Automatic Parameter Tuning of Hierarchical Incremental Checkpointing

Alfian Amrizal[1,3], Shoichi Hirasawa[1,3], Hiroyuki Takizawa[1,3], and Hiroaki Kobayashi[1,2,3]

[1] Graduate School of Information Science, Tohoku University
Sendai, Miyagi 980-8578, Japan
Email: {alfian@sc., hirasawa@sc., tacky@, koba@}isc.tohoku.ac.jp
[2] Cyberscience Center, Tohoku University
[3] Japan Science and Technology Agency, CREST

Abstract. As future HPC systems become larger, the failure rates and the cost of checkpointing to the global file system are expected to increase. To solve this problem, this paper proposes a hierarchical incremental CPR mechanism that utilizes a hierarchical storage system of local storages and global storages. Then, to adjust the parameters of the proposed mechanism, a runtime autotuning technique is presented. Evaluation results show that the timing overhead can be significantly reduced if the storage hierarchy can be exploited with appropriate checkpoint intervals.

1 Introduction

The computational power of high-performance computing (HPC) systems is exponentially growing every year and hence enables finer-grained scientific simulations. However, the exponentially-increasing number of components of the HPC systems causes the increase in the overall failure rate. Future HPC systems are predicted to experience a failure every tens of minutes [1]. Thus, fault-tolerance has become more important than ever for future HPC systems.

Checkpointing and rollback recovery (CPR) is the most widely-used fault-tolerance mechanism for HPC systems. CPR writes the state of a running process to a checkpoint file. These checkpoint files are generally stored in a stable storage, typically a global file system. In this work, the *efficiency* of a CPR mechanism is defined as a ratio of *ideal time* to *expected time*. Here, *ideal time* is the runtime if the application encounters no failures and takes no checkpoints, while *expected time* is the runtime when a checkpoint mechanism is implemented and the occurrence of failures is considered.

Since the computational capabilities are increasing faster than the bandwidth to the global file system, the checkpointing overhead to the global file system still can dominate the overall application runtime even if the checkpointing is performed infrequently. The incremental checkpointing [2] can be one of the promising technologies to decrease the huge overhead caused by checkpointing. The incremental checkpointing reduces the data size to be written into

a checkpoint file at every checkpoint by writing only updated data or updated memory pages between two consecutive checkpoints. The changed data/memory pages will be marked as dirty and only the dirty memory pages are transferred during the checkpointing.

Another promising approach to efficient CPR is the hierarchical checkpointing [3][4] that exploits a hierarchical storage system of local and global storages. Since each storage has different degrees of resiliency and checkpointing cost, the hierarchical checkpinting relies on node-local storages for restarting from more common local failures (e.g. memory errors), and the global file system for global failures (e.g. total node failures). By frequently taking inexpensive node-local checkpoints and less frequently taking expensive system-wide global checkpoints, applications can achieve both high reliability and efficiency.

Both the incremental checkpointing and the hierarchical CPR mechanism can potentially reduce the timing overhead of CPR if some parameters are adjusted properly. The important parameters that can significantly affect performance are the checkpoint interval and the ratio of local checkpoints to global checkpoints [3]. The optimal parameters depend not only on the system configuration but also on the application to be checkpointed. This is because some information which is required to determine the optimal parameters, such as the growth speed of dirty memory pages, is application-specific. Thus, a runtime autotuning technique is required because this information is unknown in advance of the execution.

In this paper, we propose a hierarchical incremental CPR mechanism and optimizes its performance through a runtime autotuning technique. As far as we know, there exists no technique to determine the optimal parameters of a hierarchical incremental CPR mechanism for a given application and a system.

The rest of this paper is organized as follows. Section 2 describes the hierarchical incremental CPR mechanism and its performance model. Section 3 discusses a runtime autotuning technique to find the optimum parameter combination for the hierarchical incremental CPR mechanism. Section 4 shows the evaluation results. The conclusion of this paper is stated in Section 5.

2 A Hierarchical Incremental Approach to High Performance Checkpointing

2.1 A Hierarchical Incremental CPR Mechanism

This paper proposes a hierarchical incremental CPR mechanism. The proposed mechanism uses local and global storages. The local storage is used for *local incremental checkpointing* and the global storage for *global incremental checkpointing*.

