A New Diskless Checkpointing Approach for Multiple Processor Failures

Ge-Ming Chiu, *Member*, *IEEE Computer Society*, and Jane-Ferng Chiu

Abstract—Diskless checkpointing is an important technique for performing fault tolerance in distributed or parallel computing systems. This study proposes a new approach to enhance neighbor-based diskless checkpointing to tolerate multiple failures using simple checkpointing and failure recovery operations, without relying on dedicated checkpoint processors. In this scheme, each processor saves its checkpoints in a set of peer processors, called checkpoint storage nodes. In return, each processor uses simple XOR operations to store a collection of checkpoints for the processors for which it is a checkpoint storage node. This study defines the concept of safe recovery criterion, which specifies the requirement for ensuring that any failed processor can be recovered in a single step using the checkpoint data stored at one of the surviving processors, as long as no more than a given number of failures occur. This study further identifies the necessary and sufficient conditions for satisfying the safe recovery criterion and presents a method for designing checkpoint storage node sets that meet these requirements. The proposed scheme allows failure recovery to be performed in a distributed manner using XOR operations.

Index Terms—Diskless checkpointing, multiple failures, rollback recovery, XOR.

1 Introduction

Fault tolerance is one of the most desirable properties for many distributed or parallel computing systems. A large-scale parallel computing system, such as IBM Blue Gene/L [1], may easily involve tens of thousands or even hundreds of thousands of processors. As the number of processors grows, the system's mean time between failures (MTBF) significantly decreases. In a different context, small and portable devices equipped with wireless communication capabilities have become readily available at low cost due to the advancement of electronics and communication technologies. As a result, mobile computing applications in which such devices are major components are becoming prevalent. However, these portable devices are often more vulnerable to component or communication failure than their counterparts in fixed networks [14].

In the above systems, a partial failure may easily halt the operation of the entire system. Thus, many systems employ the checkpoint/rollback recovery technique [11] to allow a system to continue running after an appropriate recovery action is taken after failure occurs. Normally, processors checkpoints must be stored in disk-based stable storage [12]. Although stable storage can protect against hardware and power failures, the latency of writing checkpoints to a hard disk and reading from the disk incurs significant overhead for the system and may result in significant performance degradation [27]. Researchers have devised various techniques to minimize this source of overhead. These techniques include incremental checkpointing [15], checkpoint buffering with copy-on-write [13], compression [19], and memory exclusion [28]. However, the performance of stable storage medium remains a major concern with all of these techniques for the systems.

The diskless checkpointing technique [27] eliminates the need for stable storage. This design replaces stable storage operations with memory and processor redundancy. Diskless checkpointing decreases latency in checkpointing operations, allowing more frequent checkpoints without significantly affecting application runtime [36]. Diskless checkpointing also makes checkpoint/rollback recovery possible in computing environments in which stable storage is not readily available, such as mobile computing systems. One challenge faced by diskless checkpointing is the memory overhead incurred by its implementation. Hence, a diskless checkpointing scheme must reduce memory usage as much as possible.

Diskless checkpointing methods fall into three categories: neighbor-based [3], [10], [32], parity-based [7], [24], [26], [27], and Reed-Solomon coding-based [3], [4], [5], [22]. In neighbor-based diskless checkpointing, each processor saves its checkpoints in the memory of peer processors. Each checkpoint is stored in its entirety in peer memory, and no coding is involved. Whenever a processor fails, the last checkpoint can be readily recovered from one of these peer processors. However, this approach may consume a large amount of memory to tolerate multiple failures.

Parity-based schemes use a dedicated checkpoint processor to store the parity of the checkpoints taken by all the application processors using XOR operations. This approach is simple and easy to implement. Based on some parity array coding technique, two checkpoint processors can tolerate two failures [23].

- *G.-M. Chiu is with the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, #43, Section 4, Keelung Road, Taipei 106, Taiwan. E-mail: chiu@mail.ntust.edu.tw.*
- *J.-F. Chiu is with the Department of Information Technology and Communication, Tungnan University, No. 152, Section 3, PeiShen Road, ShenKeng, Taipei 22202, Taiwan. E-mail: jfchiu@mail.tnu.edu.tw.*

Manuscript received 30 June 2009; revised 29 June 2010; accepted 17 Oct. 2010; published online 23 Nov. 2010.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2009-06-0093. Digital Object Identifier no. 10.1109/TDSC.2010.76.

The Reed-Solomon coding-based approach encodes checkpoints of multiple processors using Reed-Solomon erasure coding techniques over Galois Field arithmetic. When failures occur in the system, a consistent checkpoint can be restored for each failed processor through the decoding process. However, coding-based techniques involve relatively complex computations for checkpointing and failure recovery. Specifically, failure recovery requires n surviving processors for the decoding operation, where n is the number of application processors in the system.

Existing parity-based and Reed-Solomon coding-based techniques generally require extra dedicated checkpoint processors for storing the encoded checkpoint data. These checkpoint processors are not participants of the original system, and their addition increases failure probability. Furthermore, some systems, such as mobile computing systems, may have difficulty finding extra processors to act as checkpoint processors.

This study proposes a new approach to neighbor-based diskless checkpointing that tolerates multiple failures using simple checkpoint and failure recovery operations without relying on dedicated checkpoint processors. In the proposed scheme, each processor saves its checkpoints at a set of peer processors and in return is responsible for storing a collection of checkpoints for other peer processors using simple XOR operations. Based on the concept of safe recovery criterion, this method identifies the necessary and sufficient conditions for ensuring that any failed processor can be recovered in a single step using the checkpoint data stored by one of the surviving processors. This is possible as long as no more than a given number of failures occur in the system. This study further presents a method for designing the set of peer processors through which a processor should save its checkpoints so that the safe recovery criterion is met. Failure recovery can be performed in a distributed manner using XOR operations, and the proposed scheme requires no dedicated checkpoint processors.

The rest of the paper is organized as follows: Section 2 reviews related research. Section 3 describes the system model used in this study and the basic operations of the proposed scheme. Section 4 describes the determination of the checkpoint storage node sets on which this technique is founded. Section 5 presents experimental results and overhead analysis. Finally, Section 6 draws conclusions.

2 Related Work

Checkpointing using stable storage has been studied extensively in the literature. A detailed description and comparison of different checkpointing approaches appear in [11]. Since stable storage access introduces a considerable amount of operational overhead to the system, the number of checkpoints a system can take is typically restricted. Hence, most of the previous research efforts on checkpointing technique have been devoted to ensuring that checkpoints are useful for failure recovery [18], [37]. This issue has been addressed from different perspectives: optimal checkpoint intervals [39], coordinated checkpointing [13], communication-induced checkpointing [16], [34], [35], [37], prevention of useless checkpoints [8], message logging [6], etc. In addition, several techniques have been proposed to reduce the overhead involved in checkpointing and failure recovery operations. Copy-on-write [13], incremental checkpointing [15], compression [19], and memory exclusion [28] are good examples of this. Another study [13] attempts to minimize the number of processes required for taking a consistent checkpoint or failure recovery.

