Improving checkpointing intervals by considering individual job failure probabilities

Alvaro Frank, Manuel Baumgartner, Reza Salkhordeh, André Brinkmann
Zentrum für Datenverarbeitung
Johannes Gutenberg University Mainz
Email: {afrankra, manuel.baumgartner, rsalkhor, brinkman}@uni-mainz.de

Checkpointing is a popular resilience method in HPC and its efficiency highly depends on the choice of the checkpoint interval. Standard analytical approaches optimize intervals for big, long-running jobs that fail with high probability, while they are unable to minimize checkpointing overheads for jobs with a low or medium probability of failing. Nevertheless, our analysis of batch traces of four HPC systems shows that these jobs are extremely common.

We therefore propose an iterative checkpointing algorithm to compute efficient intervals for jobs with a medium risk of failure. The method also supports big and long-running jobs by converging to the results of various traditional methods for these. We validated our algorithm using batch system simulations including traces from four HPC systems and compared it to five alternative checkpoint methods. The evaluations show up to 40% checkpoint savings for individual jobs when using our method, while improving checkpointing costs of complete HPC systems between 2.8% and 24.4% compared to the best alternative approach.

***Index Terms*—checkpointing, HPC, resilience**

I. Introduction

Parallel MPI applications typically fail as soon as a single node of the underlying HPC system crashes. Checkpointing reduces the impact of such node failures by regularly storing the complete state of an application on persistent storage. An application can therefore restart after a node failure and continue at the point of the last checkpoint. However, checkpointing requires considerable compute and storage resources to save the state of an application. For this reason, optimizing the checkpoint interval to reduce overheads has been an active area of study in resilience [10], [19], [20], [28], [29], [32].

The most prominent theoretical method for finding optimal checkpointing intervals was proposed by Young [32] and later improved by Daly [9]. Their methods are derived from a checkpoint cost function minimization for Poisson failure distributions. Both approaches model their cost functions with the assumption that failures nearly certainly happen within the runtime of a job, which holds for very long running and big jobs. Subsequent studies maintained this assumption [10], [19], [20], [28], [29].

Our first observation in this paper reveals that many checkpointable HPC jobs only have a medium probability of failure (MPF). These MPF jobs can be estimated to have a failure chance between $0.2 < p_{fail} < 0.7$. These are not hard thresholds and depend highly on system parameters. MPF jobs can benefit on average from checkpointing, while many individual MPF jobs do not fail during their runtime. They therefore do not fulfill the modeling assumptions of previous checkpointing strategies, as either their runtimes are not long enough, they do not use many nodes, or due to long times between node failures. Not accounting for the properties of these MPF jobs in previous studies resulted in over- or underestimations of their optimal checkpointing intervals and therefore induced unnecessary extra checkpointing costs. Jobs with a low failure probability $p_{fail} < 0.2$ are not explicitly optimized in this paper, as it is typically not beneficial to checkpoint them. They nevertheless occur within our traces.

The main contributions of our work are the development of a new cost function for checkpointing MPF jobs and an iterative algorithm that finds the statistically best checkpoint intervals for MPF jobs based on the new cost function. The input of the algorithm includes the number of nodes being used by an application, the runtime of an application, and the failure properties of the underlying HPC system. The algorithm furthermore converges to the optimal results for jobs with high failure probabilities. To the best of our knowledge, no other method provides accurate estimates of checkpointing intervals for MPF jobs.

Our method considers MPF jobs by taking into account the actual number of checkpoints within the runtime of a job and by modeling the average checkpoint count in case of failures, using an exact finite sum of weighted probabilities. Our approach furthermore models the average runtime with failures using the first truncated moment of the respective failure distribution. Finally it weighs the checkpointing costs of both success and failure scenarios with the respective failure and success probabilities for the entire job's runtime.

Traditional approaches neglected these aforementioned aspects and considered failure probabilities to be extremely high. They also assumed that the number of checkpoints is high and that the average runtime can be statistically modeled using non-truncated failure distributions. Neglecting these aspects leads to inaccurate checkpoint intervals for MPF jobs with a limited number of checkpoints.

We present implementations of our algorithm and experiments based on real system traces of four HPC systems: JGU MOGON II[1], LANL Atlas[2], LANL Mustang[2] and CEA

[1] www.hpc.uni-mainz.de

[2] ftp.pdl.cmu.edu/pub/datasets/ATLAS

Curie[3]. System failures in these experiments are modeled using Weibull and Poisson failure distributions. Nevertheless our algorithm does not require an analytical formula of the applied failure distributions and it suffices to provide an empirical numerical formula of the failure properties.

The experimental results for the four HPC traces show that we can obtain cumulative checkpoint cost savings (considering all jobs and re-queuing) from 7.1% to 25.7% for Weibull failures and from 7.3% up to 27.5% for Poisson failures. The 7.1% and 7.3% cost savings are compared to the best alternative methods. MOGON II benefits most with average checkpoint savings across all alternative methods of 18.1% followed by LANL Mustang with 13.0%. LANL Atlas and Curie see 9.0% and 10.5% savings, respectively. Finally we also show that our method achieves savings between 6.0% and 26.6% compared to alternative methods when only inaccurate values for the mean time between failures (MTBF) are available.

The experimental results further show that our method numerically agrees with Daly's higher order cost approximation for jobs with a high probability of failure and failures following a Poisson distribution.

In the remainder of this paper we discuss relevant related work in Section II, the motivation for our approach in Section III, the problem formulation in Section IV, the derived algorithm in Section V, the validating simulations in Section VI and end with a conclusion in Section VII.

II. Related Work

Checkpoint and restart has become the de-facto method to protect computations from system failures. The main theoretical framework to model overhead of checkpoint and restart methods is based on Young [32] and Daly [9].

Young proposed a first order approximation of optimal checkpoints and concluded that the optimal interval is fixed and equals $\sqrt{2Mt_c}$ where M is the mean time between failures and t_c the checkpoint cost. Daly expanded Young's work taking into account restart costs, leading to the higher order approximation of $\sqrt{2Mt_c}+t_c$ in which the restart times ended up not contributing to the interval determination.

Bouguerra *et al.* proposed a method for Weibull failure distributions that searches for a checkpoint count that minimizes the average completion time of jobs [7]. The authors also showed that a periodic checkpointing strategy is optimal when all nodes rejuvenate (state becomes failure-free after a failure). This is a very restrictive requirement in practice, since HPC system would need new nodes with no previous history of failures that might affect future resilience. Our work does not require component rejuvenation and instead relies on using the available nodes within the HPC system to run re-queued jobs after a failure without requiring unused spare nodes. Our approach is also compatible with Weibull distributions, but differs by searching through intervals to minimize checkpointing costs.

Bougeret *et al.* proposed a multi-node heuristic (DP-NextFailure) to determine checkpoint intervals by maximizing the work done before a failure instead of minimizing waste [5]. Their method recursively calculates probabilities for all computational sections between checkpoints, and yields an interval for the job. It does not use the total runtime in the probability calculations, making their approach inaccurate for MPF jobs. The recursive algorithm also makes their method computationally expensive with cubic complexity. Another difference is that they use success probabilities for the computation sections, while our method uses the failure probability of the total job runtime.

Jin *et al.* [20] expanded Young's approximation to also consider multiple nodes as well as spare nodes. They arrived at a first order approximation that matches Young's method when the recovery cost is zero. As previously mentioned, our method relies on existing nodes within a batch system.

Subasi *et al.* [28] proposed a general checkpointing method with analytical formulations for arbitrary distributions, but does not include the properties of MPF jobs. Our approach also models arbitrary failure distributions, as long as density functions and truncated moment formulas are available.

Bessho *et al.* [4] adapted checkpoint intervals to multi-node systems. We accommodate this aspect into our simulations of batch systems by scaling the MTBF accordingly and treating the failures of distinct nodes as independent. This assumption was explored and validated by Aupy *et al.* [3].