When the initial checkpoint request comes, a *full checkpoint*, i.e. the whole memory data of an application, is first taken and dumped to both local and global storages. After this initial checkpoint, the type of checkpointing conducted, i.e. local incremental checkpointing or global incremental checkpointing, is determined by a parameter, which is the ratio of local to global checkpoints.

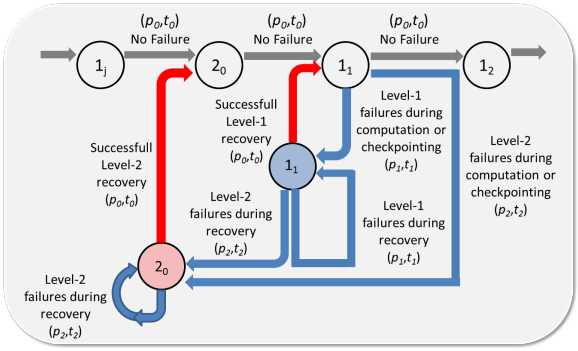

Fig. 1. Basic structure of the hierarchical checkpointing model

When the local incremental checkpointing is performed, only the dirty memory pages are transferred to the local storage. The full checkpoint file that is previously saved at the local storage during the initial checkpoint is then updated using the transferred dirty memory pages. This local incremental checkpointing process is repeated until the global checkpoint request comes. Global incremental checkpointing is performed by updating the full checkpoint file in the global storage with the global incremental data.

2.2 Performance Model

This paper extends Moody's Markov model [4] to estimate the efficiency of the proposed mechanism with a certain parameter configuration. Since the model is built using an existing model, that model's assumptions are adopted also in the performance model.

The model is constructed by combining the basic structure illustrated in Figure 1 with other similar basic structures to create a recursive structure. The basic structure has *computation* (white circle) and *recovery* (blue and red circle) states labeled by k_j. The computation state represents the period of application computation followed by a level-k checkpoint. In this case, level-1 is a local checkpoint and level-2 is a global checkpoint. Similarly, the recovery state represents the period of restoring from a checkpoint at level-k. The time required to write a level-k checkpoint is $c_k(t)$ and the time required to restore an application using a level-k checkpoint is r_k. $c_k(t)$ is a function of time since the amount of dirty memory pages transferred to a checkpoint file is changing with the change of time. However, r_k is not a function of time since a full checkpoint is required for restarting from any failure. The computation state has an internal counter j to count how many level-1 checkpoints must be done before a level-2 checkpoint is performed. The recovery state uses another internal counter to specify the state, to which the application should be rolled back when a failure occurs.

In this model, an application is transitioning from one state to another. The probability and the expected time for each of these transitions are labeled p_i

and t_i, respectively, for $i \epsilon$ 0,1,2. p_0 represents the probability of transitioning to the next right computation state and t_0 represents the expected time before the transition. The probability of transitioning to the next right computation state and the expected time before transition are $p_0(t + c_k(t))$ and $t_0(t + c_k(t))$, respectively:

$$p_0(t + c_k(t)) = e^{-\lambda(t+c_k(t))}, \tag{1}$$

$$t_0(t) = t + c_k(t). \tag{2}$$

Here, λ is the summation of local failure rates, λ_1, and global failure rates, λ_2, i.e. $\lambda = \lambda_1 + \lambda_2$. Similarly, the probability of transitioning to a level-i recovery state and the expected time before transition are $p_i(t + c_k(t))$ and $t_i(t + c_k(t))$, respectively:

$$p_i(t + c_k(t)) = \frac{\lambda_i}{\lambda}(1 - e^{-\lambda(t+c_k(t))}), \tag{3}$$

$$t_i(t + c_k(t)) = \frac{1 - (\lambda(t + c_k(t)) + 1)e^{-\lambda(t+c_k(t))}}{\lambda(1 - e^{-\lambda(t+c_k(t))})}. \tag{4}$$

Using this model under assumption that failures occur based on the Poisson distribution, the expected runtime to complete a given number of computation states can be computed. This can be done by taking the sum of the multiplications of the transition probability, p, with its corresponding expected time before transition, t, from Equations (1) to (4). Then, a calculation technique similar to [5] is applied to obtain the application's expected runtime.

