Assuming failure independence: are we right to be wrong?

Guillaume Aupy, Yves Robert, Frédéric Vivien

Assuming failure independence: are we right to be wrong?

Guillaume Aupy*§, Yves Robert‡§, Frédéric Vivien§
*Inria & LaBRI, University of Bordeaux, France
‡Laboratoire LIP, École Normale Supérieure de Lyon & Inria, France
§University of Tennessee Knoxville, USA

***Abstract*—This paper revisits the *failure*[1] *temporal independence* hypothesis which is omnipresent in the analysis of resilience methods for HPC. We explain why a previous approach is incorrect, and we propose a new method to detect failure cascades, i.e., series of non-independent consecutive failures. We use this new method to assess whether public archive failure logs contain failure cascades. Then we design and compare several cascade-aware checkpointing algorithms to quantify the maximum gain that could be obtained, and we report extensive simulation results with archive and synthetic failure logs. Altogether, there are a few logs that contain cascades, but we show that the gain that can be achieved from this knowledge is not significant. The conclusion is that we can wrongly, but safely, assume failure independence!**

I. Introduction

This work revisits the *failure temporal independence* hypothesis in HPC (High Performance Computing) systems. Assuming failure temporal independence is mandatory to analyze resilience protocols. To give a single example: the well-known Young/Daly formula for the optimal checkpointing period [25], [8] is valid only if failure inter-arrival times, or IATs, are IID (Independent and Identically Distributed) random variables. We aim at providing a quantitative answer to the following question: to what extent are failures temporally independent? We base our analysis on publicly available failure logs from LANL [16], [15] and Tsubame [24]. We show that a previously proposed approach based on degraded intervals [4] leads to incorrect results, and we propose a new algorithm to detect failure cascades, based on the study of pairs of consecutive IATs. This new algorithm is used for the largest six public logs at our disposal, and we detect cascades in one log for sure, and possibly in a second one. The first conclusion is that it is wrong to assume failure independence everywhere!

The next question is to assess which gain can be achieved when equipped with the knowledge that failure cascades are present in some logs (and in real life, on some large-scale execution platforms). We design and compare several cascade-aware checkpointing algorithms. Four algorithms are simple periodic algorithms, with a constant checkpointing period obeying the Young/Daly formula $\sqrt{2C\mu}$; they differ by the value chosen for the MTBF μ. The choices for μ are: (i) the MTBF of the entire log; (ii) the MTBF of the log expunged of failures present in degraded intervals [4]; (iii) the MTBF of the log expunged of failures whose IATs belong to the first quantile (our new algorithm); and (iv) a brute-force algorithm that searches all possible periods (used for reference). The remaining seven algorithms are more sophisticated in that they use two different regimens: a *normal regimen* for failure-free segments, with a large checkpointing period; and a *degraded regimen* which is entered after a failure, with a smaller checkpointing period to cope with potential cascades. We compare different versions based upon the work in [4] and upon our new cascade detection algorithm. Finally, we use brute-force methods and oracle-based solutions to fully quantify the maximum gain that could be achieved with omniscient knowledge of forthcoming failures. Altogether, the overall conclusion is that there is not much to gain from the knowledge of failure cascades: we can just ignore them without any significant overhead. The second and final conclusion is that we can wrongly, but safely, assume failure independence! The main contributions of this work are:

- The correct evaluation of the method in [4] to detect failure cascades (Section III-A);
- A novel method to detect failure cascades, based on the quantile distribution of consecutive IAT pairs (Section III-B);
- The design and comparison of several cascade-aware checkpointing algorithms to assess the potential gain of cascade detection (Sections IV-A and IV-B);
- Extensive evaluation via archive and synthetic logs of all algorithms (Section V).

In addition to the above sections, Section II reviews relevant related work, and Section VI provides concluding remarks.

II. Related Work

Reliability is key to future extreme-scale HPC systems. The de-facto standard approach, namely the coordinated checkpointing protocol [7] has been recently extended in several directions, such as hierarchical checkpointing [6], in-memory checkpointing [26], [19], or multi-level checkpointing [18], [5], to quote just a few. To develop better solutions and reliability techniques, it is important to understand and characterize failure behavior. Indeed, failure characteristics can be used to inform failure predictors [9], [3], or to improve fault-tolerance techniques [12], [11], [23].

[1]We use the work *failure* and *fault* interchangeably, to denote an unrecoverable interruption of resource execution, a.k.a a fail-stop error.

In their seminal work, Young [25] and Daly [8] assume that failure IATs are IID and follow an Exponential probability distribution. This assumption was key to derive their famous formula for the optimal checkpoint interval. Other distributions have been considered, such as Weibull distributions [10], [23], [12]. Formulas for the optimal checkpointing period have been obtained by Gelenbe and Hernández [10], but still under the temporal independence assumption. In stochastic terms, each checkpoint is assumed to be a renewal point, which (unrealistically) amounts to rebooting the entire platform after each failure. Tiwari et al. [23] and Heien et al [12] confirmed the observation that the Young/Daly formula is a very good approximation for Weibull distributions. In particular, Tiwari et al. [23] analyze a failure log, show that it matches well a Weibull distribution through Kolmogorov-Smirnov and other statistical tests, and report some gains when using the fact that much more than half of IATs are smaller than the expectation of the Weibull law (due to its infant mortality property).

It is important to understand the impact of the renewal condition on the works that deal with non-Exponential distribution. Consider a platform with several processors, each subject to failures whose IATs follow a Weibull (or, in fact, any other, except Exponential) probability distribution. Without the renewal condition, the distribution of failure IATs over the whole platform is no longer independent nor identically distributed, which complicates everything, including the mere definition of the platform MTBF (see [13] for details). This explains why all these previous works consider that failures are temporally independent.

However, it has been observed many times that when a failure occurs, it may trigger other failures that will strike different system components [12], [23], [4]. As an example, a failing cooling system may cause a series of successive crashes of different nodes. Also, an outstanding error in the file system will likely be followed by several others [21], [14]. Recently Bautista-Gomez et al. [4] studied nine systems, and they report periods of high failure density in all of them. They call these periods *cascade failures*. This observation has led them to revisit the temporal failure independence assumption, and to design bi-periodic checkpointing algorithms that use different periods in normal (failure-free) and degraded (with failure cascades) modes. See Sections III-A and IV-B for a full description of their approach [4].

Finally, this paper focuses on temporal properties of failures. Spatial properties of failures [11] are out of scope for this study.

III. Algorithms to Detect Failure Cascades

We informally define a failure cascade as a series of consecutive failures that strike closer in time that one would normally expect. We explain how to refine such an imprecise definition by describing two approaches below. The first approach is introduced in [4] and is based upon *degraded intervals*. The second approach is a major contribution of this paper and uses *quantiles of consecutive* IAT *pairs*.

A. Degraded Intervals

In their recent work, Bautista-Gomez et al. [4] provide Algorithm 1 to detect *degraded intervals*, i.e., intervals containing failure cascades. Consider a failure log of total duration L seconds, with n failures striking at time-steps t_i, where $0 \leq t_1 \leq \ldots \leq t_n \leq L$. By definition, the MTBF (Mean Time Between Failures) of the platform log is $\mu = \frac{L}{n}$. Intuitively, we expect failures to strike every μ seconds in average. Algorithm 1 simply divides the log into n intervals of length L/n and checks for intervals that contain two or more failures. Such intervals are called *degraded* while intervals with zero or one failure are called *normal*. Algorithm 1 returns the set $\mathcal{S}_c$ of degraded intervals. The percentage of degraded intervals is $P_{deg} = \frac{|\mathcal{S}_c|}{n}$.

Algorithm 1 Identifying the set $\mathcal{S}_c$ of degraded intervals.

```
1: procedure DetCascades({t_1, ..., t_n}, L, X)
2:   for i = 0 to n − 1 do
3:     if |{j|t_j ∈ [i L/n, (i+1) L/n]}| ≥ 2 then
4:       S_c = S_c ∪ {j|t_j ∈ [i L/n, (i+1) L/n]}
5:     end if
6:   end for
7:   return S_c
8: end procedure
```

Table I summarizes the percentage of degraded interval P_{deg} found by Algorithm 1 for the LANL and Tsubame failure logs. Altogether, 20% to 30% of system time is spent in degraded regimen. The authors of [4] conclude that Algorithm 1 has identified failure cascades. This conclusion is incorrect:

- Assume first that failure IATs are IID and follow an Exponential probability distribution $\text{EXP}[\lambda]$ with parameter λ (we have $\mathbb{P}(X \leq t) = 1 - e^{-\lambda t}$). For a failure log of size L and with n failures, we directly have $\lambda = \frac{1}{\mu} = \frac{n}{L}$. Theorem 1 shows that in this case, the expected percentage of degraded intervals is $1 - \frac{2}{e} \approx 0.264$ (or 26.4%).
- Assume now that failure IATs are IID and follow a Weibull probability distribution $\text{WEIBULL}[k, \lambda]$ with parameter shape parameter k and scale parameter λ: we have $\mathbb{P}(X \leq t) = 1 - e^{-(\frac{t}{\lambda})^k}$, and the MTBF is $\mu = \lambda\Gamma(1 + \frac{1}{k})$. For a failure log of size L and with n failures, we have $\mu = \frac{L}{n}$, hence we let $\lambda = \frac{L}{n\Gamma(1+\frac{1}{k})}$. Table II shows the value of P_{deg} for values of the shape parameter k ranging from 0.5 to 1. The value for $k = 1$ is the same as for the Exponential distribution, because $\text{WEIBULL}[1, \lambda]$ reduces to $\text{EXP}[\frac{1}{\lambda}]$. For all the values of k in Table II, the percentage of degraded intervals lies between 26% and 27.5%. These values are independent of λ and are obtained experimentally, using MonteCarlo simulations. Note that typical values of k used to model failures in the literature are ranging from $k = 0.5$ to $k = 0.9$ [17], [20], [23].

