To Checkpoint or Not to Checkpoint: Understanding Energy-Performance-I/O Tradeoffs in HPC Checkpointing

Nosayba El-Sayed Bianca Schroeder
Department of Computer Science, University of Toronto
{nosayba, bianca}@cs.toronto.edu

***Abstract*—As the scale of high-performance computing (HPC) clusters continues to grow, their increasing failure rates and energy consumption levels are emerging as two serious design concerns that are expected to become more challenging in future Exascale systems. Therefore, efficiently running systems at such large scales requires an in-depth understanding of the performance and energy costs associated with different fault tolerance techniques. The most commonly used fault tolerance method is checkpoint/restart. Over the years, checkpoint scheduling policies have been traditionally optimized and analysed from a performance perspective. Understanding the energy profile of these policies or how to optimize them for energy savings (rather than performance), remain not very well understood.**

In this paper, we provide an extensive analysis of the energy/performance tradeoffs associated with an array of checkpoint scheduling policies, including policies that we propose, as well as few existing ones in the literature. We estimate the energy overhead for a given checkpointing policy, and provide simple formulas to optimize checkpoint scheduling for energy savings, with or without a bound on runtime. We then evaluate and compare the runtime-optimized and energy-optimized versions of the different methods using trace driven simulations based on failure logs from 10 production HPC clusters. Our results show ample room for achieving high energy savings with a low runtime overhead when using non-constant (adaptive) checkpointing methods that exploit characteristics of HPC failures. We also analyze the impact of energy-optimized checkpointing on the storage subsystem, identify policies that are more optimal for I/O savings, and study how to optimize for energy with a bound on I/O time.

Keywords*—*High-performance computing; Fault tolerance; Checkpoint/Restart; Energy-efficiency; Performance.

I. Introduction

The efficient design and operation of large-scale, high-performance computing (HPC) clusters requires a good understanding of how to balance different design tradeoffs between performance, reliability, and energy-efficiency. As HPC installations continue to grow in scale and complexity, capturing the interplay between these key design factors in production environments becomes increasingly difficult, and is expected to be more challenging in future Exascale platforms. Among the most serious issues facing HPC systems are their increasing failure rates and energy consumption levels. This means that the design and optimization of fault tolerance methods requires an understanding of their performance impact, as well as their *energy cost*, and all possible tradeoffs that result when these methods are used in practice.

The most commonly used method for fault tolerance in tightly-coupled HPC applications is coordinated checkpointing, where a parallel application periodically stops execution to checkpoint its current state. In the case of a failure, the application recovers by restarting from the most recent checkpoint. Traditionally, work on checkpoint policies has been focused on optimizing the application runtime by minimizing the associated overheads, namely the time that is spent writing periodic checkpoints and the time that is spent to recover the lost work in case of a failure.

However, checkpointing policies optimized for application runtime are not necessarily optimal from an energy perspective. For example, while time spent writing a checkpoint and time spent redoing work lost after a failure count equally towards the completion time of the application, they do not affect the power budget in the same way, since writing a checkpoint consumes less energy than doing computation [1]–[4]. Understanding the energy overheads associated with different methods for *scheduling checkpoints*, as well as how they can be optimized for energy consumption, in combination with their effects on application runtime, are critical, yet not very well understood questions in the HPC community.

While there has been work on reducing the power consumed during an individual checkpoint, e.g. by using DVFS (Dynamic Voltage and Frequency Scaling) during a checkpoint [5], possibly combined with the use of energy-efficient NAND flash memory [6], less attention has been paid to the question of how to *schedule checkpoints* for energy-efficiency (see Section V for more details).

This paper has multiple contributions. First, we provide a thorough analysis of a wide array of policies for scheduling coordinated checkpoints, using trace-driven simulations based on real world HPC failure logs, to provide a better understanding of their respective energy, performance and I/O tradeoffs. We consider static checkpointing policies, which use a fixed checkpoint interval, as well as a number of more advanced, non-constant checkpointing techniques. We show how each policy can be optimized for energy-efficiency, with or without a bound on application runtime, and explore practical considerations for deployment. Finally, we provide an evaluation of the implications energy optimizations have on the I/O subsystem, and how the impact on the I/O subsystem can be limited.

978-1-4799-5548-0/14/$31.00 ©2014 IEEE

II. Energy awareness in static checkpointing

A. Optimizing static checkpoints for energy

Traditionally, the goal behind work on checkpoint scheduling has been to minimize an application's runtime or completion time. One of the oldest and simplest results is Young's formula [7], which determines the checkpoint interval based on two input parameters, the Mean Time to Fail (MTTF) in a system, and the checkpoint cost, C (i.e. the time it takes to write a checkpoint), as follows:

$$\Delta_{Runtime} = \sqrt{2 \cdot C \cdot MTTF} \tag{1}$$

We show in another paper [8] that Young's formula provides near-optimal results in terms of completion time, which are on-par with more recent, significantly more complex solutions, despite the fact that its derivation relies on a number of simplifying assumptions known to be unrealistic in practice. Young's formula works by trying to exactly balance the two types of *wasted work* associated with checkpointing and failure recovery that will delay an application's completion: the amount of time the application spends writing checkpoints ($T_{checkpt}$), and the amount of time that is lost after a failure and needs to be recomputed (T_{recomp}), i.e. the work that was done since the most recent checkpoint.

The motivation behind this paper is the observation that minimizing wasted time (and hence completion time) does not necessarily minimize the consumed energy. To see why, note that the power $P_{checkpt}$ consumed during checkpointing is lower than the power P_{comp} consumed during computation, since the storage system of an HPC installation consumes significantly less power than computation [9].

More precisely, measurements in modern HPC installations suggest that the power consumption during a checkpoint operation compared to idle power (P_{idle}) is typically in the $1.05 - 1.15 \times P_{idle}$ range [1]–[3]. The power consumption during HPC computation, on the other hand, was found to be in the $2 - 4 \times P_{idle}$ range [2], [4], depending on the underlying hardware architecture. Furthermore, the difference between power consumed during computation versus idle time is expected to grow in future architectures.

Formally, our goal in this section is to minimize wasted energy E_{waste}, i.e. energy that is spent either on writing checkpoints ($E_{checkpt}$) or redoing work that was lost after a failure (E_{recomp}), and hence did not directly contribute to an application's forward progress, for an application running on N number of nodes:

$$\begin{aligned} E_{waste} &= E_{checkpt} + E_{recomp} \\ &= (N \cdot P_{checkpt} \cdot T_{checkpt}) + (N \cdot P_{comp} \cdot T_{recomp}) \end{aligned} \tag{2}$$

(Note that we do not include in our analysis the time needed to *restart* an application to the state of the most recent checkpoint, as this time does not depend on the checkpointing interval and hence will be the same for any checkpointing policy).

