# Thermal-aware Dynamic Checkpoint Interval Tuning for High Performance Computing

Extended Abstract

Pei Li
Grad. School of Information Sciences, Tohoku University
lipei@sc.cc.tohoku.ac.jp

Mulya Agung
Grad. School of Information Sciences, Tohoku University
agung@sc.cc.tohoku.ac.jp

Muhammad Alfian Amrizal
Cyberscience Center, Tohoku University
alfian@sc.cc.tohoku.ac.jp

Ryusuke Egawa
Cyberscience Center, Tohoku University
egawa@tohoku.ac.jp

Hiroyuki Takizawa
Cyberscience Center, Tohoku University
takizawa@tohoku.ac.jp

## ABSTRACT

Checkpointing with a fixed checkpoint interval, a so-called constant checkpointing method, is commonly used in the field of fault-tolerance for high-performance computing (HPC) systems. It can achieve minimum total execution time if the failure follows an exponential distribution. Related work show that there is a high correlation between temperature and reliability. By analyzing the results of the CPU temperatures monitoring on several applications, we noticed that the change of the failure rate does not follow an exponential distribution. In this paper, a dynamic checkpoint interval adjustment method based on CPU temperature monitoring is proposed. It can minimize the execution time since the checkpoint interval is adaptive to the change of the failure rates. The method predicts the change of the failure rate by utilizing a low-overhead prediction method. The simulation results show that the method can achieve a shorter execution time compared to the constant checkpointing method.

**ACM Reference Format:**
Pei Li, Mulya Agung, Muhammad Alfian Amrizal, Ryusuke Egawa, and Hiroyuki Takizawa. 2018. Thermal-aware Dynamic Checkpoint Interval Tuning for High Performance Computing: Extended Abstract. In *Proceedings of International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2018).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

The scale of high-performance computing (HPC) systems is getting larger every year to cope with the continuous growth of computational demands. For example, the world fastest supercomputer, TaihuLight, consists of more than 10 million cores with a peak performance of 125 PFlop/s [10]. In addition, applications executed on HPC systems generally have a long execution time, and thus they have a high probability of encountering failures during their run time. Without a fault-tolerance mechanism, such applications cannot finish their computation because they have to restart from the beginning of the execution whenever a failure strikes.

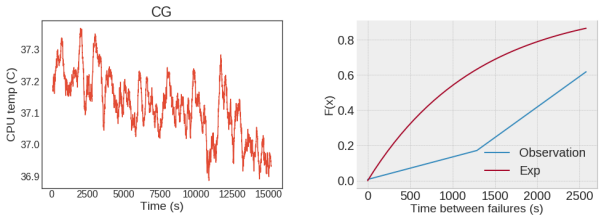



**Figure 1: Temperature monitoring results (left) and shape of the failure distribution's cumulative distribution function (right) of the CG benchmark**

The most widely-used fault-tolerance mechanism in HPC systems is *coordinated checkpointing* [1]. Coordinated checkpointing suspends application's execution, stores all the necessary data to a reliable storage, and then resumes the execution. The necessary data are stored as a checkpoint file. If a failure occurs, the execution can be recovered by restarting the application from the latest checkpoint instead of executing it again from the beginning. These checkpoints are usually taken periodically with a certain fixed interval, which is carefully selected so that the total execution time of the application is minimal.

Numerous efforts have been made to optimize the checkpoint interval in terms of execution time. Young proposed a first order approximation of the optimal checkpointing interval (OCI) [16]. Daly derived a high order estimation of the OCI [2]. Nevertheless, most of these works assume that failures are exponentially distributed with a constant failure rate, $\lambda$. By using this assumption, the OCI can be calculated conveniently. This OCI is then used constantly throughout the execution of the application. The mechanism that uses such a constant checkpointing interval is called the *constant checkpointing method*.

There are several reports that have demonstrated that the reliability of HPC systems can be affected by many factors [3, 5, 8, 12, 13].


*HPC Asia 2018, January 2018, Tokyo, Japan*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

One of the important factors is the operating temperatures of the system. Several studies [4, 9, 11] have shown that the failure rate, $\lambda$, dynamically changes with the change of operating temperature, $T$. They also discussed that the relationship between $\lambda$ and $T$ is well represented by the Arrhenius Equation [14].