Diskless checkpointing approach was first proposed in [26] to avoid the excessive overhead associated with stable storage operation. The various diskless checkpointing techniques that followed can be broadly classified into three categories: neighbor-based, parity-based, and Reed-Solomon code-based. In the neighbor-based approach, each processor saves its checkpoints in the memory of another processor. When a processor fails, the checkpoint data can be recovered from the corresponding checkpoint processor. There are three different neighbor-based checkpointing schemes [32]: mirroring [3], [10], [27], pairing [3], [10], and ring structured [3].

In the mirroring scheme, each application processor is assigned a dedicated checkpoint processor in which it stores checkpoints. The pairing scheme organizes application processors into pairs. Each processor sends checkpoints to its partner in the pair and, in return, receives and stores checkpoints for the partner processor. Therefore, dedicated checkpoint processors are not necessary. The ring-structured scheme organizes all processors into a virtual ring. Each processor sends its checkpoints to the following neighbor processor. The neighbor-based approach is simple. However, a failed processor cannot recover its state if its partner or neighbor storing its checkpoint fails at the same time.

The parity-based diskless checkpointing technique [25] requires that all application processors coordinate to take checkpoints, with parity data of the checkpoints being saved in the main memory of a dedicated parity processor. When an application processor fails, the dedicated parity processor and all the other processors that are still alive cooperate to decode the last checkpoint for the failed processor. Hence, the amount of diskless checkpoint data to be stored is small in the parity-based technique. However, the time overhead for checkpointing and failure recovery operations depends on the number of application processors in the system. To deal with the scalability problem, some schemes employ a binary tree-based XORing operation for computing checkpoint parity [5], [10]. In [5], all processors are partitioned into a number of small groups to facilitate parity computations. Parity array coding techniques [23] can easily extend this approach to tolerate more than one failure. [21] proposes a checkpoint scheme based on a multidimensional parity technique. This scheme organizes processors into a multidimensional structure in which each processor has a multidimensional address. Parity data is taken along each dimension and is stored in a dedicated parity processor. This allows a 2D parity mechanism to tolerate any double faults with $2\sqrt{n}$ dedicated processors, where n is the number of application processors in the system.

The Reed-Solomon code-based diskless checkpointing method [22] works by encoding the checkpoints of n application processors to generate an extra set of k distinct checksum data. These k pieces of encoded information are stored at k dedicated checkpoint processors. When

some application processors fail, the system is able to decode the original checkpoints for each of the failed application processors as long as the total number of failed processors (including both application processors and checkpoint processors) is no more than k. Reed-Solomon erasure code is used for encoding and decoding purposes. To implement a Reed-Solomon erasure code algorithm, it is necessary to construct a generator matrix, such as Vandermonde matrix or Cauchy matrix, to calculate checksum words with Galois Field arithmetic. Failure recovery requires the use of Gaussian elimination with at least n processors involved in the operation. Hence, implementing Reed-Solomon code algorithms is relatively expensive [4], [20], and its complex calculations can significantly decrease performance.

Considering the complexity of implementing the original Reed-Solomon code algorithm, [5] proposes a different version of the technique. This modified technique is based on floating-point arithmetic, which is simpler than Galois Field arithmetic. The main disadvantage of this algorithm is that it introduces round-off errors for application data. Hence, its applicability is restricted to scientific applications, where round-off errors of recovered data are tolerable. This method also requires application developers to be involved in the design of the checkpoint algorithm.

Another recent study [9] presents a diskless checkpoint technique, called Mutual-Aid, that is, a hybrid of neighbor-based and parity-based approaches. This scheme requires no extra processors and can tolerate any double faults using only XOR operations.

3 Basics of the Proposed Scheme

3.1 System Model

Consider a distributed system consisting of a collection of n processors (or nodes), $P_0, P_1, P_2, \ldots, P_{n-1}$, that are interconnected by a (wired or wireless) network. Each processor has physical memory and communication capability. Stable storage installation is not required in the system, and checkpoint data must be stored in the physical memory.

Assume that a computing task is partitioned into n subtasks such that each subtask is executed on a distinct processor P_i, $0 \leq i \leq n-1$, in a distributed and asynchronous manner. These subtasks communicate with each other by passing messages via the underlying network. All of the processors may fail with fail-stop mode [31]. However, communication channels are assumed to be reliable.

Each processor in the system takes checkpoints according to some criteria, which are typically dictated by the computation demand or some specific checkpoint policy, such as a periodical checkpoint requirement. When failure occurs, the state of a previous checkpoint can be recovered for the failed processors using the information maintained in the memory of surviving processors. The following section provides an overview of the proposed scheme and describes its specific goals. The next section shows how these goals can be fulfilled.

3.2 Basic Operation of the Proposed Scheme

The goal of this study is to design a diskless checkpointing scheme that enables the system to tolerate up to k simultaneous failures. When there are no more than k failures, each of the failed processors must be able to recover its previous checkpoint in a single step. The system fulfills this goal with simple checkpointing and recovery operations, without relying on the assistance of processors external to the existing system. Another goal is to make sure that the memory overhead is low.

To tolerate up to k arbitrary failures, each processor P_i, $0 \leq i \leq n-1$, must send its checkpoint to a set of at least k other processors for storage. These processors are called the *checkpoint storage nodes* of P_i. This approach ensures that if there are no more than k simultaneous failures occurring in the system, at least one of the checkpoint storage nodes will remain alive for each failed processor. To facilitate the following discussion, denote the set of checkpoint storage nodes of P_i by CS_i. Meanwhile, P_i receives checkpoint data from other processors and stores these checkpoints in its (volatile) memory. In the following discussion, the processors for which P_i is a checkpoint storage node are called the *checkpoint coverage nodes* of P_i. The set of checkpoint coverage nodes of P_i is denoted by CC_i. In other words, P_i is a checkpoint storage node for any processor in CC_i. Thus, $P_r \in CS_i$ if and only if $P_i \in CC_r$.

When there are at most k simultaneous failures in the system, a failed processor P_i will have at least one surviving checkpoint storage node. If each processor in CS_i stores the checkpoints from its checkpoint coverage nodes (including P_i) at separate locations of its memory, P_i can be recovered using the checkpoint stored by any surviving processor in CS_i. However, this approach is costly in terms of memory space required for checkpoint storage. In fact, the memory consumption is O(k) in this case. This requirement may not be acceptable for many mobile devices that are typically limited by available memory space.