Intuitively, due to the difference between P_{comp} and $P_{checkpt}$, the checkpoint interval that minimizes energy consumption is shorter than Young's, since checkpointing uses less energy than computation. We show in the appendix in this paper that, under a number of simplifying assumptions similar to those made by Young, the energy-optimized checkpoint interval for a given ($P_{comp}/P_{checkpt}$) ratio can be approximated as follows:

$$\Delta_{Energy} = \sqrt{2 \cdot C \cdot MTTF \cdot \left(\frac{P_{checkpt}}{P_{comp}}\right)} \tag{3}$$

The remainder of this section will provide, among other things, a trace-based evaluation of the energy/performance tradeoff provided by Young's formula and Equation (3), including the accuracy of Equation (3) in estimating the energy-optimized checkpoint interval.

TABLE I. Overview of the LANL clusters in our dataset.

LANL System ID	#CPU Months	MTTF (days)	#Failures	Weibull (shape)
2	636,928	0.58	7104	0.739
18	164,350	0.31	3997	0.817
19	142,196	0.33	3284	0.889
20	91,438	0.57	2478	0.646
12	22,787	2.66	259	0.615
3	12,179	2.46	299	0.823
9	5,710	2.43	280	0.546
11	5,622	2.51	268	0.564
10	5,608	2.85	237	0.545
21	1,763	1.00	110	0.685

B. Trace-driven study of energy/performance tradeoffs

In this section, we use trace-driven simulations based on failure logs from 10 real HPC installations to answer the following three questions:

1) How do the energy/runtime tradeoffs vary as a function of the checkpoint interval, over a wide range of checkpoint intervals?
2) How does the runtime-optimized checkpoint interval fare in terms of energy consumption?
3) How close does our proposed solution in Equation (3) come to minimizing energy consumption?

The failure logs we use were collected on 10 different HPC clusters at Los Alamos National Lab (LANL) over a period of 9 years and are publicly available at LANL's webpage [10]. We selected these 10 LANL clusters (out of a larger dataset) by studying the best Weibull fit to the failure data in each LANL cluster whose failure logs are made available, and choosing representative systems of varying Weibull distribution shapes. Table I describes the 10 clusters in our study in more detail, including the value for the Weibull shape parameter, which ranges from 0.545 to 0.889.

We run trace-driven simulations for each LANL system to estimate the wasted energy for a wide range of checkpoint intervals, including $\Delta_{Runtime}$ and Δ_{Energy}. We vary the ratio between compute power and checkpoint power ($P_{comp}/P_{checkpt}$) from 2X to 8X and experiment with values for the checkpointing cost C from as low as 20 seconds to as high as 60 minutes. (Note that we do not make assumptions about the absolute values for P_{comp} and $P_{checkpt}$ as Equation (3) only needs the ratio between them).

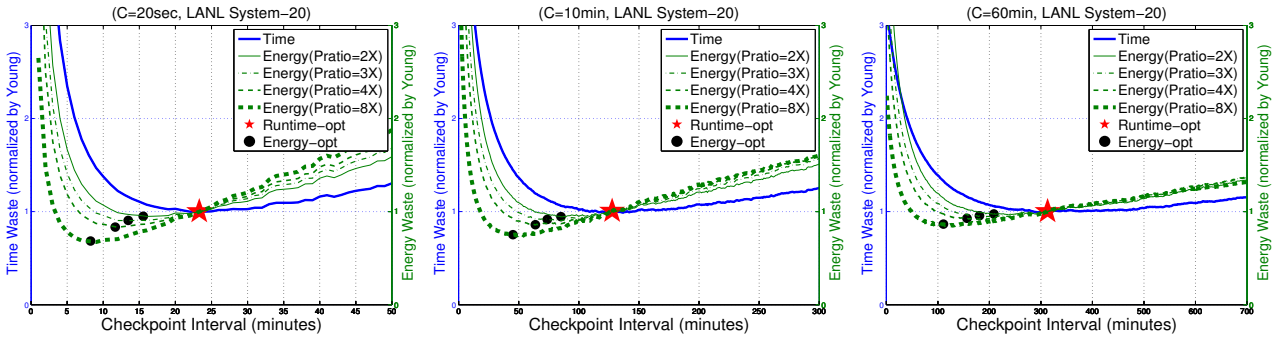

Fig. 1. Time and energy overheads as a function of the static checkpoint interval under different checkpoint costs (C) and power-ratios ($Pcomp/Pcheckpt$).

Figure 1 shows the results of the trace-driven simulation for one representative LANL system (system with ID 20). (Results for other systems are similar and are included in the tech-report [11]). Each graph corresponds to a different checkpoint cost C; the solid blue line in each graph corresponds to the wasted time (shown on the left Y-axis) and each of the other (green) lines corresponds to the wasted energy (shown on the right Y-axis) for a different $P_{comp}/P_{checkpt}$ ratio. The values for the wasted time (left y-axis) and wasted energy (right y-axis) are each normalized by the wasted time and wasted energy that result when using Young's traditional runtime-optimized interval $\Delta_{Runtime}$, respectively. The values for $\Delta_{Runtime}$ and Δ_{Energy} are marked with a star and a circle, respectively, on the X-axis.

The first observation we make from Figure 1 is that there is a relatively wide range of checkpoint intervals that leads to near-optimal amounts of wasted time, while the amount of energy consumed for checkpoint intervals in this range can vary significantly. For example, when assuming a checkpoint cost C of 10 minutes, we find that the fraction of wasted time under any checkpoint interval in the range $[90, 190]$ minutes is within 5% only of the optimal wasted time. Energy consumption, however, for the same range of intervals varies from a 5–16% *decrease* in energy waste, to a 16–25% *increase* in energy waste, compared to the energy wasted under $\Delta_{Runtime}$ (the ranges reflect different $P_{comp}/P_{checkpt}$ ratios).

The second observation is that while Young's interval ($\Delta_{Runtime}$), marked with a star on the X-axis, does produce near optimal results from a performance perspective, the same interval can be quite far from optimal in terms of energy consumptions, in particular for larger $P_{comp}/P_{checkpt}$ ratios. For example, for a checkpoint cost C of 10 minutes, $\Delta_{Runtime}$ wastes 15% and 25% more energy than the optimal energy waste achieved when assuming $P_{comp}/P_{checkpt}$ is 4X and 8X, respectively.

The third observation is that Δ_{Energy} does introduce *energy savings* (i.e. the decrease in wasted energy in the system w.r.t. the wasted energy under $\Delta_{Runtime}$), in the following ranges:

- 5–7% for $P_{comp}/P_{checkpt}$=2X (under different C values);
- 10–12% for $P_{comp}/P_{checkpt}$=3X;
- 15–19% for $P_{comp}/P_{checkpt}$=4X;
- 33–34% for the extreme case of $P_{comp}/P_{checkpt}$=8X.

