Realizing Best Checkpointing Control in Computing Systems

Purushottam Sigdel, *Member, IEEE*, Xu Yuan, *Member, IEEE*, and Nian-Feng Tzeng, *Fellow, IEEE*

Abstract—This article considers best checkpointing control realizable in real-world systems, whose mean time between failures (MTBFs) often fluctuate. The considered control scheme is based on equating aggregate checkpointing overhead over an activity sequence of interest (θ) and the expected rework amount after a failure recovery for best checkpointing, called "CHORE" (i.e., checkpointing overhead and rework equated), where θ starts from execution resumption after failure recovery and ends after restore from the following failure. CHORE lets its inter-checkpoint intervals in θ follow a pre-determined sequence independent of MTBF to aim at performance optimality and is shown analytically to keep overall execution time overhead upper bounded. When failure occurrences are tracked during job execution for real-time MTBF estimation, an enhanced CHORE (dubbed En-CHORE) is obtained to lower checkpointing overhead by skipping certain checkpoints at the beginning of each θ before taking checkpoints with the most desirable inter-checkpoint intervals determined on-the-fly for best checkpointing control. En-CHORE can outperform optimal checkpointing (which follows a fixed inter-checkpoint interval optimized for one constant global MTBF known *a prior*) both under synthetic random failures with local MTBF fluctuating markedly and under real failure traces of 22 real HPC systems (whose failure rates actually fluctuate over their trace time spans).

Index Terms—Absorbing Markov chains, checkpointing control, execution time overhead, mean time between failures (MTBFs), optimal checkpointing, rework after failure recovery

1 Introduction

Fault tolerance for computing systems becomes indispensable as their sizes grow, and it often involves checkpointing in order to permit execution state restoration after system failures occur [1], [24]. Checkpointing comes with overhead both in time and in storage, contributed mainly by halting execution temporarily for snapshotting the execution state and by transferring and storing the checkpointed data volume. Ample research articles proposed to reduce the checkpointed data volume [8], [11], [22], [23], [25], while others focus on the checkpoint placement techniques that aim to minimize overall execution time overhead [2], [3], [5], [13], [18], [26], [29].

Most of the previous checkpointing studies assume that the mean time between failures (MTBF) of a computing system (1) is known *a priori* and (2) stays constant throughout the course of job execution. A recent article analyzed the failure characteristics of high-performance computing using real failure logs [14], [38], [39], [40], revealing that various computing systems comprising the same hardware component type (i.e., identical system nodes) can have their node failure rates vary up to $50\times$ among different systems. In addition, the system's failure rate also varies during job execution since a heavy load yields a higher node temperature than a light load [28]. Likewise, execution may experience a drastically different failure rate from one spatial locality to another (i.e., from one cluster to another cluster of computing resources) [30], [38], [40] and execution during the daytime versus the nighttime [14], [30]. For example, fewer jobs are running during the night, so there will be more idle nodes that have lower chances to fail. Even during the stable operational period, it is revealed in [30] that MTBF may vary by up to $4\times$. When GPUs are equipped in its compute units, a system suffers from elevated MTBF fluctuation due to unprotected errors (like multi-bit corruption) in GPUs [31], [33], [38], [39]. Separately, the MTBF of disk drives is unveiled to vary by up to $30\times$ of the published value [16]. Those findings contradict the common, simplified assumption that a given computing system has a constant overall MTBF over its execution duration, based on which the optimal inter-checkpoint interval is determined (as reported in [2], [3], [18], [26], [29]). Meanwhile, findings in [5], [14], [30], [32] reflect that the Weibull distribution [41] with decreasing hazard represents the failures more realistically in computing systems. Under the Weibull distribution, both scale and the shape parameter (which can vary widely over time [30]) are required.

Since a real-world system may find its MTBF either unknown a *priori* or constantly changing instead of a constant, prior studies on optimal checkpointing (OPT) control under constant MTBF are usually ineffective in practice. To quantify the execution time overhead levels as the system's MTBF fluctuates away from the given MTBF, we have measured the total execution times by simulation under various inter-checkpoint intervals, each of which is optimized for a given MTBF value (called $\text{MTBF}_{\text{ideal}}$) according to prior studies [2], [3], [18], [26], [29]. As shown in Fig. 1, the execution time overhead ratio ($\boldsymbol{\mu}$) is plotted over a range of MTBF values (M) when the inter-checkpoint interval is optimized

• *The authors are with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70503.*
E-mail: {purushottam.sigdel1, xu.yuan, tzeng}@louisiana.edu.

Manuscript received 13 Sept. 2019; revised 9 July 2020; accepted 7 Aug. 2020. Date of publication 11 Aug. 2020; date of current version 31 Aug. 2020. (Corresponding author: Purushottam Sigdel.)
Recommended for acceptance by M. Becchi.
Digital Object Identifier no. 10.1109/TPDS.2020.3015805

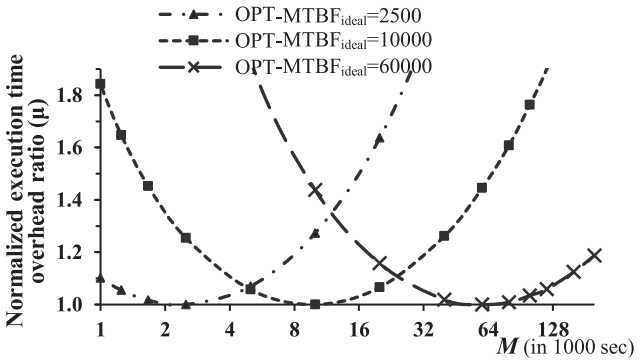


Fig. 1. Normalize execution time overhead (μ) over a range of MTBF values for a system with the inter-checkpoint interval governed respectively by $MTBF_{ideal}$ of 2500 sec, 10000 sec, and 60000 sec for optimal checkpointing, where μ is the ratio of overall execution time overhead measured under $MTBF_{ideal}$ to that under a given MTBF, and it bottoms ($= 1.0$) if MTBF equals $MTBF_{ideal}$.

for $MTBF_{ideal}$, with $MTBF_{ideal} = 2500$ sec, 10000 sec, and 60000 sec, corresponding respectively to the curves of OPT-2500, OPT-10000, and OPT-60000. The ratio of μ is defined as the overall execution time overhead measured under a given MTBF over that under $MTBF_{ideal}$, where each checkpoint is assumed to take 20 sec, so is the restoration time after a failure.

Clearly, overall execution time overhead is higher (so is μ) if a given system's MTBF deviates more from $MTBF_{ideal}$, as illustrated by curves in Fig. 1. For example, the system with its MTBF = 1000 (or 5000) sec but following periodic checkpoints at the optimal interval of $MTBF_{ideal} = 10000$ sec throughout job execution, it then takes 1.84 × (or 1.06 ×) longer than that of the best possible (in an ideal scenario where the inter-checkpoint interval is optimized for MTBF = 1000 (or 5000) sec), as can be found on the curve of OPT-$MTBF_{ideal} = 10000$. In essence, execution time overhead, in this case, rises (so does μ) when its actual MTBF is smaller than $MTBF_{ideal}$ because job execution then has a longer inter-checkpoint interval than the best desirable, involving a higher "rework" amount after each failure despite lower aggregate checkpointing overhead, where rework refers to previously executed job work that was not checkpointed and thus lost to a failure. Similarly, execution time overhead is also elevated if its actual MTBF is larger than $MTBF_{ideal}$ since job execution then has a shorter inter-checkpoint interval than the best desirable one, involving more aggregate checkpointing overhead, despite a smaller rework amount after each failure.

From the simulation results given in Fig. 1, we are motivated to address "effective checkpointing control which is oblivious to varying MTBF values during the course of job execution in a real-world computing system, while upper bounding overall execution time overhead." To this end, we have undertaken the OPT (optimal checkpointing) analysis for given MTBF using the absorbing Markov chain [15] as outlined in Section 3, arriving at a unique insight as follows: for OPT over an activity sequence of interest (denoted by θ), the aggregate checkpointing time over θ always equals the expected rework amount after failure recovery, where θ starts from execution resumption after failure recovery (or execution onset at the very beginning), and it ends after restore from the following failure. Exploiting the unique insight, our checkpointing control lets checkpoints taken in such a way that the aggregate checkpointing time within θ equates approximately the rework amount after failure recovery. Referred to as "CHORE" (short for 'checkpointing overhead and rework equated'), this proposed control targets OPT during the course of job execution in a real system, irrespective of its actual failure scenario. CHORE lets its inter-checkpoint intervals in θ follow an identified sequence oblivious to MTBF, in order to satisfy our unique insight as best as possible.

A detailed analysis is conducted on CHORE control under system failures that are assumed to occur randomly, following a Poisson distribution with given MTBF. The analytic result given in Section 3.2.2 uncovers that CHORE incurs at most 26 percent extra execution time overhead when compared with the smallest possible time overhead of its ideal counterpart whose inter-checkpoint interval is optimized for $MTBF_{ideal}$. For example, if an ideal system with constant $MTBF_{ideal}$ incurs 4.2 percent execution time overhead (over one without checkpointing at all) following its optimal inter-checkpoint interval, CHORE exhibits no more than 5.3 percent execution time overhead. In sharp contrast, OPT with a constant inter-checkpoint interval optimized for $MTBF_{ideal}$ can exhibit arbitrarily large execution time overhead in a real system if its actual MTBF fluctuates widely, as shown in Fig. 1.

When tracking failure instances during job execution for real-time MTBF estimation to realize best checkpointing control adaptively in real systems, enhanced CHORE (denoted by En-CHORE, detailed in Section 4) is obtained to (1) first skip certain early checkpoints in each θ and (2) then follow a specified inter-checkpoint interval sequence to take checkpoints until the next failure occurs (when θ ends). Our simulation evaluation for a wide range of MTBF confirms that En-CHORE lowers the time overhead ratio of μ up to 1.01 (from 1.26 under CHORE) if failures happen randomly without MTBF fluctuation, governed by a Poisson distribution with given MTBF. In addition, En-CHORE brings μ below 1.00 × (to be faster than OPT), if the MTBF of a system varies considerably (likely in practice) so that its failures to happen randomly with moderate to high fluctuation.

Evaluation results under real traces of 22 different HPC systems at Los Alamos National Lab (LANL) [36] reveal that, on an average, CHORE (or En-CHORE) incurs no more than 1.13 × (or 1.00×) execution time overhead when compared with its theoretically OPT counterparts. It should be noted that the results under OPT are *unrealizable in practice* because they require "known" MTBF values (which are derived from entire failure traces globally) prior to execution onset. Assuming the MTBF of a system to stay constant throughout job execution, OPT can be inferior to En-CHORE under actual trace evaluation, because the MTBF of a real system may fluctuate widely, making OPT that follows one fixed inter-checkpointing interval throughout entire execution slightly underperformed.

En-CHORE exhibits better execution performance than the recently considered checkpointing control schemes, known as SIMPLE [34], [35] (which tracks failure instances during job execution to estimate MTBF for determining the next inter-checkpointing interval) and SKIP [5] (which skips one checkpoint over every θ to reduce the overall checkpointing cost).

TABLE 1
Comparative Features and Performance Potentials

Schemes	Fixed and known MTBF a prior	Tracking failures during execution	Real applicability	Execution performance
CHORE	No	No	Yes	Good
OPT	Yes	No	No	Better
En-CHORE	No	Yes	Yes	Best
SIMPLE [35]	No	Yes	Yes	Better
SKIP [5]	No	Yes	Yes	Better

TABLE 2
List of Repetitively Used Symbols

Symbol	Description
P	Probability to complete an interval without failure
$\boldsymbol{w}$	Job execution per interval
$\boldsymbol{c}$	Checkpoint cost
$\boldsymbol{r}$	Restoration cost
M	MTBF
NET^2	Normalized expected turnaround time
$\ominus$	Activity sequence of interest (i.e., duration between two consecutive failures)
β (or α)	Total checkpointing and rework overhead under CHORE (or Optimal)

Table 1 lists the defining features and performance potentials of the four checkpointing control schemes: CHORE, OPT, En-CHORE, SIMPLE, and SKIP. As can be seen in the table, OPT needs the MTBF value before execution onset and assumes MTBF to be fixed throughout job execution, making it infeasible for practical use, albeit to its better performance potential. On the other hand, all other three schemes require *no knowledge about MTBF a priori* and accommodate MTBF fluctuation during job execution, rendering them applicable to real-world systems. CHORE does not track failure instances during job execution, but En-CHORE, SIMPLE, and SKIP do, with En-CHORE to exhibit the best execution performance among all five schemes.

