# On the Optimum Checkpointing Interval Selection for Variable Size Checkpoint Dumps

Samy Sadi and Belabbas Yagoubi

University of Oran1 Ahmed Benbella
Department of Computer Science
Oran, Algeria
{samy.sadi.contact,byagoubi}@gmail.com

**Abstract.** Checkpointing is a technique that is often employed for granting fault tolerance for applications executing in failure-prone environments. It consists on regularly saving the application's state in another and fault independent storage such that if the application fails, it can be continued without necessarily restarting it. In this context, fixing the checkpointing frequency is an important topic which we address in this paper. We particularly address this issue considering hybrid fault tolerance and variable size checkpoint dumps. We then evaluate our solution and compare it with state of the art models, and show that our solution brings better results.

**Keywords:** Optimum Checkpointing Interval, Hybrid Fault Tolerance, Variable Size, Simulation

## 1 Introduction

Since the accession to information technologies, a lot of efforts have been devoted by the research community in order to make computing systems more fault-tolerant and more reliable. Initially, reliability was chiefly sought to avoid job resubmissions and to lower resource utilization, in a context where the job average length is ever-growing especially after the emergence of computational science and high-performance computing (HPC). Afterwards, and notably due to the advent of Cloud Computing and due to the increasing number of business-sensitive applications, a new practical and financial dimension appeared which drew even more attention to reliability.

The fact is that computing systems are failure-prone and failures are getting more frequent as new systems appear. The reason behind this is not because their components are getting less reliable. Actually, future hardware components and in particular newer generation chips are expected to keep failure rates similar to those of the current generation [14]. However, the number of components per any single system has considerably increased since the last few years and is continually increasing which led to a lower overall system mean time between failures (MTBF). So as to emphasize these lines, a study [2] on large-scale HPC systems has observed an MTBF in the 6.5h–40h range. This value when extrapolated for

a peta-scale system, corresponds to a MTBF of only 1.25 hours [11]. Latterly, another study [16] has observed that in a typical Cloud datacenter, a proportion of 8% of the machines can expect to see at least one failure each year.

In this context, checkpoint-restart or checkpointing has been developed in order to leverage fault tolerance in computing systems. This technique consists on taking frequent snapshots of any job's state, and on saving it on a secondary and fault-independent machine [5]. When the job fails on its primary machine, the saved state on the secondary machine is used to restore the job's state and to continue it. Thus, the job does not need to restart from scratch and a lower execution delay can be expected in failure-prone environments.

A central concern when implementing such fault tolerance technique is about selecting the frequency of checkpointing or the delay between taking two checkpoints. A high checkpointing frequency will ensure to have at any moment a very recent snapshot of the job's state, thus minimizing the potential rework delay after a failure. But in the same time, this will induce a significant overhead to the job execution if the occurrence of failures is very low or nonexistent. In another hand, a too low checkpointing frequency is obviously not a wise choice either, as this will lead to a noteworthy rework delay after failures.

In the present study, the issue under scrutiny is precisely about determining the best checkpointing frequency, also known as the checkpointing interval selection problem. We particularly address this problem considering a variable checkpointing overhead. We assume that the checkpointing overhead, or the time which is necessary to save a job's state into an external device is a function of the previous computing phase's delay. Besides, as new research has been undertaken for hybrid fault tolerance and in particular to predict the occurrence of failures with fair results [7,12,13], we also take into consideration this aspect in order to reduce checkpointing frequency.

The organization of this paper is as follows. In the next section, we discuss related work. In section 3, we define and formalize the problem and we present the brought solution. In the penultimate section, we evaluate our solution and we discuss the results. Finally, in section 5 we conclude and we give an overview of our future work.

## 2 Related Work

Checkpointing and in particular checkpointing interval selection has been extensively studied in the past. In this section, we give a chronological review of most prominent research efforts in the literature.

One of the first contributions was made by Young [17] who managed to give a first order approximation to the optimum checkpointing interval. Young considered that the overhead which is due to checkpointing is (1) constant and independent from the computing phase, and (2) is negligible when compared to the system MTBF. Furthermore, failures occurrence was assumed to be independent and exponentially distributed following a known MTBF. This last assumption, even if adopted in many contributions [4,6,8,15,17,18], is only true

for the first occurrence of the failures. In fact, failures cluster in time and a failure is more likely to happen after a first failure [16].

In [6], the author showed that the optimum checkpointing interval is deterministic and is a function of the system load. In particular, the author proposed a queuing model where the duration of service interruptions is directly computable knowing the past history of the system.