Altogether, the correct conclusion from Theorem 1 and the results in Table I is that public logs exhibit the same number of degraded intervals as pure $\text{EXP}[\lambda]$ and $\text{WEIBULL}[k, \lambda]$ renewal processes. Hence we cannot conclude anything!

Id	Log Number of faults	Log MTBF in hours	Approach in [4] MTBF in hours	Approach in [4] Degraded intervals: P_{deg}	Approach in [4] Faults in cascades
LANL 2	5351	14.1	36.4	25.3%	71.1%
LANL 3	294	59.3	142.3	26.3%	69.4%
LANL 4	298	56.2	126.7	24.9%	66.8%
LANL 5	304	54.7	119.6	26.4%	66.4%
LANL 6	63	279.9	604.1	33.9%	69.8%
LANL 7	126	311.1	943.1	21.6%	74.6%
LANL 8	448	84.5	226.6	26.2%	72.5%
LANL 9	278	58.8	216.0	23.1%	79.1%
LANL 10	234	68.7	218.3	23.6%	76.1%
LANL 11	265	60.8	230.5	23.9%	80.0%
LANL 12	254	64.2	192.7	25.3%	75.2%
LANL 13	193	83.4	380.7	24.0%	83.4%
LANL 14	120	103.7	410.6	20.0%	80.0%
LANL 15	53	124.1	292.0	23.1%	67.9%
LANL 16	2262	21.9	56.2	25.2%	70.9%
LANL 17	125	216.9	526.1	21.8%	68.0%
LANL 18	3900	7.5	17.9	26.0%	68.9%
LANL 19	3222	7.9	17.1	26.4%	66.0%
LANL 20	2389	13.7	41.5	21.3%	74.1%
LANL 21	105	24.2	79.9	26.9%	78.1%
LANL 22	235	272.1	696.6	27.8%	71.9%
LANL 23	448	147.3	348.8	23.7%	67.9%
LANL 24	150	412.7	1040.6	22.1%	69.3%
Tsubame	884	14.8	36.5	23.9%	69.2%

Table I: Percentage of degraded intervals in failure logs. Expected value for an Exponential distribution is 26.4%. Red entries are within 5% of this value. Pink entries are within 10% of this value.

Shape parameter k	Degraded intervals: P_{deg}	Faults in cascades
0.50	26.0%	84.7%
0.52	26.3%	83.7%
0.54	26.6%	82.7%
0.56	26.8%	81.6%
0.58	27.0%	80.6%
0.60	27.2%	79.7%
0.62	27.3%	78.7%
0.64	27.4%	77.7%
0.66	27.4%	76.8%
0.68	27.5%	75.9%
0.70	27.5%	75.0%
0.72	27.5%	74.1%
0.74	27.5%	73.2%
0.76	27.5%	72.4%
0.78	27.4%	71.5%
0.80	27.4%	70.7%
0.82	27.3%	69.9%
0.84	27.3%	69.1%
0.86	27.2%	68.3%
0.88	27.1%	67.5%
0.90	27.0%	66.8%
0.92	26.9%	66.0%
0.94	26.8%	65.3%
0.96	26.7%	64.6%
0.98	26.6%	63.9%
1.00	26.4%	63.2%

Table II: Expected values of P_{deg} and of the number of faults in cascades for IID IATs following a Weibull distribution $\text{WEIBULL}[k, \lambda]$.

Theorem 1. *For a log duration L, and IID failure* IATs *with distribution* $\text{EXP}[\lambda]$*, the percentage of degraded intervals P_{deg} converges to* $\lim_{L\to\infty} P_{deg} = 100(1 - \frac{2}{e}) \approx 26.4\%$.

Due to lack of space, the (technical) proof of Theorem 1 is available in the companion research report [2].

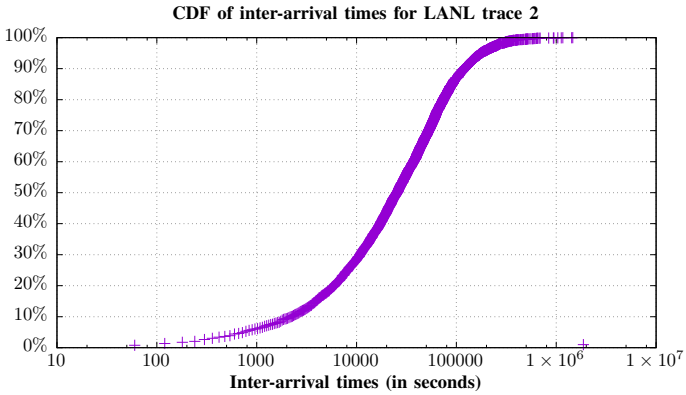

Figure 1: Cumulative plot of IATs for failure log LANL 2.

B. *Quantile distribution of consecutive* IAT *pairs*

In this section, we introduce a new method to detect failure cascades. In a nutshell, the method analyzes the distribution of pairs of two consecutive IATs. Intuitively, consider a failure log and its IATs. If small values are scattered across the log, we do not conclude anything. On the contrary, if a small value follows another small value, we may have found the beginning of a cascade. Our approach checks the frequency of having two consecutive small values, and compares this frequency with the expected value when IATs are independent. Our approach proceeds as follows. Consider a failure log with $n = N + 2$ failures, i.e., with $N + 1$ IATs z_i, $1 \leq i \leq N + 1$. Note that the first failure in the log does not correspond to an IAT since its predecessor is not included in the log. Note also that $z_i = t_{i+1} - t_i$ where t_i is the time where the i-th failure strikes, as defined in Algorithm 1. Finally, note that there are N pairs (z_i, z_{i+1}) of consecutive IATs, $1 \leq i \leq N$.

We start by sampling the failure log and computing quantiles. In Figure 1 we plot the cumulative distribution of IATs for failure log LANL2. In Figure 1, we use $Q = 10$ quantiles. The 10% smallest values constitute the first quantile, or *limit quantile* Q_{limit}, and are between 0 and 2,220 seconds. The next 10% smallest values (second quantile) are between 2,221 and 6,000 seconds, and so on. By definition, the probability that an IAT belongs to a given quantile is $\frac{1}{Q} = 0.1$. Now, if we assume (temporal) failure independence, the probability that both components of a pair (z_i, z_{i+1}) of consecutive IATs belongs to the same given quantile is $\frac{1}{Q^2}$, and the expected number of such pairs is $\frac{N}{Q^2}$. We need N to be reasonably large so that this expectation is accurate. Out of the 24 logs in Table I, we keep only the five LANL logs with $N \geq 1000$ (namely LANL 2, 16, 18, 19, 20) and the Tsubame log: see Table III. For these 6 logs, in Figures 2 to 7, we plot the ratio of the actual number of pairs in each quantile over the expected value. This ratio is called the lag plot density [1], [22]. We expect a log without cascades to exhibit a flat surface up to a few statistical artefacts. The figures lead to the conclusions reported in Table III: only log LANL2 contains cascades for sure, because the ratio for the first quantile is greater than four times its expected value; maybe LANL20 does too (ratio between 2 and 3); the other logs do not include any cascade.

Note that another phenomenon can be observed in LANL2 and, to some extent, in LANL20: there is an *anti*-cascade

Log	Number of Faults	Cascades
LANL 2	5351	Yes
LANL 16	2262	No
LANL 18	3900	No
LANL 19	3222	No
LANL 20	2389	Maybe
Tsubame	884	No

Table III: Presence of cascades in large logs.

behavior, where long IATs are followed by other long IATs more often than expected. We are not able to give any explanation to this phenomenon.

Altogether, there are indeed some cascades, albeit not very frequent, in some failure logs. Hence we were wrong to assume failure independence everywhere. The next question is whether the knowledge that cascades are present may help reduce the overhead due to the standard checkpoint/recovery approach. The rest of the paper is devoted to answering this question.

In the companion research report [2], we provide three more lag plots to help the reader understand and assess temporal correlation in failure logs: a first for an EXP$[\lambda]$ distribution, and a second one for a WEIBULL$[0.7, \lambda]$ IAT distributions: as expected, there is no cascade in these renewal processes. The third lag plot is for LANL2 after randomly shuffling the log IATs: there is no more cascade either!

IV. CASCADE-AWARE CHECKPOINTING

In this section we provide a quantitative assessment of many algorithms that can be used to improve the classical periodic checkpointing algorithm, whose period is given by the Young/Daly formula [25], [8]. We use both the previous public logs and synthetic logs to generate simulation results.