The contributions of this paper are four-fold: (1) it reveals that best checkpointing control results if the aggregate checkpointing cost always equals the expected rework cost after failure recovery, (2) it analytically proves that the overall execution time overhead under CHORE is upper bounded by $1.26\times$ of that under OPT (the optimal one), (3) it derives En-CHORE by tracking failure instances for real-time MTBF estimation to realize best checkpointing control in real systems on-the-fly, and (4) it evaluates CHORE and En-CHORE extensively under both random failures (in the absence and the presence of MTBF fluctuation) and the real failure logs of 22 different HPC systems, confirming their superior checkpointing behaviors. Note that manually adjusting checkpointing intervals in the course of job execution (according to failure instances encountered) is infeasible practically because a long-running job with checkpointing always undertakes automatic recovery from any encountered failure using the recent checkpointing file(s) *without halting job execution* (when manual checkpointing interval adjustment may then be made).

2 Pertinent Background

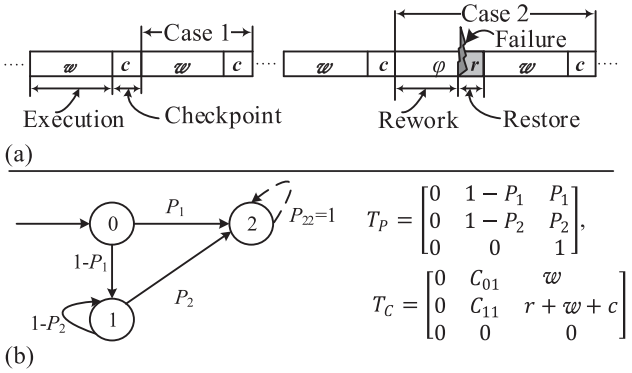
2.1 Checkpointing Techniques

Various checkpointing techniques have been considered for lowering the total execution time overhead either by reducing data volume involved in checkpoints taken during the course of application run [8], [11], [22], [23], [25] or by determining the optimal periodic checkpoint interval [2], [3], [18], [26], [29], when the checkpoint latency is often known via execution profiling before an actual application run starts, and some mechanisms [1], [6], [7] consider both techniques. Other techniques resort to dynamic checkpointing which either predicts the checkpoint cost on-the-fly to determine appropriate points of time for checkpointing [1], [6], [19] or omits certain periods like in application-initiated periodic checkpoints [17]. Still, others adopt incremental checkpointing [1], [6], [22], [25], which collects only dirty pages or memory blocks since the immediate prior checkpoint to become the next checkpoint with a reduced footprint. Similarly, multilevel checkpointing was proposed [7], [13] to reduce I/O contention over a parallel file system. The checkpoints from multiple nodes were combined and compressed with considering different data types in [8]. Hash-based incremental checkpointing is tested in [4], [12], [22] where a hash of immediate prior version of a data block is compared with the hash of current version of the same data block. Although these approaches reduce the checkpoint data volume, the additional processing is still in execution critical path. Another technique proposes to offload the checkpointing job to a separate unused core [6] while the job is executed concurrently to reduce the overall execution time if unused core exists.

Assuming a fixed MTBF value and a given checkpointing cost (which is the same for restoration) of a system, earlier work [2], [3], [18], [26], [29] estimated the optimal inter-checkpoint interval, while other studies [14], [16], [21] showed that MTBF values were not fixed in real-world systems, often varying markedly during the course of job execution. Hence, the assumption of constant MTBF is generally inapplicable to real-world systems. Checkpoint/restart can be done with different restore costs. For example, Large-scale high-performance computing systems typically do a full application abort and restart into a new set of computing nodes, while more intelligent solutions may use spare nodes in the system so that spare nodes are still available to substitute the failed participating node(s).

2.2 Checkpointing With Constant MTBFs

Earlier checkpoint analyses under constant MTBFs simplify their analytic derivation with different assumptions, but they are unrealistic for real-world systems. Specifically, Young analyzed the OPT interval without taking the restoration cost into account [29], while Benoit *et al*. assumed no failures during checkpointing and restoration for OPT derivation [2]. Both analytic results ignore the restoration cost and reveal that the optimal inter-checkpoint interval is fixed, equal to $\sqrt{2M\boldsymbol{c}}$, where M and $\boldsymbol{c}$ are MTBF and the checkpointing cost, respectively. Meanwhile, another analysis which takes the restoration cost into account [18], arrives at the OPT interval of $\boldsymbol{w} = \sqrt{2(M+\boldsymbol{r})\boldsymbol{c}}$, with $\boldsymbol{r}$ being the restoration cost, under the first order assumption, while the interval becomes $\boldsymbol{w} = \sqrt{2M\boldsymbol{c}} - \boldsymbol{c}$, for $\boldsymbol{c} < M/2$, under a high order approximation.



$$T_P = \begin{bmatrix} 0 & 1-P_1 & P_1 \\ 0 & 1-P_2 & P_2 \\ 0 & 0 & 1 \end{bmatrix}, \quad T_C = \begin{bmatrix} 0 & C_{01} & w \\ 0 & C_{11} & r+w+c \\ 0 & 0 & 0 \end{bmatrix}$$

Fig. 2. (a) Execution and checkpointing activities during job execution, where Case 1 (or Case 2) represents the scenario without failures (or with a failure), and (b) Modeling the expected time between two consecutive checkpoints, with restoration and rework costs taken into consideration, by the absorbing Markov chain, where T_P and T_C denote transition probability matrix and transition cost matrix, respectively.

3 CHORE Checkpointing Control

Based on the absorbing Markov chain, this section first analyzes *two consecutive checkpoints* to derive the mean time over two consecutive checkpoints, with (1) failures possible to happen randomly even during checkpointing and restoration and (2) all restoration and rework costs taken into consideration. Such an analysis gives rise to optimal inter-checkpointing characteristics, which signify that (1) the restore cost ($\boldsymbol{r}$) has no impact on the optimal inter-checkpoint interval and (2) there exists one and only one desirable inter-checkpoint interval $\boldsymbol{w}$ for optimal inter-checkpointing, under random failures to follow a Poisson distribution with a given constant MTBF. Our derived optimal $\boldsymbol{w}$ agrees with that reported earlier under a high order approximation [18]. This section then analyzes *optimal checkpointing durations* that cover failures (if existing) in ideal computing systems and in real-world systems, separately.

3.1 Analysis on Two Consecutive Checkpoints

Long-running scientific applications involve a sequence of execution and checkpointing activities denoted respectively by $\boldsymbol{w}$ and $\boldsymbol{c}$, as shown in Fig. 2a. If a failure happens, execution state restore (using the last checkpoint file) is invoked before execution resumption. The restore cost ($\boldsymbol{r}$) includes the time for (1) reading the immediate prior checkpoint, (2) transferring it (checkpoint file) to a substitute node, and (3) restoring the job execution state in the substitute node. On the other hand, the rework cost ($\boldsymbol{\varphi} \leq \boldsymbol{w}$) involves the re-execution of the previously executed job that has not been checkpointed yet, from the last checkpoint till the point to which the failure happens. It has been proved in [3] that the optimal (shortest) execution under a given mean failure rate is achieved if the inter-checkpoint intervals are kept the same.

An absorbing Markov chain model [15] we adopt to derive the expected execution time over two consecutive checkpoints is shown in Fig. 2b, where to simplify the derivation failures are assumed to occur randomly (following a Poisson distribution,[1] with the MTBF of M) and possibly at checkpointing and restoration durations. The model includes three states: initial state (State 0), intermediate state (State 1), and absorbing state (State 2, which incurs no cost). Beginning at State 0, the process stays there till either (1) it completes a checkpoint without experiencing any failure and then enters State 2 or (2) a failure happens before finishing a checkpoint, it then enters State 1. The preceding two scenarios correspond respectively to Case 1 and Case 2 depicted in Fig. 2a. Whenever the process transits to State 1, it invokes restoration with an overhead of $\boldsymbol{r}$ seconds followed by re-execution for $\boldsymbol{w}$ seconds and a checkpoint of $\boldsymbol{c}$ sec. The process then transits to State 2 if there is no failure during the period of $\boldsymbol{r}+\boldsymbol{w}+\boldsymbol{c}$ sec; otherwise, it remains at State 1 and repeats the aforementioned activities, as shown in Fig. 2b.

Let P_1 (or P_2) be the transition probability from State 0 to State 2 (or State 1 to State 2), with the associated state transition cost being $\boldsymbol{w}+\boldsymbol{c}$ (or $\boldsymbol{r}+\boldsymbol{w}+\boldsymbol{c}$). The state transition cost from $\boldsymbol{i}$ to $\boldsymbol{j}$ refers to the mean time spent on State $\boldsymbol{i}$ before making a transition to State $\boldsymbol{j}$. Fig. 2b lists the transition probability matrix and the transition cost matrix, denoted respectively by T_P and T_C. Since failures follow a Poisson distribution with MTBF of M, the transition probabilities of P_1 and P_2 are given respectively by $P_1 = e^{-(\boldsymbol{w}+\boldsymbol{c})/M}$ and $P_2 = e^{-(\boldsymbol{r}+\boldsymbol{w}+\boldsymbol{c})/M}$. Thus, the probability of transition from State 0 to State 1 equals $(1-P_1) = 1-e^{-(\boldsymbol{w}+\boldsymbol{c})/M}$, and that from State 1 to itself is $(1-P_2) = 1-e^{-(\boldsymbol{r}+\boldsymbol{w}+\boldsymbol{c})/M}$. Similarly, the state transition cost from State 0 to State 1 (i.e., C_{01}; see T_C in Fig. 2b) denotes the expected time spent in State 0 before transiting to State 1. This cost can be calculated by using the failure probability density function at time τ in an arbitrary execution interval of $\boldsymbol{w}+c$, as expressed by $f(\tau) = \frac{\frac{1}{M}e^{-\tau/M}}{(1-e^{-(\boldsymbol{w}+\boldsymbol{c})/M})}$ [15]. Hence, the expected random time in the range of $0 \leq \tau \leq (\boldsymbol{w}+\boldsymbol{c})$ is given by

$$C_{01} = \frac{\int_0^{\boldsymbol{w}+\boldsymbol{c}} (\tau)\frac{1}{M}e^{-\frac{\tau}{M}}d\tau}{\left(1-e^{-\frac{\boldsymbol{w}+\boldsymbol{c}}{M}}\right)} = M - \frac{\boldsymbol{w}+\boldsymbol{c}}{\left(e^{\frac{\boldsymbol{w}+\boldsymbol{c}}{M}}-1\right)} = M - \frac{(\boldsymbol{w}+\boldsymbol{c})P_1}{(1-P_1)}. \tag{1}$$

Similarly, the state transition cost from State 1 to itself (i.e., C_{11}; see T_C in Fig. 2b) is expressed by

$$C_{11} = M - \frac{(\boldsymbol{r}+\boldsymbol{w}+\boldsymbol{c}) \cdot P_2}{(1-P_2)}. \tag{2}$$

With all transition probabilities and transition costs obtained, the mean execution time of an interval is estimated by adding the expected time spent in State 0 (denoted by T_0) and State 1 (denoted by T_1), with T_0 and T_1 being calculated by

$$T_0 = P_1 \cdot (\boldsymbol{w}+\boldsymbol{c}) + (1-P_1)\left[M - \frac{(\boldsymbol{w}+\boldsymbol{c}) \cdot P_1}{(1-P_1)}\right] = (1-P_1) \cdot M, \tag{3}$$

$$\begin{aligned} T_1 &= \frac{(1-P_1)}{P_2}\left[P_2 \cdot (\boldsymbol{r}+\boldsymbol{w}+\boldsymbol{c}) + (1-P_2)\left[M - \frac{(\boldsymbol{r}+\boldsymbol{w}+\boldsymbol{c}) \cdot P_2}{(1-P_2)}\right]\right] \\ &= \frac{(1-P_1)}{P_2}(1-P_2) \cdot M. \end{aligned} \tag{4}$$

The term of $(1-P_1)$ in T_1 is the probability of transition from State 0 to State 1, while $1/P_2$ represents how many

1. A more realistic failure model is represented with the Weibull distribution [5, 14, 30, 32].

times the process transits from State 1 to itself because P_2 is the probability to exit from State 1. The total mean execution time, T, is given by

$$\begin{aligned} T &= T_0 + T_1 \\ &= M.(1 - P_1)/P_2 = M \cdot \left(1 - e^{-(\boldsymbol{w}+\boldsymbol{c})/M}\right)/e^{-(\boldsymbol{r}+\boldsymbol{w}+\boldsymbol{c})/M} \\ &= M \cdot e^{\boldsymbol{r}/M} \cdot \left(e^{(\boldsymbol{w}+\boldsymbol{c})/M} - 1\right). \end{aligned} \tag{5}$$

We are interested in normalized expected turnaround time (NET^2), defined as the ratio of the total mean execution time (T) to the base process execution time for that interval, $\boldsymbol{w} > 0$, i.e.,

$$NET^2 = T/\boldsymbol{w} = Me^{\boldsymbol{r}/M}\left(e^{(\boldsymbol{w}+\boldsymbol{c})/M} - 1\right)/\boldsymbol{w}. \tag{6}$$