Figure 1 shows the preliminary results of monitoring the CPU temperatures for the CG kernel of the NAS benchmark [7]. The left figure shows the changes of the CPU temperatures during the execution of the CG kernel. By using the data of the temperatures, we use the Arrhenius Equation to calculate the timings of the occurrence of failures. Then we plot the CDF (cumulative distribution function) of the failure distribution and compare it with that of the exponential distribution. The right figure shows the comparison result. From this figure, we can see that there is a large discrepancy between both CDFs. The relative root-mean-squared error (*RRMSE*) is 0.71. Such a high *RRMSE* value suggests that, when $\lambda$ and $T$ are highly correlated, the failure distribution will not fit well with the exponential distribution. Therefore, deriving an OCI by assuming an exponential distribution of failure with a constant $\lambda$, as in Young's [16] and Daly's [2] works, might not lead to an optimal execution time.

In this paper, we present a more advanced *adaptive checkpointing method* to cope with the dynamically-changing failure rate. Our work focuses on online checkpoint interval tuning to dynamically adjust the checkpoint interval at runtime. Here, online means that the fault-free execution time and the temperature behavior (equivalent to the failure distribution) of the application are unknown beforehand. The goal of this work is to accelerate application's total execution time by tuning the checkpoint interval based on the runtime monitoring of CPU's recent temperature. This work can be divided into two parts: (1) To monitor the temperature and to calculate the dynamic failure rate based on the existing Arrhenius model. (2) To predict the future failure rate based on the trend derived from the monitoring data, and to use the predicted value for adjusting the checkpoint interval.

The rest of the paper is organized as follows. Section II describes the problem to be discussed in this work. In Section III, a dynamic checkpoint interval tuning method is proposed. Section IV evaluates the proposed method and discusses the experimental results. Finally, the conclusions and future plan of this work are shown in Section V.

## 2 PROBLEM DESCRIPTION

This work considers that there is a high correlation between failure rate and the operating temperature. We assume that the temperature of an HPC system is constantly monitored every $t$ seconds. Then, time series data of failure rates can be obtained by substituting the monitored temperature data into Arrhenius Equation in [11].

The attempt to tune the checkpoint interval so as to minimize application's total execution time can be performed in two ways: offline or online. In the offline approach, the whole time series data must be fit into a distribution, and an OCI is analytically determined based on the distribution. The drawback of this approach is that it needs a preliminary run of the application to completely profile the CPU temperature changes so as to fully collect the time series data of the failure rates. On the other hand, the online approach

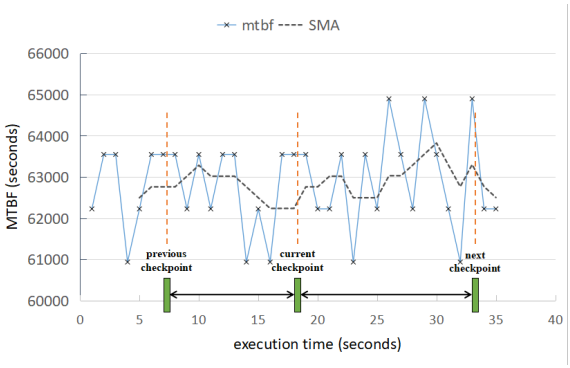



**Figure 2: An example of the failure rate prediction.**

discussed in this work does not require a preliminary run of the application. In this approach, the time series data are obtained during the execution, and those data are analyzed to predict an appropriate checkpoint interval in the near future whenever a checkpoint is taken. This interval must be re-adjusted as the time series data are updated with newly obtained data. However, such an online mechanism might potentially introduce a certain additional overhead to the application execution, and increase the total execution time. Therefore, it is important to discuss both the benefit and overhead of this approach. The main contribution of this work is to propose an online adjustment method that minimizes the execution time.

## 3 THERMAL-AWARE DYNAMIC CHECKPOINT INTERVAL TUNING

### 3.1 Methodology

In this work, a method to dynamically adjust the checkpoint interval is proposed. As discussed in Section 2, the time series data of dynamically changing failure rates are estimated by monitoring the change in CPU temperature during the application execution. The procedure of the proposed method is summarized as follows.

The first step is to monitor the CPU temperatures and to calculate the failure rate using a model based on Arrhenius Equation. Then, by using a lightweight prediction method, the failure rate in the near future is predicted. Finally, the predicted failure rate is used to calculate the interval to decide when to take the next checkpoint.