An $O(n^3)$ algorithm has been proposed in [15] to select the $(n-1)$ potential checkpoint locations which can minimize a given job's execution time. To do so, the authors considered that a job consists of a set of tasks and that checkpoints can only be taken between two consecutive tasks (and not during the task's execution). As opposed to the so far presented contributions, in this contribution authors assumed that checkpointing overhead is not constant and is task-dependant.

An aperiodic checkpointing approach where the checkpointing interval is varying from one checkpoint to another has been proposed in [9]. The authors considered a general failure-rate and no assumption was made regarding the distribution of the failures. Nevertheless, this research specifies that when the distribution of failures is exponential, the optimum placement of checkpoints is equidistant.

A higher order estimate of the optimum checkpointing interval has been provided in [4]. Daly has undertaken to continue the groundwork initiated by Young in [17], and kept most of his assumptions particularly regarding the checkpointing overhead and the failures distribution. However, Daly generalized Young's solution considering the case where the checkpointing overhead is not negligible compared to the system's MTBF.

In [10], the authors proposed an approach for checkpoint placement under incomplete failures information and when the failures distribution is unknown. The min-max principle has been employed to this extent.

An hybrid fault tolerance approach has been proposed in [8]. In this approach, it is assumed that the system has fault-prediction capabilities [7,12,13] which can be used to reduce checkpointing frequency. The authors proposed to partition the job's execution using a preset time interval. At each interval, a decision stage takes place where (1) the job is checkpointed, (2) the job is migrated or (3) no action is taken.

The checkpointing scheduling complexity has been analyzed in [3]. In this research, no assumption was made regarding failures distribution, and checkpointing overhead was assumed to be variable. The authors stated that the checkpointing problem is NP-hard even in the simple case where the failures distribution is uniform. In addition, a dynamic programming algorithm has been proposed to solve the problem.

In [18], the authors exploited failures prediction capabilities in order to define a new formula for computing checkpointing interval. Two main metrics were considered, namely the precision and the recall of failures prediction. These two metrics respectively characterize the capacity of the system to not make false predictions of failures, and the capacity of the system to predict all future

failures. We also consider these two metrics in current paper, but we consider variable checkpointing overhead.

## 3 The Checkpointing Interval Model

In this section, we develop our model for estimating the optimum checkpointing interval considering a variable checkpointing overhead. We draw our inspiration in the model proposed by Young [17] and later used in many other research [4,18].

In the next lines, we first describe the checkpointing process in both the situations where a failures prediction mechanism is employed or not. Then, we give the cost function we want to optimize. Next, and before solving the equation we place some assumptions regarding the failures distribution and the checkpointing overhead function. Once the assumptions placed, we solve the cost function and the optimum checkpointing interval is quantified. Finally, we address a special issue as regards to if the checkpointing overhead function is bounded.

The main symbols used in this paper are described in table 1.

**Table 1.** Description of used symbols

| Symbol | Description |
|---|---|
| $t$ | Checkpointing interval. |
| $\delta(t)$ | Checkpointing overhead, or the length of the save phase considering $t$ is the length of the compute phase. |
| $\delta_{max}$ | The maximum length of the checkpointing overhead. |
| $\alpha$ and $\beta$ | Values characterizing the job's checkpointing overhead. |
| $R$ | Restart delay before a job can continue on a secondary machine. |
| $C$ | Total number of real failures during job execution. |
| $C_{tp}$ | Total number of failures that were predicted (true positives). |
| $C_{fp}$ | Total number of failures that were predicted but will actually not happen (false positives). |
| $C_{fn}$ | Total number of failures that were not predicted (false negatives). |
| $p$ | Precision value in the $[0, 1]$ range for failures prediction. |
| $r$ | Recall value in the $[0, 1]$ range for failures prediction. |
| $n_i(t)$ | Number of successful computing phases between the $(i-1)^{th}$ and the $i^{th}$ failure event. |
| $w_i(t)$ | Last computing phase's delay just before the $i^{th}$ failure event happens. |
| $l_i(t)$ | Time lost due to the $i^{th}$ failure event. |
| $L(t)$ | Time lost due to checkpointing and considering all failure events. |
| $S$ | The submitted job length, or the execution time needed by the job to complete. |
| $M$ | The mean time between failures (MTBF). |
| $t_{opt}$ | Optimum Checkpointing interval. |

### 3.1 The Fundamental Checkpointing Process

The checkpointing process consists of multiple sequences of a computing phase where the job is normally executing followed by a save phase where the job's state is written to an external storage. During each save phase the job execution is paused and the job continues in the next computing phase.