Herault *et al.* [17] discussed the limitations of Daly's method when taking I/O resource contention into account. Their resulting approach yields a similar interval approximation to Daly but includes constrains for I/O utilization. Our approach requires the I/O overhead to be modeled within the cost of checkpointing.

Tiwari *et al.* [29] introduced a variable and fixed interval method that takes into account an estimate for the average fraction of lost work in their checkpoint formula. Our method also accounts for the average lost work after a failure. Unlike Tiwari, we do not use an estimate, but instead calculate accurate statistical values using truncated moments.

El-Sayed *et al.* [10] concluded that the overhead and complexity of variable checkpoint intervals was not justified in practice for their LANL systems. We follow this assessment and focus our efforts on fixed checkpoint intervals during the uninterrupted runtime of an application. Nevertheless, the checkpointing interval for a job can change based on the new residual runtime left when restarting the job after a failure. This is possible because our approach is a function of the runtime of the job.

Jayasekara *et al.* [19] proposed a checkpoint interval for batch stream processing systems that takes into account the time needed to detect a failure. Their method differs from others in their modeling of the optimal interval using a Lambert function and by not assuming a single node failure to be fatal for the complete job. In contrast, most HPC checkpointing approaches including ours consider the first encountered failure to be fatal.

Gupta *et al.* [15] showed that failures which are correlated based on physical proximity need to be modeled differently. We can support such locality of failures, if they are modeled within the failure distribution. We also require the

[3]www.cs.huji.ac.il/labs/parallel/workload/l_cea_curie

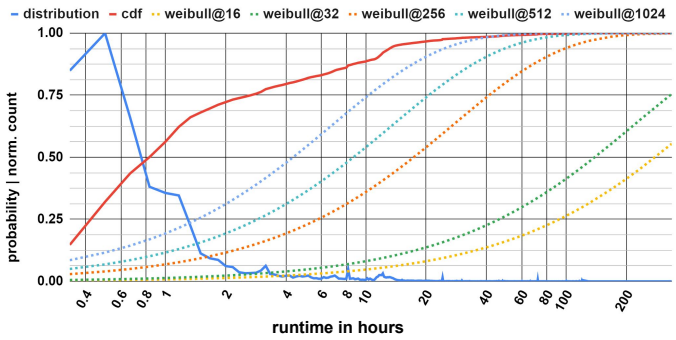

Figure 1: Runtime distribution of user jobs within our HPC system (blue). The red curve indicates the CDF of runtimes and the dotted curves the failure probability distributions for various node counts.

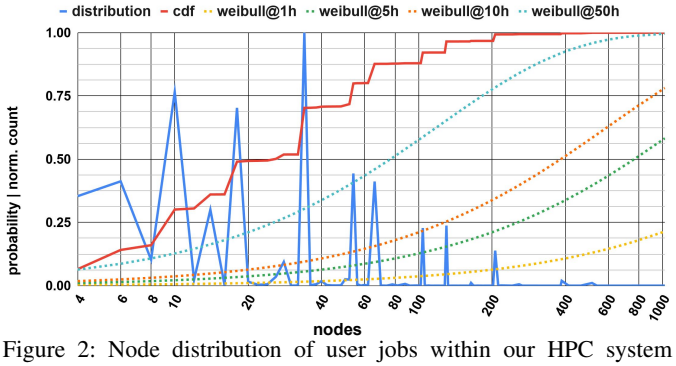

Figure 2: Node distribution of user jobs within our HPC system (blue). The corresponding red curve indicates the CDF and the dotted curves the failure probability distributions.

failure model to consider the whole system and not just sub-components. The failure distributions are therefore preferably derived from real logs.

We follow the approach of previous works [17], [20], [3] by scaling the MTBF of a system by the amount of nodes used by a job. Under this modeling a job using half a system experiences an MTBF twice as long.

The cost of checkpointing defined by our model is directly influenced by the duration of every checkpoint t_c performed. This cost is significant in file-systems that cannot handle the large volumes of data written during the creation of checkpoints. The overhead of checkpoints can be mitigated by using highly parallel on-demand file-systems [8], distributed file systems that offer high bandwidth, caching and QoS [30], [25], [31] high performance persistent client caching that can cope with the overhead [24], [26] and techniques that reduce the size of checkpoints [14] [21]. In this work we perform simulations using multiple checkpoint costs to showcase how our method offers consistent gains.

Finally our model does not need to account for multiple failures fatally interrupting the application during its runtime. This follows from focusing on parallel MPI applications, where applications fail catastrophically upon encountering the first failure. Furthermore, there is no need to consider any sequence of non-fatal failures that do not invalidate computations.

In the next section we expand on how HPC system workloads and failure characteristics motivate our approach.

III. Motivation

Our main motivation to address MPF jobs is based on the failure characteristics and job statistics of the Mogon II HPC system with more than 2,000 nodes as well as two LANL Systems, Trinity and Mustang [1]. For Mogon II, we have access to batch logs for a period of two years consisting of four million jobs [12]. We found out that $\sim$1.59% of the jobs crashed from either a hardware or system software failure (non-user related), creating a scaled MTBF of 11.15 hours for 1024 nodes.

Looking at the runtimes and node counts we can determine the probability of a job failure in conjunction with the MTBF of the system. Figure 1 shows the normalized runtime distribution of jobs in our system, the related cumulative distribution function (CDF), and four Weibull failure probability curves for jobs with node counts up to 1,024.

At first glance, we note that most jobs are shorter (3.91h) than the full system MTBF of 5.71h, and when we consider the scaled MTBF of more common node sizes used by the jobs, the MTBF increases to 428.25 hours for 16 nodes. Dotted curves in Figure 1 indicate the respective failure probabilities for 16, 64, 256, 512 and 1024 nodes The key aspect is that both the MTBF and the probability of failure decrease as the number of nodes used by a single job decreases. A job with 512 nodes, e.g., needs 8 hours of runtime to achieve a failure probability of 50%.

Figure 2 shows the node size distribution of the same job set as in Figure 1. Similarly to the runtime distribution, the node counts are commonly small and mostly in powers of two. The CDF shows at least 50% of the node counts being smaller than 26 nodes, 75% smaller than 54 nodes and 90% smaller than 104 nodes. It already takes 74 nodes running for 50 hours to achieve a coin toss failure probability of 50%. For jobs only running for 1 hour, it takes at least 3,826 nodes to surpass a probability of failure of 50%.

Browne *et al.* [22] have found similar node and runtime distributions within the *Lonestar4* and *Ranger* systems. The jobs from those systems are also skewed to lower runtimes and node sizes. Amvrosiadis *et al.* [1] surveyed the LANL Trinity and LANL Mustang traces that we have used in our simulations. They showed that 80% of the jobs lasted less than 3 hours for Mustang and 6 hours for Trinity. This is evidently shorter than the runtime needed to assure failures. The LANL traces show that MPF jobs are very common and are responsible for 90% of the jobs using 128 nodes or less.

Checkpointing these MPF jobs wastes resources if the checkpointing interval is calculated based on the traditional assumption that jobs have high failure probabilities. Our method enables users of MPF jobs to reduce the number of required core hours when considering failures and failure free-runs.

IV. Problem Formulation

Our goal is to derive a checkpoint interval length τ for each HPC job that balances the additional runtime of the job due to the creation of the checkpoints and the waste of computation between the last checkpoint and a possible failure.

