LETTER

A Checkpointing Method with Small Checkpoint Latency

Masato KITAKAMI[†a)], *Member*, **Bochuan CAI**[††], *Nonmember, and* **Hideo ITO**[†b)], *Fellow*

SUMMARY The cost of checkpointing consists of checkpoint overhead and checkpoint latency. The former is the time to stop the process for checkpointing. The latter is the time to complete the checkpointing including background checkpointing which stores memory pages. The large checkpoint latency increases the possibility that the error occurs in background checkpointing, which leads to long rollback distance. The method for small checkpoint latency has not been proposed yet. This paper proposes a checkpointing method which achieves small checkpoint latency. The proposed method divides a checkpoint interval into several subcheckpoint intervals. By using the history of memory page modification in subcheckpoint intervals, the proposed method saves some pages which are not expected to be modified in the rest of checkpoint interval in advance. Computer simulation says that the proposed method can reduce the checkpoint latency by 25% comparing to the existing methods.

key words: *dependability, checkpointing, checkpoint overhead, checkpoint latency, subcheckpoint*

1. Introduction

Recently, computers are widely used in many systems in the society. We depend on computers so deeply that computer errors have very serious damage on the society. Therefore, dependability of computers is very important.

Checkpointing and rollback have been used to improve system dependability [1]. In checkpointing, states of processes, such as the values of registers, stacks, and contents of memory pages, are stored into stable storage, such as hard disks. If the failure occurs in computers, the previous checkpoint is restored and the process is restarted from the point where the checkpoint was taken previously. This procedure is called rollback. It is apparent that some cost is required for checkpointing. The *checkpoint overhead* is a class of checkpointing cost and the time to stop the process for checkpointing.

In order to reduce the checkpoint overhead, several methods which take checkpoint for memory pages as a background procedure have been proposed. In such methods, processes can be executed during checkpointing. Although background checkpointing has a small influence on the foreground process usually, if an error occurs during the background checkpointing, it has serious damage on the process. In this case, the checkpoint is not valid and the process has to roll back to the previous checkpoint. From this, another metrics of checkpointing cost becomes important. The metrics is called *checkpoint latency*. The checkpoint latency is the time to complete checkpointing and includes background checkpointing time. The larger the checkpoint latency is, the higher the probability that the error occurs in background checkpointing is. If an error occurs during the background checkpointing, the checkpoint which is being taken in background is not valid and the process has to roll back to the previous checkpoint, that is, the rollback distance gets longer. In parallel computers, especially, checkpoint latency is larger comparing to the single-processor computer because of consistent checkpointing [2]–[4]. While many checkpointing methods achieving small checkpoint overhead have been proposed, any methods for small checkpointing latency have not been proposed yet.

This paper proposes a class of checkpointing method which can reduce checkpointing latency. The proposed method reduces checkpointing latency by saving pages which are not expected to be modified in the rest of checkpoint interval in advance. In order to determine the saved pages, a checkpoint interval is divided into some subcheckpoint intervals.

This paper includes 5 sections. Section 2 introduces checkpoint overhead and checkpoint latency; and illustrates the existing checkpointing methods. The proposed method is shown and evaluated in Sects. 3 and 4, respectively. Section 5 concludes the paper.

2. Preliminaries

2.1 Existing Methods

Many checkpointing methods which reduce the checkpointing cost have been proposed [5]–[9]. These methods are classified into two types: frozen time, i.e., overhead, reduction and checkpointing information size reduction. The followings show the existing methods for checkpointing cost reduction.

(1) Sequential Checkpointing

Sequential checkpointing is a very simple checkpointing. When an application process wants to take a checkpoint, it pauses and saves all of its states on the stable storage. The time required to save the checkpoint is practically equal to the increase in the execution time of the process.

Manuscript received February 16, 2007.
Manuscript revised November 13, 2007.

[†]The authors are with the Graduate School of Advanced Integration Science, Chiba University, Chiba–shi, 263–8522 Japan.
[††]The author is with the Graduate School of Science and Technology, Chiba University, Chiba–shi, 263–8522 Japan.
a) E-mail: kitakami@faculty.chiba-u.jp
b) E-mail: h.ito@faculty.chiba-u.jp

DOI: 10.1093/ietisy/e91–d.3.857

(2) Copy-on-Write Checkpointing

This method stops the process only during the checkpoint, excluding memory pages, is copied to the memory in checkpointing. After the checkpoint is copied to the memory, all memory pages are set to read-only mode and the process is restarted. The checkpoint stored in the memory and memory pages are stored to the stable storage as background process. The memory page which has been stored to the stable storage are set to read-write mode. If the foreground process tries to write to a memory page which has not already been stored to the stable storage, the page is set to read-write mode after it is stored to the buffer, called *copy-on-write buffer*. The page stored to the copy-on-write buffer is stored to the stable storage.