Differentiating Eq. (6) with respect to $\boldsymbol{w}$ and setting it to 0, we get the minimum value point (i.e., $\boldsymbol{w}$), as below:

$$\begin{aligned} &\frac{d(NET^2)}{d\boldsymbol{w}} = 0 = (\boldsymbol{w} - M)e^{(\boldsymbol{w}+\boldsymbol{c})/M} + M \\ &\Rightarrow M - \boldsymbol{w} = Me^{-(\boldsymbol{w}+\boldsymbol{c})/M}. \end{aligned} \tag{7}$$

Interestingly, Eq. (7) is independent of the restoration cost ($\boldsymbol{r}$). Additionally, the second derivative of NET^2 is monotonically increasing for $\boldsymbol{w} > 0$ and hence, any value of $\boldsymbol{w} > 0$ which satisfies Eq. (7) gives the minimum value of NET^2. Furthermore, the right-hand side of Eq. (7) is an exponentially decaying function with a maximum value of $Me^{-\boldsymbol{c}/M}$ (i.e., $\leq M$) and it reaches zero when $\boldsymbol{w}$ approaches ∞, while the left-hand side function is linearly decreasing with a maximum value of M, reaching zero when $\boldsymbol{w}$ approaches M. Thus, there exists one and only one desirable inter-checkpoint interval $\boldsymbol{w}$ (with $0 < \boldsymbol{w} \leq M$) that minimizes NET^2. We approximate $e^{-(\boldsymbol{w}+\boldsymbol{c})/M}$ in Eq. (7) with a second order polynomial based on $e^{-(\boldsymbol{w}+\boldsymbol{c})/M} = 1 - (\boldsymbol{w}+\boldsymbol{c})/M + (\boldsymbol{w}+\boldsymbol{c})^2/2M^2$ to get

$$\boldsymbol{w} = \sqrt{2M\boldsymbol{c}} - \boldsymbol{c}, \tag{8}$$

which happens to equal the result derived earlier by Daly under a high order approximation [18], as reviewed in Section 2.2. Adopting the Newton-Raphson method [27] with an initial value of $\boldsymbol{w}_0 = \sqrt{2M\boldsymbol{c}} - \boldsymbol{c}$, we can approach precise $\boldsymbol{w}$ quickly via iterations, with its first precision digit obtained usually in no more than three iterations.

To corroborate our analytical finding that the restore cost (i.e., $\boldsymbol{r}$) has no contribution over an inter-checkpoint interval, we simulated a long-running job with varying $\boldsymbol{r}$ values, as depicted in Fig. 3. In the figure, the normalized execution turnaround time (NET^2) plotted against the inter-checkpoint interval is governed by a convex curve, where NET^2 refers to the execution time normalized with respect to that without checkpointing and without failures. Every NET^2 value shown is averaged over 10,000 event-driven simulation runs. The system is evaluated under the parameter values of per checkpointing cost ($\boldsymbol{c}$) = 20 sec, MTBF (i.e., M) = 10000 sec, and restore cost ($\boldsymbol{r}$) ranging from 0 to 4x$\boldsymbol{c}$ sec. Started at 20 sec, the inter-checkpoint interval maintains 5-sec precision throughout simulation for all sets of results, as shown in Fig. 3. The simulation results illustrate that the optimal inter-checkpoint interval for

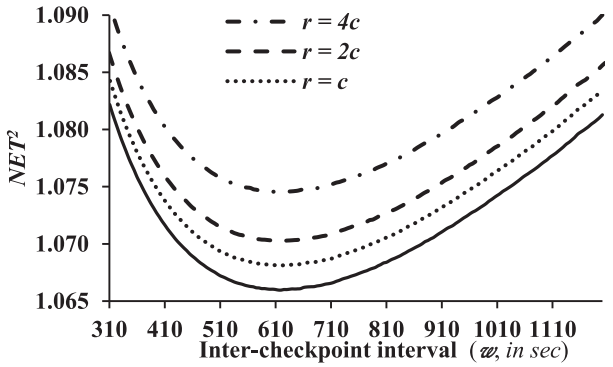


Fig. 3. Normalized expected turnaround time (NET^2) versus inter-checkpoint interval (w) under varying restore values (r), $M = 10000$ sec, and $\boldsymbol{c} = 20$ sec.

all four $\boldsymbol{r}$ values (i.e., $\boldsymbol{r} = 0$, $\boldsymbol{c}$, 2x$\boldsymbol{c}$, and 4x$\boldsymbol{c}$) stays the same, equal to 610 sec, whereas the analytical value (i.e., $\sqrt{2M\boldsymbol{c}} - \boldsymbol{c}$) is 612.45 sec. Although the optimal inter-checkpoint interval is independent of $\boldsymbol{r}$, the associated NET^2 value under the optimal inter-checkpoint interval of 610 sec increases by 0.2, 0.4, and 1.0 percent when $\boldsymbol{r}$ rises from 0 to $\boldsymbol{c}$, 2x$\boldsymbol{c}$, and 4x$\boldsymbol{c}$, respectively.

Observation 1. The absorbing Markov chain accurately models the execution time overhead of a system. For a given MTBF of M and checkpointing cost of $\boldsymbol{c}$, the optimal inter-checkpointing interval is expressed by $\boldsymbol{w} = \sqrt{2M\boldsymbol{c}} - \boldsymbol{c}$.

3.2 Analyzing Execution Duration

Each analyzed checkpointing duration here involves at most one failure because a new duration begins after failure recovery to the immediate last checkpoint. A duration starts from either execution onset (see Fig. 4) or execution resumption after failure recovery (see Fig. 4b), and it ends after restore from the following failure, involving an activity sequence of interest (denoted by θ). In other words, θ covers repeated execution ($\boldsymbol{w}$) and checkpoint ($\boldsymbol{c}$) activities, followed by a failure, which incurs restoration of $\boldsymbol{r}$ and rework of $\boldsymbol{\varphi}$.

3.2.1 Optimal Checkpointing in Ideal Systems

As considered by all previous checkpointing control studies for minimizing execution time overhead, an ideal system is assumed to have its MTBF (1) known a priori and (2) fixed during the entire course of an execution run and from one run to another [2], [3], [18], [26], [29]. From Eq. (1), $\boldsymbol{\varphi}$ for OPT (optimal checkpointing) in an ideal system with a constant MTBF of M can be derived by $M - (\boldsymbol{w}+\boldsymbol{c})/(e^{(\boldsymbol{w}+\boldsymbol{c})/M} - 1)$. Using Laurent series expansion of $1/(e^{(\boldsymbol{w}+\boldsymbol{c})/M} - 1)$ in the preceding expression, we have

$$\varphi = M - (\boldsymbol{w}+\boldsymbol{c})\left[\frac{M}{\boldsymbol{w}+\boldsymbol{c}} - \frac{1}{2} + O\left(\frac{1}{M^1}\right)\right] = \frac{\boldsymbol{w}+\boldsymbol{c}}{2} + O\left(M^{-1}\right), \tag{9}$$

which reflects that the expected rework amount is nearly equal to one half of the interval (i.e.,$\sim (\boldsymbol{w}+\boldsymbol{c})/2$, after truncating high-order terms).

Similarly, we obtain aggregate execution time overhead over θ due to checkpoints alone as $\boldsymbol{c} \cdot EC^{\#}_{ideal}$, where $EC^{ideal}_{\#}$ is the expected number of complete intervals with repeated $\boldsymbol{w}$ and $\boldsymbol{c}$ activities over θ in an ideal system. To estimate

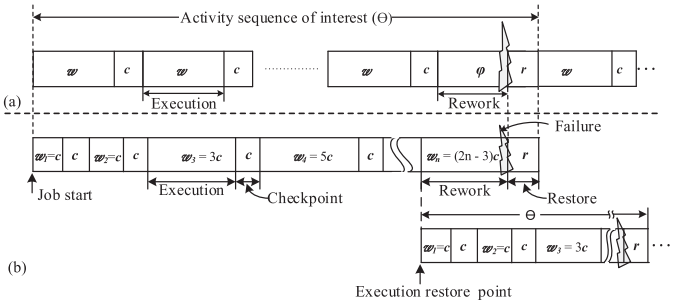


Fig. 4. Activity sequence of interest (⊖) during the course of job execution, under two scenarios: (a) optimal checkpointing, which requires known, constant MTBF, and (b) CHORE, whose inter-checkpoint intervals (w_i) follow a pre-determined sequence (of c, c, $3c$, $5c$, $7c$, etc.) oblivious to MTBF.

$EC^{\#}_{ideal}$, we use the negative binomial distribution for determining the expected number of completed intervals. Failures are assumed to occur randomly (following a Poisson distribution) with their MTBF equal to M, as commonly adopted earlier [2], [3], [18], [26], [29]. Additionally, OPT in ideal systems has a fixed inter-checkpoint interval of $w + c = \sqrt{2Mc}$ (from Eq. (8)). Let P be the probability to complete the execution and checkpointing (without a failure happens) of an interval of $(w + c)$ sec. The probability of a failure happen within an interval of $(w + c)$ is given by $(1 - P) = 1 - e^{-(w+c)/M} = 1 - e^{-\sqrt{2c/M}}$. Since the total number of completed intervals equals zero if a failure happens in the very first interval, we have the mean number of completed intervals equal to

$$\begin{aligned} EC^{\#}_{ideal} &= 0 \cdot (1-P) + 1 \cdot P(1-P) + 2 \cdot P^2(1-P) + \ldots + \\ & i \cdot P^i(1-P) + \ldots \ to \infty terms \\ &= (1-P)P(1 + 2 \cdot P + 3 \cdot P^2 + 4 \cdot P^3 + \ldots \ to \infty terms \\ &= (1-P)P/(1-P)^2 = P/(1-P) = 1/\Big(e^{\sqrt{2c/M}} - 1\Big). \end{aligned}$$

Expanding $1/(e^{\sqrt{2c/M}} - 1)$ in the preceding expression via Laurent Series, we get

$$EC^{\#}_{ideal} = \sqrt{M/2c} - 1/2 + O\Big(M^{-1/2}\Big). \tag{10}$$

Therefore, the aggregate execution time overhead over θ due to checkpoints only is given by $c \cdot EC^{\#}_{ideal} = \sqrt{Mc/2} - c/2 + O(M^{-1/2})$, and for M $\gg$ c, the value can be approximated by $c \cdot EC^{\#}_{ideal} \approx \sqrt{Mc/2} = (w + c)/2$. It reflects that the aggregated checkpointing time over θ is nearly equal to one half of the interval (i.e.,$\sim (w + c)/2$), which equals expected rework after failure recovery, according to Eq. (9). Hence, for OPT in an ideal system, the aggregate checkpointing time over θ always equals expected rework after failure recovery.

Based on the result derived above to characterize inter-checkpoint intervals, CHORE is designed to undertake checkpoints in such a way that the aggregate checkpointing time over θ approximates $\frac{1}{2}$ of the inter-checkpoint interval as best as possible on the fly during job execution to aim at performance optimality, as follows.

Observation 2. Under optimal checkpointing, aggregated checkpointing time overhead over θ is nearly equal to one half of an interval (i.e., $\sim (w + c)/2$), and it equals the expected rework after failure recovery (φ).

3.2.2 CHORE Checkpointing in Real Systems

Given MTBF of a real system has been shown to highly fluctuate [14], [16], [21], it is desirable to pursue checkpointing control for computing systems with unknown or fluctuating failure rates. To this end, we first introduce a control strategy that yields checkpointing overhead and rework equated (CHORE) over θ. Unlike employing a fixing inter-checkpoint interval for an ideal system, CHORE varies the ith inter-checkpoint interval over θ, denoted by w_i (see Fig. 4b). It first lets a job run for $w_1 = c$ sec before taking a checkpoint (with job execution paused) that involves overhead of c sec. The execution and checkpointing activities then repeat until a failure occurs, with inter-checkpoint intervals (w_i) governed by the control strategy below. Each checkpoint is assumed to involve a fixed overhead of c.

Control Strategy. *Under* CHORE control, *the* ith (*for* $i \geq 2$) *inter-checkpoint interval*, w_i, *over* θ *is set to* $w_i = (2i-3) \cdot c$, *before taking a checkpoint*.

An example of the CHORE checkpointing control strategy is illustrated in Fig. 4b, where the execution state is restored after the failure, using the most recent checkpoint file(s), and θ then concludes. The subsequent θ starts from the previous checkpoint until the next failure, and it involves rework, whose expected amount is nearly equal to one half of the very last inter-checkpoint interval, as indicated by Eq. (9). The inter-checkpoint intervals in θ under CHORE always follow the specified sequence of c, c, $3c$, $5c$, $7c$, etc. and they are oblivious to MTBF.