To deal with this memory consumption problem, this study adopts the parity technique to reduce the size of memory space required for storing checkpoint data. In the proposed scheme, each processor calculates the parity of the checkpoints from its checkpoint coverage nodes using XOR operations and stores only the parity result in its memory. This parity technique can store the checkpoint data in a memory space of size equal to the maximum of all checkpoints. Fig. 1 illustrates the conceptual framework of this scheme. The rest of this section describes the operation of the proposed scheme under the assumption that each processor P_i has already been assigned a set of checkpoint storage nodes, i.e., CS_i. Section 4 discusses how CS_i is determined.

When processor P_i takes a checkpoint, it sends its current state to all of the processors in CS_i. Meanwhile, P_i also stores a copy of the state in a distinct section of its memory. This is necessary because P_i must maintain its last checkpoint to help other failed processors decode their previous checkpoints.

Suppose that some processor P_i fails. In this case, P_i is either restarted or a new processor will replace the role of P_i. In either case, the new processor sends a recovery request to the surviving processors in CS_i. Since P_i has k or more checkpoint storage nodes, at least one checkpoint storage node will remain alive. However, the key is that we need to ensure that at least one of the surviving checkpoint storage nodes is able to help P_i recover its previous checkpoint.

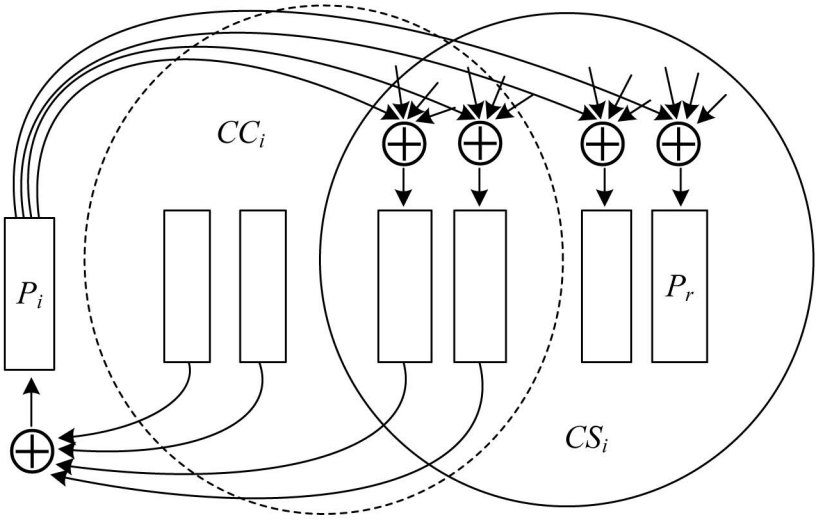


Fig. 1. The conceptual framework of the proposed diskless checkpointing approach.

Consider a surviving processor P_r in CS_i. Recall that P_r maintains the parity data of the checkpoints from all of its checkpoint coverage nodes, including P_i. Due to the property of XOR operation, all the other checkpoint coverage nodes in CC_r must stay alive to successfully decode the latest checkpoint for P_i. The proposed failure recovery mechanism is single step in the sense that each failed processor can be directly recovered by existing surviving processors. In contrast, in multistep recovery, a failed processor may have to wait for other failed processors to be recovered before it can use their recovered states to perform its own recovery. Multistep failure recovery requires extensive coordination among processors and often results in less predictable recovery time. Further, it is difficult to implement multistep failure recovery in a distributed manner. The design of checkpoint storage node sets (and thus, checkpoint coverage node sets) is nontrivial and will be addressed in detail in the next section.

Assume that one of the surviving checkpoint storage nodes, P_r, has all of its checkpoint coverage nodes, except P_i, remain alive. When P_r receives a recovery request from P_i, it asks all of the processors, except P_i, in CC_r to forward their previous checkpoints. After receiving these checkpoints, P_r then recovers the previous checkpoint for P_i by simply XORing all of these checkpoints with the parity data kept by itself. To resume the state for subsequent execution, each recovered processor P_i must request the previous checkpoints from the processors in CC_i to recover the parity data.

4 Determining the Checkpoint Storage Node Set

As described above, for any failed processor, at least one of its surviving checkpoint storage nodes must be able to help it recover its previous checkpoint, provided that there are no more than k simultaneous failures in the system. Therefore, the design of the checkpoint storage node sets must meet the following criterion:

For any processor P_i, given any subset of processors S such that $|S| \leq k$ and $P_i \in S$, there is always at least one processor $P_r \in CS_i$ such that $P_r \notin S$ and, for any processor $P_l \in CC_r$, $P_l \neq P_i$, we have $P_l \notin S$.

Here, S corresponds to the set of failed processors. To facilitate discussion, the following section calls this criterion the *safe recovery criterion*. The safe recovery criterion means that, for any failed processor, at least one surviving checkpoint storage node has all of its checkpoint coverage nodes, except the failed one, fail-free, provided that no more than k processors fail. The goal of the scheme is achieved if, and only if, the safe recovery criterion is satisfied.

4.1 Fundamentals of CS_i

This study designs CS_i's that meet the safe recovery criterion. As described earlier, the cardinality of CS_i must be at least k to ensure that at least one checkpoint storage node will survive for any failed processor when there are as many as k simultaneous failures. The proposed scheme assumes that the cardinality of CS_i is exactly k. Therefore, in the worst case, a failed processor may have only one surviving checkpoint storage node to count on for recovery. Further, each processor P_i has k checkpoint coverage nodes. Thus, the cardinality of CC_i is k to ensure good load balance.

For example, consider a system consisting of five processors, $P_0, P_1, \ldots, P_4$. If $k = 2$, the cardinality of each CS_i is two. In other words, each processor sends its checkpoint to two other processors for storage. Consider the checkpoint storage node sets CS_i and the corresponding checkpoint coverage node sets CC_i, derived from the CS_i, in Table 1a. Fig. 2a illustrates this design. Note that these CS_i's do not satisfy the safe recovery criterion. Suppose that processors P_0 and P_1 fail simultaneously. P_0 stores its previous checkpoint at P_2 and P_4 since we have $CS_0 = \{P_2, P_4\}$. However, P_1 has failed and we need P_1's checkpoint to decode the parity data kept by either of P_2 and P_4. That is, we have $CS_0 \cap CS_1 = \{P_2, P_4\}$, which has more than one element. Table 1b shows another setting of CS_i and the corresponding CC_i. Fig. 2b illustrates this design, showing that it does not satisfy the safe recovery criterion either. When P_0 and P_1 fail simultaneously, P_0 cannot be recovered in a single step because P_2 is the only surviving checkpoint storage node of P_0 and P_1's checkpoint is necessary to decode the parity data kept by P_2. That is, $P_1 \in CS_0$ but $CS_0 \cap CS_1 = \{P_2\}$. Next, consider another set of CS_i's and the corresponding CC_i's listed in Table 2. Fig. 3