The checkpointing interval $t$ represents the length of the computing phase which is also the delay between two consecutive save phases. In this paper, we assume that the save phases are equidistant and that the checkpointing interval stays unchanged during all the job's execution.

During a job's execution one or more failures may happen. After each failure, a restart delay $R$ is necessary for the job to be continued. Moreover, there is an added rework delay $w_i(t)$ corresponding to the not saved job progress due to the $i^{th}$ failure. Refer to Fig.1 for an overview of a job's execution in a context where checkpointing is used.

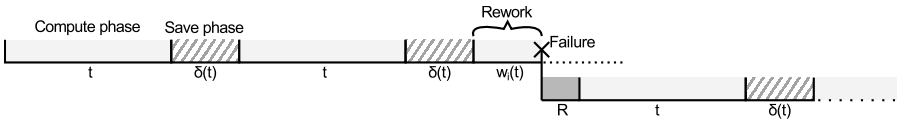


**Fig. 1.** The Checkpointing Process

### 3.2 The Hybrid Checkpointing Process

As previously discussed, there is a rapidly growing literature on failures prediction techniques. These techniques can be employed on top of periodic checkpointing in order to trigger additional save phases when a failure is predicted. After the save phase, the job is immediately continued on the secondary machine and no rework delay is necessary as the job state on the secondary machine is up to date. Another important feature of such process, is that checkpointing frequency can be reduced. The hybrid checkpointing process is displayed in Fig.2.

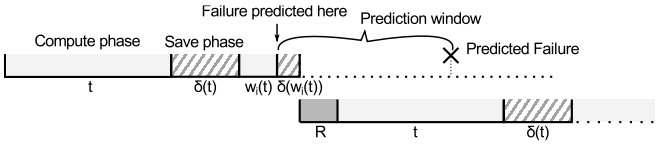


**Fig. 2.** The Hybrid Checkpointing Process

The level of trust of failures prediction is characterized by three different metrics, namely: the precision($p$), the recall($r$) and the prediction window.

The precision is the fraction of true positive predictions ($C_{tp}$) that are made by the system when compared to all the predictions made. In fact, the system might do an erroneous prediction of a future failure. These erroneous predictions are also designated as false positives ($C_{fp}$).

$$p = \frac{C_{tp}}{C_{tp} + C_{fp}} \tag{1}$$

The recall value represents the ability of the system to predict all future failures. It is the fraction of true positive predictions on the total number of failures. The missed failure predictions are designated as false negatives ($C_{fn}$) and is used when computing the recall:

$$r = \frac{C_{tp}}{C_{tp} + C_{fn}} \tag{2}$$

The prediction window is the time interval which is left before a predicted failure happens. A good prediction window is big enough so that a save phase can be engaged and completed before the failure happens. If the prediction window is too small, then the failure prediction is pointless as the save phase will not complete. Such predictions are consider to be false negatives in the current paper.

### 3.3 Cost Function

We define our cost function by considering the time lost due to checkpointing in different scenarios. We start by considering the time lost due to at most one single failure event. After that, we generalize the results to more than one failure event. A failure event is either an actual failure happening, or a failure prediction event.

**Single Failure Event Cost Function** We identify four cases as regards to the time lost in the checkpointing process when compared to a job executing in a regular and failure immune system. We present the cost functions for each case. We assume in those that $n_i(t)$ is the number of successful computing phases between the $(i-1)^{th}$ and the $i^{th}$ failure event.

The first case is when no failure events happen and the job executes normally on the primary machine. In this regard, the time lost is the sum of the delays spent in all the save phases.

$$l_{i,1}(t) = n_i(t) \cdot \delta(t) \tag{3}$$

The second case is when an unpredicted failure happens. Hereof, the time lost consists of all the save phases plus a restart delay ($R$) and eventually some rework time ($w_i(t)$) which has not yet been saved.

$$l_{i,2}(t) = n_i(t) \cdot \delta(t) + R + w_i(t) \tag{4}$$

The third case is when the system predicts the occurrence of a failure that will really happen (ie: a true positive). In this case, the time lost consists of all the save phases plus a restart delay. The last save phase being the one initiated after the failure prediction. Besides, no rework time is induced in this situation.

$$l_{i,3}(t) = n_i(t) \cdot \delta(t) + R + \delta(w_i(t)) \tag{5}$$

The fourth and final case is when the system predicts a failure which will not happen (ie: a false positive). The time lost is the same as in the third case.