(3) Incremental Checkpointing

Incremental checkpointing is a technique for reducing the checkpoint size by using virtual memory primitives. In this method, all memory pages are set to read-only mode after checkpointing. When a program attempts to write a read-only page, an access violation occurs. In this case, the memory page is set to read-write mode and its index is stored. At the next checkpointing, only pages that have caused access violations are stored.

The combination of this method and the copy-on-write checkpointing, mentioned above, is also used in some systems.

2.2 Checkpoint Overhead and Checkpoint Latency

Checkpoint overhead is a time while processes are stopped for checkpointing. In sequential checkpointing, all checkpointing time is included in checkpoint overhead. It is apparent that checkpoint overhead is small in copy-on-write checkpointing because memory pages occupy the most of the checkpoint information and are saved by background process. In this case, another type of checkpointing time cost should be considered. *Checkpoint latency* is a time required to save the checkpoint information to stable storage at check points. It includes the time for background process to save memory pages in copy-on-write checkpointing. In the copy-on-write checkpointing, checkpoint latency is much larger than checkpointing overhead, while both are equal to each other in sequential checkpointing and incremental check pointing.

It is apparent that the large checkpoint overhead require long process freeze time and decreases the performance of processing. The large checkpoint latency also decreases the performance. If an error occurs during the background checkpointing, the checkpoint is not valid and the process has to roll back to the previous checkpoint. Since probability that error occurs during background checkpointing is large if check point latency is large, large checkpoint latency leads to long rollback distance and decreases the performance. Therefore, checkpointing method with small checkpoint latency as well as small checkpoint overhead should be considered.

3. Proposed Method

Most of the existing checkpointing methods focus on reducing checkpoint overhead and their checkpoint latency is sometimes large. If the error occurs when the checkpoint is saved as background procedure, this checkpoint is not valid and the process have to roll back to the previous checkpoint. That is, large checkpoint latency leads to long rollback distance. For single-process applications, most of existing checkpointing schemes achieve small checkpoint overhead, while resulting in a large checkpoint latency [2], [3], [10].

The proposed method reduces checkpoint latency by saving pages which are not expected to be modified in the remaining checkpoint interval before the next checkpoint. In order to select such pages, the proposed method introduces *subcheckpoints* between checkpoints. A checkpoint interval is divided into N *subcheckpoint intervals*. The pages which satisfy the following conditions are saved before the next checkpoint; 1) the page is modified in subcheckpoint intervals located between the last checkpoint and the last subcheckpoint, and 2) the page is not modified in the last subcheckpoint interval.

In other words, the proposed method distributes saving pages at each checkpoint to subcheckpoints and checkpoint. That is, the number of saving pages which should be stored in each checkpoint and subcheckpoint is reduced, although the total number of saved pages are not reduced. Since processes roll back to only checkpoints, not to subcheckpoints, in the case of error, the pages saved at subcheckpoints are not counted into checkpoint latency according to the definition of checkpoint latency. Therefore, the proposed method can reduce checkpoint latency.

In the proposed algorithms, the following notations are used.

- CP_i: the ith checkpoint
- $[CP_i, CP_{i+1}]$: the checkpoint interval between CP_i and CP_{i+1}
- N: the number of subcheckpoint intervals included in a checkpoint interval
- $SCP_{i,j}$: the jth subcheckpoint in $[CP_i, CP_{i+1}]$, where $0 < j < N$
- $[SCP_{i,j}, SCP_{i,j+1}]$: the subcheckpoint interval between $SCP_{i,j}$ and $SCP_{i,j+1}$

Also, we assume that $SCP_{i,0}$ and $SCP_{i,N}$ are equivalent to CP_i and CP_{i+1}, respectively.

(1) Proposed Method 1

The proposed method 1 consists of the procedure at the subcheckpoint and the one at the checkpoint. The following show the procedures. Here, $MT_{i,j}$ is a set of indices of pages which are modified in $[SCP_{i,j-1}, SCP_{i,j}]$ and we assume that $MT_{i,0} = \emptyset$.