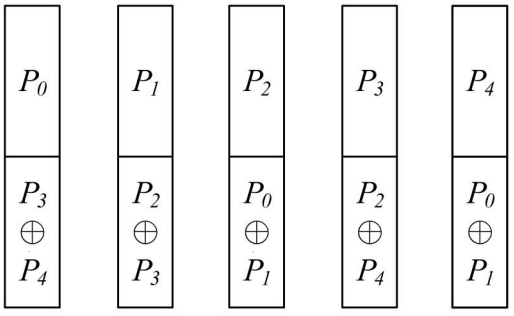
TABLE 1
Two Settings of CS_i and the Corresponding CC_i ($n = 5$ and $k = 2$) that Do Not Satisfy the Safe Recovery Criterion

i	CS_i	CC_i
0	$\{P_2, P_4\}$	$\{P_3, P_4\}$
1	$\{P_2, P_4\}$	$\{P_2, P_3\}$
2	$\{P_1, P_3\}$	$\{P_0, P_1\}$
3	$\{P_0, P_1\}$	$\{P_2, P_4\}$
4	$\{P_0, P_3\}$	$\{P_0, P_1\}$

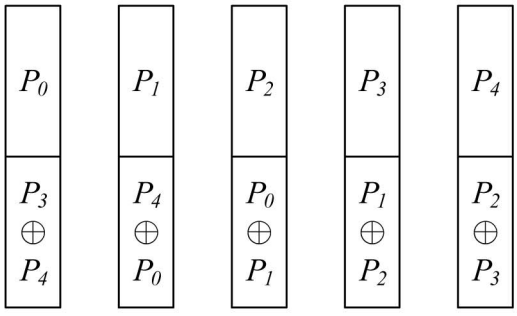
(a)

i	CS_i	CC_i
0	$\{P_1, P_2\}$	$\{P_3, P_4\}$
1	$\{P_2, P_3\}$	$\{P_0, P_4\}$
2	$\{P_3, P_4\}$	$\{P_0, P_1\}$
3	$\{P_0, P_4\}$	$\{P_1, P_2\}$
4	$\{P_0, P_1\}$	$\{P_2, P_3\}$

(b)



(a)



(b)

Fig. 2. Two designs not satisfying the safe recovery criterion: (a) corresponding to Tables 1(a); (b) corresponding to Table 1(b).

illustrates this scenario. These CS_i's clearly satisfy the safe recovery criterion.

The following section investigates fundamental properties associated with CS_i's to meet the safe recovery criterion. Two necessary conditions that govern the design of CS_i's are studied first. Suppose that two processors, P_i and P_r, share more than one checkpoint storage node, as illustrated in Fig. 4. If the set S includes P_i, P_r, and all checkpoint storage nodes of P_i that are not shared by P_r, as exemplified in Fig. 4, the safe recovery criterion is not satisfied for P_i. Fig. 2a depicts a real scenario. In other words, to meet the safe recovery criterion, any two processors in the system cannot have more than one checkpoint storage node in common. Theorem 1 formally states and proves this necessary condition.

TABLE 2
A Setting of CS_i and the Corresponding CC_i ($n = 5$ and $k = 2$) that Satisfies the Safe Recovery Criterion

i	CS_i	CC_i
0	$\{P_2, P_3\}$	$\{P_2, P_3\}$
1	$\{P_3, P_4\}$	$\{P_3, P_4\}$
2	$\{P_4, P_0\}$	$\{P_0, P_4\}$
3	$\{P_0, P_1\}$	$\{P_0, P_1\}$
4	$\{P_1, P_2\}$	$\{P_1, P_2\}$

Theorem 1. *If the safe recovery criterion is met, then the following condition must be true for all processors P_i and P_r, $i \neq r$: $|CS_i \cap CS_r| \leq 1$.*

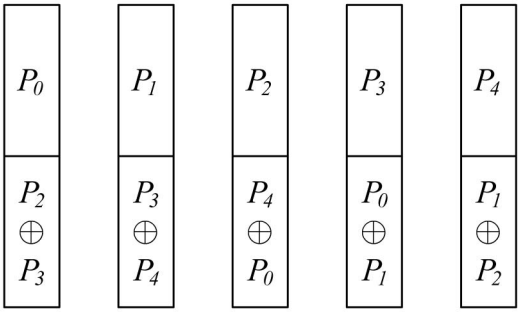


Fig. 3. A design satisfying the safe recovery criterion (Table 2).

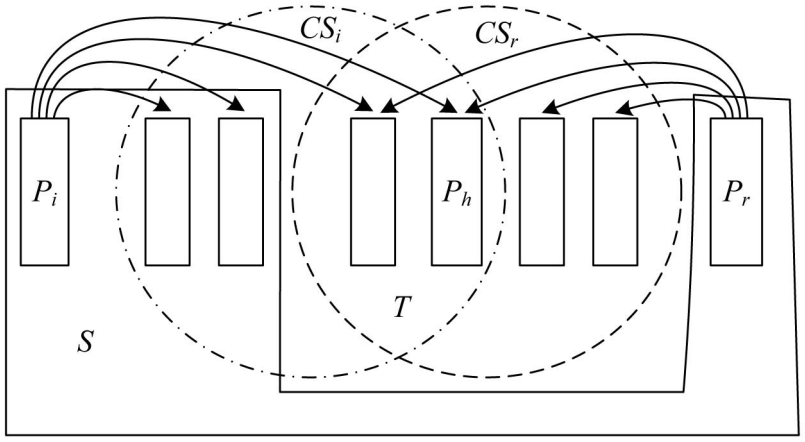


Fig. 4. System under the assumption that $|CS_i \cap CS_r| > 1$.

Proof. The example in Fig. 3 (or Table 2) illustrates this condition in which any pair of processors has at most one common checkpoint storage node. Now prove the theorem by contradiction. Assume that $|CS_i \cap CS_r| > 1$ for some $i \neq r$. Let $T = CS_i \cap CS_r$. Thus, P_i, $P_r \notin T$, because $P_i \notin CS_i$ and $P_r \notin CS_r$. Let $S = (CS_i - T) \cup \{P_i, P_r\}$ (as Fig. 4 shows). In this case, $P_i \in S$ and $|S| \leq k$. Next, consider P_i. Note that the set of processors in CS_i, but not in S is exactly T. However, for any processor $P_l \in T$, we have $P_r \in CC_l$ and $P_r \in S$. Hence, there exists no processor $P_h \in CS_i$ such that $P_h \notin S$ and, for any processor $P_l \in CC_h$, $P_l \neq P_i$, we have $P_l \notin S$; thus, a contradiction arises. □

Further, consider the situation in which P_i and $k-1$ checkpoint storage nodes of P_i form the set S, as Fig. 5 illustrates. If the sole checkpoint storage node in CS_i that is not included in S (P_h in Fig. 5) is also a checkpoint storage node of one of the checkpoint storage nodes of P_i in S (P_r in Fig. 5). This situation violates the safe recovery criterion for P_i. Fig. 2b depicts a real scenario. Theorem 2 formally states and proves this second necessary condition.