When playing an algorithm against a log, it is common practice to have the algorithm learn data from, say, the first half of the log, and play against the second half of the log. More precisely, we would start the algorithm from a randomly generated instant in the second half of the log to avoid bias. This instant must be not too close to the end of the log to ensure that there remain enough failures to strike, e.g., any instant in the thirst quarter of the log would be fine. In the following, we do not adopt this strategy. Instead, we let the algorithms learn form the entire log, and replay them from a randomly chosen instant. This gives a somewhat unfair advantage to the algorithms, but our goal is to provide an upper bound of the maximum gains that can be achieved.

A. Periodic algorithms

What can we learn from a log? The Young/Daly checkpointing algorithm needs to know the checkpoint time C and the MTBF μ_{log}, and then uses the optimal checkpointing period $T = \sqrt{2\mu_{log}C}$. As already mentioned, one can directly use $\mu_{log} = \frac{L}{N}$ for a log of length L with N failures. However, inspecting the log for cascades can lead to refined values of the MTBF:

- Remember that [4] defines two categories of intervals in the log, *normal* and *degraded*, and computes μ_{normal_int} and $\mu_{degraded\_int}$, the MTBF for each set of intervals. Typically, μ_{normal_int} will be significantly larger than μ_{log}, and using this value instead of μ_{log} will decrease the failure-free overhead incurred by checkpointing too frequently outside the cascades.
- Following the approach in Section III-B, we can divide the log into Q quantiles and separate IATs that belong to the first quantile Q_{limit} from the other ones. IATs that belong to the first quantile Q_{limit} constitute the cascades and have mean value $\mu_{cascade}$(Q). The other IATs have mean value $\mu_{non-cascade}$(Q). Just as μ_{normal_int}, $\mu_{non-cascade}$(Q) will be larger than μ_{log}, with the same desired impact. We use different values of $|Q_{limit}|$ for the simulations: $|Q_{limit}| = 10\%$, and $|Q_{limit}| = 5\%$ in this paper, and, in addition, $|Q_{limit}| \in \{2\%, 1\%, 0.5\%, 0.2\%, 0.1\%\}$ in the companion research report [2].

Altogether, we evaluate 4 periodic checkpointing algorithms, which use the period $T = \sqrt{2\mu C}$, where μ is chosen from the following:

- Algorithm Π_{Daly} uses $\mu = \mu_{log}$
- Algorithm $\Pi_{Intervals}$ uses $\mu = \mu_{normal\_int}$
- Algorithm $\Pi_{Quantiles}$ uses $\mu = \mu_{non-cascade}(Q)$
- Algorithm Π_{Best_period} uses a brute-force search and returns the best period

For each algorithm, we report the WASTE, defined as the fraction of time where the platform does not perform useful work. Experimental values of WASTE are averaged from many Monte Carlo simulations. As a side note, a first-order approximation is given by the formula

$$\text{WASTE} = \frac{C}{T} + \frac{1}{\mu}(R + \frac{T}{2}) \quad (1)$$

and is obtained as follows [13]: the first term $\frac{C}{T}$ is the overhead in a failure free execution, since we lose C seconds to checkpoint, every period of T seconds. The second term is the overhead due to failures, which strike every μ seconds in expectation; for each failure, we lose R seconds for recovery (letting $R = C$ everywhere) and re-execute half the period on average. Equation (1) explains that checkpointing algorithms aim at finding the best trade-off between checkpointing too often (large failure-free overhead) and not often enough (large overhead after a failure); the waste is minimized when $T = \sqrt{2\mu C}$. All this helps understand how WASTE depends on the value chosen for the MTBF μ.

B. Bi-periodic algorithms

We compare the following seven bi-periodic algorithms. Each bi-periodic algorithm uses two different regimens, namely the *normal* and *degraded* regimens, to borrow the terminology of [4]. In the *normal* regimen, which is the regimen by default, the algorithm runs in the absence of failures, hence uses a larger checkpointing period. In the

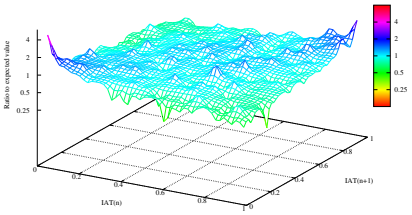

Figure 2: Lag plot for LANL2.

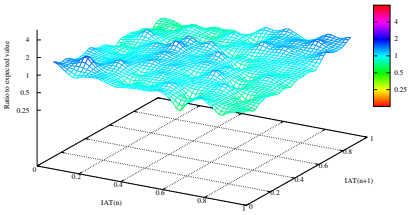

Figure 3: Lag plot for LANL16.

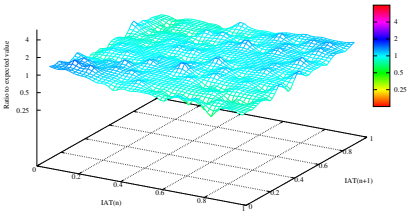

Figure 4: Lag plot for LANL18.

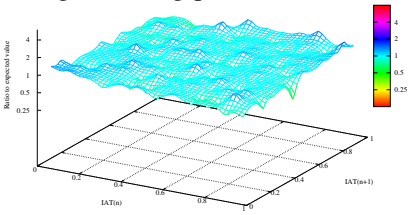

Figure 5: Lag plot for LANL19.

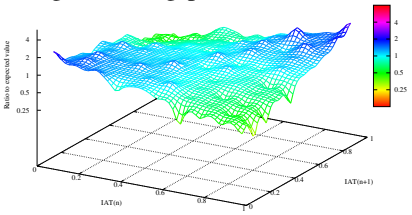

Figure 6: Lag plot for LANL 20.

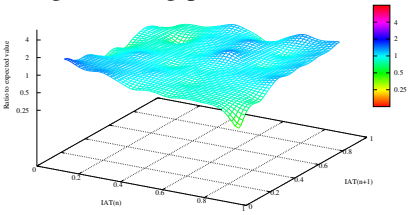

Figure 7: Lag plot for Tsubame.

degraded regimen, the algorithm uses a shorter checkpointing period, to cope with potential cascades.

The seven algorithms differ by several parameters:

- the MTBF value μ used for each regimen, which dictates the corresponding checkpointing period. In Table IV, we report the MTBF for the normal regimen μ_{normal} and for the MTBF for the degraded regimen $\mu_{degraded}$. Again, the checkpointing period for the normal regimen is $T = \sqrt{2\mu_{normal}C}$ and that for the degraded regimen is $T = \sqrt{2\mu_{degraded}C}$.
- the criteria used to enter and exit the cascade regimen. Most algorithms enter the cascade regimen as soon as a failure strikes, but *lazy* variants enter the cascade regimen only after a second failure has struck, and provided that the IAT belongs to the first quantile Q_{limit}. All algorithms exit the degraded regimen when enough time has elapsed since the last failure. Following [4], we set this timeout to $2\mu_{degraded}$.

All these parameters are listed in Table IV. For reference, here follows a detailed description of each bi-periodic algorithm. First, we have 3 algorithms based on refined values of the MTBF:

- BI-$\Pi_{Intervals}$ uses $\mu_{normal} = \mu_{normal\_int}$ and $\mu_{degraded} = \mu_{degraded\_int}$ for the checkpointing periods, as proposed by [4]. It enters the degraded mode as soon as a failure strikes, and exits it with timeout $2\mu_{degraded}$.
- BI-$\Pi_{Quantiles}$ works similarly, but with different MTBF values. $\mu_{normal} = \mu_{non-cascade}(Q)$ and $\mu_{degraded} = \mu_{cascade}(Q)$. These values are computed from the quantiles of the log IATs.
- BI-$\Pi_{Quantiles}$LAZY is a variant of BI-$\Pi_{Quantiles}$ where the degraded mode is entered only after two successive failures, provided that the second failure strikes shortly (IAT in first quantile Q_{limit}) after the first one.

The next 2 algorithms use brute-force search:

- BI-Π-BEST uses a brute-force method for everything: it computes the waste for all values of μ_{normal}, $\mu_{degraded}$ and timeout value, and retains the best triple. BI-Π-BEST is agnostic of the cascade detection algorithm; its only limitation is that it enters the degraded mode after the first failure.
- BI-$\Pi_{Quantiles}$LAZY-BEST is the lazy variant of BI-Π-BEST. However, to decide whether to enter the degraded mode, just as BI-$\Pi_{Quantiles}$LAZY, it needs to know the quantiles of the distribution to check whether the IAT of the second failure belongs to the first quantile Q_{limit}.