While the energy savings that Δ_{Energy} provides do not come for free in terms of runtime overheads, the price one pays is on-par with the gains one reaps. We observe *runtime overheads* (i.e. the increase in wasted time in the system w.r.t. the wasted time under $\Delta_{Runtime}$), in the following ranges:

- 6–8% for $P_{comp}/P_{checkpt}$=2X (under different C values);
- 10–15% for $P_{comp}/P_{checkpt}$=3X;
- 16–24% for $P_{comp}/P_{checkpt}$=4X;
- 32–56% for $P_{comp}/P_{checkpt}$=8X.

It is important to recall that the above are increases in *wasted time*, and not total completion time. E.g. if for a given system and application a total of 10% of the time is wasted under $\Delta_{Runtime}$, then even in the extreme case of a $P_{comp}/P_{checkpt} = 8X$ not more than 13.2 – 15.6% of the time is wasted under Δ_{Energy}. (Note that we use the terms *energy savings* and *runtime overheads* in the rest of this paper to describe the decrease in wasted energy and the increase in wasted time, w.r.t. the wasted energy/time under Young's $\Delta_{Runtime}$, respectively).

Summary: Methods that schedule checkpoints to optimize application runtime are not optimal for energy purposes, and optimizing the checkpoint interval for energy introduces a tradeoff between energy savings and runtime overheads (due to the added checkpointing time). We find that the percentage of increase in wasted time is either comparable or modestly higher than the percentage of decrease in wasted energy, for the majority of the (C, $P_{comp}/P_{checkpt}$) configurations that we experimented with.

C. Optimizing for energy with a bound on runtime

After observing the performance cost of optimizing the checkpoint interval for energy savings, one might ask whether we can optimize checkpoint scheduling for energy purposes by minimizing the checkpoint interval, while enforcing a bound on the maximum allowed performance degradation (i.e. runtime overhead).

More precisely, for a given threshold t on the increase in the expected fraction of wasted time W in a system, compared to the wasted time that would have resulted under Young's $\Delta_{Runtime}$, we want to find the smallest checkpoint interval

Δ_{bound} that satisfies the following inequality:

$$W(\Delta_{bound}) < t \cdot W(\Delta_{Runtime}) \quad (4)$$

Solving this problem requires an estimate of the wasted time as a function of the size of the checkpoint interval. In [8], we derive the following approximation of the fraction of time wasted:

$$W(\Delta) = \frac{C}{\Delta} + \frac{\Delta}{2 \cdot MTTF} \quad (5)$$

Since we assume that C and MTTF are known for a given system, and $\Delta_{Runtime}$ can be computed using Equation (1), it is straightforward to determine Δ_{bound} using Inequality (4) and Equation (5).

To evaluate the quality of this approach, we determine the value for Δ_{bound} for several t values and then use trace-driven simulations, based on the same data and testbed as in the previous Section, to obtain the amount of wasted time and wasted energy resulting from Δ_{bound}.

Table II presents the results for runtime threshold values of 3%, 5% and 10%, under a checkpoint cost C of 10 minutes and a $P_{comp}/P_{checkpt}$ ratio of 3X. The two right most columns show the runtime overhead and energy savings for Δ_{bound} (w.r.t. the wasted time and wasted energy from $\Delta_{Runtime}$), respectively.

The first thing we observe from Table II is that enforcing a bound on runtime did not significantly affect the quality of the energy savings achieved. In fact, under this configuration (C=10 minutes, $P_{comp}/P_{checkpt}$=3X), a threshold on the runtime overhead as small as 3% was sufficient to introduce energy gains in the 7–11% range. As the threshold goes up to 10%, the corresponding energy savings either exhibit a modest increase (in the 0.3–4% range), or do not exhibit an increase at all (see systems 9, 10, 12). This is explained by the trends in the energy lines in Figure 1: as the runtime threshold increases, the checkpoint interval decreases until the added overhead due to checkpointing time raises the overall energy consumption of the system significantly.

We also observe from Table II that using Equation 5 to put a bound on runtime worked quite accurately in 23 scenarios out of the 30 scenarios we experimented with. For the 7 cases where the actual runtime overhead exceeded the estimated overhead, we observe that 5 of these cases belong to two LANL systems (systems 9 and 10), which warranted a closer look into their failure behaviour. Going back to Table I, we find that systems 9 and 10 have the lowest values for the Weibull shape parameter among all systems. A smaller shape parameter implies a heavier tail in the distribution function of failure inter-arrivals, while the equation assumes failures arrive on average half-way through a checkpoint interval, therefore leading to this estimation error in these two systems.

Summary: Optimizing the checkpoint interval for energy with a bound on runtime can be done using simple formulas that estimate wasted time in a system. We find that relatively small margins of increase in runtime (due to increased checkpointing frequency) can lead to high energy savings.

TABLE II. OPTIMIZING THE CHECKPOINT INTERVAL FOR ENERGY WITH A BOUND ON RUNTIME (RESULTS FOR C=10 MIN, PCOMP/PCHECKPT=3X).

LANL System ID	$\Delta_{Runtime}$ (min)	Runtime Threshold %	Δ_{bound} Length (min)	Δ_{bound} %time overhead (simulation)	Δ_{bound} %energy savings (simulation)
2	129.69	3	101.57	2.31	8.12
		5	94.65	3.96	9.49
		10	83.23	7.68	11.64
3	266.32	3	208.58	2.35	8.93
		5	194.37	4.2	10.41
		10	170.91	10.5	9.24
9	264.72	3	207.33	4.27	6.77
		5	193.20	8.3	5.03
		10	169.88	13.4	6.29
10	286.46	3	224.36	2.62	9.47
		5	209.07	7.34	6.81
		10	183.83	12.1	8.63
11	269.03	3	210.71	2.96	8.24
		5	196.35	3	12.46
		10	172.65	11.19	8.76
12	276.58	3	216.62	1.23	10.84
		5	201.86	4.48	10.25
		10	177.49	10.1	10.21
18	94.76	3	74.22	1.73	8.26
		5	69.16	3.28	9.59
		10	60.81	6.63	11.76
19	96.99	3	75.96	2.02	7.75
		5	70.79	3.05	9.91
		10	62.24	6.25	12.22
20	127.70	3	100.02	2.41	8.06
		5	93.20	4.69	8.59
		10	81.95	7.97	11.42
21	169.56	3	132.80	-0.54	11.52
		5	123.75	-1.68	16.78
		10	108.82	4.41	15.22

III. ENERGY AWARENESS IN ADVANCED METHODS

Section II demonstrates the potential for energy savings by carefully choosing the checkpoint interval. While Section II focused on simple static methods with a fixed checkpoint interval, the results motivate us to investigate also advanced, non-constant checkpointing methods, which adapt the checkpointing interval dynamically. We consider three types of methods, which we propose in [8]. While the focus of our other work [8] is to minimize the impact of checkpointing and failures on application runtime, in this paper we study their effect on energy consumption and how they can be adapted for minimizing energy waste in HPC systems.