### 3.2 Failure Rate Prediction

As mentioned above, this work determines the next checkpoint interval by using the monitored failure rates during the previous checkpoint intervals. Figure 2 illustrates the failure rate prediction method. In this figure, the mean time between failures (MTBF) represents the value of $\frac{1}{failure\ rate}$. The three green bars in the figure represent three consecutive checkpoint timings, namely *previous*, *current*, and *next* checkpoint timings (from left to right). The interval between the current and the next checkpoints is adjusted by predicting the MTBF of this interval. The prediction is based on the trend of the data points of the MTBF between the previous and the current checkpoints. Since the adjustment is performed whenever a checkpoint is taken, the computation of the prediction will introduce an extra overhead. In order to minimize the overhead, in this

work, a lightweight Simple Moving Average (SMA) method is used for the prediction [15]. First, data points of the MTBF between the previous and the current checkpoints are smoothed by the SMA calculation. The smoothed results are shown as the broken lines in the figure. Then, the result of the SMA calculation, i.e., the last value of the broken lines before the current checkpoint, is used as the MTBF to calculate the interval for the next checkpoint.

### 3.3 Checkpoint Interval Tuning

In the proposed method, the next checkpoint interval is calculated by using Young's formula [16]. The OCI, $\Delta_{Young}$, is approximated by the following equation. It is based on only two parameters, checkpoint overhead $C$ and the failure rate (or MTBF) of the system.

$$\Delta_{Young} = \sqrt{2 \times C \times MTBF} \ . \quad (1)$$

In our method, the OCI for each next checkpoint is determined as follows:

$$\Delta_{i+1} = \Delta_{Young}(MTBF_i) \ . \quad (2)$$

Here $\Delta_i$ represents the OCI of the $i$-th checkpoint.

## 4 EXPERIMENTAL EVALUATION

To evaluate the effects of the proposed method, simulation-based experiments have been conducted. In this section, the experimental setup is first presented, and the experiment results are then shown for discussions.

### 4.1 Experimental Setup

Three particular kernels of NAS Parallel Benchmarks (NPB) [7], i.e., FT, LU, and CG, are used to gather the sample data of the CPU temperatures. The kernels are sequentially executed for 1,000 iterations to simulate a large-scale application run on an HPC system. The set of obtained sample data consists of the average temperatures of all processor cores for each second. Table I shows the system configuration used in the experiments.

In the monitoring step, the NPB kernels are executed under the assumption where no failure nor checkpoint happens during the execution time. The Linux *lm_sensors* tool [6] is used to monitor the CPU temperatures during the execution. The monitoring is performed every one second. Then, we use Arrhenius Equation to convert temperature data into failure rate. Finally, a simulator is implemented to mimic the occurrence of the failures under the condition of the monitored temperature changes.

The simulator injects failures during the application execution based on the integral calculation of the obtained failure rates from the monitoring step. The integral calculation is performed as shown in Equation (3).

$$F = \int_{t_{start}}^{t_{end}} \mathrm{f}(t)\,\mathrm{d}t \ . \quad (3)$$

Here $f(t)$, $t_{start}$ and $t_{end}$ represent the failure rate at monitoring time $t$, application's start time and end time, respectively. Every time $F$ reaches an integer value (1,2,3,...), the time stamp is recorded and used as a failure occurrence timing.

Besides the failure injection, the simulator emulates the proposed adaptive checkpointing and constant offline checkpointing with various checkpoint intervals.

**Table 1: Experimental Hardware Configuration**

| | |
|---|---|
| Processor: | Intel Core i7-6700 CPU @ 3.40GHz |
| CPU Frequency Range: | 800 MHz – 4000 MHz |
| Number of cores: | 4 Cores |
| Memory: | 16 GB RAM |
| Cache sizes: | 32 KB L1, 256 KB L2, 8 MB L3 |

- Constant Optimal (C-opt): The checkpoint interval is calculated by using a brute-force search. Here, the brute-force search is limited to checkpoint intervals whose values are integers.
- Constant Average (C-avg): The checkpoint interval is determined by using the average failure rate.
- Constant Optimistic (C-opms): The checkpoint interval is determined by using the minimum failure rate, which means the longest checkpointing interval.
- Constant Pessimistic (C-pes): The checkpoint interval is determined by using the maximum failure rate, which means the shortest checkpointing interval.