Finally, the last two algorithms are included for reference. They use oracles that know everything in cascades, including the future! Specifically:

- BI-$\Pi_{Quantiles}$ORACLE uses $\mu_{normal} = \mu_{non-cascade}(Q)$ in normal mode, just as BI-$\Pi_{Quantiles}$. However, as soon as a failure strikes, it knows exactly whether there will be a cascade, and when the next failures in that cascade will strike. It can thus checkpoint as late as possible, completing the checkpoint right before the failure. And it also knows in advance when the next failure is far away (IAT not in the first quantile Q_{limit}), so that it can immediately switch back to normal mode.
- BI-Π-ORACLE-BEST is the variant of BI-$\Pi_{Quantiles}$ORACLE that tests all possible values of μ_{normal} in normal mode, not just $\mu_{non-cascade}(Q)$. It behaves exactly the same after a failure. A comparison with BI-$\Pi_{Quantiles}$ORACLE will help assess whether using $\mu_{normal} = \mu_{non-cascade}(Q)$ is a good decision or not.

V. SIMULATION RESULTS

A. Simulation setup

In addition to the six large public logs of Table III, we generate synthetic logs. We first generate failures according to an Exponential distribution of MTBF $\mu_1 = 3,600$ seconds. Such an MTBF of 1 hour is much smaller than archive logs MTBFs in Table III, which are of the order of 10 hours. This is because we want the waste to be higher for synthetic logs than for archive logs, so that there is a potential significant gain with cascade detection. Next we perturb the log by randomly

Algorithm	μ_{normal}	$\mu_{degraded}$	Enter criterion	Timeout Exit Criterion
BI-$\Pi_{Intervals}$	μ_{normal_int}	$\mu_{degraded\_int}$	First failure	$2\mu_{degraded\_int}$
BI-$\Pi_{Quantiles}$	$\mu_{cascade}(Q)$	$\mu_{non-cascade}(Q)$	First failure	$2\mu_{non-cascade}(Q)$
BI-Π-BEST	Best value for μ_{normal}	Best value for $\mu_{degraded}$	First failure	Best value
BI-$\Pi_{Quantiles}$LAZY	$\mu_{cascade}(Q)$	$\mu_{non-cascade}(Q)$	Second failure in first quantile	$2\mu_{non-cascade}(Q)$
BI-$\Pi_{Quantiles}$LAZY-BEST	Best value for μ_{normal}	Best value for $\mu_{degraded}$	Second failure in first quantile	Best value
BI-$\Pi_{Quantiles}$ORACLE	$\mu_{cascade}(Q)$	Omniscient	Omniscient	Omniscient
BI-Π-ORACLE-BEST	Best value for μ_{normal}	Omniscient	Omniscient	Omniscient

Table IV: Bi-periodic algorithms

generating cascades: after each failure, we generate a cascade with frequency (probability that this failure is the start of a cascade) $f = 1\%$, $f = 5\%$ or $f = 10\%$. The length of a cascade, defined as the number of additional failures generated (thus not counting the original failure), is a random value between 3 and 5 (we write $\ell = 3-5$), or between 3 and 10 (we write $\ell = 3-10$). Finally, the failures in the cascades follow another Exponential distribution of MTBF $\mu_2 = \frac{\mu_1}{\rho}$, where the ratio ρ is chosen in $\{10, 100, 1000\}$. Altogether, a synthetic log is tagged '`Synth.` $\rho|f|\ell$'. For instance, '`Synth.` $\rho = 100|1\%|3-5$' means that a cascade is generated every 100 failures in average ($f = 1\%$), with MTBF 36 seconds, one hundredth of the original distribution ($\rho = 100$), and a number of additional failures, not including the first one, uniformly drawn between 3 and 5. The lag plot for this log is given in [2] and does show the cascades.

For each archive and synthetic log, we average results over 100 executions. We draw a starting point uniformly between the beginning of the log and $200\mu_{log}$ seconds before its end, and we run the simulation during $100\mu_{log}$ seconds, not counting checkpoint and re-execution overhead. Finally, we use a wide range of checkpoint values, namely $C = 300$, $C = 30$ and $C = 3$ seconds, in order to cover the widest range of scenarios.

B. Waste values

Due to lack of space, we report results only for $|Q_{limit}| = 5\%$. Values for $|Q_{limit}| \in \{10\%, 2\%, 1\%, 0.5\%, 0.2\%, 0.1\%\}$ are available in [2]. Results follow the same trends for all values of $|Q_{limit}|$.

Log statistics are provided in Table V. The column "Common faults" reports the percentage of failures that are detected as belonging to cascades, by both the interval-based and the quantile-based approaches.

For the waste, we report the improvement or degradation with respect to the reference periodic checkpointing algorithm Π_{Daly}. The color code in Tables VI to VIII is the following:

- Green: Improvement by at least 10%
- Lime: Improvement between 5 and 10%
- Black: Improvement between 0 and 5%
- Pink: Degradation between 0 and 5%
- Red: Degradation larger than 5%

C. Discussion

Overall, the main take-away is that cascade-aware algorithms achieve negligible gains, except for a few scenarios where the waste is already very low with the standard Π_{Daly} approach. This is true even when considering the best scenarios with (i) very short checkpointing time; (ii) high frequency of cascade failures: and (iii) knowing exactly when the next cascade failures are going to strike. In fact, small gains could be achieved only if cascade failures strike both *frequently*, say more than 10% of the time after an initial failure, and *with a relatively large MTBF*, say, not less than 10% of the log MTBF. We further discuss these statements in the following.

1) Cascade detection algorithms: Tables VI to VIII show that both cascade-aware algorithms based on intervals and on quantiles, are not more efficient than the simple Π_{Daly} approach. While the quantile-based approach ($\Pi_{Quantiles}$, BI-$\Pi_{Quantiles}$, BI-$\Pi_{Quantiles}$LAZY) seems slightly better than the interval-based approach ($\Pi_{Intervals}$, BI-$\Pi_{Intervals}$), the gain (or loss) is still within an error margin (most are between -1% and +1%). Furthermore, as one can see from Tables VI to VIII, the quantile-based approach seems to perform better than the interval-based approach: this is because it detects fewer cascades, hence its MTBF for the normal regimen is very similar to the one used by Π_{Daly}.

To better understand why cascade-aware algorithms achieve little gain, we come back to Equation(1). There are two sources of waste, one due to checkpointing overhead $\frac{C}{T}$, and one due to failures. The intuition is the following: when there are cascade failures, the work wasted remains low: in average, it is approximatively the MTBF of the degraded regimen. Additional (more frequent) checkpointing can reduce this waste, but can be an overkill too when there is no actual cascade.

Finally, recall that the MTBF μ_{log} of archive logs is approximately 10 hours, while it is 1 hour for synthetic logs. For the latter logs, using $\rho = 100$ means that $\mu_{degraded} = 36$ seconds, so there is little hope to gain anything except for $C = 3$ seconds. In that case, we do achieve some gain, up to 20%, but the waste was already low, around 5%, with the standard approach Π_{Daly}: overall, the absolute diminution of the waste reduces to 1%.

2) Assessing potential gain: Brute-force algorithms that search for the optimal MTBF in normal and degraded regimens allow us to quantify the potential gain that could be achieved with better cascade-aware algorithms. First, we observe that BI-Π-BEST, that supersedes both BI-$\Pi_{Intervals}$ and BI-$\Pi_{Quantiles}$, is not significantly better than Π_{Daly}. Second, we make a similar observation for BI-$\Pi_{Quantiles}$LAZY-BEST, that supersedes BI-$\Pi_{Quantiles}$LAZY without performing significantly better. Third, BI-Π-ORACLE-BEST and BI-$\Pi_{Quantiles}$ORACLE perform quite similarly, which is reassuring for the choice of