Theorem 2. *If the safe recovery criterion is met, the following condition must be true for any processor P_i: $CS_i \cap CS_r = \phi$ for any $P_r \in CS_i$.*

Proof. For example, consider processor P_0 in Fig. 3 (or Table 2). Thus, $CS_0 = \{P_2, P_3\}$. Note that $CS_0 \cap CS_2 = \phi$ and $CS_0 \cap CS_3 = \phi$. Next, prove the theorem by contradiction. Assume that, for some processor P_i, we have $CS_i \cap CS_r \neq \phi$ for some $P_r \in CS_i$. Obviously, $P_r \neq P_i$ follows. Let $T = CS_i \cap CS_r$. Note that we have $P_r \notin T$ and $|T| > 0$. Now let $S = (CS_i - T) \cup \{P_i\}$ (as Fig. 5 shows). Thus, P_i, $P_r \in S$, and $|S| \leq k$. Consider processor P_i. Note that the set of processors in CS_i, but not in S is exactly T. However, for any processor $P_h \in T$, we have $P_r \in CC_h$ and $P_r \in S$. Hence, there exists no processor $P_h \in CS_i$ such that $P_h \notin S$ and, for any processor $P_l \in CC_h$, $P_l \neq P_i$, we have $P_l \notin S$; thus, a contradiction arises. □

Theorem 2 shows that a processor does not share the same checkpoint storage node with any of its checkpoint storage nodes if the safe recovery criterion is satisfied. The necessary conditions specified in Theorems 1 and 2 can ensure the safe recovery criterion for the system, as stated in the Theorem 3.

Theorem 3. *If the conditions specified in Theorems 1 and 2 are satisfied, then the safe recovery criterion is met.*

Proof. Again, prove the theorem by contradiction. Assume that the system does not meet the safe recovery criterion.

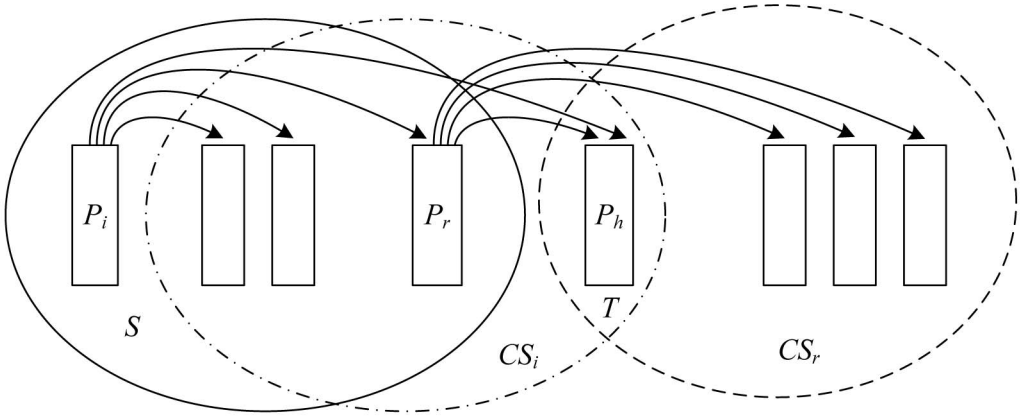


Fig. 5. System under the assumption that $CS_i \cap CS_r \neq \phi$ for some $P_r \in CS_i$.

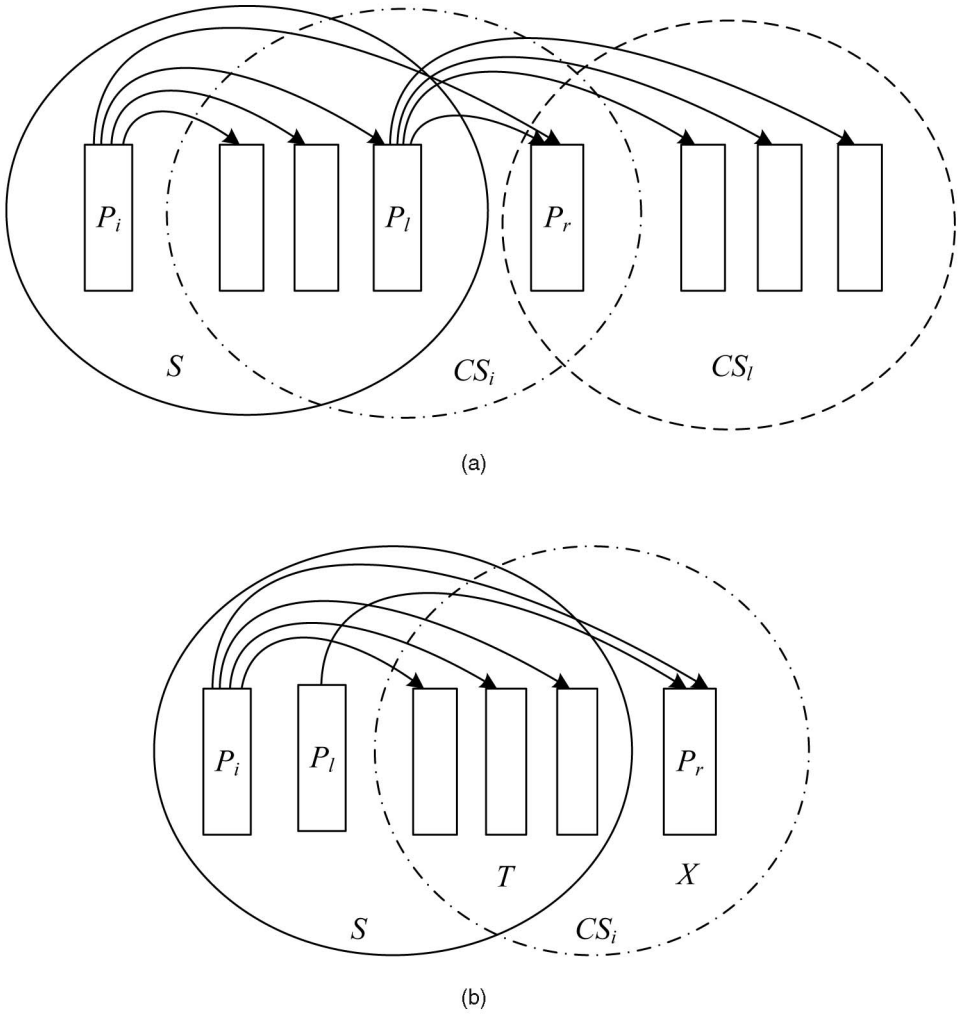


Fig. 6. (a) Some $P_r \in CS_i$, $P_r \notin S$, such that some processor $P_l \in CC_r$, $P_l \neq P_i$, $P_l \in S$, and $P_l \in CS_i$; (b) For any $P_r \in CS_i$ with $P_r \notin S$, for all $P_l \in CC_r$, $P_l \in S$, we have $P_l \notin CS_i$.

