Performance Optimization of Checkpointing Schemes with Task Duplication

Avi Ziv, *Member*, *IEEE*,
and Jehoshua Bruck, *Senior Member*, *IEEE*

Abstract—In checkpointing schemes with task duplication, checkpointing serves two purposes: detecting faults by comparing the processors' states at checkpoints, and reducing fault recovery time by supplying a safe point to rollback to. In this paper, we show that, by tuning the checkpointing schemes to a given architecture, a significant reduction in the execution time can be achieved. The main idea is to use two types of checkpoints: compare-checkpoints (comparing the states of the redundant processes to detect faults) and store-checkpoints (storing the states to reduce recovery time). With two types of checkpoints, we can use both the comparison and storage operations in an efficient way and improve the performance of checkpointing schemes. Results we obtained show that, in some cases, using compare and store checkpoints can reduce the overhead of DMR checkpointing schemes by as much as 30 percent.

Index Terms—Fault-tolerant computing, checkpointing, task duplication, parallel computing, performance optimization.

1 Introduction

CHECKPOINTING enables reducing the time to recover from a fault by saving intermediate states of the task in a reliable storage, and upon detection of a fault restoring a previous stored state. Studies have shown that the rate of transient faults in a computer system is 10 to 30 times higher than the rate of permanent faults [1]. Transient faults can be hard to detect because they can cause a change in the task state that might lead to a wrong output from the task, without causing the task to crush. Task duplication [2] can be used to detect transient faults that cannot be detected internally, and, hence, increase the reliability of the system. In task duplication, the task is executed on more than one processor and the states of the processors are compared to detect faults.

Several papers (such as [3], [4], [5], [6]) describe schemes that combine checkpointing and task duplication. In the schemes described in these papers, each checkpoint serves two purposes. The first is to save the processor state and to reduce the fault-recovery time by supplying an intermediate correct state, thus avoiding rollback to the beginning of the task. The second purpose is fault-detection, which is achieved by executing the task on more than one processor, and comparing the processors' states at each checkpoint.

The length of the optimal interval between checkpoints, that minimizes the average execution time of a task, is determined by the checkpointing overhead [7], [8]. In checkpointing schemes with task duplication, this overhead consists of the time to store the processors' states and the time to compare these states. When there is a big difference between the time to store the processors' states and the time to compare these states, the overhead time is determined mainly by the operation that takes a longer time. As a result, the operation that takes less time is not used efficiently, and, therefore, the schemes have an unnecessary overhead that can be

- *A. Ziv is with IBM Israel, Science and Technology, MATAM—Advanced Technology Center, Haifa 31905, Israel. E-mail: aziv@vnet.ibm.com.*
- *J. Bruck is with the California Institute of Technology, Mail Code 136-93, Pasadena, CA 91125. E-mail: bruck@paradise.caltech.edu.*

Manuscript revised 13 Aug. 1997.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 105662.

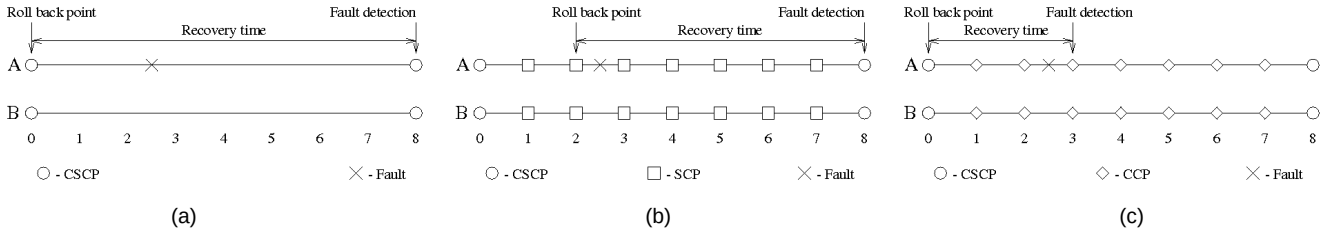


Fig. 1. Execution of one interval with the DMR scheme: (a) Traditional DMR, (b) DMR with additional store-checkpoints, (c) DMR with additional compare checkpoints.

avoided. An example for systems with a big difference between the storage and comparison times are clusters of workstations connected by a LAN. In these systems, the bandwidth of the communication subsystem (using Ethernet technology) is 300K byte/sec,[1] while the bandwidth of the local storage subsystem is about 10M byte/sec. Another example is multiprocessor supercomputers without local disks at the computing nodes, where the bandwidth of the communication subsystem is usually higher than the bandwidth of the local storage subsystem. For example, in the Intel Paragon, the bandwidth of the storage subsystem is only 600K byte/sec, while the overall bandwidth of the communication subsystem is more than 10M byte/sec.