Log Id	Number of faults	MTBF in hours	Approach: Intervals MTBF in hours	Degraded intervals	Faults in cascades	Approach: Quantiles MTBF in hours	Faults in cascades	Common faults	Number of cascades	in Cascades MTBF	Avg. length	Max length
LANL 2	5351	14.14	36.45	25.3%	71.1%	14.92	8.9%	8.9%	194	0.102	2.4	10
LANL 16	2262	21.85	56.19	25.2%	70.9%	23.01	9.2%	9.2%	95	0.113	2.2	5
LANL 18	3900	7.52	17.88	26.0%	68.9%	7.92	9.7%	9.7%	180	0.119	2.1	4
LANL 19	3222	7.87	17.06	26.4%	66.0%	8.30	10.1%	10.0%	158	0.164	2.1	4
LANL 20	2389	13.66	41.46	21.3%	74.1%	14.43	10.0%	10.0%	110	0.093	2.2	5
Tsubame	884	14.78	36.50	23.9%	69.2%	15.57	9.7%	9.7%	41	0.100	2.1	4
Synth. $\rho = 10$ — 1.00 % — 3-5	3136	0.98	1.99	25.1%	63.1%	1.03	9.5%	9.5%	141	0.017	2.1	4
Synth. $\rho = 10$ — 1.00 % — 3-10	3182	0.95	1.98	25.0%	64.1%	1.00	9.8%	9.8%	153	0.015	2.0	3
Synth. $\rho = 10$ — 5.00 % — 3-5	3591	0.85	1.87	22.4%	64.9%	0.89	9.5%	9.5%	160	0.009	2.1	4
Synth. $\rho = 10$ — 5.00 % — 3-10	4075	0.75	1.77	20.5%	66.2%	0.79	9.6%	9.6%	186	0.006	2.1	4
Synth. $\rho = 10$ — 10.00 % — 3-5	4133	0.72	1.81	22.2%	69.0%	0.76	9.5%	9.4%	184	0.007	2.1	4
Synth. $\rho = 10$ — 10.00 % — 3-10	5067	0.59	1.65	19.5%	71.2%	0.62	9.5%	9.5%	229	0.004	2.1	4
Synth. $\rho = 100$ — 1.00 % — 3-5	3136	0.98	1.99	24.9%	62.9%	1.03	8.0%	8.0%	94	0.006	2.7	6
Synth. $\rho = 100$ — 1.00 % — 3-10	3182	0.95	1.98	24.6%	63.9%	1.00	7.9%	7.9%	91	0.005	2.8	8
Synth. $\rho = 100$ — 5.00 % — 3-5	3591	0.85	1.89	21.6%	64.8%	0.89	8.9%	8.9%	139	0.002	2.3	5
Synth. $\rho = 100$ — 5.00 % — 3-10	4075	0.75	1.79	18.3%	65.7%	0.79	9.4%	9.4%	179	0.001	2.1	4
Synth. $\rho = 100$ — 10.00 % — 3-5	4133	0.72	1.84	20.2%	68.7%	0.76	9.4%	9.4%	180	0.001	2.1	4
Synth. $\rho = 100$ — 10.00 % — 3-10	5067	0.59	1.69	15.8%	70.5%	0.62	9.6%	9.6%	231	0.001	2.1	4
Synth. $\rho = 1000$ — 1.00 % — 3-5	3136	0.98	1.99	24.9%	63.0%	1.03	6.7%	6.7%	54	0.001	3.9	6
Synth. $\rho = 1000$ — 1.00 % — 3-10	3182	0.95	1.98	24.6%	63.9%	1.00	6.4%	6.4%	45	0.001	4.6	11
Synth. $\rho = 1000$ — 5.00 % — 3-5	3591	0.85	1.89	21.3%	64.8%	0.89	8.9%	8.9%	138	0.000	2.3	5
Synth. $\rho = 1000$ — 5.00 % — 3-10	4075	0.75	1.79	18.1%	65.6%	0.79	9.4%	9.4%	178	0.000	2.1	4
Synth. $\rho = 1000$ — 10.00 % — 3-5	4133	0.72	1.83	19.9%	68.6%	0.76	9.4%	9.4%	180	0.000	2.1	4
Synth. $\rho = 1000$ — 10.00 % — 3-10	5067	0.59	1.70	15.3%	70.4%	0.62	9.6%	9.6%	231	0.000	2.1	4

Table V: Statistics when $|Q_{limit}| = 5\%$.

Table VI: Waste (and gain) with C=300s and $|Q_{limit}| = 5\%$.

Log	Periodic algorithms				Bi-periodic algorithms					Omniscient oracle	
	Π_{Daly}	$\Pi_{Intervals}$	$\Pi_{Quantiles}$	Π_{Best_period}	Bi-$\Pi_{Intervals}$	Bi-$\Pi_{Quantiles}$	Bi-Π-Best	Bi-$\Pi_{Quantiles}$Lazy	Bi-$\Pi_{Quantiles}$Lazy-Best	Bi-$\Pi_{Quantiles}$Oracle	Bi-Π-Oracle-Best
LANL 2	0.127	0.145 (-14.07%)	0.129 (-1.17%)	0.127 (0.00%)	0.143 (-11.84%)	0.135 (-5.67%)	0.128 (-0.16%)	0.129 (-1.52%)	0.128 (-0.16%)	0.129 (-1.01%)	0.127 (0.01%)
LANL 16	0.100	0.112 (-11.99%)	0.099 (0.90%)	0.100 (0.00%)	0.113 (-12.35%)	0.103 (-2.95%)	0.099 (0.90%)	0.100 (0.54%)	0.099 (0.90%)	0.099 (1.08%)	0.099 (1.08%)
LANL 18	0.175	0.190 (-8.46%)	0.176 (-0.03%)	0.175 (0.06%)	0.194 (-10.79%)	0.186 (-5.92%)	0.175 (0.12%)	0.176 (-0.30%)	0.175 (0.26%)	0.175 (0.18%)	0.175 (0.18%)
LANL 19	0.174	0.187 (-7.09%)	0.173 (0.81%)	0.171 (1.90%)	0.191 (-9.38%)	0.183 (-5.04%)	0.171 (2.08%)	0.174 (0.32%)	0.171 (2.02%)	0.172 (1.31%)	0.170 (2.43%)
LANL 20	0.119	0.137 (-15.45%)	0.118 (1.05%)	0.118 (0.35%)	0.145 (-22.40%)	0.123 (-3.53%)	0.118 (1.05%)	0.118 (0.79%)	0.117 (1.30%)	0.117 (1.25%)	0.117 (1.25%)
Tsubame	0.122	0.139 (-14.62%)	0.122 (-0.75%)	0.121 (0.26%)	0.139 (-14.71%)	0.128 (-5.13%)	0.121 (0.86%)	0.123 (-1.25%)	0.120 (1.43%)	0.122 (-0.49%)	0.120 (0.93%)
Synth. $\rho = 10$ — 1.00 % — 3-5	0.680	0.727 (-6.97%)	0.686 (-0.86%)	0.679 (0.15%)	0.763 (-12.25%)	0.686 (-0.86%)	0.678 (0.25%)	0.686 (-0.86%)	0.678 (0.25%)	0.682 (-0.28%)	0.674 (0.86%)
Synth. $\rho = 10$ — 1.00 % — 3-10	0.690	0.741 (-7.41%)	0.687 (0.32%)	0.688 (0.17%)	0.784 (-13.74%)	0.687 (0.32%)	0.686 (0.60%)	0.687 (0.32%)	0.686 (0.60%)	0.686 (0.46%)	0.684 (0.75%)
Synth. $\rho = 10$ — 5.00 % — 3-5	0.688	0.729 (-5.92%)	0.692 (-0.46%)	0.688 (0.00%)	0.856 (-24.29%)	0.692 (-0.46%)	0.692 (-0.46%)	0.692 (-0.46%)	0.692 (-0.46%)	0.685 (0.46%)	0.680 (1.27%)
Synth. $\rho = 10$ — 5.00 % — 3-10	0.735	0.751 (-2.19%)	0.729 (0.79%)	0.720 (2.03%)	0.954 (-29.74%)	0.729 (0.79%)	0.721 (1.95%)	0.729 (0.79%)	0.721 (1.95%)	0.707 (3.85%)	0.706 (3.96%)
Synth. $\rho = 10$ — 10.00 % — 3-5	0.756	0.790 (-4.50%)	0.760 (-0.51%)	0.745 (1.49%)	0.998 (-32.04%)	0.760 (-0.51%)	0.742 (1.88%)	0.760 (-0.51%)	0.742 (1.88%)	0.726 (3.94%)	0.726 (3.94%)
Synth. $\rho = 10$ — 10.00 % — 3-10	0.813	0.805 (0.95%)	0.798 (1.74%)	0.775 (4.68%)	1.170 (-44.01%)	0.798 (1.74%)	0.774 (4.71%)	0.798 (1.74%)	0.774 (4.71%)	0.746 (8.19%)	0.746 (8.19%)
Synth. $\rho = 100$ — 1.00 % — 3-5	0.676	0.722 (-6.82%)	0.681 (-0.79%)	0.673 (0.42%)	0.762 (-12.75%)	0.681 (-0.79%)	0.673 (0.51%)	0.681 (-0.79%)	0.673 (0.51%)	0.679 (-0.39%)	0.670 (0.97%)
Synth. $\rho = 100$ — 1.00 % — 3-10	0.681	0.735 (-7.94%)	0.679 (0.27%)	0.681 (0.01%)	0.781 (-14.75%)	0.679 (0.27%)	0.678 (0.35%)	0.679 (0.27%)	0.678 (0.35%)	0.679 (0.28%)	0.678 (0.35%)
Synth. $\rho = 100$ — 5.00 % — 3-5	0.663	0.704 (-6.30%)	0.667 (-0.69%)	0.663 (0.00%)	0.843 (-27.19%)	0.667 (-0.69%)	0.667 (-0.69%)	0.667 (-0.69%)	0.667 (-0.69%)	0.667 (-0.73%)	0.660 (0.43%)
Synth. $\rho = 100$ — 5.00 % — 3-10	0.684	0.705 (-3.02%)	0.681 (0.41%)	0.671 (1.87%)	0.941 (-37.58%)	0.681 (0.41%)	0.672 (1.76%)	0.681 (0.41%)	0.672 (1.76%)	0.671 (1.95%)	0.671 (1.95%)
Synth. $\rho = 100$ — 10.00 % — 3-5	0.697	0.732 (-5.01%)	0.703 (-0.84%)	0.690 (0.99%)	0.963 (-38.06%)	0.703 (-0.84%)	0.689 (1.16%)	0.703 (-0.84%)	0.689 (1.16%)	0.686 (1.59%)	0.684 (1.84%)
Synth. $\rho = 100$ — 10.00 % — 3-10	0.706	0.713 (-0.97%)	0.698 (1.11%)	0.677 (4.12%)	1.175 (-66.55%)	0.698 (1.11%)	0.677 (4.08%)	0.698 (1.11%)	0.677 (4.08%)	0.673 (4.67%)	0.673 (4.67%)
Synth. $\rho = 1000$ — 1.00 % — 3-5	0.675	0.722 (-6.88%)	0.681 (-0.92%)	0.673 (0.36%)	0.761 (-12.76%)	0.681 (-0.92%)	0.672 (0.49%)	0.681 (-0.92%)	0.672 (0.49%)	0.678 (-0.50%)	0.670 (0.84%)
Synth. $\rho = 1000$ — 1.00 % — 3-10	0.681	0.730 (-7.29%)	0.679 (0.25%)	0.680 (0.07%)	0.781 (-14.77%)	0.679 (0.25%)	0.678 (0.35%)	0.679 (0.25%)	0.678 (0.35%)	0.679 (0.26%)	0.678 (0.36%)
Synth. $\rho = 1000$ — 5.00 % — 3-5	0.662	0.704 (-6.44%)	0.664 (-0.32%)	0.662 (-0.00%)	0.841 (-27.14%)	0.664 (-0.32%)	0.664 (-0.32%)	0.664 (-0.32%)	0.664 (-0.32%)	0.664 (-0.36%)	0.659 (0.45%)
Synth. $\rho = 1000$ — 5.00 % — 3-10	0.681	0.700 (-2.84%)	0.677 (0.58%)	0.665 (2.31%)	0.942 (-38.45%)	0.677 (0.58%)	0.666 (2.21%)	0.677 (0.58%)	0.666 (2.21%)	0.665 (2.25%)	0.665 (2.29%)
Synth. $\rho = 1000$ — 10.00 % — 3-5	0.692	0.730 (-5.48%)	0.697 (-0.78%)	0.682 (1.42%)	0.960 (-38.83%)	0.697 (-0.78%)	0.681 (1.54%)	0.697 (-0.78%)	0.681 (1.54%)	0.676 (2.24%)	0.676 (2.24%)
Synth. $\rho = 1000$ — 10.00 % — 3-10	0.697	0.697 (-0.04%)	0.686 (1.60%)	0.669 (4.02%)	1.178 (-69.07%)	0.686 (1.60%)	0.669 (3.95%)	0.686 (1.60%)	0.669 (3.95%)	0.663 (4.84%)	0.663 (4.84%)
Avg.	0.557	0.585 (-6.53%)	0.557 (0.07%)	0.550 (1.11%)	0.724 (-26.48%)	0.558 (-1.14%)	0.550 (1.22%)	0.557 (-0.02%)	0.549 (1.25%)	0.547 (1.46%)	0.545 (1.92%)
Weighted Avg.	0.586	0.613 (-5.79%)	0.585 (0.12%)	0.577 (1.34%)	0.778 (-29.24%)	0.587 (-0.90%)	0.577 (1.39%)	0.585 (0.05%)	0.577 (1.41%)	0.573 (1.79%)	0.571 (2.19%)