$$l_{i,4}(t) = l_{i,3}(t) = n_i(t) \cdot \delta(t) + R + \delta(w_i(t)) \tag{6}$$

**Final Cost Function** We can now express the lost time in the general case and considering multiple failure events. Note that we exclude the special case where $C = 0$, as in this case, the optimum checkpointing interval is obvious and is infinity. Furthermore, we consider that the system where the job runs is failure prone and $C > 0$. The total cost function is thus as follows:

$$L(t) = \sum_{i=1}^{C_{fn}} l_{i,2}(t) + \sum_{i=1}^{C_{tp}} l_{i,3}(t) + \sum_{i=1}^{C_{fp}} l_{i,4}(t) \tag{7}$$

Considering that the job length is $S$, we can estimate the number of successful computing phases for the job to complete as follows:

$$N(t) = \frac{S}{t} = \sum_{i=1}^{C} n_i(t) \tag{8}$$

The equation 7 can be expanded as follows.

$$\begin{aligned} L(t) = S\,\frac{\delta(t)}{t} + (C + C_{fp})\,R + C_{fn}\,w_i(t) + \\ (C_{tp} + C_{fp})\,\delta(w_i(t)) + \delta(t) \sum_{i=1}^{C_{fp}} n_i(t) \end{aligned} \tag{9}$$

The optimum checkpointing interval $t_{opt}$ is the value of $t$ which produces the smallest time lost. In other words, it is the minimum of the function $L(t)$.

### 3.4 Solving Assumptions

**Assumption on the Checkpointing overhead** We assume that the dump size produced by jobs is linear and is a function of time. Thus, and considering a fixed bandwidth allocation, the checkpointing overhead is function of $t$ and is linear. It can be expressed as follows:

$$\delta(t) = \alpha \cdot t + \beta \tag{10}$$

Of course, this simplistic formulation may not fit for any type of job. One main issue, is that the job's dump size may be bounded with a maximum. Another issue, is that the job's dump size may not be linear.

We will address the first issue in section 3.6. For the second issue, we consider that a fairer approximation can be made using the previous formula as when compared to constant checkpointing overheads. But we do not further address this issue in this paper.

**First Assumption on the Failures Distribution** We assume that failures are independent and follow an exponential distribution of mean $M$. The literature have shown that this is usually only true for the first occurrence of the failure on the machine and tends to be false once the machine have been repaired. Thus this is only pertinent if we did consider repairs.

As the failures distribution is known, we can now estimate the number of failures that will happen for a job of length $S$. As previously stated, we need $N(t)$ successful computing phases to complete the job. Besides, we know that the probability to complete one computing phase (followed by a save phase) is:

$$Q_1(t) = P(x > t+\delta(t)) = 1-P(x \leq t+\delta(t)) = 1-(1-e^{-(t+\delta(t))/M}) = e^{-(t+\delta(t))/M} \tag{11}$$

Thus the number of tries to complete $N(t)$ computing phases is:

$$Q_n(t) = \frac{N(t)}{P(x > t + \delta(t))} = N(t) \cdot e^{\frac{t+\delta(t)}{M}} \tag{12}$$

Finally, the number of failures is:

$$C = Q_n(t) - N(t) = N(t) \cdot (e^{\frac{t+\delta(t)}{M}} - 1) \tag{13}$$

**Second Assumption on the Failures Distribution** We assume that the mean time between failures ($M$) in the system is big enough such that the checkpointing interval ($t$) and the restart delay ($R$) are negligible when compared to it. And as a direct consequence to this assumption, the checkpointing overhead ($\delta(t)$) is also negligible when compared to $M$.

We know that the first degree Taylor's series expansion is a good approximation for small values. Thus, the equation 13 can be reformulated as follows:

$$C = N(t) \cdot \frac{t + \delta(t)}{M} = S \cdot \frac{t + \delta(t)}{t \cdot M} \tag{14}$$

**Assumption on the rework time** We assume that on average, the failure event happens in the middle stage as regards to the computing phase. Because, failures are independent and exponentially distributed, this value is fair enough.

$$w_i(t) = \frac{t}{2} \tag{15}$$

**Assumption on the moment of failures predictions** We need a final assumption as regards to the moment of the failures predictions when those are false predictions (false positives). We assume that those are uniformly distributed as regards to the whole job execution. In other words, we can make the following approximation:

$$\sum_{i=1}^{C_{fp}} n_i(t) = N(t) \cdot \frac{C_{fp}}{C} = \frac{S}{t} \cdot \frac{C_{fp}}{C} \tag{16}$$