In this paper, we present two methods to reduce the average execution time of checkpointing schemes with task duplication. The first method is to tune the scheme to the specific system it is implemented on, and use both the compare and store operations efficiently. The second method is to reduce the comparison time by using signatures.

Tuning the scheme to the system is done by using two types of checkpoints, *compare-checkpoints (CCP)* and *store-checkpoints (SCP)*. The compare-checkpoints are used to compare the states of the processors without storing them, while, in the store-checkpoints, the processors store their states without comparison. The two operations can still be used together in the same checkpoint. We refer to this type of checkpoint, with both store and compare operations, as a *compare-and-store checkpoint (CSCP)*. Using two types of checkpoints enables choosing different frequencies for the two checkpoint operations, and utilizing both operations in an efficient way. When the checkpoints that are associated with the operation that takes less time are used more frequently than the checkpoints associated with operation that takes more time, the recovery time after fault can be reduced without increasing the checkpoint overhead. This leads to a significant reduction in the average execution time of a task.

A different method to use two types of checkpoints in distributed systems was described in [9]. In this paper, Vaidya showed that using two levels of checkpoints, one which can handle most failures and is cheap and another that can handle all failures but takes longer time, can improve the performance of checkpointing schemes for distributed systems.

To illustrate how store and compare checkpoints can be used, we show how to modify the DMR (Double Modular Redundancy) scheme to take advantage of both types of checkpoints. We analyze the DMR scheme with store and compare checkpoints, and use the analysis results to compare the average execution time of a task using the traditional DMR scheme with the average execution time of a task using the proposed DMR scheme with store and compare checkpoints. The comparison results show that, in both types of systems, a significant reduction of up to 30 percent in the overhead of the execution time can be achieved when two types of checkpoints are used.

The execution time of checkpointing schemes can be reduced even more if, instead of comparing the whole states of the processors, a comparison of short signatures of the states is performed. In systems with high comparison time, signatures can significantly reduce the checkpoint overhead, and, hence, reduce the execution time of a task. Even in systems where the overhead of compare-checkpoints is small compared to the overhead of store-checkpoints, signatures can still be used to reduce the execution of the task, because with signatures, the frequency of compare checkpoints can be increased, causing reduction in the recovery time. The use of signatures for state comparison was previously suggested by Pradhan and Vaidya in [10].

The rest of the paper is organized as follows. In Section 2, we describe how the store and compare checkpoints can be used to reduce the execution time of a task, and provide the analysis of the DMR scheme with store and compare checkpoints. In Section 3, we discuss the use of signatures to shorten the comparison time. Section 4 concludes the paper.

2 Checkpointing Schemes with Store and Compare Checkpoints

In this section, we show how store and compare checkpoints can be used to improve the performance of existing checkpointing schemes. To illustrate how the modifications to the existing schemes are done, and to show how the modified schemes can be analyzed, we use the DMR (Double Modular Redundant) scheme. In this scheme, the task is executed on two processors. At each checkpoint, the states of the two processors are compared. If the states match, a correct execution is assumed, and the processors continue to the next interval. If the states do not match, both processors are rolled back to the previous checkpoint, and the execution of the same interval is repeated.

In the execution example in Fig. 1a, the states of the processors are compared and stored at the end of the interval (checkpoint 8). The states do not match because processor *A* had a fault. Hence, both processors are rolled back to checkpoint 0 and the whole interval is executed again.

In Figs. 1b and 1c, the execution of the interval that was shown in Fig. 1a is repeated with additional SCPs (Fig. 1b) or CCPs (Fig. 1c). In Fig. 1b, seven additional SCPs are placed between the CSCPs. In the given example, the fault in processor *A* is still detected at checkpoint 8. But, unlike the traditional DMR example shown in Fig. 1a, after the fault is detected, the task can be rolled back to checkpoint 2, instead of checkpoint 0, yielding a reduction in the recovery time. Store checkpoints were presented in [11], where we showed how they can be used to reduce the average execution time in LAN based distributed systems.