In other words, there is some processor P_i for which there exists a subset of processors S, $|S| \leq k$ and $P_i \in S$, such that, for any processor $P_r \in CS_i$, $P_r \notin S$, there is at least one processor $P_l \in CC_r$ such that $P_l \neq P_i$ and $P_l \in S$. Consider the two following cases:

Case 1. $P_l \in CS_i$ (Fig. 6a).

Since $P_l \in CC_r$, then $P_r \in CS_l$, which leads to $P_r \in CS_i \cap CS_l$. This result means $CS_i \cap CS_l \neq \phi$ with $P_l \in CS_i$, and so the condition specified in Theorem 2 is violated.

Case 2. $P_l \notin CS_i$ (Fig. 6b).

Let $T = S \cap CS_i$ and $t = |T|$. Then, there are $k - t$ processors in CS_i, but not in S. Let X denote the set of these $k - t$ processors, i.e., $X = CS_i - S$. Consider the two subcases below:

Case 2.1. There are two distinct processors $P_u, P_v \in X$, for which there is some $P_l \neq P_i$, $P_l \in S$, such that $P_l \in CC_u \cap CC_v$.

In this case, $P_u, P_v \in CS_l$. Hence, $|CS_i \cap CS_l| > 1$ follows, which violates the condition specified in Theorem 1, with respect to processors P_i and P_l.

Case 2.2. There is no $P_l \neq P_i$, $P_l \in S$, such that $P_l \in CC_u \cap CC_v$ for any P_u, $P_v \in X$.

Recall that, for each processor P_r in X, there is at least one processor $P_l \in CC_r$ such that $P_l \neq P_i$ and $P_l \in S$. Note that, in the case considered, there cannot be another processor $P_h \in X$, $P_h \neq P_r$, for which $P_l \in CC_h$. In other words, such P_l's are distinct processors. There are at least $k - t$ such processors and all of them do not belong to CS_i. Considering that S also contains P_i and the processors in T, $|S| \geq k + 1$, which contradicts the assumption of $|S| \leq k$. □

Theorems 1, 2, and 3 readily lead to the following corollary:

Corollary 1. *The safe recovery criterion is met if, and only if, the conditions specified in Theorems 1 and 2 are both true.*

Note that the example shown in Fig. 3 can be validated by Corollary 1. Figs. 2a and 2b violate the conditions specified in Theorems 1 and 2, respectively.

4.2 Design of CS_i's

This section presents a method for realizing the safe recovery criterion. The approach of assigning checkpoint storage node sets to processors in the system is based on the

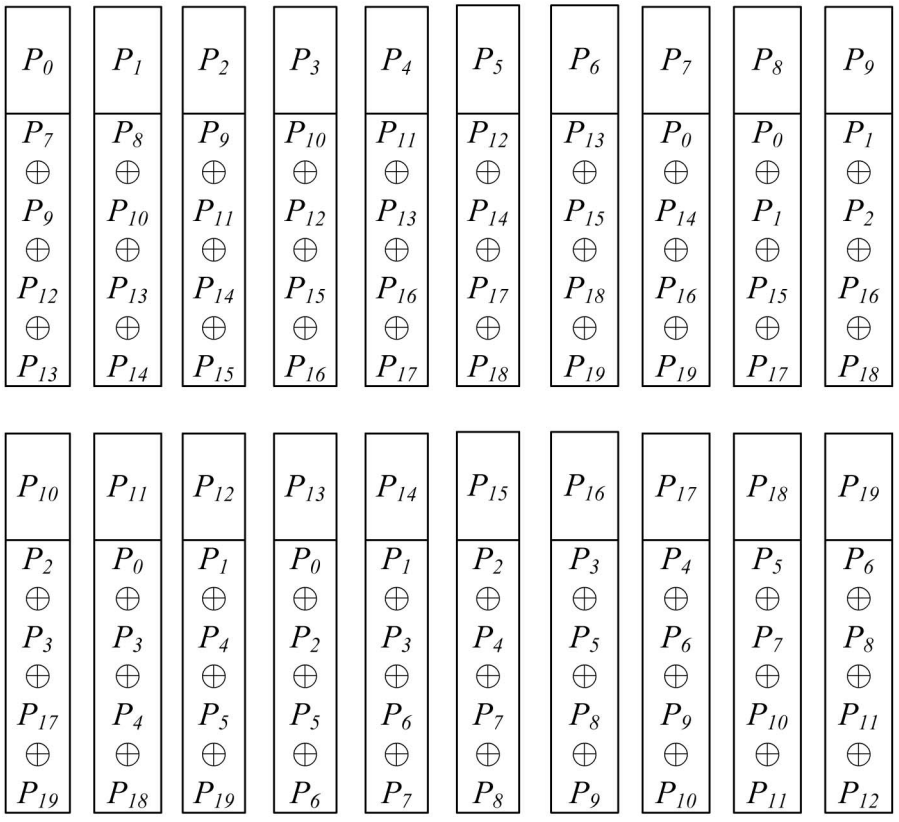


Fig. 7. A design for $n = 20$ and $k = 4$.

concept of cyclic design. That is, for any processor P_i, $0 < i \leq n-1$, CS_i can be derived from CS_0 as follows:

$$CS_i = \{P_r | r = (a+i) \bmod n, \forall P_a \in CS_0\}.$$

In other words, checkpoint storage node sets can all be cyclically generated from CS_0. Once CS_i's are designated for all processors, their counterparts CC_i's are also determined. The fact that all CS_i's and CC_i's have the same size implies that load balance can be achieved for all the processors in the system. Another advantage of this approach is that it evenly distributes the task of checkpoint operation among all processors. This approach avoids possible bottlenecks and is positive in terms of network scalability. With this approach, we only need to focus on the design of CS_0. The checkpoint storage node sets of both Figs. 2b and 3 are generated in this manner.