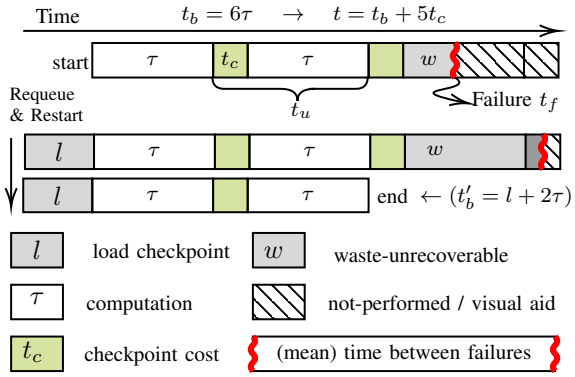

Figure 3: Checkpointing diagram showing the cyclical processes of periodic checkpoints between computation phases. In the event of a failure the application is re-queued and restarted by loading the checkpoint.

Let t_c denote the time needed to create a single checkpoint (see Figure 3). For an estimated base runtime t_b of the job, e.g. given by the user, the total runtime of the job extends to $t = t_b + nt_c$ if n checkpoints are made and no error occurs.

In order to be able to optimize the checkpoint interval τ at all, the following Assumption 1 is made:

Assumption 1. *The base runtime of applications without checkpoint overheads can be accurately estimated by either users or other predictive systems.*

Given that most users provide an upper bound of their job's runtime on submission, this assumption may be considered a comparatively weak requirement.

We use a random variable $\mathbb{F}$ to model the occurrence of failures within the HPC systems. For a given job, a realization of $\mathbb{F}$ creates a time $0 \leq t_f$ at which a failure occurs. If $t_f < t$, the failure occurs within the runtime of the job and the job fails, otherwise the job succeeds.

The expected average cost $\hat{S}$ of checkpointing excluding useful computation time is given by Equation 1. Our goal is to model the actual average cost $\hat{S}$ of performing periodic checkpoints through a simulation of F events for a job runtime t and interval τ. Any runtime not used for computations is considered as wasted, and hence is a cost. This is shown in Equation 1, where the first term (1a) accounts for the cost of checkpointing t_c multiplied by the actual amount of performed checkpoints $\lfloor \min(t, t_f)/(\tau + t_c) \rfloor$ in each simulated event. This cost is always paid by a checkpointed job, regardless of a failure occurring.

$$\hat{S} = \frac{1}{F} \sum_{f=1}^{F} t_c \lfloor \min(t, t_f)/(\tau + t_c) \rfloor \tag{1a}$$

$$+ t_f \bmod (\tau + t_c) \quad \text{if } t_f < t. \tag{1b}$$

Additionally, term (1b) adds the wasted time after the last successful checkpoint ($t_f \bmod (\tau + t_c)$) in case a failure occurs at time $t_f < t$. In Section VI we will show how Equation 1 is approximated by the traditional methods when the event $t_f < t$ occurs with high probability. Furthermore we will show how our method deals with the case of $t_f > t$ for MPF jobs. In contrast to previous models, our cost is discontinuous since we included floor functions to model the actual number of checkpoints at lower probabilities. In Section VI-A we will show how the effects of the floor function are smoothed out at failure probabilities close to one.

Our approach to model $\hat{S}$ works with arbitrary failure distributions $\mathbb{F}$, as long as approximations of the truncated moments (cf. Eq. (12)) are available and the relevant failure assumptions described in this section hold. Nevertheless we focus on the two most common distributions: Weibull for its flexibility [27] and Poisson for its use in previous work [32], [9].

HPC jobs are traditionally run using a batch system that queues jobs in HPC systems and waits for resources to be available before executing. If a job fails but a valid checkpoint is available it can be re-queued and can become a new job with a new base runtime t'_b. The re-queued job receives available nodes from within the already existing HPC system. Re-queuing and re-starting requires waiting until the re-queued job starts. Re-queued jobs are then checkpointed and are susceptible to failures. As we focus on the checkpointing costs during the runtime of a job, we neglect the waiting time of the job, since waiting consumes no system resources. Moreover, the initial waiting time of the first job-submission is traditionally also not taken into account. For the additional time needed to restart from the last checkpoint, we make the following Assumption 2:

Assumption 2. *The restart time for a failed and re-queued job is known and is part of the new base runtime t'_b.*

A re-queued job is considered as a new job in itself and a new checkpoint interval τ' may be computed for the new base runtime t'_b. Previous works show no contribution from the restart time [9], [17], [29], [20], [19].

Assumption 3 accounts for spatial locality and multiple failures affecting the same runtime of an application by considering failures to be catastrophic and applying subsequent related failures to re-queued jobs.

Assumption 3. *An application fails catastrophically on all its nodes upon encountering the first node failure. Only failures that stop or invalidate computations are meaningful for checkpointing. Multiple correlated failures can affect newly re-queued jobs and should be modeled within the failure distributions of the whole system.*

Assumption 3 requires that the failures of a system are modeled as a whole for the HPC system instead of modeling failures per node (component rejuvenation).

Additionally any changes in parameters like MTBF or cost of checkpointing can be used to update the model and compute new interval values.

In the following section, we derive a mathematical model of our problem formulation and show in Section VI that our approach offers lower expected checkpointing costs for MPF jobs compared to Daly's or other methods.

V. Algorithm Derivation

An analytical determination of the optimal checkpoint interval length is challenging due to the floor functions in Equation 1 and not the focus of our study. Moreover, a fixed analytical formulation of the failure distribution would interfere with our goal of providing a method that is able to work with any failure distribution.

Instead we will provide an iterative algorithm that evaluates values of τ to find a minimal cost tailored to the expected runtime of the application and its probability of failure. Although this approach does not necessarily find the optimal value of τ in a mathematical sense, it provides a very fast and efficient method that can be implemented in production.

We summarize the combined job and system settings needed to model checkpointing as the vector

$$s = (t_c, \tau, M, t, t_f),$$

where t_c is the time required for a single checkpoint, τ the computing time between two checkpoints, M the MTBF of the system, t the runtime of an application and t_f is the time of a failure.

A. Cost function

Traditional methods for the determination of checkpoint intervals are based on the assumption that a failure nearly definitely happens during the job's runtime. Our approach removes this limitation by weighting the costs of failure and success events with their respective probabilities. Considering a single job with total runtime t and a failure occurring at t_f, we consider the cost function

$$C(t_f, t) \coloneqq \begin{cases} C_{\text{ok}} & \text{if } t_f > t \\ C_{\text{fail}} & \text{if } 0 \leq t_f \leq t \end{cases} \tag{2}$$

evaluating the costs if a failure occurs at time t_f. The individual costs are defined as

$$C_{\text{ok}} \coloneqq \left\lfloor \frac{t}{\tau + t_c} \right\rfloor t_c, \tag{3a}$$

$$C_{\text{fail}} \coloneqq t_f - \left\lfloor \frac{t_f}{\tau + t_c} \right\rfloor \tau. \tag{3b}$$

The cost function in (2) accounts for the additional runtime of the job in case the job terminates successfully, or the wasted time in case of a failure. The function C_{ok} accumulates the additional time spent in creating the checkpoints. Note that this function is independent of t_f and is our extension to Young and Daly. The function C_{fail} represents the traditional checkpointing cost, i.e. the difference of the time t_f at which the failure occurs and the successfully completed full computation intervals $t_u = \tau + t_c$ until t_f. In the failure scenario the computation progress is accounted for by the computation chunk τ. The rest is wasted work that constitutes cost. In either cases, the total time $t = t_b + nt_c$ depends on the value of the time interval τ between two checkpoints.

Both cases include a floor function rendering this approach different from previous works. The floor function transforms the number of checkpoints into a whole number (*e.g.* $3.34 \mapsto 3$) to account for only the cost of checkpoints that are actually performed, making the method more accurate. Any residual is considered part of the wasted computation from failures and is accounted for later in Equation 6c. Floor functions are necessary to model the actual cost for MPF jobs in that it exactly computes the number of completed checkpoints. The floor function produces a non-continuous cost function for MPF jobs but retains relative smoothness for traditional jobs where the probability of failure approaches 1.

