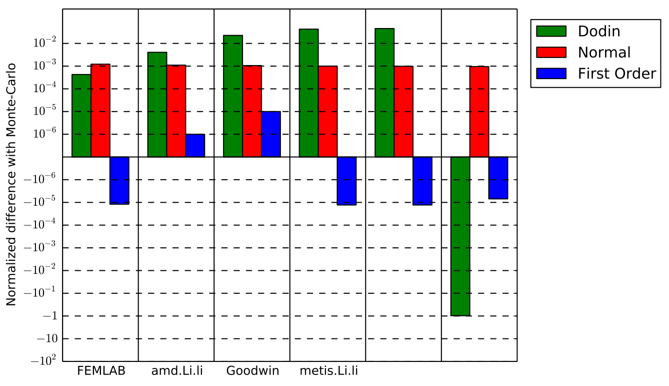

Figure 9: LU, $p_{\text{fail}} = 0.0001$

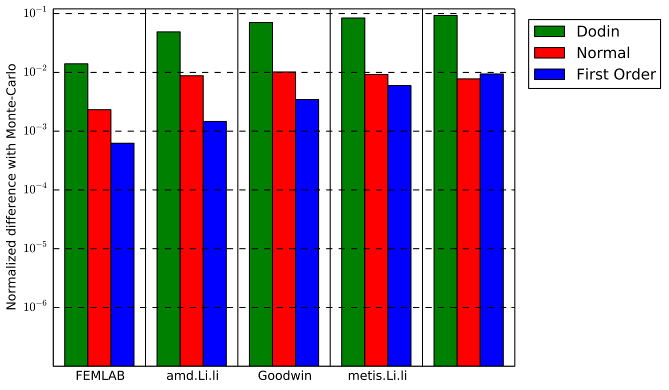

Figure 10: QR, $p_{\text{fail}} = 0.01$

Dodin leads to the largest errors overall. For $p_{\text{fail}} = 0.01$, out approach leads to similar errors of the same order of magnitude as the errors for Normal (and in this case Normal is most often an underestimation while First Order is an overestimation).

Finally, Figures **??**, **??**, and **??** show results for QR DAGs and similar trends are observed. First Order drastically improves on Dodin and Normal for $p_{\text{fail}} = 0.001$ and $p_{\text{fail}} = 0.0001$. For $p_{\text{fail}} = 0.01$, First Order leads to similar or lower error than Normal. Here again, Dodin leads to the highest errors across the board.

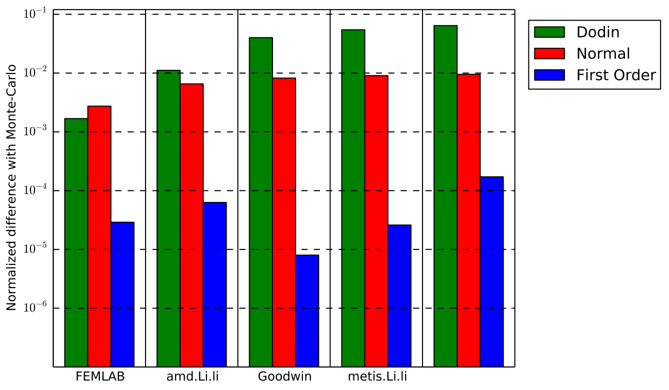

Figure 11: QR, $p_{\text{fail}} = 0.001$

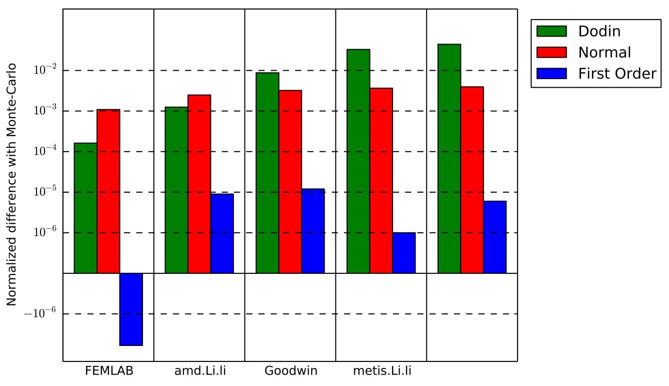

Figure 12: QR, $p_{\text{fail}} = 0.0001$

E. Scalability Results

To assess the scalability of the three approximation methods, we run experiments with $k = 20$ for the LU DAG, i.e., $2,870$ tasks, and $p_{\text{fail}} = 0.0001$. For this large graph, we ran the Monte Carlo for ten hours, so as to ensure the accuracy of the ground truth. Error (normalized difference with Monte Carlo) and execution time results are shown in Table **??**. We see that for this large DA, Dodin exhibits very large error. First Order is roughly two orders of magnitude more accurate than Normal. We also see that First Order can be computed within one second, while Normal requires about 20 minutes, i.e., about three orders of magnitude longer.

F. Summary of Results

Our results show that our proposed approximation, First Order, is accurate as long as the probability of task failure is sufficiently low, and in particular much more accurate than its two competitors. When the task failure probability is high, its accuracy is comparable to or better than the Normal approximation. Across the board the Dodin approximation leads to high error. This is because the DAGs that we consider are far from being series-parallel. As a result, the series-parallel graph constructed by Dodin is a poor approximation of the original DAG, hence the large errors. Finally, not only is First Order more accurate, but also it is faster. Its execution on the large problem with $2,870$ tasks requires negligible time, as opposed to several minutes for Dodin and Normal.

	Dodin	Normal	First 0rder
Normalized difference with Monte Carlo	-0.97	954×10^{-6}	7×10^{-6}
Execution time	around 2 minutes	around 20 minutes	less than 1 second

Table I: LU with $k = 20$ and $p_{\text{fail}} = 0.0001$.

VI. Conclusion

We have proposed an algorithm to compute a first-order approximation of the expected makespan of a Directed Acyclic Graph (DAG) of tasks in which tasks are subject to silent errors that may require task re-execution from scratch due to corrupted results, which we term a failure. Our approximation can be computed in $O(V^2 + V.E)$ time for a DAG with V vertices and E edges, and it is exact in the first order and neglects second order $O(\lambda^2)$ terms where λ is the exponential failure rate. This amounts to assuming that each task may need to be re-executed at most once. The problem of computing the expected makespan of a DAG of tasks with this assumption is actually a computationally difficult problem (#P-complete). As a result, techniques to approximate the expected makespan have been proposed in previous works [**?**], [**?**]. We have evaluated our proposed approximation and these previously proposed techniques for three classes of application graphs from the field of numerical linear algebra. These evaluations quantify the approximation error with respect to a ground truth computed via a brute-force Monte Carlo method.

Our results show that our proposed approximation outperforms its competitors, by several orders of magnitude at low failure rates. In addition, it can be computed much more quickly than previously proposed approximations, which is crucially important for solving problems at scale. Overall, we have proposed a novel and improved approximation of the expected makespan of probabilistic DAGs in which tasks are subject to silent errors that mandate task re-executions.

Our general approach to obtain our first order approximation can be used to obtain a (more complicated but still tractable) second order approximation. While the improvement due to including the second order would be negligible for low failure rates, it may be significant for relatively high failure rates. A broader future direction is to adapt existing list scheduling algorithms, or to develop novel such algorithms, that rely on our proposed approximation to make scheduling decisions.

Acknowledgments: This work was supported in part by the European project SCoRPiO, by the LABEX MI-LYON (ANR-10- LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR), by the PIA ELCI project, and by NSF award SI2-SSE:1642369 . Yves Robert is with Institut Universitaire de France.