The following definition facilitates the description of the proposed design method:

Definition 1. *A sequence of r positive integers $d_0, d_1, d_2, \ldots, d_{r-1}$ is defined as a partial sum restricted sequence (or PSR sequence) if there exists no l, m, p, and q, $0 \leq l \leq m < p \leq q \leq r-1$, for which $\sum_{i=l}^{m} d_i = \sum_{i=p}^{q} d_i$.*

For example, 2, 1, 5, 3 is not a PSR sequence because $2 + 1 = 3$. However, 1, 3, 5, 2 is a PSR sequence of four elements. This design is based on the PSR concept. As shown later, PSR ensures that any two processors do not share more than one checkpoint storage node. Moreover, the system must include enough processors to ensure that a processor does not share checkpoint storage node with any of its checkpoint storage nodes. For example, assume that a system must tolerate four simultaneous failures, i.e., $k = 4$. First construct a PSR sequence of three (i.e., $k-1$) positive integers. Although there are many such PSR sequences, use $d_0 = 1$, $d_1 = 3$, and $d_2 = 2$ because this sequence gives the minimum sum, d, of 6. This sequence defines the spacing between two consecutive checkpoint storage nodes of a processor. CS_0 is constructed as follows: The first checkpoint storage node in CS_0 is P_7, whose subscript is one more than the value of d. The other three checkpoint storage nodes in CS_0 are obtained by d_0, d_1, and d_2 as respective increments. That is, the second checkpoint storage node in CS_0 is P_8, the third one is P_{11}, and the last one is P_{13}. Eventually, this leads to $CS_0 = \{P_7, P_8, P_{11}, P_{13}\}$, whose elements follow the spacing requirement stated above. The proposed method requires that the total number of processors in the system is no less than $3d + 2 = 20$. For $n = 20$, we can derive other CS_i's cyclically from CS_0. Fig. 7 illustrates this design, which meets the safe recovery criterion. Formally, this design is based on the following theorem. Here, we only consider $k \geq 2$.

Theorem 4. *Suppose that $d_0, d_1, d_2, \ldots, d_{k-2}$ is a PSR sequence of $k-1$ positive integers. Let $d = \sum_{i=0}^{k-2} d_i$. If $n \geq 3d + 2$, the safe recovery criterion is met if CS_0 is given as $CS_0 = \{P_{m_i} | 0 \leq i \leq k-1; m_0 = d+1; \forall i > 0, m_i = d + 1 + \sum_{j=0}^{i-1} d_j\}$.*

Proof. Fig. 8 illustrates the design (showing processor IDs in the figure). Note that there are k processors in CS_0. Consider any two processors P_i and P_r, $i \neq r$. First, $|CS_i \cap CS_r| \leq 1$ is proved. Due to the property of the cyclic design, we may assume, without loss of generality, that

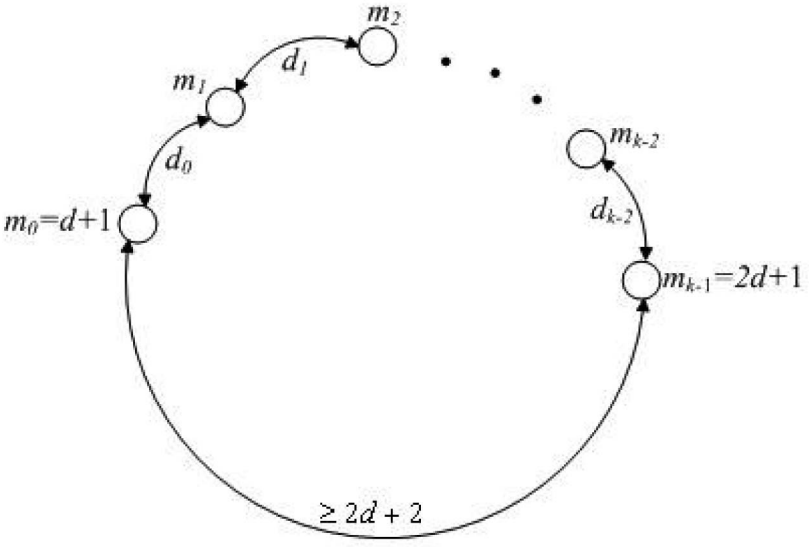


Fig. 8. The design of Theorem 4.

$P_i = P_0$. By contradiction, assume $|CS_0 \cap CS_r| > 1$, that is, CS_0 and CS_r share at least two elements. Let P_{m_a} and P_{m_b} be any two such elements, with $0 \leq a < b \leq k-1$. In addition, assume that, for CS_r, P_{m_a} and P_{m_b} are derived from P_{m_g} and P_{m_h} of CS_0, respectively. Since $0 < r \leq n-1$, we must have $a \neq g$ and $b \neq h$. Treat the elements of CS_0 as lying in a circle of n elements (from P_0 to P_{n-1}). Since we have $m_{k-1} = 2d+1$ and $n \geq 3d+2$, the (shortest) distance between m_u and m_v, for $0 \leq u < v \leq k-1$, is $\sum_{q=u}^{v-1} d_q$. Note that the distance between m_a and m_b equals the distance between m_g and m_h due to the property of cyclic generation. Since $m_0, m_1, \ldots, m_{k-1}$ fall in the range $[d+1, 2d+1]$, $n \geq 3d+2$, and $a < b$, it must be true that $g < h$. Therefore, $\sum_{q=g}^{h-1} d_q = \sum_{q=a}^{b-1} d_q$ follows. Consider the two cases below:

Case 1. $g < a$.

Consider the two subcases below:

Case 1.1. $h \leq a$.

In this case, we have $g \leq h-1 < a \leq b-1$ such that $\sum_{q=g}^{h-1} d_q = \sum_{q=a}^{b-1} d_q$, thus contradicting the system assumption.

Case 1.2. $h > a$.

Note that we must have $h < b$ in this case. Thus, $h \leq b-1$. The fact that $\sum_{q=a}^{b-1} d_q = \sum_{q=g}^{h-1} d_q$ leads to $\sum_{q=h}^{b-1} d_q = \sum_{q=g}^{a-1} d_q$. Hence, we have $g \leq a-1 < h \leq b-1$ such that $\sum_{q=g}^{a-1} d_q = \sum_{q=h}^{b-1} d_q$, thus contradicting the system assumption.

Case 2. $g > a$.

An argument similar to Case 1 applies in this case.