3 A Runtime Autotuning Technique for the Hierarchical Incremental CPR Mechanism

An analytical solution to find the optimal value of the expected runtime of the Markov model is too difficult to derive. Hence, one must numerically explore a huge parameter space of the checkpoint interval and the ratio of local to global checkpoints. This paper presents a runtime autotuning technique to optimize the checkpoint interval and the ratio of local to global checkpoints.

A rough approximation of the optimal checkpoint interval was originally proposed by Young [6]. Then, a higher order estimation was proposed by Daly [7]. Young and Daly show that the optimal checkpoint interval t_{opt} is actually a function of system's MTTF (Mean Time to Failure) M, which is equal to $\frac{1}{\lambda}$, and the time to write checkpoint file c, as shown in Equation (5).

$$t_{opt} = \sqrt{2 \times c \times M}. \tag{5}$$

This equation is designed for a non-hierarchical and non-incremental CPR mechanism where the checkpoint cost is constant. In this paper, Young equation is extended for the runtime autotuning of the proposed CPR mechanism.

In the case of hierarchical CPR, the checkpoint cost, δc, can be approximated by the following equation.

$$\delta c = P \times \delta c_1 + (1 - P) \times \delta c_2. \tag{6}$$

Here, P and $1-P$ are the percentages of local checkpoints and global checkpoints, respectively. δc_1 and δc_2 are the average checkpoint time overheads for local checkpointing and global checkpointing, respectively.

Let μ_1 and μ_2 be the average numbers of dirty memory pages being checkpointed to local storages and global storages, respectively. If it is assumed that the bandwidth to a local storage, B_1, and the bandwidth to a global storage, B_2, are constant and can be measured beforehand, then $\delta c_1 = \mu_1/B_1$ and $\delta c_2 = \mu_2/B_2$. Both μ_1 and μ_2 can be obtained by monitoring the number of dirty memory pages at runtime.

Let the mean time to local failure and the mean time to global failure be M_1 and M_2, respectively. Either a global checkpoint file or a local checkpoint file is sufficient for recovery from local failures. On the other hand, a global checkpoint file is necessary for recovery from a global failure. Therefore, the global checkpoint interval must be optimized for the mean time to global failure M_2. Hence the optimal global checkpointing interval, t_{opt2}, can be calculated with the following equation.

$$t_{opt2} = \sqrt{2 \times \delta c_2 \times M_2}. \tag{7}$$

Then, the optimal local checkpointing interval, t_{opt1}, is calculated by considering hierarchical checkpointing and local failures.

$$t_{opt1} = \sqrt{2 \times \{P \times \delta c_1 + (1 - P) \times \delta c_2\} \times M_1}. \tag{8}$$

For hierarchical CPR, the ratio of local to a global checkpoint can be defined as the following equation.

$$P = 1 - \frac{t_{opt1}}{t_{opt2}}. \tag{9}$$

Thus, δc_1 and δc_2 can be obtained if the growth speed of dirty memory pages and the storage bandwidths of the system are known. The growth speed of dirty memory pages of an application can be monitored at runtime. Storage bandwidths are system-specific parameters. The MTTF M can be estimated using the history data or the failure log of the system. Thus, unknown variables are now t_{opt1}, t_{opt2}, and P. By solving Equations (7), (8), and (9), these unknown parameters can be obtained. Then, the obtained values are used for the runtime parameter tuning in the hierarchical incremental CPR mechanism.