Algorithm 1: Procedure at subcheckpoint $SCP_{i,j}$

1. Store indices of pages which are modified in $[SCP_{i,j-1}, SCP_{i,j}]$ into $MT_{i,j}$.
2. If $j = 1$, the procedure at $SCP_{i,j}$ is terminated; otherwise, go to Step 3.
3. Save the pages whose indices are in $MT_{i,j-1} \cap \overline{MT_{i,j}}$, where $\overline{A}$ is a complementary set of A.
4. Let $MT_{i,j} = \left(MT_{i,j-1} - (MT_{i,j-1} \cap \overline{MT_{i,j}})\right) \cup MT_{i,j}$.

Algorithm 2: Procedure at checkpoint CP_{i+1}

1. Freeze the process.
2. Save the checkpoint information excluding memory pages.
3. Restart the process.
4. Save the pages whose indices are in $MT_{i,N-1}$ by background precess.

(2) Proposed Method 2

The proposed method 2 reduces the number of the saved pages by using the queue FQ. The queue is constructed based on the frequency of modification. At subcheckpoints, pages whose indices are located near the top of FQ are stored first, because modification frequency of these pages are low and are the probability that these pages are modified again is considered to be low. The following shows the procedure at subcheckpoint.

Algorithm 3: Procedure at subcheckpoint $SCP_{i,j}$

1. Store indices of pages which are modified in $[SCP_{i,j-1}, SCP_{i,j}]$ into $MT_{i,j}$.
2. If $j < \lfloor N/2 \rfloor$, the procedure at $SCP_{i,j}$ is terminated; otherwise, go to Step 3.
3. If $j = \lfloor N/2 \rfloor$, construct the queue FQ satisfying the following conditions; 1) the less frequently the page appears in $MT_{i,1}, \cdots, MT_{i,\lfloor N/2 \rfloor - 1}$, the former part of FQ the index of the page is located at; 2) the pages which are not modified in $[CP_i, SCP_{i,\lfloor N/2 \rfloor - 1}]$ are located at the top of the queue; 3) the indices having the same frequency are ordered so that smaller index is closer to the top. And let $MT_{i,\lfloor N/2 \rfloor} = \bigcup_{j=1}^{\lfloor N/2 \rfloor - 1} MT_{i,j}$.
4. Save the pages whose indices are in $MT_{i,j-1} \cap \overline{MT_{i,j}}$, according to the order located in FQ.
5. Let $MT_{i,j} = \left(MT_{i,j-1} - (MT_{i,j-1} \cap \overline{MT_{i,j}})\right) \cup MT_{i,j}$.

Here, $MT_{i,0} = \emptyset$ and $\lfloor x \rfloor$ is the largest integer not greater than x.

The procedure at the checkpoint is same as the above Algorithm 2.

Figure 1 (a) shows an example for the proposed method 1 for $N = 4$. At the $SCP_{i,1}$, since the memory pages 1, 2, and 3 are modified in the last subcheckpoint interval, $MT_{i,1} = \{1, 2, 3\}$. In the same manner, $MT_{i,2} = \{3\}$ is obtained. Since $MT_{i,1} \cap \overline{MT_{i,2}} = \{1, 2\}$, the memory pages 1 and 2 are saved as checkpoint information for CP_{i+1} at $SCP_{i,2}$. The pages 1 and 2 which should be saved at CP_{i+1} are saved by using background process. The procedure for the following SCPs

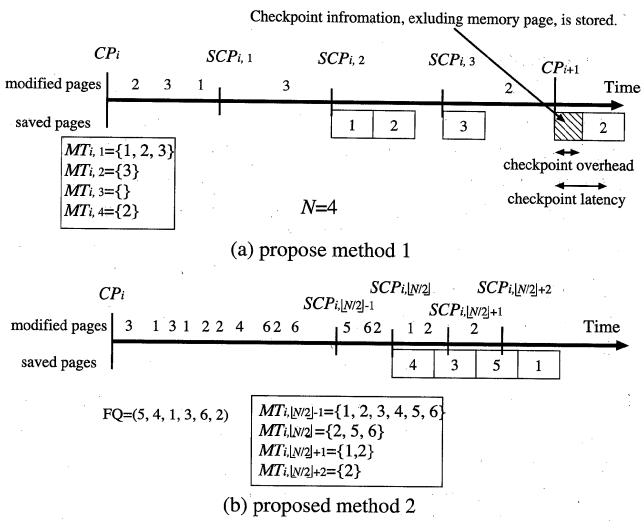

Fig. 1 Examples of the proposed methods.

is same as the above.