In Fig. 1c, seven CCPs are added between the CSCPs. In the given example, the fault in processor *A* is detected at checkpoint 3, instead of checkpoint 8. Although the task still has to be rolled back to checkpoint 0 after the fault is detected, the early detection time cause a reduction in the recovery time.

1. This bandwidth includes round trip delays and operating system overhead, so it is much lower than the 10M bit/sec Ethernet bandwidth.

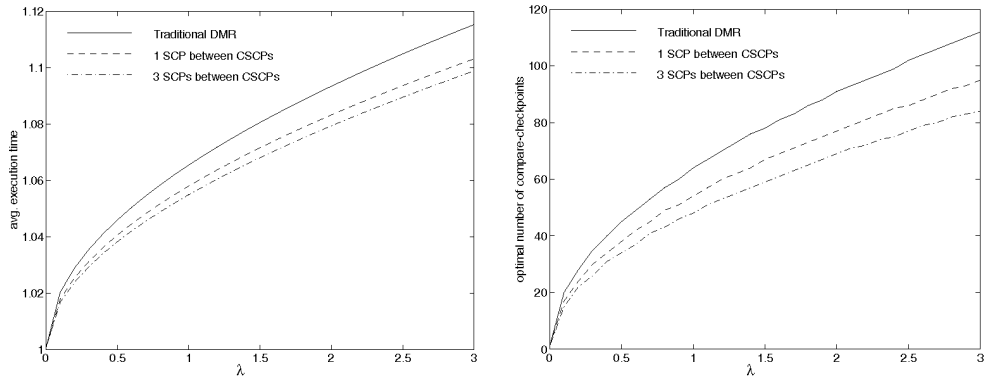


Fig. 2. Comparison between DMR scheme with and without additional store-checkpoints: (a) average execution time, (b) optimal number of CSCPs.

To analyze the average execution time of a task using the DMR scheme with additional store or compare checkpoints, we assume that a task of length 1 has to be executed. The task is divided into $m \cdot n$ intervals of length $t_I = \frac{1}{m \cdot n}$ and, at the end of each interval, a checkpoint is placed. A CSCP is placed every n intervals. The task is executed on two processors using the DMR scheme.

The processors that execute the task are vulnerable to transient faults. The faults occur in each processor according to a Poisson random process with rate λ. The faults in the processors are independent of each other. We assume that faults can occur while the processors execute the task, but not during checkpoints.

Let t_s denote the time to store the processors' states, t_{cp} denote the time to compare the processors' states, and t_r denote the time to rollback the processors to the last saved state. Let c be the probability that no faults occurred in both processors, while executing a single interval. Because the faults in the processors are independent of each other, we can write the following expression for c;

$$c = \left(e^{-\frac{\lambda}{m \cdot n}}\right)^2 = e^{-\frac{2\lambda}{m \cdot n}}.$$

The probability that no fault occurred between the CSCPs is simply c^n.

Let I denote last interval before the first fault occurred, that is, no fault occurred in intervals 1, 2, …, I, but a fault occurred in interval $I + 1$. Let Q be the number of CSCPs with identical states before the fault is detected, that is, $Q = \left\lfloor \frac{I}{n} \right\rfloor$. I and Q are geometric random variables with parameters c and c^n, respectively.

2.1 Analysis of DMR with Additional SCPs

In schemes with additional SCPs, after a fault is detected, we need to find the most recent checkpoint with identical states and roll back to it. To make the search for the most recent identical checkpoints efficient, a binary search is performed on the Huffman tree [12] induced by the probabilities of rollback to each SCP. The average number of comparisons, $\overline{C}$, using the Huffman tree, is approximately $\log_2 n$, when a CSCP is placed every n checkpoints. We assume that the time to rollback the processors is included in the time to find the most recent checkpoint with identical states.

In the example in Fig. 1b, seven additional SCPs, numbered 1 to 7, are placed between checkpoints 0 and 8. During the execution, a fault occurred in processor A, and the comparison at checkpoint 8 fails. In the first step of the search for the most recent matching checkpoint, the states at checkpoint 4 are compared. The states do not match because processor A had an earlier fault. Next, the states at checkpoint 2 are compared. These states match. After that step, we know that the required checkpoint is either 2 or 3. Finally, the states at checkpoint 3 are compared. They do not match and a rollback to checkpoint 2 is done.