Table VII: Waste (and gain) with C=30s and $|Q_{limit}| = 5\%$.

Log	Periodic algorithms				Bi-periodic algorithms					Omniscient oracle	
	Π_{Daly}	$\Pi_{Intervals}$	$\Pi_{Quantiles}$	Π_{Best_period}	Bi-$\Pi_{Intervals}$	Bi-$\Pi_{Quantiles}$	Bi-Π-Best	Bi-$\Pi_{Quantiles}$Lazy	Bi-$\Pi_{Quantiles}$Lazy-Best	Bi-$\Pi_{Quantiles}$Oracle	Bi-Π-Oracle-Best
LANL 2	0.037	0.042 (-13.64%)	0.038 (-0.86%)	0.037 (0.49%)	0.041 (-10.60%)	0.040 (-5.91%)	0.037 (0.40%)	0.038 (-1.54%)	0.037 (0.75%)	0.037 (0.06%)	0.037 (1.29%)
LANL 16	0.030	0.033 (-11.25%)	0.030 (0.04%)	0.030 (0.67%)	0.033 (-10.43%)	0.030 (-1.89%)	0.029 (2.34%)	0.030 (0.92%)	0.029 (1.45%)	0.030 (1.02%)	0.029 (1.73%)
LANL 18	0.050	0.055 (-9.85%)	0.049 (0.52%)	0.050 (0.10%)	0.054 (-9.13%)	0.054 (-8.43%)	0.049 (1.40%)	0.050 (-0.02%)	0.049 (1.02%)	0.049 (2.16%)	0.049 (2.32%)
LANL 19	0.049	0.052 (-6.11%)	0.048 (1.99%)	0.049 (1.08%)	0.053 (-6.93%)	0.052 (-5.90%)	0.048 (2.46%)	0.049 (1.59%)	0.048 (2.49%)	0.047 (4.45%)	0.047 (4.55%)
LANL 20	0.035	0.040 (-13.47%)	0.035 (0.17%)	0.035 (1.12%)	0.042 (-20.95%)	0.037 (-4.93%)	0.034 (2.33%)	0.035 (0.03%)	0.035 (1.50%)	0.035 (1.14%)	0.035 (1.14%)
Tsubame	0.035	0.040 (-15.42%)	0.037 (-5.64%)	0.035 (-0.00%)	0.040 (-15.53%)	0.037 (-6.30%)	0.035 (0.32%)	0.037 (-6.65%)	0.035 (0.83%)	0.036 (-4.46%)	0.034 (1.56%)
Synth. $\rho = 10$ — 1.00 % — 3-5	0.150	0.158 (-5.72%)	0.152 (-1.22%)	0.149 (0.23%)	0.167 (-11.57%)	0.157 (-4.74%)	0.149 (0.79%)	0.152 (-1.13%)	0.148 (1.18%)	0.148 (1.05%)	0.147 (1.71%)
Synth. $\rho = 10$ — 1.00 % — 3-10	0.154	0.162 (-5.38%)	0.154 (-0.09%)	0.152 (0.81%)	0.172 (-12.20%)	0.162 (-5.71%)	0.153 (0.54%)	0.153 (0.11%)	0.152 (0.74%)	0.150 (2.01%)	0.150 (2.58%)
Synth. $\rho = 10$ — 5.00 % — 3-5	0.158	0.164 (-3.71%)	0.159 (-0.49%)	0.158 (0.00%)	0.188 (-18.71%)	0.168 (-6.02%)	0.158 (-0.12%)	0.160 (-0.95%)	0.158 (0.18%)	0.151 (4.68%)	0.148 (6.28%)
Synth. $\rho = 10$ — 5.00 % — 3-10	0.168	0.177 (-5.44%)	0.167 (0.33%)	0.166 (1.33%)	0.205 (-21.92%)	0.167 (0.33%)	0.167 (0.40%)	0.167 (0.33%)	0.167 (0.83%)	0.152 (9.44%)	0.151 (9.92%)
Synth. $\rho = 10$ — 10.00 % — 3-5	0.177	0.184 (-4.27%)	0.175 (1.32%)	0.172 (2.59%)	0.210 (-18.70%)	0.175 (1.32%)	0.173 (2.22%)	0.175 (1.32%)	0.172 (3.02%)	0.154 (13.18%)	0.153 (13.57%)
Synth. $\rho = 10$ — 10.00 % — 3-10	0.191	0.205 (-7.66%)	0.190 (0.53%)	0.190 (0.66%)	0.244 (-27.87%)	0.190 (0.53%)	0.189 (1.16%)	0.190 (0.53%)	0.187 (2.20%)	0.154 (19.02%)	0.154 (19.02%)
Synth. $\rho = 100$ — 1.00 % — 3-5	0.148	0.156 (-5.59%)	0.150 (-1.42%)	0.148 (0.12%)	0.166 (-12.13%)	0.150 (-1.42%)	0.146 (0.88%)	0.150 (-1.42%)	0.146 (0.88%)	0.148 (-0.10%)	0.147 (0.47%)
Synth. $\rho = 100$ — 1.00 % — 3-10	0.150	0.158 (-5.50%)	0.150 (0.19%)	0.149 (0.74%)	0.171 (-13.60%)	0.150 (0.19%)	0.149 (0.75%)	0.150 (0.19%)	0.149 (0.75%)	0.150 (0.27%)	0.149 (0.83%)
Synth. $\rho = 100$ — 5.00 % — 3-5	0.149	0.153 (-2.71%)	0.150 (-0.64%)	0.148 (0.43%)	0.184 (-23.43%)	0.150 (-0.64%)	0.148 (0.62%)	0.150 (-0.64%)	0.148 (0.62%)	0.149 (-0.17%)	0.147 (1.26%)
Synth. $\rho = 100$ — 5.00 % — 3-10	0.152	0.156 (-2.26%)	0.152 (0.12%)	0.150 (1.46%)	0.202 (-32.72%)	0.152 (0.12%)	0.150 (1.77%)	0.152 (0.12%)	0.150 (1.77%)	0.150 (1.93%)	0.150 (1.93%)
Synth. $\rho = 100$ — 10.00 % — 3-5	0.159	0.159 (-0.28%)	0.156 (2.22%)	0.153 (4.01%)	0.206 (-29.60%)	0.156 (2.22%)	0.153 (3.94%)	0.156 (2.22%)	0.153 (3.94%)	0.150 (5.41%)	0.150 (5.41%)
Synth. $\rho = 100$ — 10.00 % — 3-10	0.161	0.163 (-1.07%)	0.160 (0.79%)	0.153 (4.81%)	0.244 (-51.34%)	0.160 (0.79%)	0.152 (5.80%)	0.160 (0.79%)	0.152 (5.80%)	0.149 (7.20%)	0.149 (7.20%)
Synth. $\rho = 1000$ — 1.00 % — 3-5	0.147	0.156 (-5.85%)	0.149 (-1.22%)	0.147 (0.25%)	0.166 (-12.30%)	0.149 (-1.22%)	0.146 (0.79%)	0.149 (-1.22%)	0.146 (0.79%)	0.148 (-0.17%)	0.147 (0.44%)
Synth. $\rho = 1000$ — 1.00 % — 3-10	0.149	0.158 (-5.77%)	0.149 (0.01%)	0.148 (0.62%)	0.170 (-13.97%)	0.149 (0.01%)	0.149 (0.56%)	0.149 (0.01%)	0.149 (0.56%)	0.149 (0.03%)	0.149 (0.58%)
Synth. $\rho = 1000$ — 5.00 % — 3-5	0.147	0.150 (-2.02%)	0.148 (-0.69%)	0.146 (0.80%)	0.182 (-23.50%)	0.148 (-0.69%)	0.147 (0.47%)	0.148 (-0.69%)	0.147 (0.47%)	0.148 (-0.50%)	0.145 (1.31%)
Synth. $\rho = 1000$ — 5.00 % — 3-10	0.149	0.153 (-2.50%)	0.149 (0.04%)	0.147 (1.35%)	0.200 (-33.82%)	0.149 (0.04%)	0.147 (1.59%)	0.149 (0.04%)	0.147 (1.59%)	0.147 (1.26%)	0.147 (1.43%)
Synth. $\rho = 1000$ — 10.00 % — 3-5	0.155	0.155 (-0.10%)	0.152 (2.05%)	0.149 (3.77%)	0.202 (-30.50%)	0.152 (2.05%)	0.149 (4.08%)	0.152 (2.05%)	0.149 (4.08%)	0.148 (4.37%)	0.148 (4.75%)
Synth. $\rho = 1000$ — 10.00 % — 3-10	0.154	0.156 (-1.28%)	0.153 (0.90%)	0.147 (4.44%)	0.239 (-55.42%)	0.153 (0.90%)	0.146 (5.30%)	0.153 (0.90%)	0.146 (5.30%)	0.145 (5.82%)	0.145 (5.82%)
Avg.	0.127	0.133 (-5.70%)	0.127 (-0.04%)	0.125 (1.33%)	0.158 (-21.54%)	0.129 (-1.89%)	0.125 (1.70%)	0.127 (-0.13%)	0.125 (1.78%)	0.122 (3.30%)	0.121 (4.05%)
Weighted Avg.	0.134	0.139 (-5.15%)	0.133 (0.20%)	0.131 (1.53%)	0.168 (-23.36%)	0.135 (-1.55%)	0.131 (1.87%)	0.133 (0.12%)	0.131 (1.98%)	0.127 (4.08%)	0.126 (4.64%)