A. Description of methods and their adaptation for energy

The three classes of methods that we consider all rely on the same principle: they exploit different characteristics of real world HPC failure processes to dynamically obtain (and continuously update) an accurate MTTF estimate and then plug this estimate into Young's formula (recall Equation (1)) to determine the length of the next checkpoint interval. As such, each of these methods can easily be adapted for optimizing for energy usage, rather than application runtime, by simply applying the MTTF estimates they provide to Equation (3), rather than Equation (1).

1) Variability in the system MTTF: Empirical studies of failures in HPC systems report that a system's MTTF varies over its lifetime [12]. We therefore propose that a checkpointing policy dynamically maintains an estimate of the system's MTTF using one of three implementations of Moving Averages (MA):

- SMA: This approach uses a Simple Moving Average, i.e. it simply calculates the MTTF as the average value of the failure inter-arrival times within an observation window consisting of the last w days and uses that average as an estimate of the expected time to the next failure. w is a parameter that needs to be determined.

- WMA: This method works like SMA, but uses a Weighted Moving Average, i.e. it assigns a weight for each value in the observation window, with more recent observations having greater weights. WMA, therefore, considers recent failure inter-arrival times more predictive of the time to next failure, than older ones.

- EMA: EMA uses an Exponential Moving Average to estimate the MTTF, i.e. the weights of older observations decrease exponentially, giving past values a diminishing contribution to the calculated average. Unlike SMA and WMA which only consider values within the observation window, EMA is a cumulative calculation that includes all the historical observations (the entire history of failures).

2) Failure autocorrelation: In this method, we propose to take information about the burstiness of the failure process into account when making checkpointing decisions, using autoregression (AR) to model the time between failures in a system, and make predictions about future failures.

More precisely, we fit an AR model to the observed sequence of failure inter-arrivals for each LANL system in our data, then use the fitted model to predict the time to next failure each time a checkpoint scheduling decision is to be made; i.e., after a system failure occurs. The new checkpointing interval is then determined by plugging the estimate from the AR model into Young's formula.

3) Decreasing Hazard Rates: In this method we study the potential of exploiting the statistical distribution of the time between failures, and particularly, the system's hazard rate function, in checkpoint scheduling. The motivation behind this approach is that, unlike the exponential distribution, empirical distributions of failures in HPC systems often exhibit decreasing hazard rates [13]. We observe this property in all 10 LANL systems in our data (as indicated by a Weibull shape parameter less than 1 in Table I).

A decreasing hazard rate function implies that if a long time has elapsed since the last failure then the expected remaining time until the next failure is long. The intuition behind exploiting this property in checkpoint scheduling is that one can reduce the checkpoint frequency if a long time has passed without seeing any failures.

We formalize this intuition in a method that we call Hazard, which maintains for a given system a table holding the empirical values of the (average) expected time to fail (ETTF) as a function of the elapsed time since the last failure (TSLF). Then, it uses this table to dynamically estimate the ETTF after a failure event or a checkpoint (based on the TSLF), and adapts the length of the checkpoint interval accordingly.

B. Trace-driven study of energy/performance tradeoffs

In this section, we use trace-driven simulations to examine the energy/performance tradeoffs of the methods described above, both when optimized for runtime or for energy consumption.

Table III shows the results achieved under the runtime-optimized and energy-optimized versions of the adaptive methods when running trace-based simulations for the 10 LANL systems in our data, assuming $P_{comp}/P_{checkpt}$ is 3X and the checkpoint cost C is 5 minutes. Note that for the sake of completeness we include in our comparative analysis Young's static interval as well as the higher order approximation of the optimum (static) checkpoint interval proposed by Daly [14].

1) Runtime-optimized methods: The three columns under 'Runtime-optimized methods' in Table III describe the runtime overhead, the energy overhead, and the fraction of time spent writing checkpoints when the different methods are optimized for runtime. We observe the following:

- Despite being optimized for runtime, all the adaptive methods, with the exception of Hazard, introduce both runtime and energy savings (i.e. decrease in wasted time and wasted energy) for the 10 LANL systems in our data, compared to the static methods.

- The average savings in wasted time and wasted energy when using the moving averages (SMA, WMA, EMA) over Young's interval are 5% and 7%, respectively. The best performing MA method overall is EMA, which results in runtime savings up to 8–13% and energy savings up to 10–18% (see systems 9, 10, 11, 12).

- We find that AR method produces average improvements of 6.5% and 7.6% in runtime and energy wastes, respectively. In some systems, exploiting failure autocorrelation resulted in significantly high runtime savings (9–17%) and energy savings (10–17%); see systems 3, 10, 11, 21.

- For the Hazard method, we find that the runtime improvements are on average quite marginal (1%), and no energy savings are introduced by the runtime-optimized version of this method for any LANL system in our data (energy overhead is 10.8% on average).

2) Energy-optimized methods: We repeat our analysis after optimizing all the methods for energy savings, as described in Section III-A, and observe the following (see the columns under 'Energy-optimized methods' in Table III):

- The energy-optimized methods do produce higher energy savings than the runtime-optimized ones with varying degrees of runtime overheads. Again, the adaptive methods outperform the static ones.

- The moving windows (SMA, WMA, EMA) now result in an average of 15.4% energy savings, but with an average runtime overhead of 11%. We find that EMA offers the best energy/runtime tradeoff among all MA methods.

- The average energy/runtime tradeoff for AR is comparable to the tradeoff for EMA: we find that AR results in an average of 17% energy savings, 7.5% runtime overhead.