After a CSCP is reached and the fault recovery process is completed, the next CSCP is placed n intervals from the last matching checkpoint. For example, in the execution in Fig. 1b, after the rollback to checkpoint 2, the next CSCP is placed at checkpoint 10, and checkpoint 8 becomes a store-checkpoint.

PROPOSITION 1. *The average execution time of task of length 1, using the DMR scheme with additional SCPs, denoted by* $\overline{T}_S$, *with* $m \cdot n$ *checkpoints, CSCP every n intervals, and faults according to independent Poisson processes with rate* λ *in the processors is*

$$\overline{T}_S = \frac{n(1-c)}{c\left(1-c^n\right)} \cdot \left(1 + mnt_s + m\left[1 + \left(1-c^n\right)\overline{C}\right]t_{cp}\right). \qquad (1)$$

PROOF. To calculate the average execution time of a task, we need to find the progress, measured in intervals, and elapsed time from the time the last rollback is completed (or the beginning of the execution) until the first fault is detected and its rollback is completed.

After the fault is detected, the task is rolled back to the last matching checkpoint. Therefore, the progress X_S is equal to the last interval without error. The average progress until the first fault is

$$\overline{X}_S = \overline{I} = \frac{c}{1-c}.$$

The time between CSCPs is the time to execute n intervals of the task, the time to store the states of the processors after each interval, and the time to compare the states of the processors at the CSCP. The first fault is detected at the Q + 1st CSCP. The time to find the last matching checkpoint after the fault is detected is $\overline{C} \cdot t_{cp}$. Therefore, the amount of time until the first fault is detected and the rollback to the last matching checkpoint is completed is

$$D_S = (Q+1)\left(n \cdot \left(t_I + t_s\right) + t_{cp}\right) + \overline{C} \cdot t_{cp},$$

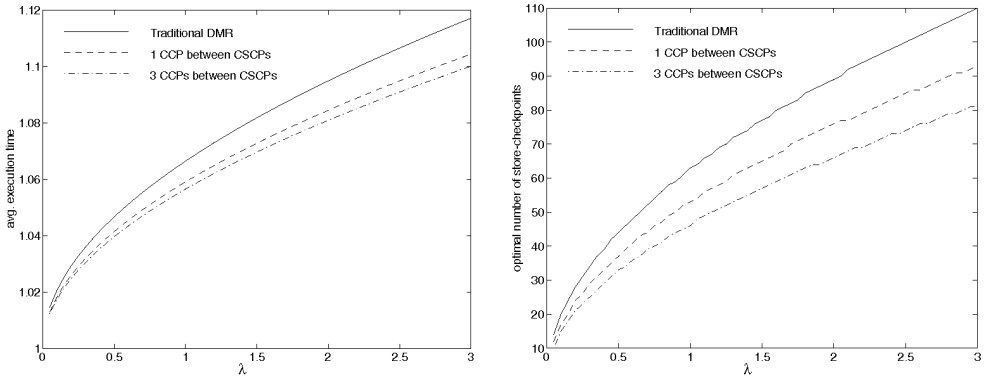
and the average time is

$$\overline{D}_S = \left(\overline{Q} + 1\right)\left(n \cdot \left(t_I + t_s\right) + t_{cp}\right) + \overline{C} \cdot t_{cp} = \frac{n \cdot \left(t_I + t_s\right) + t_{cp}}{1-c^n} + \overline{C} \cdot t_{cp}.$$

The average time to execute the whole task is

$$\overline{T}_S = n \cdot m \cdot \frac{\overline{D}_S}{\overline{X}_S} = \frac{n(1-c)}{c\left(1-c^n\right)} \cdot \left(1 + mnt_s + m\left[1 + \left(1-c^n\right)\overline{C}\right]t_{cp}\right).$$

□



(a) (b)

Fig. 3. Comparison between DMR scheme with and without additional compare-checkpoints: (a) average execution time, (b) optimal number of CSCPs.