In summary, the cost function C comprises the additional runtime of a successful job caused by the checkpoint-restart technique and the wasted computation time for a failing job. Restart and resume costs are conceptually modeled within job re-queueing after catastrophic failures.

Substituting the random variable $\mathbb{F}$, modeling the occurrence of a failure (see examples in Section V-C), into the cost function (2) yields the new random variable $C(\mathbb{F}, t)$ and its expected value

$$E_C \coloneqq E[C(\mathbb{F}, t)] \tag{4}$$

corresponds to the expected cost of a job that performs checkpoints. Note, since the cost function in Equation (2) depends on the time interval τ between two consecutive checkpoints, the expected cost in Equation (4) is a function of τ. The (mathematical) optimal value of τ is the minimum of the expected cost (4). This approach was taken by Young/Daly and Bougeret utilizing their own smooth cost functions. However, our cost function that accounts for MPF jobs is not smooth due to the floor functions, thus we do not analytically compute the optimal value. The use of floor functions also makes our cost more realistic, since only completed checkpoints are actually considered.

By definition, the expected cost E_C (4) may be written as

$$E_C = \int_0^t C_{\text{fail}}(x, t) p(x)\,\mathrm{d}x + \int_t^\infty C_{\text{ok}}(x, t) p(x)\,\mathrm{d}x \tag{5a}$$

$$= \int_0^t C_{\text{fail}}(x, t) p(x)\,\mathrm{d}x + \left\lfloor \frac{t}{\tau + t_c} \right\rfloor t_c (1 - P(t)) \tag{5b}$$

where the CDF $P(t) = \int_0^t p(x)\,\mathrm{d}x$ is the probability of a failure occurring before t. The fail part in (5) is evaluated as

$$\int_0^t C_{\text{fail}}(x, t) p(x)\,\mathrm{d}x \tag{6a}$$

$$= \int_0^t x p(x)\,\mathrm{d}x - \tau \int_0^t \left\lfloor \frac{x}{\tau + t_c} \right\rfloor p(x)\,\mathrm{d}x \tag{6b}$$

$$= \mu^t - \tau \hat{n}_f \tag{6c}$$

where $\mu^t = \int_0^t x p(x)\,\mathrm{d}x$ is the first right truncated moment, also known as the partial average of the distribution from $\mathbb{F}$

[23]. Using the truncated moment allows us to model average runtimes before a failure for MPF jobs. This approach is new among checkpoint determination techniques. In the second integral in Equation (6b), τ can be taken out, while the integral itself evaluates to the mean amount of checkpoints in case of failures $\hat{n}_f$. Equation (6) may be interpreted as the mean time up to failure, minus the mean time spent doing actual computations. Effectively, this is the expected time spent doing checkpoints and wasted computations that need to be redone upon a failure. Unlike Bougeret where computing sections are weighted, we weight the costs with the respective probabilities of the entire jobs runtime. This allows for modeling of MPF jobs.

The expected number of checkpoints before a failure at t_f in (6c) can be numerically calculated using the formula

$$\hat{n}_f = \sum_{i=0}^{t/t_u} i \cdot [\ P((i+1)t_u) - P(it_u)\ ], \tag{7}$$

resulting from splitting up the second integral from Equation (6b) to intervals where the integrand is smooth. Within the sum, each checkpoint count i is weighted by the probability of a failure occurring in the particular interval. This represents an average number of checkpoints upon encountering the first failure of MPF jobs and is to our knowledge also the first formulation of it.

The expected costs E_C can therefore be summarized in Equation (8) as the sum of the partial costs for failures (8a) and successes (8b):

$$E_C = \quad \mu^t - \tau \sum_{i=0}^{t/t_u} i \cdot [\ P((i+1)t_u) - P(it_u)\ ] \tag{8a}$$

$$+ \left\lfloor \frac{t}{\tau + t_c} \right\rfloor t_c(1 - P(t)) \tag{8b}$$

This final equation is used within Algorithm 1 to find the the value of t_u that minimizes it.

B. Iterative Algorithm

Algorithm 1: Compute t_u for which E_C is minimal.

Data: system s with input t_b.
Result: t_u with minimum expected cost.
$\mathbb{U} = [\frac{1}{60}, \frac{2}{60}, \frac{3}{60} \ldots, t - t_c]$; $t_u \in \mathbb{U}$ `in hours`
$c_{\min} = maxflag$;
$u_{\min} = undefined$;
foreach t_u *in* $\mathbb{U}$ **do**
  $c = E_C(t_b, t_u)$; `Eq. 8`
  if $c < c_{\min}$ **then**
    $c_{\min} = c$;
    $u_{\min} = t_u$;
return $u_{\min}$

Algorithm 1 searches for an approximated optimal checkpointing interval τ. It iterates over the set $\mathbb{U}$ of t_u choices for $t_u = \tau + t_c$ with a resolution of one minute. The algorithm then computes the associated expected cost using Equation 8 and finally keeps track of the minimum cost to select the value of t_u which keeps the waste at a minimum.

C. Poisson and Weibull examples

Common choices for the distribution of the random variable $\mathbb{F}$ are the Poisson and the Weibull distribution [9], [15], [16], [27]. Here we provide examples for both models but emphasize that Equation 8 is derived independently of them, making the algorithms of Section V-B general. Algorithms must be aware of all properties from $\mathbb{F}$, which for Weibull includes the scaling factor w. The probability density function of the example random variable $\mathbb{F}$ is

$$p(x) = \frac{w}{\lambda_w}\Big(\frac{x}{\lambda_w}\Big)^{w-1} e^{-(\frac{x}{\lambda_w})^w} \qquad \text{for } x \geq 0 \tag{9}$$

with the parameter

$$\lambda_w = \frac{M}{\Gamma(1 + \frac{1}{w})} \tag{10}$$

where $w = 1$ gives the Poisson distribution and $w \neq 1$ the Weibull distribution. For these distributions, the first truncated moment μ^t (the incomplete mean) is given by, e.g. [18],

$$\mu^t = \int_0^t x p(x)\,\mathrm{d}x \tag{11}$$

$$= \begin{cases} M\gamma\left(\frac{1}{w} + 1, \left(\frac{t\Gamma(\frac{1}{w}+1)}{M}\right)^w\right) & \text{if Weibull} \\ M\big(1 - \big(\frac{t}{M} + 1\big)\big)\exp\big(-\frac{t}{M}\big) & \text{if Poisson} \end{cases} \tag{12}$$

where $\Gamma(\cdot)$ is the one parameter gamma function, $\gamma(\cdot,\cdot)$ is the two parameter gamma function and w the Weibull factor that models Weibull failure distributions [23]. Our introduction of truncated moments for checkpoint determination allows us to determine modeling parameters numerically.

Additionally the failure probability until time t is

$$P(t) = \begin{cases} 1 - \exp\left(-\left(\frac{t\Gamma\left(1+\frac{1}{w}\right)}{M}\right)^w\right) & \text{if Weibull} \\ 1 - \exp\big(-\frac{t}{M}\big) & \text{if Poisson} \end{cases} \tag{13}$$

D. Usability

Real world HPC scenarios show that MPF jobs are significant and hint that previous approaches can miscalculate their intervals leading to higher expected costs and wasted core hours. Using Equation 8 and Algorithm 1, system administrators may implement a small calculator for their users that can provide them at job start with the best checkpointing interval for reducing average waste. Users need to only provide the job characteristics through already existing batch system submission scripts. The calculator can then compute an interval for the remaining runtime and the user may tune their checkpointing method for that interval. The result is less expected costs from using interval based checkpoints.