Figure 1 (b) shows an example for the proposed method 2. At $SCP_{i,1}, \cdots, SCP_{i,\lfloor N/2 \rfloor - 1}$, we just write the indices of memory pages modified in last subcheckpoint interval. At $SCP_{i,\lfloor N/2 \rfloor}$, we analysis $MT_{i,1}, \cdots, MT_{i,\lfloor N/2 \rfloor - 1}$ and store indices of pages into FQ based on the frequency of the modification of each memory page. Here, we assume that $FQ = (5, 4, 1, 3, 6, 2)$, as shown in Fig. 1 (b). At $SCP_{i,\lfloor N/2 \rfloor}$, since $MT_{i,\lfloor N/2 \rfloor - 1} \cap \overline{MT_{i,\lfloor N/2 \rfloor}} = \{1, 3, 4\}$, the pages 1, 3, and 4 would be saved. Since the first index which appears in FQ is "4", the memory page 4 should be saved at first. At the $SCP_{i,\lfloor N/2 \rfloor + 1}$ the pages 1 and 5 would be saved. Although the page 1 has not been saved by the termination of $SCP_{i,\lfloor N/2 \rfloor}$, we should save the page 5 at first with the reference to FQ. In this case, the memory page 1 would be saved only once, while the proposed method 1 would save the page 1 twice in the same case. This shows the proposed method 2 can reduce the number of saved pages compared to the proposed method 1.

It is noted that Fig. 1 (b) is an example case where the proposed method 2 has advantage. On the other hand, some pages are stored at the former half part of checkpoint interval in the proposed method 1, while no pages are stored in the proposed method 2. This leads that the number of pages stored at $SCP_{i,\lfloor N/2 \rfloor}$ in the proposed method 2 is larger than that in the proposed method 1. This is advantage of the proposed method 1. From the above, the proposed methods 1 and 2 have advantages and disadvantages.

Performance degradation in checkpoinitnig for the proposed methods are almost equal to that for incremental checkpointing because the pages are saved by background processes. It is apparent that error recovery methods for the proposed methods 1 and 2 are same as the ones for conventional checkpointing. That is, in the case of error, the processes roll back to the last valid checkpoint.

4. Evaluations

The proposed methods are evaluated by computer simulation. The simulation program is multiplication of 128 by 128 double-float matrices. This program is executed on 2 computers which form master-client communication model. The following lists the specifications of the both computers:

- CPU: Pentium III 700 MHz
- Memory: 500 MBytes
- OS: Solaris 2.8.

The calculation time excluding checkpointing time is 43 minutes and 28 seconds. The checkpoint interval is 4 minutes.

In the master-client model, the master processor starts checkpointing by broadcasting a CHECKPOINT-BEGIN message to all other processors. In receiving this message, each processor halts computation and checks whether all messages that it has sent have been received. When a processor recognizes that it has no outstanding messages in the network, it sends a CHECKPOINT-READY message back to the master. In receiving CHECKPOINT-READY messages from all client processors, the master broadcasts a COMMIT-CHECKPOINT message, which instructs all processors to commit their checkpoints to disk. When a processor has committed its checkpoint, it sends a CHECKPOINT-COMMITTED message back to the master and resumes its computation. When the master receives these messages from all client processors, the checkpoint is finished, and it notifies the client processors that may delete previous checkpoints.

Figure 2 shows overhead and latency of the proposed methods 1 and 2 with $N = 10$ subcheckpoint intervals in a checkpoint interval obtained by the above simulation. For comparison, the cases of the existing methods, sequential checkpointing, incremental checkpointing, copy-on-write checkpointing, and combinations of incremental checkpointing and copy-on-write checkpointing, are also shown. Here, the size of a checkpoint for sequential checkpointing and copy-on-write checkpointing is 9,429.5 kbytes and that for the other methods is 3,313.8 kbytes. This figure says the latency of the proposed methods is 25% smaller than that of the incremental checkpointing, which achieves the smallest latency among the existing methods. This also says the overhead of the proposed methods is slightly larger than that of copy-on-write checkpointing, which achieves the smallest overhead.

Figure 3 shows the relation between the number of the subcheckpoint intervals in a checkpoint interval and overhead / latency. This says that the overhead decreases as the number of the subcheckpoint intervals increases and that the latency decreases as the number of subcheckpoint interval decreases. That is, there exists a trade-off relation between the overhead and the latency.