4 Evaluation

In this paper, efficiency is used as the performance metric to evaluate the impact of the proposed mechanism for a particular application and system

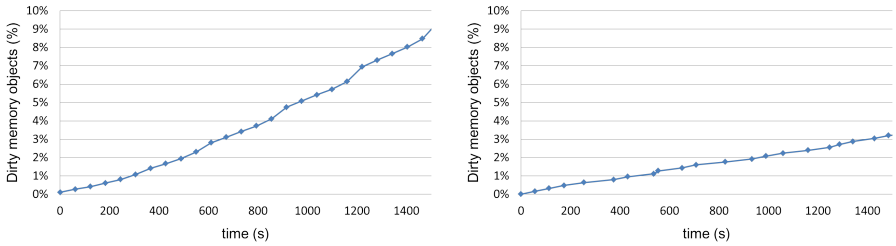

Fig. 2. The growth speed of dirty memory pages for structured diagonal sparse matrix (left) and unstructured random sparse matrix (right).

configuration. Two sparse matrix multiplication kernels are used in the evaluation. To calculate the efficiency, the values of all of the parameters of the Markov model must be known. One of these parameters is the checkpoint time, $c_k(t)$. In order to obtain $c_k(t)$, the growth speed of dirty memory pages of the matrix kernels must be monitored at runtime. Figure 2 shows the monitoring results. In this figure, the x-axis shows the elapsed time. The y-axis shows the number of dirty memory pages. This figure indicates that the value of $c_k(t)$ at a certain time instance can be obtained through the following equation:

$$c_k(t) = \frac{dirtydata(t)}{bandwidth_k}. \tag{10}$$

Here, $dirtydata(t)$ is the number of dirty memory pages obtained by monitoring the memory access behaviors at runtime, and $bandwidth_k$ is the bandwidth for performing a level-k checkpoint.

The evaluation is conducted via a simulation by assuming that a RAM disk and a Lustre system are used as the local storage and the global storage, respectively. The write bandwidth data in [4] and the failure data in [8] are used in the evaluation. The bandwidth to local storage, the bandwidth to global storage, the local failure rate, and the global failure rate are set to 6.25 Gbps, 125 Mbps, 0.1757×10^{-2} and 0.1778×10^{-4}, respectively.

At the beginning of the simulation, the checkpoint interval is set to 100 s and the ratio of local to global checkpoint is set to 2:1. Each time a global checkpoint is taken, the average number of $dirtydata(t)$ up until that point is recorded. Then, $c_k(t)$ is calculated using Equation (10). Using the obtained value of $c_k(t)$, the autotuning technique is used to adjust the checkpoint interval and the ratio of local to global checkpoints. This process is repeated until the application reaches its end. Then, the Markov model is used to evaluate the gain in efficiency from the autotuning technique.

Figure 3(a) presents the efficiency comparison of the hierarchical incremental CPR mechanism with and without the autotuning technique when running the random sparse matrix kernel (Figure 2). The results with the autotuning technique implemented are labelled “auto-inc”and those without the autotuning technique as “inc”. As future HPC systems become larger, the failure rates and

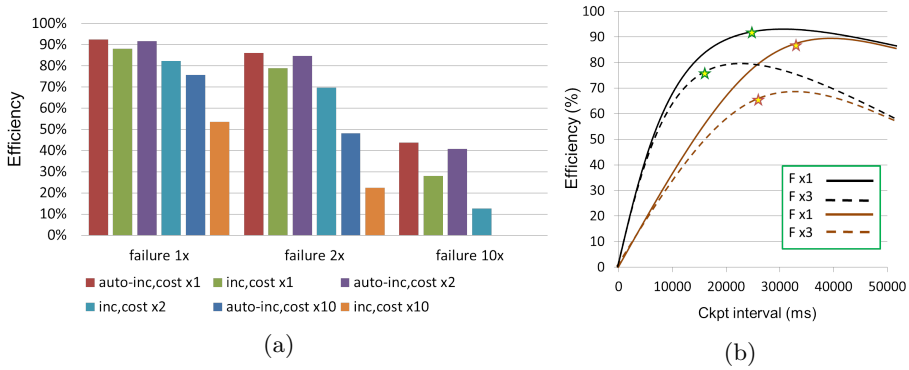

Fig. 3. (a) Comparison of hierarchical incremental CPR's efficiency with and without autotuning mechanism. (b) Efficiency versus compute interval for different growth speed of dirty memory pages.

the cost of accessing the global file system are expected to increase. To explore these effects, the base failure rates and the level-2 checkpoint costs are increased by factors of two and ten. The groups of bars along the x-axis correspond to failure rates that are one, two, or ten times higher than the base value. Within each group, the cost of the level-2 checkpoint is increased by one, two, and ten times higher than the base value.