In Fig. 2, the traditional DMR scheme performance is compared to the performance of the DMR scheme with one or three store checkpoints between CSCPs ($n = 2$ or $n = 4$). Fig. 2a shows the average execution time of a task as a function of the fault rate λ. The time to store the processors' states and compare them are $t_s = 10^{-5}$ and $t_{cp} = 5 \cdot 10^{-4}$. For each value of n and for each λ, the interval between CSCPs is chosen such that the execution time is minimized. It can be seen from the figure that using two types of checkpoints gives a significant reduction in the overhead of the execution time.

In Fig. 2b, the number of CSCPs that achieves the average execution time of Fig. 2a is shown. The figure shows us that using two types of checkpoints enables placing the CSCPs further apart and, hence, reduce the needed synchronization intervals between the processors.

2.2 Analysis of DMR with Additional CCPs

The analysis of the average execution time of the DMR scheme with additional CCPs is similar to the analysis of the DMR scheme with additional SCPs presented above. At each compare-checkpoint, the states of the processors are compared. If the states are identical, then the execution continues with the next interval. If the states are different, the execution is rolled back to the last stored state. When a CSCP is reached and the states of the processors are identical, these states are saved and they can be used as a point to rollback to.

The average execution time of a task using the DMR scheme with additional compare checkpoints is given in Proposition 2.

PROPOSITION 2. *The average execution time of task of length 1, using the DMR scheme with additional CCPs, denoted by* $\overline{T}_C$*, with* $m \cdot n$ *checkpoints, CSCP every n intervals, and faults according to independent Poisson processes with rate* λ *in the processors, is*

$$\overline{T}_C = \frac{1-c^n}{nc^n(1-c)}\left(1+mnt_{cp}\right) + mt_s + \frac{m\left(1-c^n\right)}{c^n}t_r. \qquad (2)$$

PROOF. To calculate the average execution time of a task, we need to find the progress, measured in intervals, and elapsed time from the time the last rollback is completed (or the beginning of the execution) until the first fault is detected and its rollback is completed.

After the fault is detected at the end of interval $I + 1$, a rollback to the end of interval $n \cdot Q$ is performed. Therefore, the progress between faults is $X_C = n \cdot Q$, and the average progress until the first fault is

$$\overline{X}_C = n \cdot \overline{Q} = \frac{n \cdot c^n}{1-c^n}.$$

The time between CSCPs is the time to execute n intervals of the task, the time to compare the states of the processors after each interval, and the time to store the states at the CSCP. The first fault is detected between CSCPs Q and $Q + 1$, after interval $I + 1$. The elapsed time until this fault is detected and the rollback is completed is

$$D_C = (I+1)(t_I + t_{cp}) + Q \cdot t_s + t_r,$$

and the average time is

$$\overline{D}_C = \left(\overline{I}+1\right)\left(t_I + t_{cp}\right) + \overline{Q} \cdot t_s + t_r = \frac{t_I + t_{cp}}{1-c} + \frac{c^n}{1-c^n}t_s + t_r.$$

The average time to execute the whole task is

$$\overline{T}_C = n \cdot m \cdot \frac{\overline{D}_C}{\overline{X}_C} = \frac{1-c^n}{nc^n(1-c)}\left(1+mnt_{cp}\right) + mt_s + \frac{m\left(1-c^n\right)}{c^n}t_r.$$

□

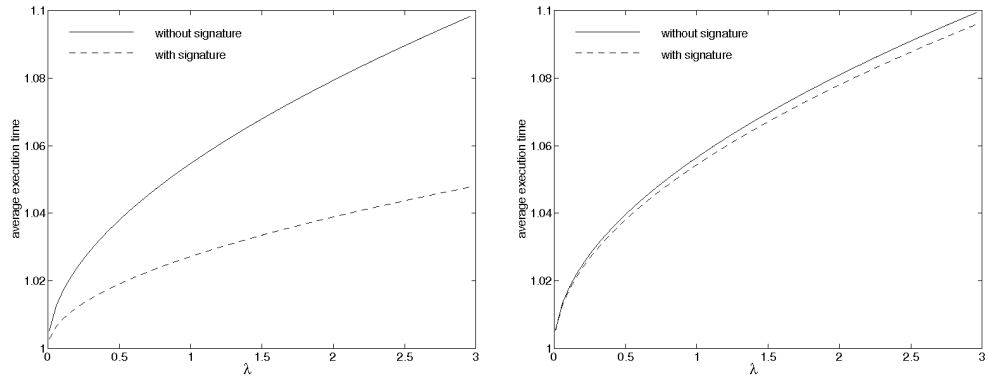
In Fig. 3, the traditional DMR scheme performance is compared to the performance of the DMR scheme with one or three compare checkpoints between CSCPs ($n = 2$ or $n = 4$). Fig. 3a shows the average execution time of a task as a function of the fault rate λ when the faults in the processors are iid Poison processes with rate λ. The time to compare the processors' states, store them and rollback are $t_{cp} = 2.5 \cdot 10^{-5}$, $t_s = 5 \cdot 10^{-4}$, and $t_r = 5 \cdot 10^{-4}$. For each value of n and for each λ, the interval between checkpoints is chosen such that the execution time is minimized. It can be seen from the figure that the usage of compare checkpoints gives a significant reduction in the overhead of the execution time.