VI. Simulation and Results

This section explores the simulation of jobs within HPC systems to compare our MPF aware algorithm to Daly's (2006) second order estimate [9], Bouguerra's (2009) Weibull approach [7], Subasi's (2017) general method [28], Jayasekara's (2019) method considering failure detection costs [19] and Tiwari's (2014) waste-aware (OCI) approach [29].

We also perform an initial comparison to Bougeret's (2011) heuristic DPNextFailure (DPNF) [5]. Unfortunately computing intervals with DPNF for the simulated jobs is computationally extremely expensive due to its $\mathcal{O}(t^3)$ complexity. Since the DPNF method seems not to provide significant improvements in MPF scenarios compared to the other five methods, we just performed initial comparisons to corroborate Bougeret's findings that DPNF is suited for high node counts equivalent to our high probabilities of failure.

In the following simulations we model the failure characteristics with a synthetic Weibull failure distribution and a weight factor of 0.8. Weibull is a good fit for the LANL ATLAS system as shown by Yuan *et al.* [33]. We also provide summarized simulations with Poison failures for completion.

The MTBF is then scaled to the node count of each job through

$$\mathrm{MTBF}_{\mathrm{job}} = \mathrm{MTBF}_{\mathrm{machine}} \cdot \frac{N_{\mathrm{machine}}}{N_{\mathrm{job}}} \tag{14}$$

where N_{machine} and N_{job} denote the node number of the whole machine and the job, respectively. This scaling is consistent with modeling in previous works [15], [17].

To ensure robust statistic, we simulated $F = 10,000$ failure events. The average number of successes and failed attempts coincide with the failure probability of the job. We refer to our checkpoint intervals as the "**aware intervals**", since their computation explicitly accounts for the reduced failure probability of jobs. In order to compare the methods, we compare against the average checkpoint cost produced by the simulation ($\hat{S}$) from Equation 1. This cost includes the cost of checkpointing and the cost of computations lost from failures, but we refeer to it only as checkpointing cost.

We implemented a single-threaded version of Algorithm 1 with Python 2.7 using the NumPy 1.16 and SciPy 1.2 libraries and an Intel Xeon E5-2650 CPU @ 2.00GHz. Using this setup, an implementation of Equation 8 evaluates 25,000 waste costs per second and Algorithm 1 computes at least 400 aware intervals per second. This algorithm offers rates way above the requirement of a typical HPC batch system, while the algorithm can be trivially parallelized.

A. Interval and cost function simulation

We start by comparing all algorithms (using Equation 8) against the simulated average checkpointing costs of Equation 1 using F failure events at time t_f.

Figure 4 shows the simulated and the estimated costs of four distinct application runtimes t as a function of the checkpoint interval τ. We mark the location of each interval, the full final costs for the simulation (blue) and our model E_C (red). All panels shown are examples of jobs running on a system with

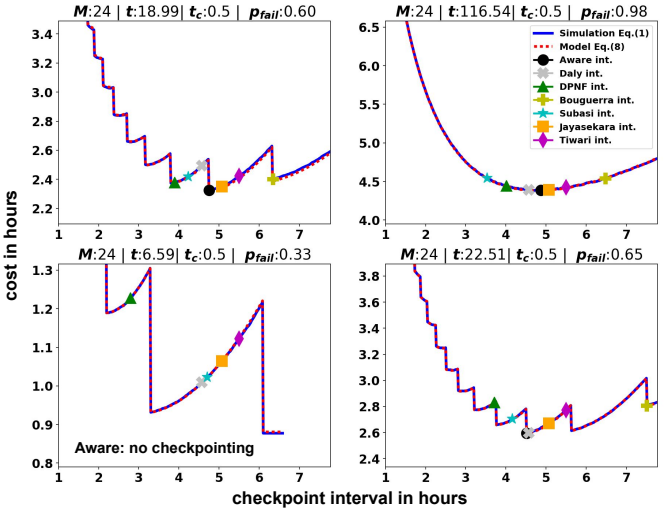

Figure 4: Simulated and aware-algorithm costs of checkpointing for a range of intervals. The panels show the costs for jobs with 0.33%, 60%, 65% and 99% probability of failing. Our aware algorithm is shown in a black circle.

a MTBF of 24 hours and checkpoint cost of half an hour. The jobs have a runtime of 6.59h, 18.99h, 22.51h and 142.90h as well as failure probabilities 0.33%, 60%, 65% and 99%.

The first noteworthy feature is the accuracy of the model in relation to the stochastically simulated events. Numerically, the standard error of costs for our model is consistently below 10^{-2} or 1.6 minutes. Qualitatively, the total cost follows closely the simulated curves across the entire range of checkpointing intervals.

The next interesting property is the stepped shape of the cost function that is more pronounced for jobs with low and medium failure probabilities as seen by comparing the left panels to the right ones. These sharp steps arise from the floor function within our cost function. The steps cause inefficient cost estimation for the alternative methods that do not weight the checkpoint costs by the full checkpoint count. This assumption is acceptable for large failure probabilities where the steps smooth out making the traditional methods consistent with their own modeling algorithms. However, this traditional assumption fails to model the costs of MPF jobs, while our method is able to find the interval with the lowest value in this stepped curve.

As seen in Figure 4, it is by coincidence that alternative methods report an interval with a low cost at the bottom of a step depending on the parameters t, t_c, M. In the top left example Jayasekara and Daly were close to our aware interval, with the first having a lower cost than the former. Although DPNF and Tiwari were further off than Daly, they obtained lower costs by sitting in lower parts of the steps. This shows how other methods might have increased checkpoint costs with comparable intervals to our aware method. We can report that this inexact behavior at lower probabilities is present regardless of the distribution or failure parameters. The bottom left panel shows how our method can recommend not checkpointing at all when other methods would recommend at least one checkpoint. As the probability of failure increases, the difference to

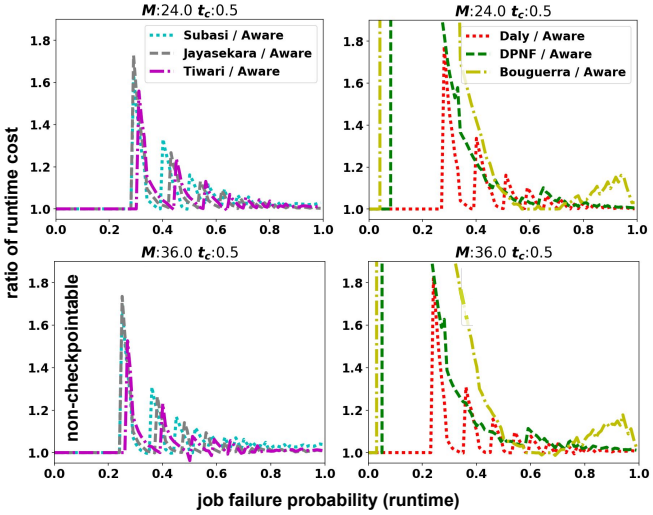

Figure 5: Relative cost between alternative methods and our aware checkpointing approach for applications with runtimes corresponding to failure probabilities of $0 < P < 1$.

the Daly cost as given by its interval diminishes. This is seen on the bottom right panel and top right panel.

Qualitatively the overall shape of the curve is smooth and concave in the limit $P \to 1$, thus a minimum can be analytically derived in the classical scenario. The cost difference from our method to the alternatives only consistently shrinks as $P \to 1$. This is shown in the right panel where the simulated and modeled cost curves become smoother. In this scenario our method converges to Daly's method because we achieve the traditional assumption that jobs will definitely fail. The cost of Jayasekara also approaches Daly's estimate for $P \to 1$. The other methods might offer diverging intervals in this scenario, but the cost differences are much smaller as a result of the smooth curve. At this probability the cost among methods becomes smaller and the choice of method can be left to usage requirements that match the specialization of the alternative methods.