TABLE III. COMPARISON OF THE PERFORMANCE/ENERGY TRADEOFFS INTRODUCED BY DIFFERENT CHECKPOINTING METHODS UNDER ($P_{comp}/P_{checkpt}$)=3X, C=5 MINUTES *(Note: a negative sign in the time/energy overhead columns indicates savings).*

LANL System ID	Method	Runtime-optimized Methods			Energy-optimized Methods		
		runtime overhead% (w.r.t. Young-runtime)	energy overhead% (w.r.t. Young-runtime)	*I/O fraction% (of total time)*	runtime overhead% (w.r.t. Young-runtime)	energy overhead% (w.r.t. Young-runtime)	*I/O fraction% (of total time)*
2	YOUNG	0	0	4.905	13.025	-11.836	8.368
	DALY	-0.010	-1.627	5.089	14.778	-11.929	8.664
	SMA	-2.295	-3.260	4.903	10.925	-14.595	8.340
	WMA	-2.416	-3.521	4.912	11.010	-14.585	8.353
	EMA	-2.414	-3.026	4.856	10.834	-14.056	8.264
	AR	-3.003	-2.806	4.736	9.302	-14.640	8.081
	Hazard	-0.145	8.650	3.899	2.683	-11.984	6.703
3	YOUNG	0	0	2.518	13.584	-11.913	4.329
	DALY	-2.332	-4.169	2.565	12.954	-14.143	4.406
	SMA	-1.329	-3.171	2.591	14.479	-12.558	4.441
	WMA	-1.788	-3.189	2.554	12.692	-14.056	4.379
	EMA	-2.773	-4.442	2.544	13.189	-13.055	4.363
	AR	-8.185	-10.204	2.428	6.885	-18.761	4.169
	Hazard	-1.046	1.111	2.367	9.985	-12.621	4.073
9	YOUNG	0	0	2.541	16.963	-9.579	4.363
	DALY	-1.675	-3.394	2.588	18.912	-8.095	4.437
	SMA	-9.807	-12.563	2.436	10.656	-15.050	4.159
	WMA	-10.031	-12.061	2.392	8.235	-17.302	4.089
	EMA	-11.753	-13.048	2.310	5.202	-19.229	3.954
	AR	-3.994	-8.006	2.649	16.307	-12.791	4.480
	Hazard	-2.331	17.384	1.448	-5.474	-7.310	2.498
10	YOUNG	0	0	2.354	15.195	-11.721	4.050
	DALY	1.928	2.075	2.392	15.834	-12.112	4.116
	SMA	-3.593	-6.247	2.401	13.816	-14.901	4.107
	WMA	-3.560	-5.685	2.376	11.458	-17.374	4.057
	EMA	-8.422	-11.487	2.308	9.640	-17.879	3.949
	AR	-9.001	-14.075	2.394	9.773	-19.024	4.016
	Hazard	-5.778	8.396	1.514	-3.072	-9.562	2.604
11	YOUNG	0	0	2.501	14.116	-13.097	4.298
	DALY	-3.769	-6.445	2.549	16.347	-11.137	4.368
	SMA	-8.144	-11.636	2.483	11.921	-15.065	4.231
	WMA	-7.058	-9.537	2.456	10.312	-16.593	4.186
	EMA	-8.577	-10.717	2.400	9.454	-16.259	4.102
	AR	-17.013	-17.211	2.086	-1.756	-22.906	3.579
	Hazard	-0.742	17.760	1.501	-7.113	-12.147	2.59
12	YOUNG	0	0	2.434	15.643	-10.321	4.181
	DALY	-4.489	-7.474	2.482	14.178	-13.901	4.256
	SMA	-8.794	-11.909	2.384	7.437	-20.404	4.080
	WMA	-8.697	-12.069	2.400	9.096	-18.363	4.100
	EMA	-13.602	-18.015	2.335	8.288	-17.410	3.987
	AR	-3.131	-4.949	2.453	11.759	-15.881	4.174
	Hazard	-0.975	13.652	1.641	-7.423	-18.590	2.841
18	YOUNG	0	0	6.450	10.805	-13.241	10.960
	DALY	-0.450	-2.708	6.779	12.909	-13.595	11.485
	SMA	-0.876	-1.813	6.542	11.640	-12.842	11.083
	WMA	-2.069	-3.459	6.537	11.101	-13.492	11.065
	EMA	-1.004	-1.509	6.465	10.821	-13.269	10.967
	AR	-1.178	-1.199	6.377	9.246	-14.666	10.838
	Hazard	0.130	6.642	5.426	3.307	-13.272	9.292
19	YOUNG	0	0	6.317	10.983	-13.209	10.750
	DALY	-0.353	-2.536	6.632	13.633	-12.698	11.248
	SMA	-0.129	-0.290	6.334	11.408	-12.726	10.767
	WMA	-0.057	-0.273	6.347	10.990	-13.438	10.787
	EMA	-0.057	-0.168	6.331	11.131	-13.090	10.763
	AR	-0.025	0.270	6.270	10.531	-13.334	10.671
	Hazard	0.173	4.470	5.664	6.406	-12.694	9.674
20	YOUNG	0	0	4.981	14.852	-9.702	8.480
	DALY	0.571	-0.831	5.167	17.172	-9.033	8.781
	SMA	-4.979	-6.610	4.916	10.369	-15.074	8.357
	WMA	-4.848	-5.411	4.803	8.767	-15.739	8.172
	EMA	-4.624	-5.556	4.855	9.378	-15.686	8.265
	AR	-4.212	-2.592	4.589	7.748	-14.274	7.842
	Hazard	0.180	12.721	3.580	0.198	-10.134	6.152
21	YOUNG	0	0	3.84	12.451	-12.749	6.572
	DALY	-1.097	-2.787	3.949	13.702	-13.012	6.755
	SMA	-1.775	-8.7126	4.392	18.506	-13.898	7.449
	WMA	-6.444	-12.156	4.103	12.117	-17.637	6.966
	EMA	-5.305	-10.786	4.126	13.525	-16.009	7.001
	AR	-15.734	-15.916	3.252	-4.718	-26.022	5.564
	Hazard	1.487	17.124	2.498	-5.95	-14.166	4.346

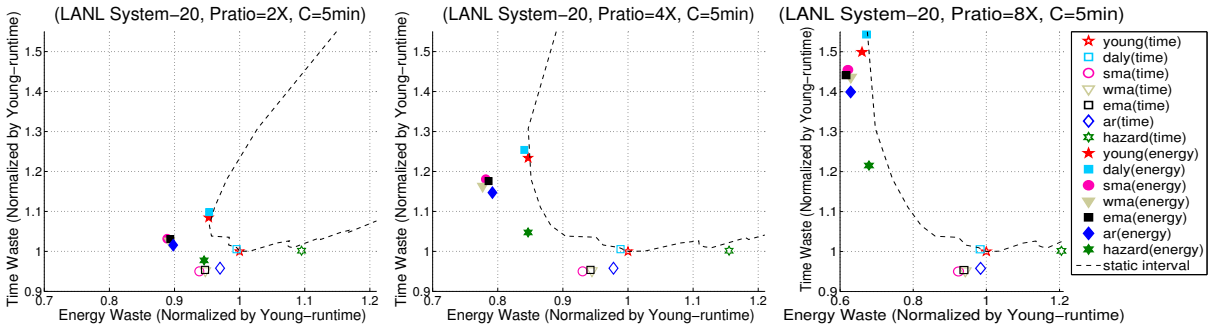

Fig. 2. Energy/performance tradeoffs for all checkpointing policies under different power-ratio assumptions ($P_{comp}/P_{checkpt}$) and assuming C=5 minutes, in LANL System 20. (Hollow markers represent runtime-optimized methods; filled markers represent energy-optimized methods).