Table VIII: Waste (and gain) with C=3s and $|Q_{limit}| = 5\%$.

Log	Periodic algorithms				Bi-periodic algorithms					Omniscient oracle	
	Π_{Daly}	$\Pi_{Intervals}$	$\Pi_{Quantiles}$	Π_{Best_period}	Bi-$\Pi_{Intervals}$	Bi-$\Pi_{Quantiles}$	Bi-Π-Best	Bi-$\Pi_{Quantiles}$Lazy	Bi-$\Pi_{Quantiles}$Lazy-Best	Bi-$\Pi_{Quantiles}$Oracle	Bi-Π-Oracle-Best
LANL 2	0.012	0.013 (-11.94%)	0.011 (0.21%)	0.011 (0.83%)	0.013 (-10.41%)	0.012 (-5.73%)	0.011 (1.35%)	0.011 (0.22%)	0.011 (1.29%)	0.011 (2.12%)	0.011 (2.41%)
LANL 16	0.009	0.011 (-14.17%)	0.009 (-0.07%)	0.009 (0.54%)	0.010 (-11.20%)	0.010 (-5.63%)	0.009 (1.40%)	0.009 (0.07%)	0.009 (1.33%)	0.009 (1.98%)	0.009 (2.97%)
LANL 18	0.015	0.017 (-10.27%)	0.015 (0.42%)	0.015 (0.00%)	0.016 (-9.06%)	0.016 (-8.00%)	0.015 (2.74%)	0.015 (-0.12%)	0.015 (1.79%)	0.015 (2.67%)	0.015 (3.54%)
LANL 19	0.015	0.016 (-8.82%)	0.015 (0.28%)	0.015 (1.05%)	0.016 (-8.50%)	0.016 (-10.12%)	0.015 (1.95%)	0.015 (-0.35%)	0.015 (0.95%)	0.014 (2.99%)	0.014 (3.32%)
LANL 20	0.011	0.012 (-13.00%)	0.011 (0.99%)	0.011 (1.97%)	0.013 (-19.57%)	0.011 (-4.22%)	0.010 (3.00%)	0.011 (0.50%)	0.011 (2.07%)	0.010 (3.05%)	0.010 (3.05%)
Tsubame	0.011	0.012 (-7.71%)	0.011 (-0.29%)	0.011 (4.63%)	0.012 (-11.97%)	0.012 (-6.89%)	0.011 (3.16%)	0.011 (-1.44%)	0.011 (3.36%)	0.011 (1.98%)	0.011 (5.33%)
Synth. $\rho = 10$ — 1.00 % — 3-5	0.043	0.045 (-4.94%)	0.043 (0.03%)	0.043 (-0.00%)	0.048 (-11.18%)	0.047 (-8.41%)	0.043 (0.90%)	0.043 (-0.63%)	0.043 (0.99%)	0.043 (0.85%)	0.042 (2.72%)
Synth. $\rho = 10$ — 1.00 % — 3-10	0.044	0.046 (-5.49%)	0.044 (-0.67%)	0.044 (0.45%)	0.049 (-11.10%)	0.047 (-6.36%)	0.043 (1.10%)	0.044 (-1.10%)	0.043 (1.29%)	0.043 (2.07%)	0.042 (3.46%)
Synth. $\rho = 10$ — 5.00 % — 3-5	0.045	0.049 (-8.40%)	0.045 (0.22%)	0.045 (0.16%)	0.053 (-15.87%)	0.049 (-7.95%)	0.045 (1.79%)	0.046 (-0.92%)	0.045 (1.01%)	0.042 (7.75%)	0.042 (8.41%)
Synth. $\rho = 10$ — 5.00 % — 3-10	0.048	0.052 (-8.30%)	0.048 (-0.13%)	0.048 (0.37%)	0.058 (-20.35%)	0.050 (-5.20%)	0.047 (1.20%)	0.048 (-0.66%)	0.047 (1.18%)	0.042 (12.02%)	0.042 (12.39%)
Synth. $\rho = 10$ — 10.00 % — 3-5	0.050	0.055 (-9.60%)	0.050 (-0.11%)	0.050 (0.05%)	0.059 (-18.86%)	0.053 (-5.20%)	0.049 (1.83%)	0.050 (-0.63%)	0.049 (1.45%)	0.042 (15.22%)	0.042 (15.22%)
Synth. $\rho = 10$ — 10.00 % — 3-10	0.054	0.061 (-11.63%)	0.055 (-0.69%)	0.054 (0.00%)	0.068 (-24.08%)	0.057 (-4.02%)	0.053 (1.90%)	0.055 (-0.92%)	0.054 (0.79%)	0.043 (20.70%)	0.043 (21.94%)
Synth. $\rho = 100$ — 1.00 % — 3-5	0.043	0.045 (-4.56%)	0.043 (0.07%)	0.043 (0.00%)	0.048 (-11.46%)	0.045 (-5.20%)	0.042 (0.42%)	0.043 (-0.11%)	0.042 (1.01%)	0.043 (0.01%)	0.042 (1.93%)
Synth. $\rho = 100$ — 1.00 % — 3-10	0.043	0.045 (-4.71%)	0.043 (-0.80%)	0.043 (0.61%)	0.049 (-13.07%)	0.045 (-3.47%)	0.043 (0.68%)	0.043 (-0.73%)	0.043 (1.17%)	0.043 (0.52%)	0.042 (1.94%)
Synth. $\rho = 100$ — 5.00 % — 3-5	0.044	0.045 (-4.09%)	0.044 (0.28%)	0.043 (0.51%)	0.052 (-19.78%)	0.044 (-1.47%)	0.044 (0.28%)	0.043 (0.33%)	0.043 (1.20%)	0.042 (3.93%)	0.042 (4.68%)
Synth. $\rho = 100$ — 5.00 % — 3-10	0.045	0.047 (-4.55%)	0.045 (0.55%)	0.044 (2.06%)	0.058 (-28.60%)	0.046 (-0.90%)	0.045 (1.53%)	0.045 (0.66%)	0.044 (3.24%)	0.042 (7.34%)	0.042 (7.34%)
Synth. $\rho = 100$ — 10.00 % — 3-5	0.047	0.049 (-4.26%)	0.047 (-0.09%)	0.046 (1.28%)	0.059 (-26.69%)	0.047 (-1.42%)	0.046 (1.88%)	0.047 (0.21%)	0.045 (3.48%)	0.043 (8.97%)	0.042 (9.25%)
Synth. $\rho = 100$ — 10.00 % — 3-10	0.050	0.050 (-0.19%)	0.050 (-0.18%)	0.048 (3.13%)	0.070 (-41.47%)	0.050 (-0.18%)	0.047 (4.71%)	0.050 (-0.18%)	0.047 (5.47%)	0.043 (13.32%)	0.043 (14.11%)
Synth. $\rho = 1000$ — 1.00 % — 3-5	0.042	0.044 (-4.12%)	0.042 (-0.18%)	0.042 (-0.00%)	0.047 (-11.95%)	0.043 (-1.72%)	0.042 (0.09%)	0.042 (-0.19%)	0.042 (0.53%)	0.043 (-0.85%)	0.042 (1.18%)
Synth. $\rho = 1000$ — 1.00 % — 3-10	0.043	0.045 (-5.02%)	0.043 (-0.72%)	0.042 (0.60%)	0.048 (-13.50%)	0.043 (-0.72%)	0.042 (0.78%)	0.043 (-0.72%)	0.042 (0.79%)	0.043 (-0.65%)	0.042 (0.79%)
Synth. $\rho = 1000$ — 5.00 % — 3-5	0.042	0.044 (-3.61%)	0.042 (0.39%)	0.042 (0.63%)	0.051 (-21.67%)	0.042 (0.39%)	0.042 (0.39%)	0.042 (0.39%)	0.042 (0.45%)	0.042 (1.13%)	0.041 (1.81%)
Synth. $\rho = 1000$ — 5.00 % — 3-10	0.043	0.044 (-3.80%)	0.042 (0.46%)	0.042 (2.19%)	0.057 (-32.79%)	0.042 (0.46%)	0.042 (1.75%)	0.042 (0.46%)	0.042 (1.75%)	0.042 (1.98%)	0.042 (1.98%)
Synth. $\rho = 1000$ — 10.00 % — 3-5	0.044	0.045 (-4.02%)	0.044 (0.23%)	0.043 (2.12%)	0.057 (-30.84%)	0.044 (0.23%)	0.043 (2.37%)	0.044 (0.23%)	0.043 (2.37%)	0.042 (3.24%)	0.042 (3.24%)
Synth. $\rho = 1000$ — 10.00 % — 3-10	0.045	0.044 (0.96%)	0.044 (0.53%)	0.042 (4.95%)	0.068 (-51.43%)	0.044 (0.53%)	0.043 (4.28%)	0.044 (0.53%)	0.043 (4.28%)	0.043 (3.89%)	0.042 (4.85%)
Avg.	0.037	0.039 (-6.51%)	0.037 (0.03%)	0.037 (1.17%)	0.045 (-19.81%)	0.038 (-3.80%)	0.036 (1.73%)	0.037 (-0.21%)	0.036 (1.80%)	0.035 (4.84%)	0.034 (5.74%)
Weighted Avg.	0.039	0.041 (-6.26%)	0.039 (0.04%)	0.038 (1.16%)	0.048 (-21.43%)	0.040 (-3.49%)	0.038 (1.80%)	0.039 (-0.16%)	0.038 (1.88%)	0.036 (5.59%)	0.036 (6.37%)