### 3.5 Solution

Before relaxing the previous assumptions, it is worth to note that $C_{tp}$, $C_{fp}$ and $C_{fn}$ can be expressed as follows (based on equations 1 and 2):

$$C_{tp} = r \cdot C \tag{17}$$

$$C_{fp} = r \cdot C \cdot \frac{1-p}{p} \tag{18}$$

$$C_{fn} = (1-r) \cdot C \tag{19}$$

Now, after relaxing the assumptions on 9 we obtain (assuming $K$ is a $t$ independent variable):

$$L(t) = \frac{S\,(\alpha+1)(\alpha r - p\,r + p)}{2\,p\,M}\,t + \frac{S\,\beta\,(r\,\beta + (r - r\,p + p)\,(R+M))}{p\,M}\,t^{-1} + K \tag{20}$$

The optimum checkpointing interval is the minimum of the function $L(t)$. We also know that $L(t)$ attains its minimum when its first derivative function is zero. We thus need to solve:

$$\frac{\mathrm{d}}{\mathrm{d}t}L(t) = 0 \tag{21}$$

From equation 20 we have:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}L(t) = -\frac{S\,\beta\,(r\,\beta + (r - r\,p + p)\,(M+R))}{p\,M}\,t^{-2} + \\ \frac{S\,(\alpha+1)(r\,\alpha - r\,p + p)}{2\,p\,M}\end{aligned} \tag{22}$$

We can thus compute optimum checkpointing interval $t_{opt}$ as follows:

$$t_{opt} = \sqrt{\frac{2\,\beta\,((M+R)\,p - (M+R)\,p\,r + (M+R+\beta)\,r)}{(\alpha+1)\,(p - p\,r + \alpha\,r)}} \tag{23}$$

We can get rid of the $R$ and the $\beta$ terms since we assumed that the restart delay and checkpointing overhead ($\delta(t) \Rightarrow \beta$) are negligible when compared

to $M$. Thus, we obtain after simplification the final formula for the optimum checkpointing interval:

$$t_{opt} = \sqrt{\frac{2\,\beta\,M\,(p - p\,r + r)}{(\alpha + 1)\,(p - p\,r + \alpha\,r)}} \tag{24}$$

We can note that the restart delay does not appear in the previous function which agrees with the result brought by Daly [4]. Besides when $r = 0$ and $\alpha = 0$, in other words when no failures predictions are made and when constant checkpointing overhead is assumed, then the optimum checkpointing interval is the same as the one predicted by Young [17]. However, the formula 24 is slightly different from the result brought in [18] when assuming $\alpha = 0$.

### 3.6 Bounding the Checkpointing Overhead

In the following lines, we address the issue related to when the checkpointing overhead is bounded with a known maximum ($\delta_{max}$). Such use case can be envisioned if the job works on fixed size files. Once a file is totally modified the checkpoint dump size will be the same even if further modifications are made on that file. Thus we can write the following equation:

$$\delta(t) \leq \delta_{max} \tag{25}$$

Replacing $\delta(t)$ using equation 10, $t$ can be bounded and the new optimum checkpointing interval is as follows:

$$t_{opt}^{'} = \min\,(t_{opt}, \frac{\delta_{max} - \beta}{\alpha}) \tag{26}$$

## 4 Evaluation

We have used the ACS simulator [1] in order to simulate and evaluate our model against state of the art models.

We have run multiple simulations for each tested model using the following simulation input. First, we consider different failure properties. Therefore, different system MTBF values are tested ranging from 1 hour to $10^4$ hours. As regards to the precision and recall, we have tested two combinations corresponding to the results reported in the literature [7]. Next, for each simulation, a total of 1000 jobs with a mean length of 500 hours are launched. And finally, for each job, we have considered empirical values for the $\alpha$, $\beta$ and $R$ parameters.

Given a simulation input, the simulator computes the finish time for each job and the average job finish time considering all the jobs. This value is used to compare different models including Young's [17], Daly's [4] and Zhu's [18]. We have observed in different simulation scenarios that when using our model, we obtain lower average job finish time when compared to other models. For brevity, we only include the results of the comparison of our model with Zhu's [18] model

which is the only model that takes into consideration hybrid fault tolerance among previously compared models. The results are depicted in Fig.3 and Fig.4, and display the gain percentage on the average job finish time when using our model instead of Zhu's model.