B. Checkpoint cost overhead

Figure 5 shows the relative extra cost of the alternative methods compared to our aware approach. The costs are shown for a range of application runtimes as a function of their failure probability $0 < P(t) < 1$. Each point in the curve shows the relative costs when using the computed interval of the compared methods.

The step-wise cost change is clear by the diverging costs between the methods that occur regularly. The ratio of all but one method (Bouguerra) approaches one for $P \to 1$. Bouguerra diverges from the rest with up to 19% extra checkpoint cost. The ratio for all methods is larger for MPF jobs and becomes smaller as jobs become more likely to fail. From the ratio of costs depicted in the right panels we see how Bouguerra and DPNF have the highest cost overhead at very low probabilities and that the Daly interval can produce 80% more overhead than our aware method. In the left two panels we see that the stepped cost overhead of the Subasi, Jayasekara and Tiwari methods start at 60% and lowers rapidly to 20% ending at minimal overhead for high $P \to 1$. In the lower left panel Subasi has slightly higher costs than the rest (except Bouguerra) at high probabilities close to one. Depending on the runtime and failure characteristics, the alternative methods offer similar costs to our approach for $P \to 1$, but incur significant overheads at medium probabilities. This is consistent with the modeled equations and examples of Figure 4. In our experience, MPF jobs with $0.2 < p_{fail} < 0.7$ benefit most from our method depending on the parameters.

C. HPC system traces

In this section, we move from synthetic benchmarks to queues of jobs sampled from four different systems: Mogon II (2,000 nodes) [13], LANL Trinity (9,408 nodes) and LANL Mustang (1,600 nodes) [2] and TGCC Curie (5,544 nodes) [11]. The Mogon II system has the most MPF jobs and Curie the least.

We simulate the runtime of all the checkpointable jobs from these traces while performing checkpoints using the intervals provided by each interval method. We present results for MPF and non-MPF jobs within the traces (where $t_c+\tau < t$), explore the MTBF-values $M \in \{24, 36\}$ hours and the checkpoint times $t_c \in \{6, 15, 30\}$ minutes.

We report the absolute checkpoint costs in hours and relative savings to the alternative methods. In addition we test four different failure versions with the $w = 0.8$ Weibull failures, a Poisson failure distribution and two system where the MTBF is uncertain and off by -20% and +20% (Weibull). This leads to a total of 480 different simulations (4 systems $\times$ 5 methods $\times$ 6 parameters $\times$ 4 versions). Each of the 480 simulations performs 10,000 failures for each job, where the average checkpoint cost per job is summed. The final sums are then used to calculate the checkpointing cost savings. Each simulation compares an alternative method to our aware approach and is unique with distinct random failure events for each job. This effectively compares our aware approach to each alternative method individually. The aware method is executed again within each unique simulated event. This explores as many distinct random sample events from the same failure distribution as possible, but makes it hard to compare between alternative methods.

1) Weibull: Table I-a lists the summarized total checkpoint savings of Weibull failures considering all jobs, including MPF and non-MPF jobs. The summarized row shows average total savings across all systems of at least 7.1% compared to Jayasekara and Daly. We consider these two methods the best performing alternatives. The checkpoint cost savings compared to Tiwari, Subasi and Bouguerra are 18.2%, 25.7% and 16.2% on average across all systems.

Looking at specific systems, our approach saves between 11.3% and 19.6% compared to Tiwari for the Mogon II traces and between 4.1% and 5.0% for the Curie system. Our aware approach offers the most savings compared to the Subasi method, with 56.8% cost savings for configurations with $0.10h$ checkpoint costs. This is because Subasi performs worse for low checkpoint costs on MPF jobs. The savings against Jayasekara and Daly are between 9.1% and 24.4% for

Weibull $w = 0.8$			Total checkpoint cost savings of all jobs (MPF and not) of Alg. 1 compared to method...					e)
System ↓	Parameters							
	MTBF	t_c	Tiwari [29]	Subasi [28]	Jayasekara [19]	Daly [9]	Bouguerra [6]	← Avg.
MUSTANG	36h	0.50h	7.1%	6.2%	5.6%	5.6%	16.6%	13.0%
		0.25h	7.3%	25.0%	5.5%	5.4%	19.0%	
		0.10h	7.0%	57.6%	5.3%	5.2%	22.0%	
	24h	0.50h	6.2%	5.5%	5.3%	5.3%	15.5%	
		0.25h	6.4%	23.2%	4.5%	4.7%	17.8%	
		0.10h	6.8%	56.8%	5.1%	5.0%	21.0%	
ATLAS	36h	0.50h	8.0%	6.6%	6.2%	6.5%	6.5%	9.0%
		0.25h	7.2%	15.6%	5.5%	7.0%	8.4%	
		0.10h	5.8%	44.7%	5.7%	6.0%	13.1%	
	24h	0.50h	6.2%	7.3%	7.5%	8.0%	2.4%	
		0.25h	6.6%	15.4%	5.8%	5.3%	8.5%	
		0.10h	6.1%	42.9%	4.6%	4.1%	14.5%	
MOGON II	36h	0.50h	11.3%	10.5%	9.4%	10.4%	18.4%	18.1%
		0.25h	11.9%	30.3%	9.1%	9.5%	19.8%	
		0.10h	17.0%	39.3%	13.0%	13.0%	26.7%	
	24h	0.50h	11.6%	10.8%	10.0%	10.1%	16.0%	
		0.25h	16.8%	24.2%	24.4%	22.4%	20.1%	
		0.10h	19.6%	39.9%	18.8%	18.5%	30.7%	
CURIE	36h	0.50h	5.0%	5.1%	4.0%	4.0%	13.4%	10.5%
		0.25h	4.7%	18.9%	3.6%	3.4%	16.1%	
		0.10h	4.8%	55.3%	3.3%	3.1%	18.7%	
	24h	0.50h	4.3%	4.7%	3.4%	3.4%	12.5%	
		0.25h	4.1%	17.0%	3.0%	2.8%	14.7%	
		0.10h	4.2%	54.5%	2.8%	2.6%	16.9%	
a) ↑ This Weibull			8.2%	25.7%	7.1%	7.1%	16.2%	
b) With Poisson			27.5%	25.1%	7.3%	7.7%	10.9%	
c) Same Weibull -20% MTBF			11.1%	24.9%	6.0%	6.0%	15.9%	
d) Same Weibull +20% MTBF			12.5%	26.6%	7.5%	7.5%	16.3%	

Table I: Savings when using our approach for four HPC systems with a combination of two MTBF values $24h, 36h$ and two checkpoint costs $0.25h, 0.5h$. The simulations of a), c) and d) are done using Weibull failures with $w = 0.8$. Poisson failures are used for b).

the Mogon II system and around 3% to 5% for Mustang, Atlas and Curie.

Against the Bouguerra method our approach saves between 12.5% and 30.7% for the Mustang, Mogon II and Curie traces. The big savings on Mogon II against Bouguerra can be attributed to MPF jobs. The savings on Curie are because of the degrading performance of Bouguerra on big probabilities of failure as seen in Figure 5. Bouguerra performs best on the Atlas system with savings of 2.4% for the 24h MTBF and 0.50h checkpoint cost. For the Curie system our method shows consistent savings less than 5% for the Tiwari, Jayasekara and Daly methods and between 13.4% and 18.7% compared to the Bouguerra method.

In Table I-e we can see average savings of 13.0% across all methods for the Mustang traces. The Mogon II system sees the most savings across all methods with 18.1%. Finally the Atlas and Curie traces see 9.0% and 10.5% savings.

For completion we show the absolute values used for the relative savings on Table II. The cost from each alternative method is shown (right) against the respective cost of our aware method (left). The absolute costs of the methods are only comparable to the respective aware cost due to the uniqueness of each simulation as described in Section VI-C. This was done to avoid re-running costly simulations as new alternative methods were tested and to explore as many distinct sample events from the same failure distribution as possible. The drawback of this approach is that it becomes difficult to compare alternative methods with each other.