$\mu_{non-cascade}(Q)$ as the MTBF in normal regimen. Overall, the results show that a significant gain is possible only for the latter two algorithms equipped with an omniscient oracle: when entering a cascade, BI-Π-ORACLE-BEST and BI-$\Pi_{Quantiles}$ORACLE checkpoint right on time before the failures. Even so, one can see that the gains are very limited.

The only case where the gain becomes larger is when (i) the degraded MTBF is not too small in front of the log MTBF, i.e., when $\rho = 10$, and (ii) the proportion of failures that turn out to be actual cascade failures is large, i.e., when $f = 10\%$. In that case, the maximal potential gain is 8% with $C = 300$, 19% with $C = 30$ and 22% with $C = 3$ (the latter two values leading to small absolute gains, because the waste with Π_{Daly} is small in these cases). However, this scenario may not be realistic, because (i) cascade failures, as their name merely indicates, are expected to strike within a few instants, hence $\mu_{degraded}$ will likely be less than one minute; and (ii) cascades are expected to be (more or less) rare events, so experiencing large values of f will probably never happen in practice.

VI. CONCLUSION

In this paper, we have revisited failure temporal independence. Recent work [4] has proposed a cascade-detection method, and we have shown that their approach was inconclusive. Then we have introduced a new approach based on pairs of consecutive IATs, and we have been able to put in evidence the presence of cascade failures. A few publicly available failure logs do contain cascades.

In a second step, we have discussed the usefulness of cascade-aware checkpointing algorithms. For this, we have used both public and synthetic logs. We used the latter to explicitly create "artificial" cascades. We have shown that current cascade-aware bi-periodic checkpointing algorithms are not really more efficient than the standard periodic checkpointing approach that considers failures to be independent. Finally, by using a brute-force search over all possible bi-periodic algorithms and considering omniscient oracles that know exactly when cascade failures will strike, we have shown that only insignificant gain should be expected from designing future cascade-aware checkpointing algorithms. The conclusion is that we can wrongly, but safely, assume failure independence!

ACKNOWLEDGEMENTS

We thank Leonardo Bautista-Gomez, Paolo Gonçalves and Arnaud Legrand for useful discussions.

REFERENCES

[1] A. Anderson and D. Semmelroth. *Statistics for Big Data For Dummies*. For Dummies, 2015.

[2] G. Aupy, Y. Robert, and F. Vivien. Assuming failure independence: are we right to be wrong? Research report RR-9078, INRIA, 2017.

[3] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni. Checkpointing algorithms and fault prediction. *Journal of Parallel and Distributed Computing*, 74(2):2048–2064, 2014.

[4] L. Bautista-Gomez, A. Gainaru, S. Perarnau, D. Tiwari, S. Gupta, C. Engelmann, F. Cappello, and M. Snir. Reducing waste in extreme scale systems through introspective analysis. In *IPDPS*, pages 212–221. IEEE, 2016.

[5] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka. FTI: High performance fault tolerance interface for hybrid systems. In *Proc. SC'11*, 2011.

[6] G. Bosilca, A. Bouteiller, E. Brunet, F. Cappello, J. Dongarra, A. Guermouche, T. Herault, Y. Robert, F. Vivien, and D. Zaidouni. Unified model for assessing checkpointing protocols at extreme-scale. *Concurrency and Computation: Practice and Experience*, 26(17):2772–2791, 2014.

[7] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.

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

[9] A. Gainaru, F. Cappello, M. Snir, and W. Kramer. Fault prediction under the microscope: A closer look into hpc systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 77. IEEE Computer Society Press, 2012.

[10] E. Gelenbe and M. Hernández. Optimum checkpoints with age dependent failures. *Acta Informatica*, 27(6):519–531, 1990.

[11] 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 *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 37–44. IEEE, 2015.

[12] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello. Modeling and tolerating heterogeneous failures in large parallel systems. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–11. IEEE, 2011.

[13] T. Hérault and Y. Robert, editors. *Fault-Tolerance Techniques for High-Performance Computing*, Computer Communications and Networks. Springer Verlag, 2015.

[14] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta. Making cloud intermediate data fault-tolerant. In *Proc. 1st ACM Symposium on Cloud Computing*, SoCC '10. ACM, 2010.

[15] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. *Cluster Computing and the Grid, IEEE International Symposium on*, pages 398–407, 2010.

[16] LANL. Computer failure data repository. https://www.usenix.org/cfdr-data, 2006.

[17] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. Scott. An optimal checkpoint/restart model for a large scale high performance computing system. In *IPDPS'08*. IEEE, 2008.

[18] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, pages 1–11, 2010.

[19] X. Ni, E. Meneses, and L. V. Kalé. Hiding checkpoint overhead in HPC applications with a semi-blocking algorithm. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 364–372. IEEE Computer Society, 2012.

[20] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proc. of DSN*, pages 249–258, 2006.

[21] K. Schroiff, P. Gemsjaeger, and C. Bolik. Cascading failover of a data management application for shared disk file systems in loosely coupled node clusters, 2006. US Patent 6,990,606.

[22] Y. A. Shardt. *Statistics for chemical and process engineers : a modern approach*. Springer, 2015.

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

[24] Tsubame. Failure history. http://mon.g.gsic.titech.ac.jp/trouble-list/index.htm, 2017.

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

[26] G. Zheng, L. Shi, and L. V. Kale. FTC-Charm++: an in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI. In *Cluster Computing, 2004 IEEE International Conference on*, pages 93–103. IEEE Computer Society, 2004.