Following CHORE control, checkpointing is adaptive to the failure characteristics of a system, without knowing its MTBF *a priori* or fixing its MTBF. Execution and checkpoint activities over θ under CHORE always satisfy that the aggregate checkpoint cost equals the mean rework amount after failure recovery, to target optimality, as proved next.

Proposition 1. *The aggregate checkpointing cost over* θ under *CHORE always equals mean rework after failure recovery.*

Proof. To prove the proposition, we use the negative binomial distribution with varying probability for determining the expected interval number at which a failure occurs. Failures are assumed to occur randomly (following a Poisson distribution) with their MTBF equal to M. □

Let P_i be the probability to complete the execution and its subsequent checkpointing (without a failure) of ith interval for $w_i + c$ sec, where $w_i = (2i-3) \cdot c$ for $i \geq 2$, with $w_1 = c$, as shown in Fig. 4b. The probability that failure happens in the ith interval is thus given by $(1 - P_i) = 1 - e^{-(w_i+c)/M}$, leading to the expected interval number at which the failure occurs as

$$\begin{aligned} E^{\#} &= (1-P_1) + 2 \cdot P_1(1-P_2) + 3 \cdot P_1P_2(1-P_3) + \ldots + \\ & i \cdot P_1P_2 \ \ldots \ P_{i-1}(1-P_i) + \ldots \ to \infty \ terms. \end{aligned}$$

Substituting the value of P_i for all i's in the above expression followed by some algebraic manipulations, we have

$$E^{\#} = 1 + \sum_{i=1}^{\infty} e^{-(2+i\cdot(i-1))\,\cdot\, \boldsymbol{c}/M}. \tag{11}$$

Over an activity sequence of interest (θ), a failure strikes at the last interval, i.e., Interval $E^{\#}$. Since $\boldsymbol{w}_i$ under CHORE is $(2i-3)\cdot \boldsymbol{c}$ according to the control strategy given above, the total length of Interval $E^{\#}$ equals $(2\cdot E^{\#} - 3)\cdot \boldsymbol{c} + \boldsymbol{c} = 2\cdot \boldsymbol{c}\times (E^{\#} - 1)$, which becomes $2\cdot \boldsymbol{c} \times \sum_{i=1}^{\infty} e^{-(2+i\cdot(i-1))\,\cdot\, \boldsymbol{c}/M}$ from Eq. (11). One half of the very last interval in θ represents the expected rework amount (see Eq. (9)), which equals

$$\boldsymbol{c} \times \sum_{i=1}^{\infty} e^{-(2+i\cdot(i-1))\,\cdot\, \boldsymbol{c}/M}. \tag{12}$$

Each completed interval over θ incurs the checkpointing cost of $\boldsymbol{c}$. If failure strikes in the very first interval, none interval has been completed, yielding the mean number of completed intervals before failure as

$$EC^{\#} = 0\cdot(1-P_1) + 1\cdot P_1(1-P_2) + 2\cdot P_1P_2(1-P_3) + \ldots + i\cdot P_1P_2\ \ldots\ P_i(1-P_i) + \ldots\ to\ \infty\ terms.$$

Replacing the value of P_i for all i's in the above expression followed by some algebraic manipulations, we have

$$EC^{\#} = \sum_{i=1}^{\infty} e^{-(2+i\cdot(i-1))\,\cdot\, \boldsymbol{c}/M}.$$

The aggregate checkpointing cost over θ thus equals $\boldsymbol{c}\times EC^{\#} = \boldsymbol{c}\times \sum_{i=1}^{\infty} e^{-(2+i\cdot(i-1))\,\cdot\, \boldsymbol{c}/M}$, which is identical to the mean rework amount after failure recovery, as expressed by Eq. (12). This completes the proof. ■

Multiple θ sequences may exist during job execution, dictated by the number of failures present. Note that if the whole job execution duration experiences no failure, no matter how long the duration is, the system at hand must be failure-free, signifying no checkpointing at all. CHORE suitably fits in this scenario as the inter-checkpoint interval grows unboundedly without encountering a failure.

- *Execution time overhead under CHORE*

Oblivious to MTBF, CHORE is analyzed to get its execution performance for contrasting against that achieved on an ideal system with a known and fixed MTBF. Given that CHORE incurs an equal amount of aggregated checkpointing cost and rework after failure recovery over θ (as proved in Proposition 1), the total overhead due to checkpointing and rework (say, β) over θ is twice the value expressed by Eq. (12), i.e.,

$$\beta = 2\cdot \boldsymbol{c} \times \sum_{i=1}^{\infty} e^{-(2+i\cdot(i-1))\cdot\, \boldsymbol{c}/M}. \tag{13}$$

The infinite series (i.e., $\sum_{i=1}^{\infty} e^{-(2+i\cdot(i-1))\,\cdot\, \boldsymbol{c}/M}$) in Eq. (13) has no close form solution. For the estimation purpose, we have numerically evaluated the infinite series for various sets of $\boldsymbol{c}$ and M values, as shown in Fig. 5. The Y-axis of the figure represents the normalized sum (γ, defined as the ratio of the sum of the infinite series to $\sqrt{M/\boldsymbol{c}}$), while its X-axis denotes MTBF. Each curve represents γ for its associated $\boldsymbol{c}$ value over a wide range of M, and it levels off after a sufficient large MTBF (M), with γ always smaller than 0.887 for all $\boldsymbol{c}$ values examined. Curves in Fig. 5 signify that (1) for a given M, γ is smaller for a large $\boldsymbol{c}$, and (2) for a given $\boldsymbol{c}$, γ rises monotonically as M increase to plateau just below 0.89. As a result, we arrive at: $\sum_{i=1}^{\infty} e^{-(2+i\cdot(i-1))\,\cdot\, \boldsymbol{c}/M} < 0.887\cdot\sqrt{M/\boldsymbol{c}}$, giving rise to $\beta < 2\times 0.887\cdot\sqrt{M\boldsymbol{c}}$.

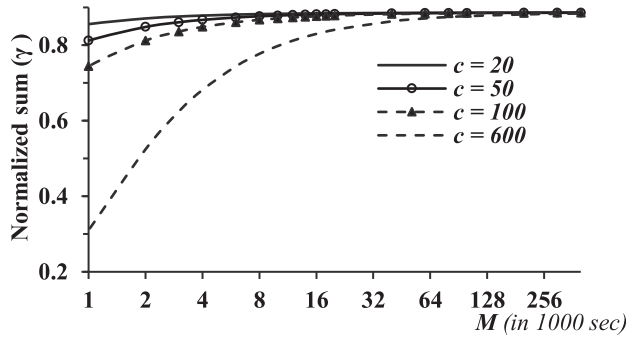


Fig. 5. Sums (γ) of the infinite series, normalized w.r.t. $\sqrt{\frac{M}{c}}$, versus MTBF, for various costs ($\boldsymbol{c}$).

On the other hand, optimal execution in an ideal system has a fixed inter-checkpoint interval of $(\boldsymbol{w}+\boldsymbol{c}) = \sqrt{2M\boldsymbol{c}}$ (from Eq. (8)), with the mean number of completed intervals over θ expressed by Eq. (10) as $EC^{\#}_{ideal} = \sqrt{M/2\boldsymbol{c}} - 1/2 + O(M^{-1/2})$. The optimal execution over θ incurs aggregate checkpoint overhead of $EC^{\#}_{ideal}\cdot \boldsymbol{c}$ and expected rework of $\sqrt{2M\boldsymbol{c}}/2$ (i.e., one half of an inter-checkpoint interval). Thus, overall checkpoint and rework overhead for optimal execution on an ideal system over θ (say, α) is given by

$$\alpha = \left(\sqrt{M/2\boldsymbol{c}} - 1/2 + O\left(M^{-1/2}\right)\right)\cdot \boldsymbol{c} + \left(\sqrt{2M\boldsymbol{c}}\right)/2,$$

which can be approximated as $\alpha \approx \sqrt{2M\boldsymbol{c}}$ for $M \gg \boldsymbol{c}$.

The degree to determine how close CHORE reaches the execution performance under OPT can be quantified by the ratio of β to α, i.e., $\beta/\alpha = (2\times 0.887\sqrt{M\boldsymbol{c}})/\sqrt{2M\boldsymbol{c}} = 1.2544$. It signifies that CHORE incurs less than 26 percent additional checkpointing and rework time overhead as compared to that under optimized checkpointing with a known, constant failure rate throughout job execution. For example, if optimized checkpointing has 4 percent execution time overhead (for any M and $\boldsymbol{c}$), CHORE incurs approximately 5 percent execution time overhead when M fluctuates and unknown during job execution. It is interesting to note that the time overhead increase amount of CHORE is upper bounded by 26 percent, irrespective of M and $\boldsymbol{c}$, as proved below.

- *Upper bound on time overhead under CHORE*

Excluding the restoration cost, the ratio of mean execution time overhead under CHORE to that under optimal checkpointing (i.e., $\beta/\alpha < 1.26$), serves as the basis for our proof of an upper bound for a non-zero restoration cost.

Proposition 2. *The mean execution time overhead under CHORE is upper bounded by 1.26 × of that under optimal checkpointing.*

Proof. Let $\delta \geq 0$ be the execution time overhead incurred by restoration cost only. For $\delta = 0$, we already observed that the expected execution time overhead under CHORE (i.e., β) is 1.26× as compared to that under optimal checkpoint (i.e., α). Now, for $\delta \neq 0$, assuming identical restoration cost upon failure, as shown in Fig. 4, both methods incur an

equal amount of additional execution time overhead (i.e., δ). Thus, the modified expected execution time overhead ratio became $(\beta + \delta)$ / $(\alpha + \delta)$. If we prove the modified ratio $(\beta + \delta)$ / $(\alpha + \delta) < \beta/\alpha$, it completes the proof. ■

Suppose the ratio satisfies the inequality, we have

$$(\beta+\delta)/(\alpha+\delta) < \beta/\alpha \Rightarrow 0 < \beta\cdot(\alpha+\delta) - -\alpha\cdot(\beta+\delta)$$
$$\Rightarrow 0 < \delta\cdot(\beta-\alpha).$$

Since $\beta > \alpha$, the inequality always holds for any $\delta > 0$, thereby completing the proof.

Note that results derived in this section assume that the checkpointing cost is constant ($\boldsymbol{c}$) throughout job execution. In practice, this assumption may not hold true. Nonetheless, the control strategy can use the immediate prior $\boldsymbol{c}$ as a basis to decide how long it should wait before placing another checkpoint. Note also that other control strategies are possible, expected to result in various bounds on execution time overhead.

Observation 3. CHORE with the ith (for i $\geq$ 2) inter-checkpoint interval, $\boldsymbol{w_i}$, over θ given by $\boldsymbol{w_i} = (2i-3)\cdot\boldsymbol{c}$, satisfies the optimality condition under unknown MTBF, and its execution time overhead is upper bounded by 1.26 $\times$ of that of optimal checkpointing under a given MTBF.

4 Enhanced CHORE (En-CHORE)

CHORE checkpoint control works superbly without knowing failure rates *a priori* or tracking failure instances during job execution for checkpointing control. With failure tracking for real-time MTBF estimation, however, CHORE can be enhanced to lower total execution time overhead by "skipping some early checkpoints" in each θ (an activity sequence of interest; see Fig. 4b), arriving at En-CHORE (enhanced CHORE). Specifically, the failure instances during job execution are tracked under En-CHORE to determine adaptively (1) if one or multiple early checkpoints in θ can be skipped and (2) the suitable inter-checkpointing intervals after checkpoint skipping until θ ends, when a failure occurs. In Fig. 4b, for example, if the failure instances gathered till the last failure (i.e., immediately before the onset of the current θ) indicate that skipping the very first two checkpoints is beneficial in lowering overall time overhead, En-CHORE skips those two early checkpoints, knowing that should a failure happen before the checkpoint following $\boldsymbol{w_3} = 3\boldsymbol{c}$ in Fig. 4b is completed, all work done from the start of θ until the failure point has to be repeated, incurring higher rework-after-failure overhead than that under CHORE. Note that although En-CHORE and the earlier control schemes of SKIP [5] and SIMPLE [34], [35] track failure instances during job execution for real-time MTBF estimation (as listed in Table 1), En-CHORE bases its estimated MTBF to determine two control parameters for minimal execution time overhead: (1) the optimal interval of checkpoint skips at the beginning of each θ and (2) the most desirable inter-checkpoint intervals, whereas SIMPLE uses the estimated MTBF to decide the next inter-checkpointing interval and SKIP skips just one (say, the second) checkpoint.

The two control parameters of En-CHORE to realize the best checkpointing are derived below in sequence.