In Fig. 3b, the number of CSCPs that achieves the average execution time of Fig. 3a is shown. The figure shows us that the usage of compare checkpoints enables reducing the number of checkpoints, and, hence, reduce the load of the I/O subsystem.

3 SIGNATURES

So far, we assumed that, at each compare-checkpoint, the complete states of the processors are compared. Comparison of the complete states ensures detection of faults at the earliest possible compare-checkpoint, but it might result in a long checkpointing overhead. In this section, we show how signatures can be used to significantly reduce the overhead of compare-checkpoints, without increasing the recovery time, causing overall reduction in the execution time of a task.

A signature is a mapping of the original space into a much smaller space. For example, a parity bit is a signature that maps the original space into a single bit. When signatures are used at



(a) (b)

Fig. 4. Comparison between DMR schemes with and without signatures: (a) LAN-based system, (b) supercomputer system.

compare-checkpoints, each processor calculates a signature of its state, and this signature is used to check if the states of the processors are identical. If the signatures are different, then the states that correspond to the signatures are different and faults have occurred. If the signatures are identical, we assume that the originating states are also identical and no fault has occurred. Note that, because the signature space is much smaller than the program-state space, many different states are mapped into the same signature, and there is a possibility that a fault might not be detected.

Signatures are used to detect and correct faults in many applications, such as communication channels and storage systems. Specific signatures are created to fit the application and environment in which they are used, so that the most likely fault patterns are always going to be detected. For example, many communications protocols use CRCs to detect errors [13], because they can detect bursts of errors.

In Sections 3.1 and 3.2, we describe the modifications to checkpointing schemes with additional SCPs or CCPs when signatures are used. To avoid reduction in the reliability of the schemes, the modified schemes combine signatures and full comparisons of the states.

The analysis of the DMR scheme with additional SCPs or CCPs and signatures is similar to the analysis of the schemes without signatures, and, therefore, it is not included in the paper. The details of the analysis can be found in [14]. In this section, we assume that the time to calculate and compare signatures is t_{sig}, and the probability of misdetections is ϵ. K denotes the number of signature comparisons that are made after the fault has occurred, until the fault is detected. K is a geometric random variable with parameter ϵ and mean $\frac{1}{1-\epsilon}$. All other assumptions made in the analysis in Section 2 are used here as well.

3.1 Signatures in Schemes with Additional SCPs

In systems with low communication bandwidth, the dominant time in comparing the states of two processors is the time to send this state from one processor to another. When signatures are used, the amount of data that needs to be sent is much smaller; therefore, the comparison time can be significantly reduced, causing a big reduction in the average execution time.

Because there is a possibility of misdetection when signatures are used, we can no longer assume that the state stored after the last CSCP is a correct state. Hence, to avoid unnecessary rollbacks, we need to keep all the stored states. Keeping all the stored states can very easily overload the storage system, and, therefore, is not practical. Therefore, the scheme that we use keeps only the last checkpoint that was verified by a full comparison and the states of the store checkpoints following the most recent CSCP. In the scheme, all the comparisons are of signatures, except the comparison at the end of the program and comparisons to detect the last matching states. When the signatures of the states at a compare checkpoint are not identical, a binary search, similar to the one described in Section 2, is performed to find the most recent identical states among the store-checkpoints states. The comparisons in this search are full comparisons. If no identical states are found, the states of the last CSCP are compared. If these states are not identical, then a misdetection occurred, and the task is rolled back to the last correct saved checkpoint. If, during the search, identical states are found, then the task is rolled back to that state, and the state is saved as the last correct state.