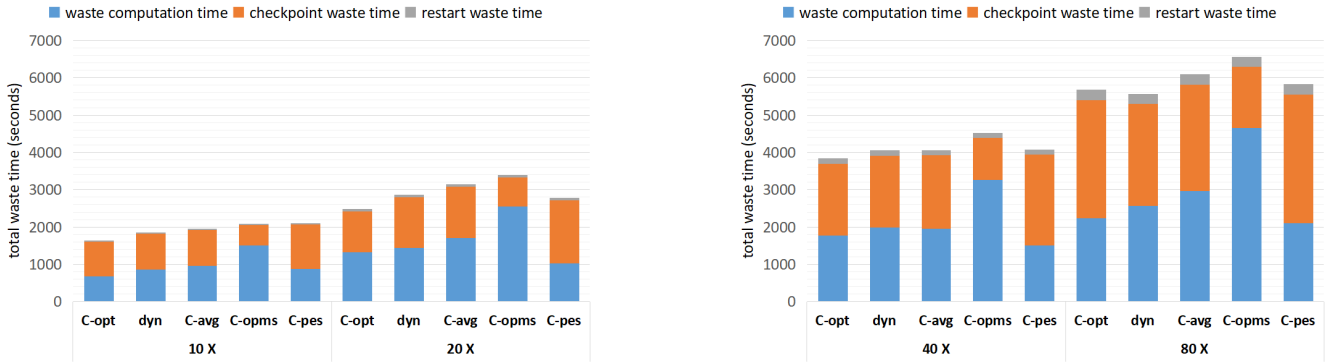
### 4.2 Performance Comparison

To simulate the effects of the CPU temperature changes on the failure rates in a large scale system, the failure rates are multiplied by the factors of 10, 20, 40 and 80. This is because the failure rates increase with the number of CPUs. The overhead of each checkpoint is assumed to be small, such as in the case when a memory-based checkpointing is used [17]. Thus, the temperature change during the checkpoint and restart is negligible. The checkpoint overhead $C$ and restart overhead $R$ are set to 1 second and 2 seconds, respectively, in the following experiments.

Figure 3 shows the total waste time of all the tested methods. The total waste time is defined as a sum of wasted computation time and the total time spent for checkpointing and restarting the application.

For the 10x, 20x, and 40x failure rate configurations, it can be observed that the total waste time of Constant Optimal is the shortest among the methods. This is because of two reasons: first, it is an offline approach where the whole temperature data is known beforehand and second, it uses brute-force search to test all possible intervals. For the 10x and 40x failure rate configurations, our proposed method can achieve the second-best results in terms of total waste time even without the prior knowledge of the temperature data. It is interesting to notice that the ratio of wasted computation time and checkpoint waste time in our method is almost the same (ratio 1:1). This is because we applied Young's formula for tuning the checkpoint interval. Note that Young's formula attempts to minimize the total execution time by making the wasted computation time and the checkpoint waste time equal [16].

Also, Figure 3 shows that Constant Pessimistic method can achieve the shortest wasted computation time among the methods because it overestimates the number of failures and takes checkpoint very frequently compared to other methods. Thus, the wasted computation time can be greatly reduced in exchange to an increase in the checkpoint waste time. Constant Pessimistic method achieves




(a) Failure rates multiplied by 10 and 20

(b) Failure rates multiplied by 40 and 80

**Figure 3: Evaluations of waste time under different failure rate multiply cases.**

the second-best result when failure rate configuration is 20x, outperforming our proposed method. This is because the checkpoint overhead used in the simulation is relatively low, i.e, 1 second, and thus the total checkpoint waste time has less contribution to the total waste time compared to the total waste computation time resulted from the more frequent checkpoint.

As the failure rate increases, the gap between Constant Optimal and Constant Pessimistic becomes closer. When the failure rate is increased to 80x, the total waste time of Constant optimal becomes nearly identical to that of Constant pessimistic. Here, the constant checkpointing method already reaches its limit because checkpointing is performed with the shortest checkpointing interval. On the other hand, our proposed method starts to outperform Constant optimal at this point. This highlights the benefit of dynamically adjusting the checkpoint interval at runtime.