2) Poisson: In Table I-b we see that the average cross system savings for Poisson failures increase to 27.5% against Tiwari and remain similar for Jayasekara and Daly at 7.3% and 7.7% respectively. The savings against Subasi and Bouguerra decrease from 14.1% to 12.5% and 10% respectively. The savings remain at an average of over 7% against all methods.

3) MTBF Uncertainty: Next we address the savings that our method can provide against the alternatives in case of a not accurately known MTBF. This represents a common scenario since new systems have not yet collected enough data to determine their own MTBF and would therefore have to extrapolate MTBF guesses from external data. We address this uncertainty by computing the optimal checkpointing intervals with an uncertain value for the MTBF, perturbed by factors -20% and $+20\%$, while still simulating the costs using the "real" MTBF values of 24 and 36 hours for Weibull failures. Both the aware and alternative methods use the uncertain MTBF to determine checkpoint intervals.

Weibull $w = 0.8$			Total overhead time in hours (aware & alternative) from the checkpoint cost of all jobs (MPF and not) in each simulation				
System ↓	Parameters						
	MTBF	t_c	Tiwari [29]	Subasi [28]	Jayasekara [19]	Daly [9]	Bouguerra [6]
MUSTANG	36h	0.50h	52820 / 56840	52967 / 56455	53474 / 56635	53759 / 56936	52603 / 63090
		0.25h	42622 / 45956	42579 / 56801	42880 / 45362	42971 / 45406	42579 / 52571
		0.10h	30698 / 33009	30698 / 72488	30731 / 32449	30745 / 32432	30698 / 39355
	24h	0.50h	66044 / 70412	66293 / 70158	66915 / 70625	67301 / 71037	65729 / 77819
		0.25h	52347 / 55949	52285 / 68062	52693 / 55203	52828 / 55431	52285 / 63578
		0.10h	37279 / 40011	37271 / 86363	37341 / 39354	37367 / 39320	37271 / 47195
ATLAS	36h	0.50h	802 / 872	823 / 881	820 / 874	836 / 894	746 / 797
		0.25h	707 / 763	693 / 821	714 / 755	722 / 776	693 / 757
		0.10h	556 / 590	555 / 1003	559 / 593	561 / 596	555 / 638
	24h	0.50h	1063 / 1133	1097 / 1183	1086 / 1173	1126 / 1224	1023 / 1048
		0.25h	897 / 960	877 / 1037	918 / 974	925 / 977	878 / 960
		0.10h	684 / 729	680 / 1191	686 / 719	687 / 717	680 / 795
MOGON II	36h	0.50h	2303 / 2597	2321 / 2595	2387 / 2636	2474 / 2760	2298 / 2815
		0.25h	2328 / 2643	2322 / 3330	2368 / 2604	2387 / 2639	2322 / 2895
		0.10h	6250 / 7528	6250 / 10292	6252 / 7188	6253 / 7185	6250 / 8525
	24h	0.50h	3359 / 3801	3381 / 3793	3424 / 3803	3491 / 3881	3215 / 3827
		0.25h	7511 / 9031	7504 / 9898	7532 / 9965	7540 / 9713	7504 / 9392
		0.10h	10694 / 13298	10694 / 17796	10706 / 13180	10722 / 13152	10694 / 15436
CURIE	36h	0.50h	98445 / 103589	98743 / 104090	98943 / 103087	99232 / 103335	98017 / 113145
		0.25h	73681 / 77332	73594 / 90689	73907 / 76645	73982 / 76553	73594 / 87673
		0.10h	549360 / 51850	49348 / 110411	49421 / 51091	49434 / 51011	49348 / 60729
	24h	0.50h	124587 / 130151	124853 / 131017	125057 / 129434	125292 / 129749	123935 / 141707
		0.25h	91944 / 95835	91446 / 110187	91805 / 94604	92202 / 94902	91446 / 107246
		0.10h	60564 / 63190	60325 / 132490	60418 / 62131	60435 / 62038	60325 / 72588

Table II: Total checkpoint cost (time h) for all 10,000 simulations of each job using the six alternative methods(on the right) and our aware approach (on the left). The values shown correspond to the simulations of Table I.

The savings for uncertain MTBF simulations are shown to be above 6.0% and 7.5% in Tables I-c and I-d. Savings against Tiwari increase slightly to 11.1% and 12.5% for an inaccurate MTBF of -20% and $+20\%$ respectively. The savings over Jayasekara and Daly are diminished by 1.1% for the under estimation and increased by 0.4% for the over estimation.

Performing no checkpointing can save the entire overhead of checkpointing, but computations are lost every time a failure happens, incurring in a cost nevertheless. We observed that our aware approach saves 49.39% of the failure costs on average compared to no checkpointing at all.

4) Takeaway: From the simulations we can see how our aware method can provide significant savings for entire HPC workloads that include MPF and non-MPF jobs. Against the best alternative methods our approach can obtain average savings of 7.1%. Furthermore our approach is more robust against non-accurately known MTBF values with average cost savings of at least 6.0%. The savings are more significant for systems like Mogon II where MPF workloads are very prominent. Systems with other types of jobs like Curie benefit less when a good enough alternative method is used.

The best alternative methods are Jayasekara and Daly and we attribute the performance of Jayasekara to their solution using a Lambert function that solves the exponential equation present in the Poisson and Weibull failure distributions. The performance of Daly is related to our method converging to it at $p_{fail} \to 1$.

Our gains over the alternatives come from their assumption that jobs fail with a high probability. Further gains over Daly and Jayasekara come from them not considering Weibull failures. Additional gains over Subasi and Tiwari come from their simple mathematical form that slightly deviates from Daly's by using a hazard function and work-lost factor respectively. Gains over Bouguerra can be attributed to the its bad performance at large probabilities of failures. All these methods may have merits under their own modeling for $p_{fail} \to 1$, but are sub-optimal for MPF jobs.

VII. Conclusion

Checkpointing is an important resilience method in HPC systems to reduce the loss of computations in case of failures. However, the runtime overhead of checkpointing highly depends on the choice of the checkpoint interval and many interval optimization studies have focused on jobs with a high probability of failure, e.g., long-running jobs that use a large part of the HPC system. Our observations of real HPC systems revealed that jobs with a medium probability of failure (MPF) are common and that current techniques fail to consider them in checkpoint interval calculations.

In this paper we have proposed an MPF job aware method for finding the checkpointing interval that minimizes the expected cost of checkpointing for jobs with a medium probability of failure. This aware method also contains the classical solutions for jobs with a high probability of failure. Moreover, in contrast to previous works, our approach is independent of a specific failure distribution and offers new examples of how to accurately model cost properties using truncated moments and weighted costs. The experiments shown employ job traces of four HPC systems, five alternative methods, two failure distributions and six failure parameters.

The savings for individual jobs with a medium probability of failure may amount up to 40% of the checkpointing cost

average. For the four job traces of the HPC systems we report cumulative checkpoint savings (of all jobs) from 7.1% to 25.7% when considering Weibull failures and between 7.3% to 27.5% for Poisson failures. The Mogon II HPC system sees the highest average savings across all methods with 18.1% checkpoint cost savings followed by the LANL Mustang system with 13.0% savings. The LANL Atlas and Curie HPC systems see 9.0% and 10.5% savings respectively. Finally we also showed that our method has savings between 6.0% and 26.6% when only inaccurate MTBF values are available.

ACKNOWLEDGMENT

This work was supported by the German Ministry for Education and Research (BMBF) under grant 01|H16010A.

REFERENCES

[1] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben. Bigger, longer, fewer: what do cluster jobs look like outside google? Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-17-104, 2017.