In two LANL systems, however, AR performs significantly better where it results in 23% and 26% energy savings while also introducing 1.75% and 4.7% *improvements* in runtime waste over Young (see systems 11 and 21).

- For Hazard, we observe energy savings of 12% on average across all systems, which is lower than MAs and AR, but the performance results are better than for other methods (on average 1% *savings* in wasted time).

3) Results for different power ratios: The results in Table III were obtained under the assumption that P_{comp} is 3X higher than $P_{checkpt}$. We now repeat this analysis for other power ratios. Figure 2 plots for each checkpointing method the wasted runtime (Y-axis) versus the wasted energy (X-axis) that result under this method when it is optimized for runtime (hollow markers), and when it is optimized for energy (filled markers), in one of the LANL systems in our data, system 20. Each graph corresponds to a different $P_{comp}/P_{checkpt}$ ratio, from 2X (far left) to 8X (far right). The dashed black lines in the graphs show the energy/runtime results for a range of static checkpoint intervals. (Results for the other LANL systems can be found in the tech-report [11]).

The graphs in Figure 2 show that the energy-optimized adaptive methods consistently result in higher quality energy/runtime tradeoffs than static checkpoints, under different power ratios (i.e. higher energy savings with lower runtime costs). We also observe that the MA methods and AR exhibit similar trends for different power ratios, while Hazard consistently reports modestly lower energy savings (than MAs and AR), but with a much smaller runtime overhead.

Summary: Using adaptive techniques to schedule checkpoints dynamically introduces runtime and energy improvements over the static methods, both when optimized for runtime or for energy savings. The highest energy savings are introduced by the energy-optimized MAs and AR methods, while energy-optimized Hazard saves less energy but with a significantly lower runtime cost, if any.

C. Practical considerations

The high quality energy/performance results obtained under the adaptive checkpointing techniques motivated us to explore how easily they can be adopted in practice. We next examine different implementation issues for all the checkpointing methods included in our study:

1) 'Hazard' in practice: Our analysis shows that the energy-optimized Hazard results in a good balance between energy savings and runtime overhead (if any). But how feasible is implementing Hazard in practice? Recall that Hazard assumes a-priori knowledge of the system's hazard rate function (i.e. knowledge of how the expected time to fail changes as a function of the time since the last failure). We explore two ways to implement Hazard in practice, whenever accurate knowledge of the system's hazard function is not available:

a) Calculating the hazard function dynamically: We propose calculating the values for the (ETTF|TSLF) table dynamically using past failure observations. Practically, this method requires keeping a log of the system's failure history over time and updating the (ETTF|TSLF) table after a failure event. The updated table would then be used to estimate the ETTF every time a checkpoint interval is computed (i.e. after a failure or a checkpoint). We refer to this approach as Hazard(dynamic).

b) Approximating the hazard function using the Weibull shape parameter: This approach estimates the hazard function using the shape parameter for the Weibull fit of failures in a system. We assume knowledge of the shape parameter and use it to compute the hazard function (i.e. the (ETTF|TSLF) table) *prior* to the application run, instead of relying on actual failure traces. We call this approach Hazard(shape).

Figure 3 evaluates the quality of these two approaches using trace-driven simulations for the four largest LANL systems in our data in terms of CPU-days (systems 2, 18, 19, 20), by comparing the energy/runtime results of Hazard(dynamic) and Hazard(shape) to the results obtained under the original Hazard method. The bars for Hazard(shape) are the mean value of 20 runs; the error bars represent 95% confidence levels.

We find that the two approximation methods result in generally comparable energy savings to Hazard, with the exception of system 18 where Hazard(dynamic) introduces significantly lower energy savings. The runtime overhead for Hazard(dynamic) was close to the original Hazard in 3 of the 4 systems, while the runtime overhead for Hazard(shape) was consistently higher.

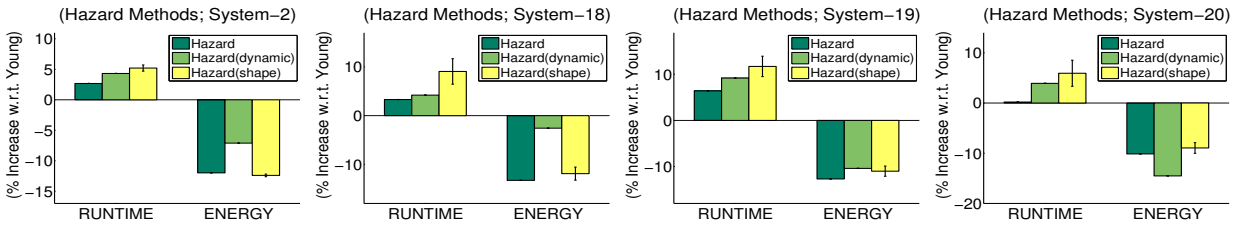

Fig. 3. Evaluation of different approaches to estimating energy-optimized *Hazard* in practice for four LANL systems, under ($P_{comp}/P_{checkpt}$)=3X.

2) MAs and AR in practice: The only requirement of the MA methods is to maintain a log of the most recent times of failures (for SMA, WMA), or a complete record of the times of failures (for EMA), which is easy to implement in practice. Implementing AR is more involved as it requires constructing an accurate autoregression model of the failure inter-arrivals in the system.

3) Young and Daly in practice: The simplicity of the static methods Young and Daly make them appear attractive for implementation purposes. However, these formulas rely on the a-priori knowledge of the system's real MTTF (unlike the MA methods that compute the MTTF online), which makes them more challenging to apply in practice.

Summary: From a practical point of view, the MA methods are the easiest to implement of all the methods we consider in our study. This observation, coupled with the high quality energy/runtime tradeoffs introduced by the MA methods, particularly EMA, makes them strong practical candidates for energy-optimized checkpointing. To achieve lower runtime overheads, however, we propose different techniques to implement the method Hazard in practice.

IV. Implications for the I/O Subsystem

Energy-optimized checkpoint scheduling increases the load on the I/O subsystem, since the system writes more frequent checkpoints to avoid the higher power-cost of redoing lost work (in case of a failure). In this section, we study the impact of energy-optimized checkpointing on the I/O subsystem, and investigate how to minimize this impact.