In all the cases, the autotuning technique results in a higher efficiencies as shown in Figure 3(a). Moreover, the advantage increases with either increasing failure rates or higher global storage checkpoint costs. The gain in efficiency ranges from 4% to 28%. These results highlight the benefits of the proposed autotuning technique.

The results in Figure 3(a) show that the autotuning technique for the hierarchical incremental CPR is essential for future HPC systems. Even with systems that are 10× less reliable, the efficiency achieved by the proposed approach exceeds 40% as long as the global file system performance is unchanged. On the other hand, a higher failure rate cannot be tolerated if the cost of global checkpointing increases. In particular, if a system becomes 10× less reliable and if the cost of saving application state to the global file system rises by 10×, even with the proposed approach, the application will not be able to finish its computation.

Then, the accuracy of the autotuning technique is investigated. Figure 3(b) shows the expected efficiency when running the random sparse matrix kernel (black curve) and the diagonal sparse matrix kernel (brown curve). The growth speed of dirty memory pages of the diagonal sparse matrix is approximately three times higher than the random sparse matrix. The failure rates (labelled "F") are also changed to be three times higher than their base values. The plots were produced assuming that the ratio of local to global checkpoints is 4 and the compute interval is changed from 0 to 50 seconds.

Overall, a broad range of checkpoint intervals that result in near-optimal efficiencies can be observed. The range becomes narrower as failure rates and growth speed of dirty memory pages increase. The star marks in Figure 3(b) are the efficiencies obtained by tuning the checkpoint interval. The runtime autotuning technique can successfully reach the near-optimal checkpoint interval parameters that lead to these near-optimal efficiencies.

5 Conclusions

This paper proposed a hierarchical incremental CPR mechanism to reduce the timing overhead of checkpointing for a large scale HPC system. This mechanism can reduce the checkpoint time overhead by adjusting tuning parameters. To adjust these parameters, a runtime autotuning technique is presented. The evaluation results show that the runtime autotuning technique can find near-optimal parameter configurations to reduce the checkpoint time overhead and increases system efficiency. In the future work, this technique will be extended to account for additional features of CPR such as checkpoint compression.

6 Acknowledgments

This research is partially supported by JST CREST "An Evolutionary Approach to Construction of a Software Development Environment for Massively-Parallel Heterogeneous Systems" and Grant-in-Aid for Scientific Research(B) #25280041. One co-author, Alfian Amrizal, is financially supported by Monbukagakusho.

References

1. Schroeder, B., Gibson, G. A.: Understanding failures in petascale computers. Journal of Physics: Conference Series, vol. 78, no. 012022, 2007.
2. Sancho, J. C., Pertini, F., Johnson, G., Fernandez, J., Frachtenberg E.: On the feasibility of incremental checkpointing for scientific computing. In Proceedings of IPDPS 2004, pp. 58-67, 2004.
3. Amrizal, A., Hirasawa, S., Komatsu, K., Takizawa, H., Kobayashi, H.: Improving the scalability of transparent checkpointing for GPU computing systems. In Proceedings of the 2012 IEEE Region 10 Conference, pp. 989-994, 19-22 November 2012.
4. Moody, A., Bronevetsky, G., Mohror, K., Supinski, B. R.: Design, modeling, and evaluation of a scalable multi-level checkpointing system. In Proceedings of SC'10, 2010.
5. Vaidya, N. H., A case for two-level recovery schemes. IEEE Transactions on Computers , Vol.47, No.6, pp. 656–666, June 1998.
6. Young, J. W., A first order approximation to the optimum checkpoint interval, Communications of the ACM, vol. 17, no. 9, pp. 530–531, 1974.
7. Daly, J. T., A higher order estimate of the optimum checkpoint interval for restart dumps, Future Generation Computer Systems, vol. 22, no. 3, pp. 303-312, 2006.
8. Vivek Sarkar, E., Exascale software study: software challenges in exascale systems, 2009.