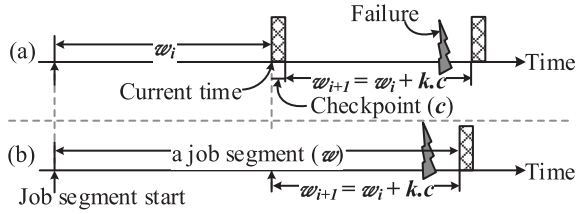


Fig. 6. Job execution segment of length w resided at the beginning of $\ominus$ and spanning two consecutive inter-checkpoint intervals of $\boldsymbol{w_i}$ and $\boldsymbol{w_{i+1}}$ under (a) checkpoint after $\boldsymbol{w_i}$ and (b) checkpoint skip.

4.1 Deriving Checkpoint Skips

For En-CHORE to determine the optimal number of checkpoint skips at the beginning of each θ, we consider a job execution segment of length $\boldsymbol{w}$ which spans two consecutive inter-checkpoint intervals of $\boldsymbol{w_i}$ and $\boldsymbol{w_{i+1}}$, as depicted in Fig. 6. Assuming failures to occur randomly (following a Poisson distribution) with MTBF of M, the probability of having a failure within interval $\boldsymbol{w_{i+1}}$ is governed by $(1 - e^{-\boldsymbol{w}_{i+1}/M})$. As the expected rework amount equals one half of the interval (see Eq. (9)) and $\boldsymbol{w_i}$ has been checkpointed in Fig. 6a, the total mean execution time overhead to complete the job segment of length $\boldsymbol{w}$ is

$$O_1 = \boldsymbol{c} + \left(1 - e^{-\frac{\boldsymbol{w}_{i+1}}{M}}\right)\cdot \boldsymbol{w}_i + 1/2.$$

On the other hand, execution of $\boldsymbol{w_i}$ in Fig. 6b is completed but not checkpointed, and hence, should a failure happen before completing $\boldsymbol{w_{i+1}}$, $\boldsymbol{w_i}$ is counted toward rework to get the total mean execution time overhead of

$$O_2 = \left(1 - e^{-\frac{\boldsymbol{w}_{i+1}}{M}}\right)\cdot(\boldsymbol{w}_i + \boldsymbol{w}_{i+1}/2).$$

Including a checkpoint will lower the overall cost, if and only if $O_1 \leq O_2$, namely,

$$\boldsymbol{c} + \left(1 - e^{-\frac{\boldsymbol{w}_{i+1}}{M}}\right)\cdot \boldsymbol{w}_{i+1}/2 \leq \left(1 - e^{-\frac{\boldsymbol{w}_{i+1}}{M}}\right)\cdot(\boldsymbol{w}_i + \boldsymbol{w}_{i+1}/2).$$

to yield

$$\boldsymbol{c} \leq \left(1 - e^{-\frac{\boldsymbol{w}_{i+1}}{M}}\right)\cdot \boldsymbol{w}_i. \tag{14}$$

The preceding non-linear inequality can be solved using the Newton-Raphson method [27], with an initial solution of $\boldsymbol{w_i} = \frac{1}{2}[\sqrt{\boldsymbol{c^2k^2} + 4\boldsymbol{c}M} - \boldsymbol{c}\,.\boldsymbol{k}]$, which is obtained by approximating $e^{-\boldsymbol{w}_{i+1}/M}$ in Eq. (14) with a first-order polynomial based on $e^{-\boldsymbol{w}_{i+1}/M} = 1 - \boldsymbol{w}_{i+1}/M = 1 - (\boldsymbol{w}_i + c.k)/M$. The first precision digit of $\boldsymbol{w_i}$ can be obtained by an approximate initial solution followed by usually no more than three iterations.

With a predicted MTBF of M, a known checkpointing cost of $\boldsymbol{c}$, and for a given value of $\boldsymbol{k}$, En-CHORE skips all checkpoints up to $\boldsymbol{w_i}$ which satisfies Eq. (14), to reduce execution time overhead. Based on our observed insight which reveals that optimal checkpointing results if the expected checkpointing cost over θ equals the rework amount after failure, we have En-CHORE governed by equalizing the total checkpointing cost and the rework amount after failure in θ, as

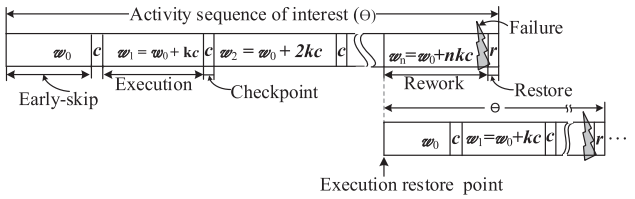


Fig. 7. Activity sequence of interest (⊖) under En-CHORE, in which the inter-checkpoint interval (w) after early checkpoint skipping is adaptively adjusted to equate total checkpointing cost with rework after a failure.

shown in Fig. 7, where $\boldsymbol{w_0}$ denotes the job execution before taking any checkpointing (i.e., the portion of θ to skip checkpointing), and $\boldsymbol{w_i} = \boldsymbol{w_0}+i\cdot\boldsymbol{c}\cdot\boldsymbol{k}$ for $\boldsymbol{i} \geq 0$. The value of $\boldsymbol{w_0}$ is obtained by solving Eq. (14).

4.2 Deriving Inter-Checkpointing Intervals

En-CHORE employs the same observed insight that motives CHORE design for the best checkpointing control (illustrated in Fig. 4), namely, *equating the aggregate checkpointing time within $\boldsymbol{\theta}$ and the rework amount after a failure recovery*. For a given M and a known value of $\boldsymbol{w_0}$, we can accurately estimate the value of $\boldsymbol{k}$, known as the incremental factor which dictates the inter-checkpointing interval, such that the overall checkpointing cost over θ equals the expected rework amount after failure, as discussed next.

Let P_i be the probability of completing the execution of ith interval and its subsequent checkpointing (without a failure) for a total duration of $\boldsymbol{w_i} + \boldsymbol{c}$, where $\boldsymbol{w_i} = \boldsymbol{w_0} + i\cdot\boldsymbol{c}\cdot\boldsymbol{k}$ for $i \geq 0$ as shown in Fig. 7. The probability that failure happens in the ith interval is thus given by $(1 - P_i) = 1 - e^{-(\boldsymbol{w_i}+\boldsymbol{c}+i\cdot\boldsymbol{c}\cdot\boldsymbol{k})/M}$, leading to the mean inter-checkpoint interval count where the failure to occur as

$$\eta = (1 - P_0) + 2 \cdot P_0(1 - P_1) + 3 \cdot P_0P_1(1 - P_2) + \ldots + i \cdot P_0P_1 \ \ldots \ P_{i-2}(1 - P_{i-1}) + \ldots \ to \ \infty \ terms.$$

Substituting the value of P_i for all i's in the above expression followed by some algebraic manipulations, we have

$$\eta = 1 + \sum_{i=1}^{\infty} e^{-i.(\boldsymbol{w_0}+\boldsymbol{c})\ /M} \cdot e^{-i.(i-1)\ \cdot\ \boldsymbol{k}.\boldsymbol{c}/2M}. \tag{15}$$

The expected length of the interval at which the failure to happen equals $(\boldsymbol{w_0} + \boldsymbol{c} + \eta \cdot \boldsymbol{c} \cdot \boldsymbol{k})$, and half of the interval represents the mean rework amount after failure (see Eq. (9)), which is $(\boldsymbol{w_0} + \boldsymbol{c} + \eta \cdot \boldsymbol{c} \cdot \boldsymbol{k})/2$. Since the total number of successfully completed intervals equals $(\eta - 1)$ and each completed interval incurs the checkpointing cost of $\boldsymbol{c}$, we have the checkpointing cost over θ equal to $(\eta - 1) \times \boldsymbol{c}$. Equating the total checkpointing cost and the rework amount over θ gives rise to $(\eta - 1) \times \boldsymbol{c} = (\boldsymbol{w_0} + \boldsymbol{c} + \eta \cdot \boldsymbol{c} \cdot \boldsymbol{k})/2$, simplifying

$$\sum_{i=1}^{\infty} e^{-i\cdot(\boldsymbol{w_0}+\boldsymbol{c})\ /M} \times e^{-i\cdot(i-1)\cdot\boldsymbol{k}\cdot\boldsymbol{c}/2M} = \frac{(\boldsymbol{w_0} + \boldsymbol{c}\cdot(\boldsymbol{k}+1))}{\boldsymbol{c}\cdot(2-\boldsymbol{k})}. \tag{16}$$

For given $\boldsymbol{w_0}$, M, and $\boldsymbol{c}$, the value on the left side of Eq. (16) decreases as $\boldsymbol{k}$ rises, but the value on the right side grows when $\boldsymbol{k}$ increases from 0 to 2. Each value of $\boldsymbol{k}$ which satisfies Eq. (16) for various combinations of M and $\boldsymbol{c}$ is illustrated in Fig. 8, where $\boldsymbol{k}$ value is zero if the value of the left side of Eq. (16) is smaller than that of the right side evaluated at $\boldsymbol{k} = 0$. From the figure, $\boldsymbol{k}$ is seen to increase as M rises, signifying that under a lower failure rate, the inter-checkpointing interval can grow more aggressively. Additionally, the $\boldsymbol{k}$ value under $\boldsymbol{c} = 100$ sec (or $\boldsymbol{c} = 600$ sec) is zero for MTBF $<$ 2000 sec (or 12000 sec), reflecting that if $M/\boldsymbol{c} < 20$, $\boldsymbol{k}$ equals zero. Fig. 9 depicts $\boldsymbol{k}$ versus $M/\boldsymbol{c}$, with $\boldsymbol{k}$ for $M/\boldsymbol{c} < 20$ discarded as it is then equal to zero. From the figure, the computed $\boldsymbol{k}$ values and the values given by the curve of $0.6214 - 2.694 \times e^{(-0.5142\times ln(M/c))}$ are virtually identical. Thus, any $\boldsymbol{k}$ value used for subsequent evaluations is obtained from this fitted curve.

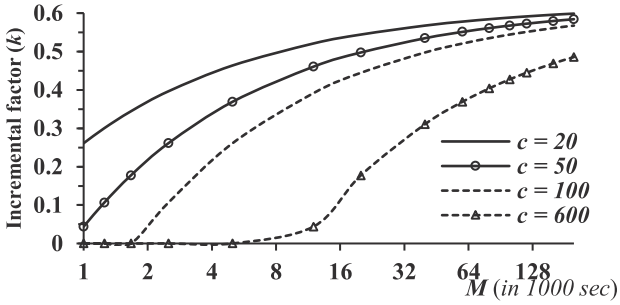


Fig. 8. Incremental factor ($\boldsymbol{k}$) versus M under various costs ($\boldsymbol{c}$).

Observation 4. For an estimated MTBF of M and a known checkpointing cost of $\boldsymbol{c}$, the ith inter-checkpointing interval over θ given by $\boldsymbol{w_i} = \boldsymbol{w_0}+i\cdot\boldsymbol{c}\cdot\boldsymbol{k}$ for $\boldsymbol{i} \geq 0$ under En-CHORE leads to optimality, where $\boldsymbol{k} = 0.6214 - 2.694^{*}exp(-0.5142^{*}ln(M/\boldsymbol{c}))$ and $\boldsymbol{w_0}$ is the solution of equation $\boldsymbol{c} = (1 - e^{-(\boldsymbol{w_0}+\boldsymbol{c}.\boldsymbol{k})/M})\cdot \boldsymbol{w_0}$

5 Performance Evaluation

This section includes comprehensive evaluation under both synthetic failure models and under real-world failure traces. It first presents the failure model and failure trace details assumptions used for extensive evaluation of CHORE and En-CHORE, followed by outlining the simulation setup and assumptions.

5.1 Failure Models and Failure Traces

Failure models govern synthetic failure occurrences, assuming failures to occur randomly, following a Poisson distribution with MTBF of M. Two types of failures exist: transient and permanent failures. The former refers to ones that are not persistent and their involved compute cores may resume execution successfully after restore, whereas the

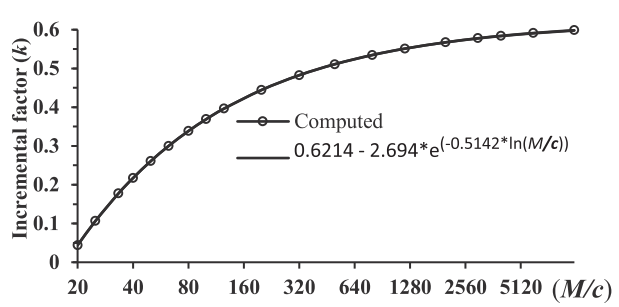


Fig. 9. Incremental factor ($\boldsymbol{k}$) versus the ratio of $M/\boldsymbol{c}$.