## 5 CONCLUSION AND FUTURE WORK

In this work, we present a thermal-aware dynamic checkpoint interval tuning mechanism. The proposed method adjusts the checkpoint interval by using the dynamic failure rate caused by the CPU temperature changes. A constant checkpoint method cannot achieve optimal performance because it does not take into account the CPU temperature changes. On the other hand, the proposed method can achieve comparable performance to the method whose the average failure rate is already known before the application execution.

Our future work will focus on improving the accuracy of the failure rate prediction. This work considers the checkpoint intervals tuning for the applications that run on a single machine. However, when a distributed application runs on a cluster of machines, the CPU temperature trends of each machine may vary. Thus, the future work will also focus on improving the method for the distributed applications.

## ACKNOWLEDGMENTS

This research is partially supported by JST CREST "An Evolutionary Approach to Construction of a Software Development Environment for Massively-Parallel Heterogeneous Systems", DFG SPPEXA ExaFSA, and Grant-in-Aid for Scientific Research(B) #16H02822.

## REFERENCES

[1] Guohong Cao and Mukesh Singhal. 1998. On coordinated checkpointing in distributed systems. *IEEE Transactions on Parallel and Distributed Systems* 9, 12 (1998), 1213–1225.
[2] John T Daly. 2006. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future generation computer systems* 22, 3 (2006), 303–312.
[3] Nosayba El-Sayed, Ioan A Stefanovici, George Amvrosiadis, Andy A Hwang, and Bianca Schroeder. 2012. Temperature management in data centers: Why some (might) like it hot. *ACM SIGMETRICS Performance Evaluation Review* 40, 1 (2012), 163–174.
[4] Paul Ellerman. 2012. Calculating Reliability using FIT & MTTF: Arrhenius HTOL Model. *microsemi, Tech. Rep.* (2012).
[5] Saurabh Gupta, Devesh Tiwari, Christopher Jantzi, James Rogers, and Don Maxwell. 2015. 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*. IEEE, 37–44.
[6] lm sensors. 2017. Linux Hardware Monitoring. (2017). http://lm-sensors.org/
[7] NASA. 2017. NAS Parallel Benchmarks. (2017). https://www.nas.nasa.gov/publications/npb.html
[8] Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H Rogers. 2016. A large-scale study of soft-errors on gpus in the field. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 519–530.
[9] F Shoji, S Matsui, M Okamoto, F Sueyasu, T Tsukamoto, A Uno, and K Yamamoto. 2015. Long term failure analysis of 10 peta-scale supercomputer. *HPC in Asia session at ISC2015, Frankfurt, Germany, July* (2015), 12–16.
[10] Top 500 Supercomputer. 2017. TOP500 Lists. (2017). https://www.top500.org/
[11] Kun Tang, Devesh Tiwari, Saurabh Gupta, Ping Huang, Qiqi Lu, Christian Engelmann, and Xubin He. 2016. Power-capping aware checkpointing: On the interplay among power-capping, temperature, reliability, performance, and energy. In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*. IEEE, 311–322.
[12] Devesh Tiwari, Saurabh Gupta, George Gallarno, Jim Rogers, and Don Maxwell. 2015. Reliability lessons learned from GPU experience with the Titan supercomputer at Oak Ridge leadership computing facility. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*. ACM, 38.
[13] Devesh Tiwari, Saurabh Gupta, James Rogers, Don Maxwell, Paolo Rech, Sudharshan Vazhkudai, Daniel Oliveira, Dave Londo, Nathan DeBardeleben, Philippe Navaux, et al. 2015. Understanding gpu errors on large-scale hpc systems and the implications for system design and operation. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 331–342.
[14] Wikipedia. 2017. Arrhenius equation. (2017). https://en.wikipedia.org/wiki/Arrheniusequation
[15] Wikipedia. 2017. Moving average. (2017). https://en.wikipedia.org/wiki/Movingaverage
[16] John W Young. 1974. A first order approximation to the optimum checkpoint interval. *Commun. ACM* 17, 9 (1974), 530–531.
[17] Gengbin Zheng, Lixia Shi, and Laxmikant V Kalé. 2004. FTC-Charm++: an in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI. In *Cluster Computing, 2004 IEEE International Conference on*. IEEE, 93–103.