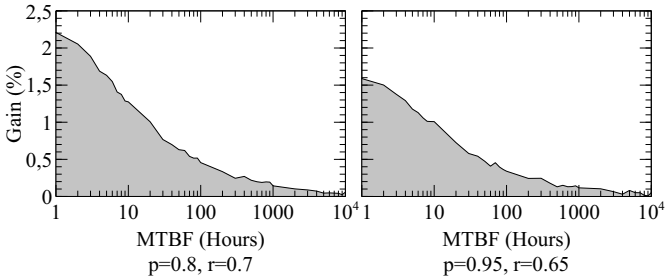


**Fig. 3.** Time gain percentage of Our Model when compared to Zhu's Model [18] assuming a Constant Checkpointing Overhead ($\alpha = 0$, $\beta = 5min$, $R = 10min$)

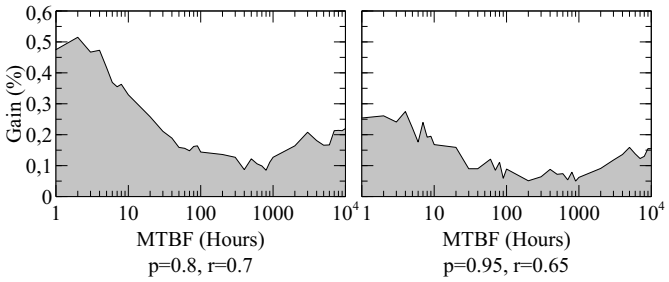


**Fig. 4.** Time gain percentage of Our Model when compared to Zhu's Model [18] assuming a Variable Checkpointing Overhead ($\alpha = 0.3$, $\beta = 5min$, $R = 10min$)

## 5 Conclusion

In this paper, we have brought a new formula for computing the optimum checkpointing interval in systems where hybrid fault tolerance is applied and considering variable size checkpoint dumps. The formula has been compared to state of the art approaches and the results show that our formula brings better results as regards to the job execution time in failure prone environments.

We have considered in this paper that the checkpoint dump size is a function of the execution time and is linear. Therefore, a good direction for future work is considering more general functions for expressing the size of checkpoint dumps.

## References


1. Acs - advanced cloud simulator, 2014.
2. D. AReed. High-end computing: The challenge of scale. In *Director's Colloquium, Los Alamos National Laboratory*, 2004.
3. M.-S. Bouguerra, D. Trystram, and F. Wagner. Complexity analysis of checkpoint scheduling with variable costs. *Computers, IEEE Transactions on*, 62(6):1269–1275, 2013.
4. J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303–312, 2006.
5. I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *The Journal of Supercomputing*, 65(3):1302–1326, 2013.
6. E. Gelenbe. On the optimum checkpoint interval. *Journal of the ACM (JACM)*, 26(2):259–270, 1979.
7. P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White. A meta-learning failure predictor for blue gene/l systems. In *Parallel Processing, 2007. ICPP 2007. International Conference on*, pages 40–40. IEEE, 2007.
8. Y. Li and Z. Lan. Exploit failure prediction for adaptive fault-tolerance in cluster computing. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 8–pp. IEEE, 2006.
9. Y. Ling, J. Mi, and X. Lin. A variational calculus approach to optimal checkpoint placement. *Computers, IEEE Transactions on*, 50(7):699–708, 2001.
10. T. Ozaki, T. Dohi, H. Okamura, and N. Kaio. Distribution-free checkpoint placement algorithms based on min-max principle. *Dependable and Secure Computing, IEEE Transactions on*, 3(2):130–140, 2006.
11. I. Philp. Software failures and the road to a petaflop machine. In *HPCRI: 1st Workshop on High Performance Computing Reliability Issues, in Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA-11)*, 2005.
12. R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–435. ACM, 2003.
13. F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3):10, 2010.
14. B. Schroeder, E. Pinheiro, and W.-D. Weber. Dram errors in the wild: a large-scale field study. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 193–204. ACM, 2009.
15. S. Toueg and Ö. Babaoglu. On the optimum checkpoint selection problem. *SIAM Journal on Computing*, 13(3):630–649, 1984.
16. K. V. Vishwanath and N. Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204. ACM, 2010.
17. J. W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, 1974.
18. L. Zhu, J. Gu, Y. Wang, and T. Zhao. Research on optimum checkpoint interval for hybrid fault tolerance. In *Advanced Parallel Processing Technologies*, pages 367–380. Springer, 2013.