TABLE 3
MTBF (in Minutes) Derived From Failure Logs of Real Systems [36], Represented in a Tuple (S#, MTBF)

(S#, MTBF)	(S#, MTBF)	(S#, MTBF)	(S#, MTBF)	(S#, MTBF)	(S#, MTBF)	(S#, MTBF)	(S#, MTBF)	(S#, MTBF)	(S#, MTBF)	(S#, MTBF)
(S2, 880)	(S4, 3404)	(S6, 17773)	(S8, 5120)	(S10, 4543)	(S12, 4024)	(S14, 3791)	(S16, 1310)	(S18, 467)	(S20, 861)	(S23, 8772)
(S3, 3590)	(S5, 3290)	(S7, 18236)	(S9, 3740)	(S11, 3860)	(S13, 5332)	(S15, 7329)	(S17, 13795)	(S19, 483)	(S21, 1438)	(S24, 24124)

latter refers to ones that render involved compute cores wholly unavailable.

In addition, our evaluation also employs real failure logs of 22 different HPC systems at Los Alamos National Lab (LANL) from January 1997 through August 2005, available to the public [36]. Each entry in the failure logs contains detailed information about the node outage that occurred during its operation, such as the time when the failure happened, the time when it was resolved, the system ID and nodes affected, and the root cause of the failure. For our evaluation purpose, a failure that brought down multiple nodes is recorded as one single failure instance.

The MTBFs computed from the failure logs of real systems [36] used by OPT (optimal checkpointing) control are listed in Table 3. In the following sections, the terms of System # and S# are used interchangeably.

5.2 Simulation Assumptions and Operations

With checkpointing, job execution in real-world systems under CHORE and En-CHORE involves a sequence of repeated events, as shown respectively in Figs. 4 and 7. We assume that whenever a failure arises, a monitoring mechanism detects it and immediately triggers the restoration process from the most recent checkpoint file(s). Additionally, the most recent checkpoint file(s) is (are) assumed to be always accessible for restoration purpose.

Checkpointing and restoration costs both depend on the checkpoint data volume, and they are mostly determined by the application memory footprint and hardware resources involved. Since the restoration cost is irrelevant to the checkpointing decision, we set the restoration cost equal to the checkpointing cost (i.e., $\boldsymbol{c} = \boldsymbol{r}$), unless stated otherwise, as in [2], [3], [5], [19], [26]. The simulator takes fixed $\boldsymbol{c}$ and $\boldsymbol{r}$ as parameters, and it is fed with either random errors governed by a Poisson distribution or failure instances registered in real failure traces. Under CHORE, the simulator takes the checkpointing cost ($\boldsymbol{c}$) as the very first inter-checkpoint interval and linearly increases the interval until it encounters a failure and reset the inter-checkpoint interval to $\boldsymbol{c}$, as stated in the control strategy and shown in Fig. 4b. On the other hand, the simulator for OPT in ideal systems uses a fixed inter-checkpoint interval, given by $\boldsymbol{w} = \sqrt{2M\boldsymbol{c}} - \boldsymbol{c}$ (see Eq. (8)), throughout the job execution, as depicted in Fig. 4a. Under En-CHORE, the simulator tracks failure instances during job execution for estimating MTBF to determine the checkpoint skip distance of w_0 at the beginning of the next θ, based on Eq. (15). En-CHORE after skipping w_0 starts to take checkpoints with inter-checkpoint intervals governed by Eq. (16) for best checkpointing control until θ ends (when a failure occurs). Note that En-CHORE assumes an initial MTBF (before failures are tracked for real-time MTBF estimation) to be a typical value (say, 5 years per core) upon a job execution onset, and so does SIMPLE [34], [35].

Both CHORE and En-CHORE are compared with theoretically OPT (with constant, known MTBF *a priori* to follow a fixed inter-checkpoint interval throughout job execution), SIMPLE [34], [35] and SKIP [5], which skips one checkpoint over every θ to reduce the overall checkpointing cost. It should be noted that OPT relies on the *full failure log* of a given system to estimate its MTBF for checkpointing control, with the failure rate assumed to always stay constant (for ideal systems). OPT achieves optimal checkpointing only upon rerunning the job controlled by estimated MTBF. If the MTBF of a system varies during job execution (likely to happen in practice), however, OPT *may not achieve optimal checkpointing any more* since it resorts to one fixed inter-checkpoint interval. On the other hand, both CHORE and En-CHORE have no MTBF knowledge of a system upon the onset of job execution, with En-CHORE tracking failure instances during job execution to achieve the best checkpointing control. Although SIMPLE and SKIP also observe failure occurrences during job execution to estimate subsequent inter-checkpoint intervals, they both are outperformed by En-CHORE soundly, as evidenced by our evaluation results in Section 6.2.

6 Evaluation Results and Discussion

Evaluation results on execution time overhead for CHORE and En-CHORE are presented in sequence, under (1) synthetic failure occurrences governed by given MTBF following a Possion process and (2) real system failure traces.

6.1 CHORE Evaluation

6.1.1 Failures Governed by MTBF

To quantify CHORE performance, we first gather total execution time overhead by simulation under CHORE for the computing system in which failures happen randomly following the Poisson distribution with various MTBF values, under $\boldsymbol{c} = \boldsymbol{r} = 20$ sec. The gathered time overhead result for each MTBF is then normalized against the lowest total execution time overhead incurred under OPT determined by the MTBF for an ideal system counterpart, to get the normalized execution time overhead ratio ($\boldsymbol{\mu}$). System failure occurrences governed by MTBF are assumed to exhibit no fluctuation throughout job execution in our analyses on the time overhead bound. In practice, system failures tend to occur with various degrees of MTBF fluctuation (dictated by factors like the system load, ambient temperature, component variations, etc.). In fact, a heavier system load and a higher temperature tend to yield more failure rates. Besides, failure rates of the same type of system may fluctuate widely, as revealed earlier [14], [16], [30], where constituent nodes of different systems are found to have their MTBF values vary by up to 50×. For a given MTBF, failure rate fluctuation may lower $\boldsymbol{\mu}$ of CHORE considerably, with more fluctuation to result in smaller $\boldsymbol{\mu}$.

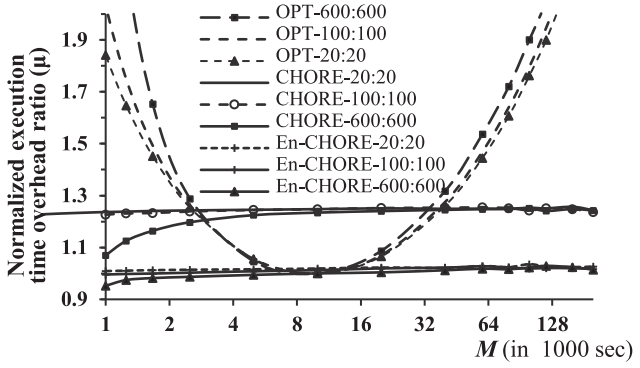


Fig. 10. Normalized execution time overhead ratio (μ) versus M under CHORE, En-CHORE, and OPT-$MTBF_{ideal} = 10000$ for various checkpointing costs (c) and restoration costs (r), designated respectively by CHORE-c:r, En-CHORE-c:r, and OPT-c:r, where failures happen randomly following the Poisson distribution without MTBF fluctuation.

- *Synthetic failure occurrences without MTBF fluctuation*

Results of μ versus M for CHORE when failure occurrences are synthesized without MTBF fluctuation is shown in Fig. 10. Every μ value in the figure is averaged over 1,000 event-driven simulation runs of a long-running job (say, 1000 hours). Experimental results confirm that CHORE upper bounds extra execution time overhead beyond what is the smallest achievable time overhead under OPT. Oblivious to MTBF, CHORE takes checkpoints over θ during execution with their inter-checkpoint intervals following the sequence of c, c, $3c$, $5c$, $7c$, etc. so as to equate the aggregate checkpointing cost with the rework after failure, yielding a near-constant μ (of 1.26) over the wide MTBF range examined.

To demonstrate the effectiveness of CHORE, we have measured the μ values of CHORE under various checkpointing costs (c) and restore costs (r) for M ranging from 1000 sec to 200000 sec, denoted by the curves of CHORE-c:r in Fig. 10. It is evidenced from the figure that each curve moves up gradually when M increases before leveling off at a value < 1.26. For example, μ under CHORE-600:600 is 1.07 at $M = 1000$ sec, and it plateaus at 1.24 for $M \geq 20000$ sec. Similarly, μ at $M = 1000$ sec equals 1.23 under CHORE-100:100, and it levels off at 1.25 for $M \geq 10000$ sec. CHORE is confirmed to upper bound its μ by 1.26 for a wide range of c and r values examined.

The effects of checkpointing and restoration costs on μ under OPT optimized for $MTBF_{ideal} = 10000$ are also included in Fig. 10, with OPT-c:r denoting the outcomes of OPT for $MTBF_{ideal} = 10000$ under given c and r. As can be found from the curves, OPT-c:r rises monotonically without upper bounds if M deviates more from $MTBF_{ideal} = 10000$ at faster paces for larger c and r, in sharp contrast to CHORE-c:r whose μ is upper bounded always by 1.26. At $M = 1000$ sec, for example, OPT-20:20 is 1.84, as opposed to 2.92 for OPT-600:600. This results mainly from bigger aggregate restore and rework during job execution caused by larger r and an inadequate number of checkpoints taken at the constant interval optimized for $MTBF_{ideal} = 10000$ (rather than for actual MTBF of 1000 sec).

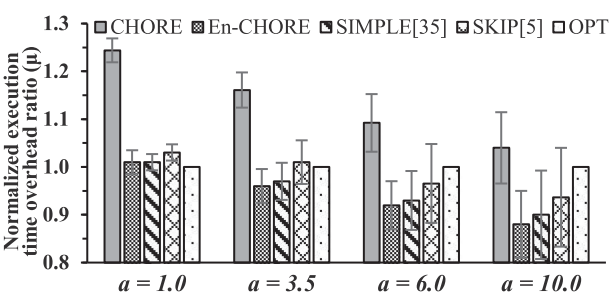


Fig. 11. Normalized execution time overhead ratio (μ) under synthetic failures for $c = r = 20$ sec, with varying MTBF fluctuation degrees in that local M values range randomly from $a \times MTBF_{ideal}$ to $1/a \times MTBF_{ideal}$, for $MTBF_{ideal} = 10000$ sec.

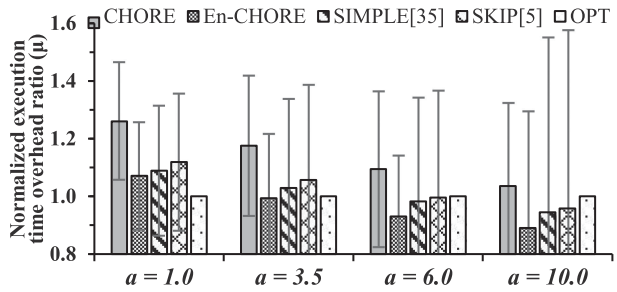


Fig. 12. Normalized execution time overhead ratio (μ) under synthetic failures for $c = r = 10$ minutes, with varying MTBF fluctuation degrees in that local M values range randomly from $a \times MTBF_{ideal}$ to $1/a \times MTBF_{ideal}$, for $MTBF_{ideal} = 6700$ minutes (averaged over the MTBFs of 22 real systems listed in Table 3).

- Synthetic failure occurrences with MTBF fluctuation

CHORE performance under synthetic failure occurrences with various MTBF fluctuation degrees is depicted in Figs. 11 and 12, where MTBF fluctuates during job execution such that a randomly chosen number of failures between 1 and 100 are to occur locally with their MTBF equal to an arbitrary value within ($a \times MTBF_{ideal}$ to $1/a \times MTBF_{ideal}$), respectively for $MTBF_{ideal} = 10000$ sec and $MTBF_{ideal} = 6700$ minutes locally. Such MTBF fluctuation characterization can also be found in earlier work [14], [16], [30], [35], [37]. Here, failure occurrences are synthesized for $a = 3.5$, 6.0, and 10.0, to reflect three different MTBF fluctuation degrees, with a larger a signifying larger fluctuation (and $a = 1.0$ for no fluctuation). Each normalized mean execution time overhead ratio (μ) with $c = r = 20$ sec (or $c = r = 10$ minutes) in Fig. 11 (or Fig. 12) is averaged over 1000 evaluation runs of a long-running job (for 1000 hours). From Fig. 11, CHORE sees its μ to drop down to 1.04 with $a = 10$ from 1.16 with $a = 3.5$, while CHORE has a higher μ value of 1.26 in the absence of MTBF fluctuation. Similarly, with $MTBF_{ideal}$ equal to the average MTBFs of 22 real systems and $c = r = 10$ minutes illustrated in Fig. 12, μ under CHORE reduces down to 1.04 with heavy fluctuation (i.e., $a = 10$) from 1.26 with no fluctuation (i.e., $a = 1.0$). In general, the execution time of CHORE comes closer to that of OPT if MTBF fluctuates more, and it in fact involves considerably lower than 26 percent additional execution time overhead than that under OPT in a real system, as demonstrated next under failure traces.