The average execution time of a task using the DMR scheme with additional store checkpoints and signatures is given in Proposition 3.

PROPOSITION 3. *The average execution time of task of length 1, using the DMR scheme with additional SCPs and signatures, denoted by* $\overline{T}_S'$, *with* $m \cdot n$ *checkpoints, CSCP every n intervals, and faults according to independent Poisson processes with rate* λ *in the processors, is*

$$\overline{T}_S' = \frac{n\left(1-\epsilon c^n\right)(1-c)}{(1-\epsilon)^2\left(1-c^n\right)c}\left(1+m\cdot n\cdot t_s+m\cdot t_{sig}\right)+\frac{m\cdot n(1-c)}{(1-\epsilon)c}\overline{C}t_{cp}. \quad (3)$$

Fig. 4a shows the average execution time of a task using the DMR scheme with store-checkpoints, with and without signatures. The figure shows the average execution time as a function of the fault rate λ. The time to store the processors' states is $t_s = 10^{-5}$. The time to compare the states' signatures is $t_{sig} = 10^{-4}$, and the time to compare the whole state is $t_{cp} = 5 \cdot 10^{-4}$. The probability of misdetection of failure in signature comparison is $\epsilon = 10^{-4}$. The values of m and n, were chosen such that the average execution time is minimized for both the scheme with the signature and without it. The figure shows that, for systems with low communication bandwidth, signature can significantly reduce the execution time of a task.

3.2 Signatures in Schemes with Additional CCPs

In systems with high communication bandwidth, the processors can share data very fast. Therefore, to benefit from signatures, the signatures have to be simple and fast to calculate. In these systems, reducing the comparison time enables placing more compare-checkpoints between store-checkpoints, and, hence, reduce the fault detection and recovery time. Although this reduction in recovery time does not result in as big an improvement in the performance as in systems with low communication bandwidth, still, some improvement is possible.

To avoid unnecessary stores in case of misdetection of faults and to avoid rollback behind the last stored state, a full comparison of the states is performed before each store-checkpoint. As the time to compare the full states is still short compared to the store time, this full comparison has almost no effect on the checkpoint overhead. On the other hand, this full comparison ensures that every stored state is a correct state, and, so, when a fault is detected, we can roll back to the last saved checkpoint.

The average execution time of a task using the DMR scheme with additional compare checkpoints and signatures is given in Proposition 4.

PROPOSITION 4. *The average execution time of task of length 1, using the DMR scheme with additional CCPs and signatures, denoted by* $\overline{T}_C$*, with* $m \cdot n$ *checkpoints, CSCP every n intervals, and faults according to independent Poisson processes with rate* λ *in the processors, is*

$$\overline{T}_C' = \frac{\left(1-c^n\right)(1-c\epsilon)}{nc^n(1-c)(1-\epsilon)}\left(1+mnt_{sig}\right)+mt_s'+\frac{m\left(1-c^n\right)}{c^n}t_r, \qquad (4)$$

where t_s' *is the difference between the checkpointing time in CSCP and the the checkpointing time in CCP, that is,*

$$t_s' = t_s + t_{cp} - t_{sig}.$$

Fig. 4b shows the average execution time of a task using the DMR scheme with compare-checkpoints, with and without signatures. The figure shows the average execution time as a function of the fault rate λ. The time to store the processors' states and the rollback time are t_s, $t_r = 5 \cdot 10^{-4}$. The time to compare the states' signatures is $t_{sig} = 1.5 \cdot 10^{-5}$, and the time to compare the whole state is $t_{cp} = 2.5 \cdot 10^{-5}$. The probability of misdetection of failure in signature comparison is $\epsilon = 10^{-4}$. The values of m and n were chosen such that the average execution time is minimized for both the scheme with the signature and without it. The figure shows that, even in systems with high communication bandwidth, where most of the checkpointing overhead is caused by storing the processors' states, signatures can still be used to reduce the execution time of a task.

4 Conclusions

In this paper, we have introduced two methods to improve the performance of checkpointing schemes with task duplication. One improvement method is tuning the scheme to the specific system it is implemented on, and the other improvement method is shortening the comparison time of a scheme by using signatures.