A. Static Checkpointing

We begin with a comparison of the I/O cost under static checkpointing when using the energy-optimized interval (Δ_{Energy}) versus the runtime-optimized interval ($\Delta_{Runtime}$). The column "I/O fraction%" in Table III shows the fraction of time the system spends writing checkpoints, when assuming a checkpoint cost of C=5 minutes and $P_{comp}/P_{checkpt}$=3X. We observe that, not surprisingly, the fraction of time spent on I/O under Δ_{Energy} is higher than under $\Delta_{Runtime}$, however it does not significantly exceed 10% in any of the cases. 10% is a typical target load that HPC storage systems are designed for [5].

For a more general understanding of how I/O time contributes to wasted time as a function of the checkpoint interval, Figure 4 plots the breakdown of wasted time into I/O time and lost computation across a wide range of checkpoint intervals. Results are shown for LANL systems 2 and 9, representatives

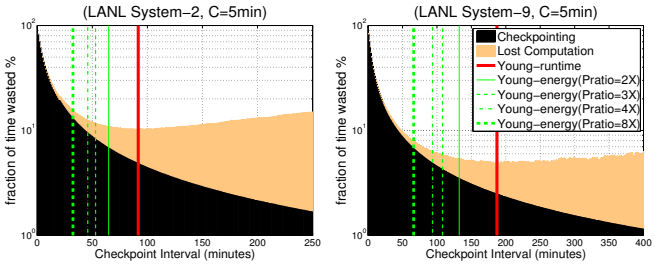

Fig. 4. Breakdown of wasted work across a wide range of static checkpoint intervals for two LANL systems (Y-axis is logscale).

of systems with higher and lower Weibull shape parameter, respectively. Each graph contains vertical lines marking the values for Δ_{Energy}, when assuming different power ratios (green dashed lines), and under $\Delta_{Runtime}$ (thick red line).

We make two observations. First, even under the aggressive assumption of $P_{comp}/P_{checkpt}$=8X the fraction of time spent on I/O remains within 7% and 13% in systems 9 and 2, respectively. (Note the log scale of the Y-axis).

Second, the graphs in Figure 4 show significant potential for using the checkpoint interval as a tuning knob to control the I/O load of a system: We observe that I/O time can be reduced significantly with only moderate runtime overhead, just by increasing the checkpoint interval. For example, within a 10% increase in the overall wasted time (compared to the wasted time under Young's $\Delta_{Runtime}$) the fraction of time spent on I/O can be reduced by nearly 50% in systems 2 and 9. Naturally, these savings come at the expense of increased *energy cost*: a 31% and 28% increase in energy waste in systems 2 and 9, respectively (under $P_{comp}/P_{checkpt}$=3X). However, in systems that are I/O bottlenecked this might be a cost worth paying.

B. Adaptive Methods

In this section, we study the I/O cost for the various adaptive checkpointing methods. Looking at the column "I/O fraction%" in Table III, we make several observations.

We find that the adaptive method Hazard consistently introduces the lowest fraction of I/O time across all methods, both when optimized for runtime or for energy. The reason is that this method reduces the checkpointing frequency whenever the expected time to fail is high. In fact, we find that the fraction of time spent on I/O under the energy-optimized Hazard remains below 10% in each system in our data. The MA methods on the other hand, when optimized for energy, consume I/O time that is comparable to the I/O time under

TABLE IV. OPTIMIZING THE CHECKPOINT INTERVAL FOR ENERGY WITH A BOUND ON I/O (SIMULATION RESULTS UNDER C=10 MINUTES, $P_{comp}/P_{checkpt}$=3X).

LANL System ID	I/O thresh %	Δ_{Energy}				$\Delta_{EnergyIO}$			
		Length (min)	%time overhead (w.r.t.Young)	%energy overhead (w.r.t.Young)	I/O time% *(of total time)*	Length (min)	%time overhead (w.r.t.Young)	%energy overhead (w.r.t.Young)	I/O time% *(of total time)*
2	10	75	11.65	-12.68	11.28	90	5.31	-10.38	9.49
9	5	153	16.95	-9.04	6.03	190	6.89	-8.31	4.88

the energy-optimized static methods (MAs however introduce higher energy savings than the static methods or Hazard). The method AR mostly consumes less I/O time than the MA methods but more than Hazard.

C. Optimizing for energy with a bound on I/O

This section explores checkpoint placement for energy savings that places a bound on I/O time. We approximate the fraction of time a system spends checkpointing as $C/(\Delta+C)$, since it spends on average roughly every $\Delta + C$ time units, C time units to checkpoint. Hence to stay within a certain bound b_{IO} on the fraction of time spent checkpointing, we need to ensure that $C/(\Delta+C) < b_{IO}$. We therefore choose the checkpoint interval $\Delta_{EnergyIO}$ that optimizes energy within a bound on the I/O cost as follows:

$$\Delta_{EnergyIO} = \max\{\Delta_{Energy},\, C/b_{IO} - C\} \qquad (6)$$

Table IV shows the simulation results from using Equation (6) for the two LANL systems in Figure 4 (systems 2 and 9). We chose I/O thresholds of 10% and 5% in systems 2 and 9, respectively, as those are moderately lower than the fraction of time spent on I/O under Δ_{Energy}. The table summarizes the results for runtime/energy/IO under Δ_{Energy} and under the interval that optimizes energy with a bound on the estimated I/O time ($\Delta_{EnergyIO}$).

We find that the I/O times under the new interval $\Delta_{EnergyIO}$ are within the desired I/O bounds. We also observe that the overall runtime overhead for $\Delta_{EnergyIO}$ is significantly lower than the runtime overhead under Δ_{Energy} (due to the reduced checkpointing time), but interestingly, the energy savings under $\Delta_{EnergyIO}$ drop only modestly. This agrees with our findings in Section II-C where we observed how moderate runtime overheads (w.r.t. Young's runtime interval) can lead to high energy savings. (We find similar results for the rest of LANL systems in our data; full table is included in the tech-report [11]).

Summary: The I/O pressure in checkpointing systems can be effectively reduced either by using adaptive methods, such as Hazard, which we show to have inherently lower I/O requirements, or by using a simple method we propose to determine a (static) checkpoint interval that optimizes energy savings while enforcing a bound on I/O time.

V. RELATED WORK

We are aware of only two papers related to checkpoint scheduling in the context of energy consumption [2], [15]. Both papers derive a formula for determining a constant checkpoint interval that optimizes energy, Meneses et al. [2] in order to compare the energy-efficiency of checkpoint/restart to other fault tolerance protocols and Aupy et al. [15] in order to derive projections on the energy-efficiency for future HPC platforms. While our work also starts by providing a formula to optimize the checkpoint interval, we make a series of very different contributions.

Our first contribution is a detailed evaluation study of energy, performance and I/O tradeoffs introduced by a wide array of checkpoint scheduling policies when optimized for energy purposes. Towards this end, we run trace-driven simulations using a decade worth of failure logs from 10 different HPC production clusters (compared to the synthetic failure loads used in related work [2], [15]), while experimenting with a broad range of parameters for power configurations and checkpoint/restart knobs.