6.1.2 Under Real System Failure Traces

We have evaluated CHORE under real failure logs of 22 different HPC systems at Los Alamos National Lab (LANL), with MTBF (in minutes) of each system obtained from its failure log listed in Table 3 for use by theoretically optimal checkpointing, OPT. Execution time overhead of CHORE

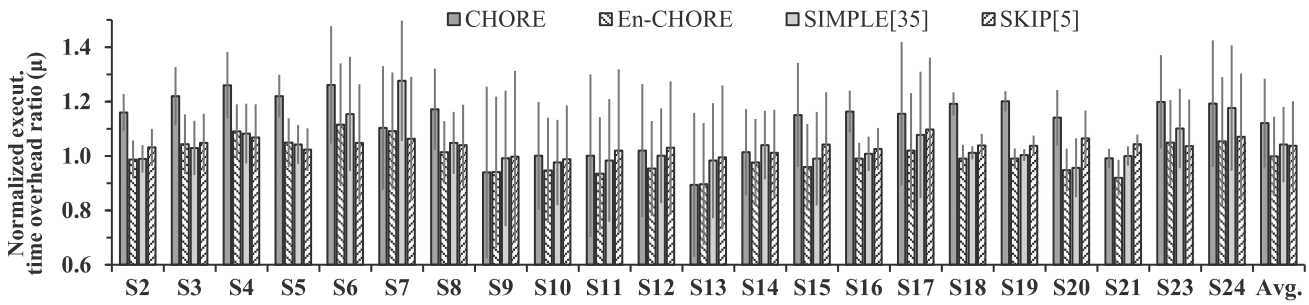


Fig. 13. Execution time overhead of CHORE, EN-CHORE, SIMPLE, and SKIP normalized w.r.t. that of OPT (μ) under the real failure traces of 22 systems with $c = r = 10$ minutes, where the standard deviation of gathered μ of each system over 1000 runs is marked at its corresponding bar.

normalized with respect to that of OPT for $c = r = 10$ minutes (μ) is shown in Fig. 13. Every μ value in the figure is averaged over 1,000 evaluation runs of a long-running job, with failure occurrences following real system traces. Each evaluation run starts from a randomly chosen time point in the real traces and computes the overall execution time overhead incurred until completing 1000-hour effective work per node. From the figure, μ under CHORE ranges from 0.89 (for System 13) to 1.26 (for System 4), with the mean μ value of all 22 systems equal to 1.13, markedly less than that of the analytical upper bound (of 1.26). The standard deviation of the gathered μ range of each system over 1000 evaluation runs for CHORE is marked at its corresponding bar. It is found that the μ values under CHORE for five systems (i.e., S9−S11, S13, and S21) are smaller than those under OPT. In addition, fourteen systems (all but S2−S5, S16, S18−S20) see their μ ranges to stretch below 1.0, indicating that their actual MTBFs fluctuate widely and deviate greatly from those derived globally from corresponding whole real traces (i.e., $\text{MTBF}_{\text{ideal}}$'s).

6.2 En-CHORE Evaluation

6.2.1 Failures Governed by MTBF

Execution time performance under synthetic failure occurrences without MTBF fluctuation and with fluctuation is presented below in sequence.

- *Synthetic failure occurrences without MTBF fluctuation*

The result of μ versus examined M values of En-CHORE under various checkpointing costs (c) and restore costs (r) for M ranging from 1000 sec to 200000 sec, denoted by the curves of En-CHORE-c:r is shown in Fig. 10. As evidenced by the curves, μ of En-CHORE-20:20 stays at a constant level of 1.02, but En-CHORE-100:100 and En-CHORE-600:600 see their μ values to go below 1.0 under a small M. For example, μ of En-CHORE-600:600 is 0.95 at $M = 1000$ sec, and it creeps up as M rises to reach 1.0 at $M = 10000$ sec; it plateaus at 1.02 for $M = 100000$ sec.

- *Synthetic failure occurrences with MTBF fluctuation*

Figs. 11 and 12 depict comparative μ results under En-CHORE, SIMPLE, and SKIP for three MTBF fluctuation degrees, with two extreme c ($= r$) values examined. They reveal that En-CHORE consistently outperforms SIMPLE, with its μ equal to 1.01 (or 1.07), 0.96 (or 0.99), 0.92), and 0.88 (or 0.89), respectively for $a = 1.0$, 3.5, 6.0, and 10.0 under $c = r = 20$ seconds (or $= 10$ minutes), as opposed to SIMPLE's μ of 1.01 (or 01.09), 0.97 (or 1.03), 0.93 (or 0.98), and 0.90 (or 0.94), respectively. Similarly, it is observed in Figs. 11 and 12 that En-CHORE consistently outperforms SKIP. Clearly, the gaps of μ values under En-CHORE and under SIMPLE and SKIP become noticeable for large c ($= r$), in the presence of MTBF fluctuation. In addition, En-CHORE shortens its execution time more if MTBF has a wider fluctuation degree locally. En-CHORE is faster than OPT under moderate to high fluctuation (with $a \geq 3.5$), as the result of its adaptive checkpointing control enabled by tracking failure occurrences on the fly. Likewise, SIMPLE and SKIP may outperform OPT, albeit under larger fluctuation degrees.

6.2.2 Under Real System Failure Traces

The μ results, averaged over 1000 runs, under En-CHORE, SIMPLE, and SKIP for various systems with $c = r = 10$ minutes are presented in Fig. 13, where the cost of 10 minutes for c and r was chosen earlier in the SIMPLE study [34], [35] as well. For each evaluation run, En-CHORE, SIMPLE, and SKIP assume an initial MTBF of 5 years per core for determining when to take the very first checkpoint after execution onset. They update their MTBFs upon encountering each failure accordingly for deciding the best checkpoint starting point in the next θ. Additionally, En-CHORE also estimates the best inter-checkpoint intervals based on Eq. (16). In this evaluation, each control scheme incorporates all failures encountered over the period of job execution for updating its MTBF.

According to Fig. 13, En-CHORE exhibits marked reduction in execution time overhead, bringing μ down below 1.0 for 13 systems out of 22 systems. Given that En-CHORE outperforms optimal checkpointing if system's MTBF fluctuates by a factor of 1.5 or more (as observed in Fig. 10), the results of Fig. 13 signify a majority of real systems possess highly fluctuating MTBF's, as also pointed out in previous articles [14], [16], [28], [30], [31]. The mean μ value under En-CHORE ranges from 1.11 (for S6) to 0.89 (for S13), with an average of 1.00 over all 22 systems. In contrast, SIMPLE (or SKIP) yields the mean μ values of all systems to range from 1.28 for S7 (or 1.1 for S17) to 0.96 for S20 (or 0.99 for S10), with 8 (or 3) systems to have their μ values less than 1.0. Here, SIMPLE and SKIP assume an initial MTBF of 5-year per core as well, when job execution starts. SIMPLE and SKIP, both with the mean μ of 1.04 over all systems, are outperformed soundly by En-CHORE, which achieves the best checkpointing control realizable in real computing systems by tracking failure instances during job execution for adaptively skipping some early checkpoints in θ before taking checkpoints with inter-checkpointing intervals appropriately determined.

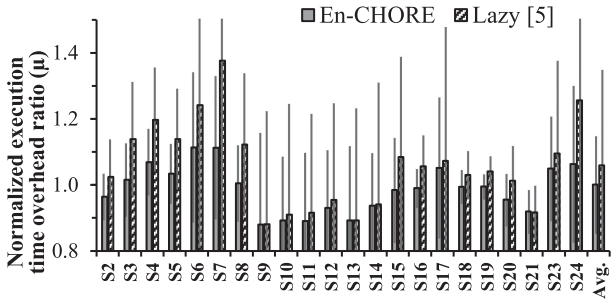


Fig. 14. Execution times overhead of EN-CHORE and Lazy Checkpointing (with the shape parameter of 0.7) normalized w.r.t. that of OPT (μ) under real failure traces of 22 systems with $r = c = 10$ minutes, where the standard deviation of gathered μ of each system over 1000 runs is marked at its corresponding bar.

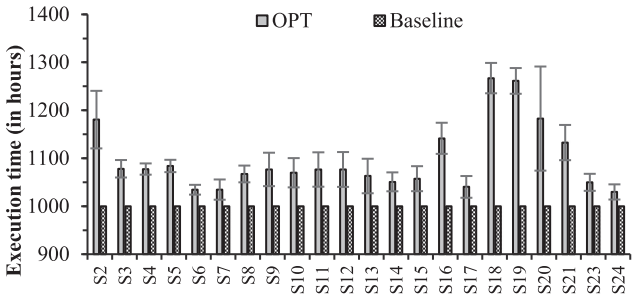


Fig. 15. Execution times taken to complete 1000-hour productive work per node under OPT and Baseline for the real failure traces of 22 systems with $c = r = 10$ minutes, where the standard deviation of gathered execution times (in hours) of each system over 1000 runs is marked at its corresponding bar.

6.2.3 Comparison With Lazy Checkpointing

Realizing temporal locality in failures, authors in [5] proposed a simplified version of Weibull distribution-based checkpointing, called Lazy checkpointing, to capture the temporal locality of failures. In addition to MTBF, Lazy checkpointing also takes into consideration, the Weibull shape parameter of the system's failure distribution. The shape parameter value below 1.0 indicates that the failure rate decreases over time, signifying a gradual increase in the inter-checkpointing interval to be favorable. However, the shape parameter is not known ahead of time in practice, and hence, we tuned Lazy checkpointing with a typical shape parameter (say, 0.7) for fair comparison against En-CHORE, as suggested in [5], and estimated the MTBF of a system in each run on-the-fly.

The normalized execution time overhead ratios (μ) of En-CHORE and Lazy checkpointing for $c = r = 10$ minutes to *complete 1000-hour productive work (without checkpointing overhead under failure-freeness) per node* under various systems are shown in Fig. 14. Every μ value in the figure is averaged over 1,000 evaluation runs of a long-execution job, with failure occurrences following real system traces. From the figure, En-CHORE is seen to consistently outperform Lazy checkpointing under all systems except System 21 (denoted by S21). Lazy checkpointing is outperformed soundly by En-CHORE for all 22 systems with $c = r = 10$ minutes and their mean μ values equal to 1.06.

6.2.4 OPT Results Under Real System Failure Traces

This subsection presents the total execution times under OPT and under the Baseline (i.e., without checkpointing under failure-freeness) for real failure logs of 22 different HPC systems [36]. Each entry in failure logs contains various information, but this evaluation uses only the system-outage timestamp, which registers the time of a system failure event. Specifically, all system-outage timestamps of a given system are used for deriving a constant global MTBF of the system according to its failure trace, as listed in Table 3. Additionally, our evaluation assumes that (1) every system executes a long-running application, (2) all system nodes are involved in the execution, and (3) adequate backup nodes are available for substituting permanent failure nodes upon execution recovery from the very last checkpoint. The inter-checkpointing interval (i.e., w in Fig. 2a) of a system under OPT is obtained according to the previously computed constant global MTBF of the system (see Table 3) and its checkpointing cost (c), which depends on the memory footprint of the application of interest only.

Fig. 15 depicts the results of total execution times taken by OPT to complete the work amount of the Baseline in 1000 hours for various systems with $c = r = 10$ minutes, where each result is the average over 1000 runs. Every evaluation run starts from a randomly chosen time point in real traces and ends upon completing 1000-hour productive work per node. It is found from the figure that S24 incurs the shortest mean execution time (of 1030 hours) to complete the 1000-hour productive work per node due to its largest MTBF (see Table 3), while S18 takes the longest time (of 1267 hours) to complete because of its smallest MTBF. As expected, Fig. 15 reveals that a system completes the execution of a long-running job quicker under a larger MTBF in general. For example, S2 with MTBF of 880 minutes takes 1180 hours to complete the job, whereas S7 with MTBF of 18236 minutes requires 1034 hours for job completion under OPT. Note that Fig. 15 also includes the base execution time results of 1000 hours (denoted by Baseline) for reference.

7 Conclusion

This article deals with efficient checkpointing control for real-world computing systems, whose MTBFs are unknown *a priori* or fluctuate widely during job execution. The proposed control strategy aims to realize the best checkpointing with the lowest execution time overhead possible in real-world computing systems. It results from analyzing optimal checkpointing over an activity sequence of interest (θ) to discover that the aggregate checkpointing cost and expected rework after a failure recovery over θ are equated for performance optimality, dubbed "CHORE" (checkpointing overhead and rework equated). Execution checkpoints under CHORE are taken with their inter-checkpoint intervals following a specified sequence oblivious to system's MTBF. CHORE exhibits an upper bound on the normalized execution time overhead ratio (μ, which is with respect to the time overhead of optimal checkpointing, OPT). If failure instances are tracked during job execution for estimating the MTBF of a system in real time, CHORE can be enhanced to lower execution time overhead by appropriate early checkpoint skipping over θ before taking checkpoints with suitable inter-checkpointing intervals, yielding En-CHORE.