Tuning the scheme to the system is done by using two types of checkpoints: compare-checkpoints (comparing the states of the redundant processes to detect faults) and store-checkpoints (storing the states to reduce recovery time). Separating the comparison and store operations enables choosing the optimal interval for each operation, without concern about the other.

Another method to improve the performance of checkpointing schemes is to shorten the comparison time by using signatures. We have shown that simple signature can be used to reduce the execution time without affecting the reliability of the scheme.

A challenging research direction is to create schemes that simultaneously address both relevant aspects of checkpointing in parallel and distributed systems, namely, the design of an efficient checkpointing scheme that achieves a consistent global state [15] and uses task duplication for fault detection. The combined checkpointing schemes have to address the complexity associated with both the detection of faults and maintaining of a consistent state. For example, if two copies of the same process receive messages in a different order, their states are going to be different and a comparison of the states will fail, even if no fault occurred. Also, when task duplication is used for fault detection, it may cause a delayed detection of faults, as faults are detected only when comparison is performed. Namely, faults in one process can spread to other processes when this process sends messages to other processes.

Acknowledgments

The research reported in this paper was supported in part by U.S. National Science Foundation Young Investigator Award CCR-9457811, by the Sloan Research Fellowship, by a grant from the IBM Almaden Research Center, San Jose, California, and by a grant from the AT&T Foundation. This research was performed, in part, using the CSCC parallel computer system operated by Caltech on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by Caltech.

References

[1] D.P. Siewiorek and R.S. Swarz, *The Theory and Practice of Reliable System Design*. Digital Press, 1982.
[2] P. Agrawal, "Fault Tolerance in Multiprocessor Systems without Dedicated Redundancy," *IEEE Trans. Computers*, vol. 37, no. 3, pp. 358-362, Mar. 1988.
[3] J. Long, W.K. Fuchs, and J.A. Abraham, "Forward Recovery Using Checkpointing in Parallel Systems," *Proc. 19th Int'l Conf. Parallel Processing*, pp. 272-275, Aug. 1990.
[4] D.K. Pradhan, "Redundancy Schemes for Recovery," Technical Report TR-89-cse-16, ECE Dept., Univ. of Massachusetts, Amherst, 1989.
[5] D.K. Pradhan and N.H. Vaidya, "Roll-Forward Checkpointing Scheme: Concurrent Retry with Nondedicated Spares," *Proc. IEEE Workshop Fault-Tolerant Parallel and Distributed Systems*, pp. 166-174, July 1992.
[6] A. Ziv and J. Bruck, "Analysis of Checkpointing Schemes for Multiprocessor Systems," *Proc. 13th Symp. Reliable Distributed Systems*, pp. 52-61, Oct. 1994.
[7] K.M. Chandy and C.V. Ramamoorthy, "Rollback and Recovery Strategies for Computer Programs," *IEEE Trans. Computers*, vol. 21, no. 6, pp. 546-556, June 1972.
[8] A. Duda, "The Effects of Checkpointing on Program Execution Time," *Information Processing Letters*, vol. 16, pp. 221-229, June 1983.
[9] N.H. Vaidya, "A Case for Two-Level Distributed Recovery Schemes," *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, pp. 64-73, May 1995.
[10] D.K. Pradhan and N.H. Vaidya, "Roll-Forward and Rollback Recovery: Performance-Reliability Trade-Off," *Proc. 24th IEEE Int'l Symp. Fault-Tolerant Computing*, June 1994.
[11] A. Ziv and J. Bruck, "Efficient Checkpointing Schemes Over Local Area Networks," *Proc. 1994 IEEE Workshop Fault-Tolerant Parallel and Distributed Systems*, June 1994.
[12] T.A. Cover and J.A. Thomas, *Elements of Information Theory*. John Wiley, 1991.
[13] A.S. Tanenbaum, *Computer Networks*. Prentice Hall, 1989.
[14] A. Ziv, "Analysis and Performance Optimization of Checkpointing Schemes with Task Duplication," PhD thesis, Stanford Univ., 1995.
[15] B. Randell, "System Structure for Software Fault Tolerance," *IEEE Trans. Software Eng.*, vol. 1, pp. 220-232, June 1975.