Unlike other studies [2], [15] we not only include static checkpointing policies, but extend our work to study the energy efficiency of more advanced, non-constant checkpointing policies, how to optimize them for energy-efficiency and how to implement them in practice. Moreover, our paper is the first to provide a detailed analysis of the impact of checkpointing policies and their energy optimizations on the I/O subsystem, and to present and evaluate formulas for optimizing checkpoint scheduling for energy savings while enforcing *bounds* on application runtime or I/O time.

VI. CONCLUSION

Checkpoint scheduling policies in HPC platforms have been traditionally designed to optimize application runtime. However, with energy-efficiency becoming a key driver in the design of future Exascale architectures, optimizing the checkpointing process needs to be revisited and analyzed from an energy point of view.

In this paper, we provide a comprehensive evaluation of the different tradeoffs introduced by energy-optimized checkpoint scheduling policies in HPC platforms. We provide insights into the energy overhead, as well as the performance impact, associated with a wide array of checkpointing policies, using traces from 10 real world HPC clusters. We study policies that vary in their design and complexity, from basic, static checkpoint intervals to more advanced techniques that exploit HPC failure properties to adapt the checkpoint interval dynamically.

We show that optimizing checkpoints for energy savings, with or without a bound on application runtime, can be done through simple formulas that we propose and evaluate in this paper. Interestingly, our analysis shows that relatively small margins of increase in runtime (due to increased checkpointing operations) can produce high energy savings.

In our comparative analysis of the different checkpointing policies, we find that the energy-optimized *adaptive* policies result in higher quality energy/runtime tradeoffs than the static (constant) policies. We explore different practical considerations for these adaptive techniques and identify candidate methods that are easy to implement in practice (while producing high energy savings and relatively low runtime overheads), such as *exponential moving averages*.

Another contribution we make in this paper is the detailed assessment of the impact of energy optimized checkpointing on the I/O subsystem. We identify opportunities for I/O bottlenecked systems to achieve high I/O savings, either

through adaptive checkpointing methods that have low I/O requirements (e.g. *hazard-rate based* methods), or by using a simple, effective formula that we propose to optimize static checkpoints for energy with a bound on the I/O load.

VII. Acknowledgment

We would like to thank LANL for making their HPC failure data publicly available. We would also like to thank our anonymous reviewers for their insightful feedback, especially our shepherd Kamil Iskra. This work has been funded by an NSERC discovery grant.

Appendix
Derivation of Δ_{Energy} (Equation (3))

- Assuming an application spends C time units writing a checkpoint roughly every Δ time units, we get $T_{checkpt} = C/\Delta$.
- Assuming failures are equally likely to happen anywhere in a checkpoint interval, the amount of work lost every time a failure happens (i.e. on average once every MTTF time units) is $\Delta/2$; then $T_{recomp} = \left(\frac{\Delta}{2 \cdot MTTF}\right)$.
- Accordingly, Equation (2) for wasted energy becomes:

$$E_{waste(\Delta)} = (N \cdot P_{checkpt} \cdot C/\Delta) + (N \cdot P_{comp} \cdot \Delta/(2 \cdot MTTF))$$

And the first order derivative:

$$E'_{waste(\Delta)} = \left(-N \cdot P_{checkpt} \cdot C/\Delta^2\right) + (N \cdot P_{comp}/(2 \cdot MTTF))$$

- Solving $E'_{waste}(\Delta) = 0$ to find the value of Δ that minimizes E_{waste}, we get:

$$\Delta_{Energy} = \sqrt{2 \cdot C \cdot MTTF \cdot \left(\frac{P_{checkpt}}{P_{comp}}\right)}$$

References

[1] M. El Mehdi Diouri, O. Gluck, L. Lefevre, and F. Cappello, "Energy considerations in checkpointing and fault tolerance protocols," in *Proc. of DSN-W'12*, Boston, USA.

[2] E. Meneses, O. Sarood, and L. Kale, "Assessing energy efficiency of fault tolerance protocols for HPC systems," in *Proc. of SBAC-PAD'12*, NY, USA.

[3] M. E. M. Diouri, O. Glück, L. Lefèvre, and F. Cappello, "ECOFIT: A Framework to Estimate Energy Consumption of Fault Tolerance protocols during HPC executions," in *Proc. of CCGrid'13*, Delft, Pays-Bas. [Online]. Available: http://hal.inria.fr/hal-00806500

[4] D. Hackenberg, R. Schöne, D. Molka, M. Müller, and A. Knüpfer, "Quantifying power consumption variations of HPC systems using SPEC MPI benchmarks," *Computer Science - Research and Development*, vol. 25, no. 3-4, pp. 155–163, 2010. [Online]. Available: http://dx.doi.org/10.1007/s00450-010-0118-0

[5] B. Mills, R. E. Grant, K. B. Ferreira, and R. Riesen, "Evaluating energy savings for checkpoint/restart," in *Proc. of International Workshop on Energy Efficient Supercomputing*, ser. E2SC '13, NY, USA.

[6] T. Saito, K. Sato, H. Sato, and S. Matsuoka, "Energy-aware I/O optimization for checkpoint and restart on a NAND flash memory system," in *Proc. of the 3rd Workshop on Fault-tolerance for HPC at extreme scale*, NY, USA.

[7] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Commun. ACM*, vol. 17, no. 9, Sep. 1974.

[8] N. El-Sayed and B. Schroeder, "Checkpoint/restart in practice: When 'Simple is better'," in *Proc. of CLUSTER'14*.

[9] M. L. Curry, H. L. Ward, G. Grider, J. Gemmill, J. Harris, and D. Martinez, "Power use of disk subsystems in supercomputers," in *Proc. of PDSW'11*, NY, USA.

[10] "Operational Data to Support and Enable Computer Science Research, Los Alamos National Laboratory," http://institute.lanl.gov/data/fdata/.

[11] N. El-Sayed and B. Schroeder, "To checkpoint or not to checkpoint: Understanding energy-performance-IO tradeoffs in HPC checkpointing," University of Toronto, Tech. Rep. TECHNICAL REPORT CSRG-621, 2014.

[12] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proc. of DSN'06*.

[13] ——, "A large-scale study of failures in high-performance computing systems," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 337–350, Oct 2010.

[14] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Gener. Comput. Syst.*, vol. 22, no. 3, pp. 303–312, Feb. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.future.2004.11.016

[15] G. Aupy, A. Benoit, T. Hérault, Y. Robert, and J. Dongarra, "Optimal checkpointing period: Time vs. energy," *CoRR*, vol. abs/1310.8456, 2013.