TABLE 4
Mean μ Values of Different Checkpointing Methods Under $M_1 = 10000$ Seconds and $M_2 = 6700$ Minutes

Schemes	Synthetic random failures with						Real system failure traces
	no fluctuation		low fluctuation		high fluctuation		
	M_1	M_2	M_1	M_2	M_1	M_2	
CHORE	1.26	1.26	1.16	1.18	1.04	1.04	1.13
En-CHORE	1.01	1.07	0.96	0.99	0.88	0.89	1.00
SIMPLE[35]	1.01	1.09	0.97	1.03	0.90	0.94	1.04
SKIP[5]	1.03	1.12	1.01	1.06	0.94	0.96	1.04

Like CHORE, En-CHORE attains best checkpointing, with its control parameters derived by equating the checkpointing cost with the rework amount after failure.

Extensive simulation evaluation has been conducted to assess CHORE and En-CHORE, under both synthetic failure occurrences and real system failure traces. Evaluated results of the average μ over 1000 independent runs are summarized in Table 4. Specifically, CHORE is confirmed to have its μ stay below 1.26 (or at 1.04) for various checkpoint and restore costs, when failures happen randomly following the Poisson distribution without (or with) MTBF fluctuation. In addition, CHORE and En-CHORE see their μ values to drop markedly for synthetic failure occurrences with MTBF fluctuation. When evaluated under the real failure traces of 22 different HPC systems at Los Alamos National Lab (LANL), our En-CHORE exhibits the average μ value of 1.00, with respect to its OPT counterpart. Note that En-CHORE may see its μ to drop below 1.0 for certain systems, since OPT is then no longer "optimal" for every system given that the MTBF of a system is derived globally from its whole real trace, whose local failure rates may, in fact, fluctuate widely over the logged time span. En-CHORE, SIMPLE [34], [35], and SKIP [5] track failures during job execution for runtime MTBF estimation to enable different checkpointing control schemes, with En-CHORE seen to clearly outperform its two counterparts, whose mean μ values across all 22 systems both equal 1.04. Similarly, En-CHORE consistently outperforms Lazy checkpointing [5], whose checkpointing control follows a simplified Weibull distribution. Note that if the knowledge of one constant global MTBF of a real system is known *a priori*, OPT does remarkably well for that system.

En-CHORE is readily applicable to real-world system checkpointing in support of long-running job execution, when system failures are tracked for best checkpointing control. In contrast, manually adjusting checkpointing intervals in the course of job execution (according to failure instances encountered) is infeasible practically because such job execution always undertakes automatic recovery from any encountered failure using the recent checkpointing file (s) without halting job execution to make manual checkpointing interval adjustment.

Acknowledgments

This work was supported in part by National Science Foundation under Award Numbers: CNS-1527051 and III-1652107, and in part by the Louisiana Board of Regents under Contract Number: LEQSF(2018-21)-RD-A-24.

References

[1] P. Sigdel and N.-F. Tzeng, "Coalescing and deduplicating incremental checkpoint files for restore-express multi-level checkpointing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 12, pp. 2713–2727, Dec. 2018.

[2] A. Benoit, A. Cavelan, V. L. Fèvre, Y. Robert, and H. Sun, "Towards optimal multi-level checkpointing," *IEEE Trans. Comput.*, vol. 66, no. 7, pp. 1212–1226, Jul. 2017.

[3] S. Di, Y. Robert, F. Vivien, and F. Cappello, "Toward an optimal online checkpoint solution under a two-level HPC checkpoint model," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 244–259, Jan. 2017.

[4] T. Knauth and C. Fetzer, "Vecycle: Recycling VM checkpoints for faster migrations," in *Proc. 16th Annu. Middleware Conf.*, 2015, pp. 210–221.

[5] D. Tiwari, S. Gupta, and S. S. Vazhkudai, "Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems," in *Proc. 44th IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, 2014, pp. 25–36.

[6] I. Jangjaimon and N.-F. Tzeng, "Adaptive incremental checkpointing via delta compression for networked multicore systems," in *Proc. 27th IEEE Int. Parallel Distrib. Process. Symp.*, 2013, pp. 7–18.

[7] K. Sato *et al.*, "Design and modeling of a non-blocking checkpointing system," in *Proc. IEEE/ACM Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2012, pp. 1–10.

[8] T. Z. Islam, K. Mohror, A. Moody, and R. Eigenmann, "mcrEngine: A scalable checkpointing system using data-aware aggregation and compression," in *Proc. Int. Conf. High-Perform. Comput. Netw. Storage Anal.*, 2012, pp. 1–11.

[9] N. Bessho and T. Dohi, "Comparing checkpoint and rollback recovery schemes in a cluster system," in *Proc. 12th Int. Conf. Algorithms Architectures Parallel Process.*, 2012, pp. 531–545.

[10] G. Zheng, X. Ni, and L. V. Kalé, "A scalable double in-memory checkpoint and restart scheme towards exascale," in *Proc. 2nd Workshop Fault-Tolerant HPC Extreme Scale*, 2012, pp. 1–6.

[11] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello, "Modeling and tolerating heterogeneous failures in large parallel systems," in *Proc. Int. Conf. High Perf. Comput. Netw. Storage Anal.*, 2011, pp. 1–11.

[12] K. B. Ferreira1, R. Riesen, R. Brighwell, P. Bridges, and D. Arnold, "libhashckpt: Hash-based incremental checkpointing using GPU's," in *Proc. 18th Eur. MPI Users' Group Conf. Recent Adv. Message Passing Interface*, 2011, pp. 272–281.

[13] A. Moody, G. Bronevetsky, K. Mohror, and B.R. de Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *Proc. IEEE/ACM Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2010, pp. 1–11.

[14] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Depend. Secure Comput.*, vol. 7, no. 4, pp. 337–350, Oct. 2010.

[15] K. S. Trivedi, *Probability & Statistics With Reliability, Queuing and Computer Science Applications*. Hoboken, NJ, USA: Wiley, 2008.

[16] B. Schroeder and G. Gibson, "Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you?," *ACM Trans. Storage (TOS)*, vol. 3, no. 3, Oct. 2007, Art. no. 8.

[17] A. J. Oliner, L. Rudolph, and R. K. Sahoo, "Cooperative checkpointing: A robust approach to large-scale systems reliability," in *Proc. 20th Annu. Int. Conf. Supercomputing*, 2006, pp. 14–23.

[18] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Gener. Comput. Syst.*, vol. 22, pp. 303–312, Jun. 2006.

[19] S. Yi, J. Heo, Y. Cho, and J. Hong, "Adaptive page-level incremental checkpointing based on expected recovery time," in *Proc. ACM Symp. Appl. Comput.*, 2006, pp. 1472–1476.

[20] P. H. Hargrove and J. C. Duell, "Berkeley lab checkpoint/restart (BLCR) for linux clusters," *J. Phys.: Conf. Ser.*, vol. 46, pp. 494–499, Sep. 2006.

[21] R. K. Sahoo, M. S. Squillante, A. Sivasubramaniam, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," in *Proc. Int. Conf. Depend. Syst. Netw.*, 2004, pp. 772–781.

[22] S. Agarwal, R. Garg, M. Gupta, and J. Moreira, "Adaptive incremental checkpointing for massively parallel systems," in *Proc. 18th Annu. Int. Conf. Supercomputing*, 2004, pp. 277–286.

[23] Y. Zhang and K. Chakrabarty, "Energy-aware adaptive checkpointing in embedded real-time systems," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2003, pp. 918–923.

[24] E. Elnozahy, L. Alvisi, Y. Wang, and D. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Comput. Surv.*, vol. 34, no. 3, pp. 375–408, Sep. 2002.
[25] J. S. Plank, K. Li, and M. A. Puening, "Diskless checkpointing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 10, pp. 972–986, Oct. 1998.
[26] N. H. Vaidya, "A case for two-level recovery schemes," *IEEE Trans. Comput.*, vol. 47, no. 6, pp. 656–666, Jun. 1998.
[27] T. J. Ypma, "Historical development of the newton–Raphson method," *SIAM Rev.*, vol. 37, no. 4, pp. 531–551, Dec. 1995.
[28] R. K. Iyer, S. E. Butner, and E. J. McCluskey, "A statistical failure/load relationship: Results of a multicomputer study," *IEEE Trans. Comput.*, vol. 31, no. 7, pp. 697–706, Jul. 1982.
[29] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Commun. ACM*, vol. 17, pp. 530–531, Sep. 1974.
[30] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "Failures in large scale systems: Long-term measurement, analysis, and implications," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2017, pp. 1–12.
[31] C. Zimmer, D. Maxwell, S. McNally, S. Atchley, and S. S. Vazhkudai, "GPU age-aware scheduling to improve the reliability of leadership jobs on Titan," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2018, pp. 83–93.
[32] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehabo, M. Paun, and S. L. Scott, "An optimal checkpoint/restart model for a large scale high performance computing system," in *Proc. Int. Symp. Parallel Distrib. Process.*, 2008, pp. 1–9.
[33] D. Tiwari, S. Gupta, G. Gallarno, J. Rogers, D. Maxwell, "Reliability lessons learned from GPU experience with the titan supercomputer at oak ridge leadership computing facility," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2015, pp. 1–12.
[34] N. El-Sayed and B. Schroeder, "Checkpoint/restart in practice: When 'simple is better'," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2014, pp. 84–92.
[35] N. El-Sayed and B. Schroeder, "Understanding practical tradeoffs in HPC checkpoint-scheduling policies," *IEEE Trans. Depend. Secure Comput.*, vol. 15, no. 2, pp. 336–350, Mar. 2018.
[36] D. John, "EESEUR-code-data," 2016. Accessed: Jul. 2020. [Online]. Available: https://github.com/Derek-Jones/ESEUR-code-data/tree/master/reliability/LA-UR-05-7318-failure-data-1996-2005.csv.xz.
[37] E. Wu, B. Li, and J. Stathis, "Modeling of time-dependent non-uniform dielectric breakdown using a clustering statistical approach," *Appl. Phys. Lett.*, vol. 103, Oct. 2013, Art. no. 152907.
[38] B. Nie, J. Xue, S. Gupta, C. Engelmann, E. Smirni, and D. Tiwari, "Characterizing temperature, power, and soft-error behaviors in data center systems: insights, challenges, and opportunities," in *Proc. Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2017, pp. 22–31.
[39] B. Nie, D. Tiwari, S. Gupta, E. Smirni, and J.H. Rogers, "A large-scale study of soft-errors on GPUs in the field," in *Proc. Int. Symp. High Perform. Comput. Architecture*, 2016, pp. 519–530.
[40] S. Gupta, D. Tiwari, C. Jantzi, J. Rogers, and D. Maxwell, "Understanding and exploiting spatial properties of system failures on extreme-scale HPC systems," in *Proc. Int. Conf. Depend. Syst. Netw.*, 2015, pp. 37–44.
[41] W. Weibull, "A statistical distribution function of wide applicability," *J. Appl. Mechanics*, vol. 18, pp. 293–297, Sep. 1951.

Purushottam Sigdel (Member, IEEE) received the BE degree in electronics and communication engineering, in 2000, the MS degree in information and communication, in 2006 both from Trubhuvan University, Kathmandu, Nepal, and the MS degree in computer science from the University of Louisiana at Lafayette, in 2014. Currently, he is working toward the PhD degree with the Center for Advanced Computer Studies, the University of Louisiana at Lafayette. From 2000 to 2012, he worked as an instructor with the Department of Electronics and Computer Engineering in Pulchowk, Nepal. His research interests include high-performance computer systems, distributed computing data compression, fault tolerance, and computer system architecture.

Xu Yuan (Member, IEEE) received the BS degree from Nankai University, Tianjin, China, in 2009, and the PhD degree from Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, in 2016. From 2016 to 2017, he was a postdoctoral fellow of electrical and computer engineering at the University of Toronto, Toronto, ON, Canada. He is currently an assistant professor with the School of Computing and Informatics, the University of Louisiana at Lafayette, Lafayette, Louisiana, USA. His research interest focuses on algorithm design and optimization for spectrum sharing, coexistence, cognitive radio networks, and network security.

Nian-Feng Tzeng (Fellow, IEEE) has been with Center for Advanced Computer Studies, School of Computing and Informatics, the University of Louisiana at Lafayette, since 1987. His current research interest include the areas of high-performance computer systems, and parallel and distributed processing. He was on the editorial board of the *IEEE Transactions on Parallel and Distributed Systems*, 1998 – 2001, and on the editorial board of the *IEEE Transactions on Computers*, 1994 – 1998. He was the chair of Technical Committee on *Distributed Processing*, the IEEE Computer Society, from 1999 till 2002. He is the recipient of the Outstanding Paper Award of the 10th IEEE International Conference on Distributed Computing Systems, May 1990, and received the University Foundation Distinguished Professor Award, in 1997.