[2] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben. On the diversity of cluster workloads and its impact on research results. In *USENIX Annual Technical Conference (ATC), Boston, MA, USA, July 11-13*, pages 533–546, 2018.

[3] G. Aupy, Y. Robert, and F. Vivien. Assuming failure independence: Are we right to be wrong? In *Int. Conf. on Cluster Computing (CLUSTER), USA*, pages 709–716, 2017.

[4] N. Bessho and T. Dohi. Comparing checkpoint and rollback recovery schemes in a cluster system. In *12th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), Fukuoka, Japan, September 4-7*, pages 531–545, 2012.

[5] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien. Checkpointing strategies for parallel jobs. In *Conference on High Performance Computing Networking, Storage and Analysis (SC), Seattle, WA, USA, November 12-18*, pages 33:1–33:11, 2011.

[6] M. Bouguerra, A. Gainaru, L. A. Bautista-Gomez, F. Cappello, S. Matsuoka, and N. Maruyama. Improving the computing efficiency of HPC systems using a combination of proactive and preventive checkpointing. In *27th IEEE International Symposium on Parallel and Distributed Processing (IPDPS), May 20-24*, pages 501–512, 2013.

[7] M. Bouguerra, T. Gautier, D. Trystram, and J. Vincent. A flexible checkpoint/restart model in distributed systems. In *8th International Conference on Parallel Processing and Applied Mathematics (PPAM), Wroclaw, Poland, September 13-16.*, volume 6067, pages 206–215, 2009.

[8] A. Brinkmann, K. Mohror, W. Yu, P. H. Carns, T. Cortes, S. Klasky, A. Miranda, F. Pfreundt, R. B. Ross, and M. Vef. Ad hoc file systems for high-performance computing. *J. Com. Sci. Tech.*, 35(1):4–26, 2020.

[9] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computing Systems (FGCS)*, 22(3):303–312, 2006.

[10] N. El-Sayed and B. Schroeder. Checkpoint/restart in practice: When 'simple is better'. In *2014 IEEE International Conference on Cluster Computing (CLUSTER), Madrid, Spain, September 22-26*, pages 84–92. IEEE Computer Society, 2014.

[11] J. Emeras. The cea curie log. https://www.cs.huji.ac.il/labs/parallel/workload/l_cea_curie, 2012.

[12] A. Frank, T. Süß, and A. Brinkmann. Effects and benefits of node sharing strategies in HPC batch systems. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 43–53. IEEE, 2019.

[13] A. Frank, D. Yang, A. Brinkmann, M. Schulz, and T. Süß. Reducing false node failure predictions in HPC. In *26th IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC), Hyderabad, India, December 17-20*, pages 323–332, 2019.

[14] R. Gad, S. Pickartz, T. Süß, L. Nagel, S. Lankes, A. Monti, and A. Brinkmann. Zeroing memory deallocator to reduce checkpoint sizes in virtualized HPC environments. *J. Sup.*, 74(11):6236–6257, 2018.

[15] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari. Failures in large scale systems: long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver, CO, USA, November 12 - 17*, pages 44:1–44:12, 2017.

[16] E. M. Heien, D. Kondo, A. Gainaru, D. Lapine, B. Kramer, and F. Cappello. Modeling and tolerating heterogeneous failures in large parallel systems. In *Conference on High Performance Computing Networking, Storage and Analysis (SC), Seattle, WA, USA, November 12-18*, pages 45:1–45:11, 2011.

[17] T. Hérault, Y. Robert, A. Bouteiller, D. C. Arnold, K. B. Ferreira, G. Bosilca, and J. J. Dongarra. Optimal cooperative checkpointing for shared high-performance computing platforms. In *IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Vancouver, BC, Canada*, pages 803–812, 2018.

[18] J. W. Jawitz. Moments of truncated continuous univariate distributions. *Advances in Water Resources*, 27(3):269 – 281, 2004.

[19] S. Jayasekara, A. Harwood, and S. Karunasekera. A utilization model for optimization of checkpoint intervals in distributed stream processing systems. *Future Gener. Comput. Syst.*, 110:68–79, 2020.

[20] H. Jin, Y. Chen, H. Zhu, and X. Sun. Optimizing HPC fault-tolerant environment: An analytical approach. In *39th International Conference on Parallel Processing (ICPP), California, USA, 13-16 September*, pages 525–534, 2010.

[21] J. Kaiser, R. Gad, T. Süß, F. Padua, L. Nagel, and A. Brinkmann. Deduplication potential of HPC applications' checkpoints. In *IEEE International Conference on Cluster Computing, CLUSTER , Taipei, Taiwan*, pages 413–422. IEEE Computer Society, 2016.

[22] C. Lu, J. C. Browne, R. L. DeLeon, J. Hammond, W. L. Barth, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra. Comprehensive job level resource usage measurement and analysis for XSEDE HPC systems. In *Extreme Science and Engineering Discovery Environment, XSEDE13, San Diego, CA, USA, July 22 - 25*, pages 50:1–50:8, 2013.

[23] R. P. McEwen and B. R. Parresol. Moment expressions and summary statistics for the complete and truncated weibull distribution. *Communications in Statistics - Theory and Methods*, 20(4):1361–1372, 1991.

[24] Y. Qian, X. Li, S. Ihara, A. Dilger, C. Thomaz, S. Wang, W. Cheng, C. Li, L. Zeng, F. Wang, D. Feng, T. Süß, and A. Brinkmann. LPCC: hierarchical persistent client caching for lustre. In M. Taufer, P. Balaji, and A. J. Peña, editors, *Proceedings SC 2019, Denver, Colorado, USA, November 17-19, 2019*, pages 88:1–88:14. ACM, 2019.

[25] Y. Qian, X. Li, S. Ihara, L. Zeng, J. Kaiser, T. Süß, and A. Brinkmann. A configurable rule based classful token bucket filter network request scheduler for the lustre file system. In B. Mohr and P. Raghavan, editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017, Denver, CO, USA, November 12 - 17, 2017*, pages 6:1–6:12. ACM, 2017.

[26] K. Sato, K. Mohror, A. Moody, T. Gamblin, B. R. de Supinski, N. Maruyama, and S. Matsuoka. A user-level infiniband-based file system and checkpoint strategy for burst buffers. In *14th Int. Symp. on Cluster, Cloud and Grid Comp., CCGrid 2014, USA*, pages 21–30.

[27] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. *IEEE Trans. Dependable Secur. Comput.*, 7(4):337–351, 2010.

[28] O. Subasi, G. Kestor, and S. Krishnamoorthy. Toward a general theory of optimal checkpoint placement. In *IEEE International Conference on Cluster Computing (CLUSTER), September 5-8*, pages 464–474, 2017.

[29] D. Tiwari, S. Gupta, and S. S. Vazhkudai. Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems. In *44th Annual Int. Conf. on Dependable Systems and Networks*, pages 25–36, 2014.

[30] M. Vef, N. Moti, T. Süß, M. Tacke, T. Tocci, R. Nou, A. Miranda, T. Cortes, and A. Brinkmann. Gekkofs - A temporary burst buffer file system for HPC applications. *J. Comp. Sci. Tech.*, 35(1):72–91, 2020.

[31] T. Wang, K. Mohror, A. Moody, K. Sato, and W. Yu. An ephemeral burst-buffer file system for scientific applications. In J. West and C. M. Pancake, editors, *Proc. of the Int. Conf. for High Perf. Comp., Net., St. and Analysis, SC 2016, USA*, pages 807–818. IEEE Computer Society.

[32] J. W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, Sept. 1974.

[33] Y. Yuan, Y. Wu, Q. Wang, G. Yang, and W. Zheng. Job failures in high performance computing systems: A large-scale empirical study. *Computers & Mathematics with Applications*, 63(2):365–377, 2012.