Simulation results for other types of programs will be lead to the similar conclusion because of locality of reference.

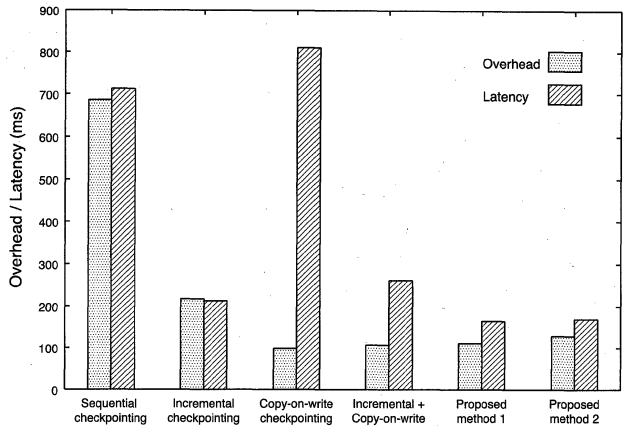

Fig. 2 Checkpoint overhead and latency.

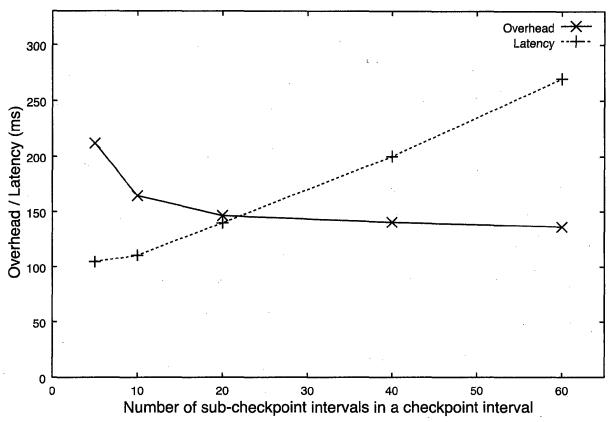

Fig. 3 Relation between the number of subcheckpoints and the checkpoint overhead / latency.

5. Conclusion

This paper has proposed checkpointing method with small checkpoint latency. The proposed method saves the page which are not expected to be modified in the rest of checkpoint interval before the corresponding checkpoint. Computer simulation says that the proposed method can reduce checkpoint latency by 25% comparing to the existing method

Future study remains in executing computer simulations by using other types of program, such as the one which requires large amount of disk I/O.

References

[1] E.N. Elnozahy, L. Alvisi, Y-M. Wang, and D.B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," ACM Comput. Surv., vol.34, no.3, pp.375–408, Sept. 2002.
[2] J.S. Plank, "An overview of checkpointing in uniprocessor and distributed systems, focusing on implementation and performance," Technical Report of University of Tennessee, July 1997.
[3] K. Hwang and Z. Xu, Scalable Parallel Computing, WCB/McGraw-Hill, 1998.

[4] N.H. Vaidya, "Staggered consistent checkpointing," IEEE Trans. Parallel Distrib. Syst., vol.10, no.7, pp.694–702, July 1999.
[5] H. Nam, J. Kim, S.J. Hong, and S. Lee, "Probabilistic checkpointing," IEICE Trans. Inf. & Syst., vol. E85-D, no.7, pp.1093–1104, July 2002.
[6] F. Karabieh, R.A. Bazzi, and M. Hicks, "Compiler-assisted heterogeneous checkpointing," Proc. 20th IEEE Symp. Reliable Distributed Systems, pp.56–65, New Orleans, USA, Oct. 2001.
[7] N.H. Vaidaya, "Another two-level failure recovery scheme: Performance impact of checkpoint placement and checkpoint latency," Technical Report of Texas A&M University, 1994.
[8] J.S. Plank, M. Beck, and G. Kingsley, "Libcpt: Transparent checkpointing under unix," 1995 USENIX Technical Conference, pp.213–223, 1995.
[9] S. Plank, M. Beck, and G. Kingsley, "Compiler-assisted memory exclusion for fast checkpointing," IEEE Technical Committee on Operating Systems and Application Environments, vol.7, no.4, pp.10–14, Winter 1995.
[10] N.H. Vaidya, "Impact of checkpoint latency on overhead ratio of a checkpointing scheme," IEEE Trans. Comput., vol.46, no.8, pp.942–947, Aug. 1997.