Next, prove that, for any processor P_i, $CS_i \cap CS_r = \phi$ for any $P_r \in CS_i$. Again, without loss of generality, assume that $P_i = P_0$. By contradiction, assume that there is some $P_{m_a} \in CS_0$, $0 \leq a \leq k-1$, such that $CS_0 \cap CS_{m_a} \neq \phi$. Let P_{m_b} be any processor in $CS_0 \cap CS_{m_a}$. Since $P_{m_b} \in CS_{m_a}$, there must be some g, $0 \leq g \leq k-1$, such that $(m_g + m_a) \bmod n = m_b$. However, since $d+1 \leq m_b, m_g, m_a \leq 2d+1$, it must be true that $2d+2 \leq m_g + m_a \leq 4d+2$. Considering $n \geq 3d+2$, it is clear that $(m_g + m_a) \bmod n$ falls in the range from $2d+2$ to d on the circle, i.e., $(m_g + m_a) \bmod n \in [2d+2, 2d+3, \ldots, n-1, 0, 1, \ldots, d-1, d]$. Thus, the equation $(m_g + m_a) \bmod n = m_b$ cannot possibly hold, and therefore, a contradiction arises.

According to Corollary 1, the safe recovery criterion is satisfied. □

Theorem 4 states that a solution for CS_0 can be derived by finding a PSR sequence of $k-1$ positive integers, $d_0, d_1, d_2, \ldots, d_{k-2}$, such that $n \geq 3d+2$, with $d = \sum_{i=0}^{k-2} d_i$. Note that, for a given k, a solution of CS_0 for n is also a solution for $n+1$ and above. Hence, for a given k, the minimum value of n for which there exists a solution for CS_0 that satisfies the safe recovery criterion can be found by first identifying a PSR sequence $d_0, d_1, d_2, \ldots, d_{k-2}$ such that $d = \sum_{i=0}^{k-2} d_i$ is minimum. Then, $n = 3d+2$ gives the minimum n. For pragmatic purpose, Table 3 lists the minimum n, along with one possible sequence of d_i's, for $k = 2$ to 10.

5 Implementation and Performance Analysis

5.1 Overhead Analysis

This section analyzes the overhead associated with the proposed diskless checkpointing scheme. Comparing this scheme with the Reed-Solomon coding-based approach is difficult because they checkpoint in different ways and have different requirements. However, this section shows the corresponding overhead for the Reed-Solomon coding-based scheme for reference. Assume that each processor can send a message to another processor and simultaneously receive a message from a possibly different processor. In addition, any two distinct communications can take place at the same time. There are two major types of overhead for checkpointing and recovery operations: time and memory space. The following analysis assumes that the size of each checkpoint is S_{ckp} words.

Consider the operation of generating parity checkpoint data when a new checkpoint is taken by each of the n processors. In the proposed scheme, each processor must send a local checkpoint to k checkpoint storage nodes and

TABLE 3
Minimum n and One Possible PSR Sequence for $k = 2$ to 10

k	Possible $d_0, \cdots, d_{k-2}$	Minimum d	Minimum n
2	1	1	5
3	1, 2	3	11
4	1, 3, 2	6	20
5	1,3,5,2	11	35
6	1,7,3,2,4	17	53
7	1,3,6,8,5,2	25	77
8	1,3,5,6,7,10,2	34	104
9	1,4,7,13,2,8,6,3	44	134
10	1,5,4,13,3,8,7,12,2	55	167

receives from its k checkpoint coverage nodes their respective local checkpoints. The entire process takes k steps. In the first step, every processor P_i sends a copy of its local checkpoint to the checkpoint storage node $P_{(m_0+i) \bmod n}$, where m_0 is defined in Theorem 4. Due to the cyclic design property, P_i will be the destination of exactly one of these data transmissions. That is, no two transmissions are addressed to the same destination node. Therefore, each P_i receives a checkpoint from one of its checkpoint coverage nodes. P_i stores the checkpoint in its memory. In the second step, each processor P_i sends a second copy of its local checkpoint to the checkpoint storage node $P_{(m_1+i) \bmod n}$, where m_1 is defined in Theorem 4. Again, P_i receives a checkpoint from a second checkpoint coverage node. P_i will then XOR this checkpoint into the one received in the first step. P_i only keeps the resulting parity checkpoint data. Each P_i performs a similar transmission-receiving-XOR operation for the remaining $k-2$ steps. Note that each P_i can perform the transmission of a checkpoint and XOR operation in parallel. Let T_{ckp} denote the time required to complete the generation of the parity data for the entire system. Assuming that computation and transmission do not overlap at a processor, T_{ckp} is given by

$$T_{ckp} = k \cdot \frac{S_{ckp}}{R_{net}} + (k-1) \cdot \frac{S_{ckp}}{R_{XOR}},$$

where R_{net} represents the bandwidth of the network and R_{XOR} represents the rate of performing XOR operation. If computation and transmission can be performed in parallel, T_{ckp} can be reduced to

$$T_{ckp} = 2 \cdot \frac{S_{ckp}}{R_{net}} + (k-2) \cdot \max\left(\frac{S_{ckp}}{R_{net}}, \frac{S_{ckp}}{R_{XOR}}\right) + \frac{S_{ckp}}{R_{XOR}}.$$

For the Reed-Solomon coding-based scheme, T_{ckp} is given as [22]

$$T_{ckp} = k \cdot \left(\frac{S_{ckp}}{R_{net}} \cdot (\log n + 1) + \frac{S_{ckp}}{R_{XOR}} \cdot \log n\right) + (k-1) \cdot \frac{S_{ckp}}{R_{GFmul}},$$

where R_{GFmul} represents the rate of performing Galois Field multiplications.

Next, consider the failure recovery operation. Suppose that k failures occur simultaneously in the system. Since the proposed scheme satisfies the safe recovery criterion, each of the k failed processors can be recovered by a distinct surviving checkpoint storage node. Let P_i be a failed processor and assume that $P_r \in CS_i$ is to help P_i recover its previous checkpoint. P_r will ask all the processors, except P_i, in CC_r to send their previous checkpoints, one by one. Upon receiving any of these checkpoints, P_r then XOR's the checkpoint into the parity checkpoint data kept by itself. This operation can be completed in $k-1$ steps. P_r then sends the recovered checkpoint to P_i for recovery operation. Assuming that computation and transmission do not overlap, the time it takes to recover the previous checkpoint for P_i, denoted by T_{rcv}, is

$$T_{rcv} = k \cdot \frac{S_{ckp}}{R_{net}} + (k-1) \cdot \frac{S_{ckp}}{R_{XOR}}.$$

If computation and transmission can be performed in parallel, T_{rcv} can be reduced to

$$T_{rcv} = 2 \cdot \frac{S_{ckp}}{R_{net}} + (k-2) \cdot \max\left(\frac{S_{ckp}}{R_{net}}, \frac{S_{ckp}}{R_{XOR}}\right) + \frac{S_{ckp}}{R_{XOR}}.$$

For the Reed-Solomon coding-based scheme, T_{rcv} should be slightly greater than T_{ckp} [22]. Since the checkpointing and recovery operations involve XOR only, the proposed scheme does not require the use of any arithmetic table to facilitate the computation.

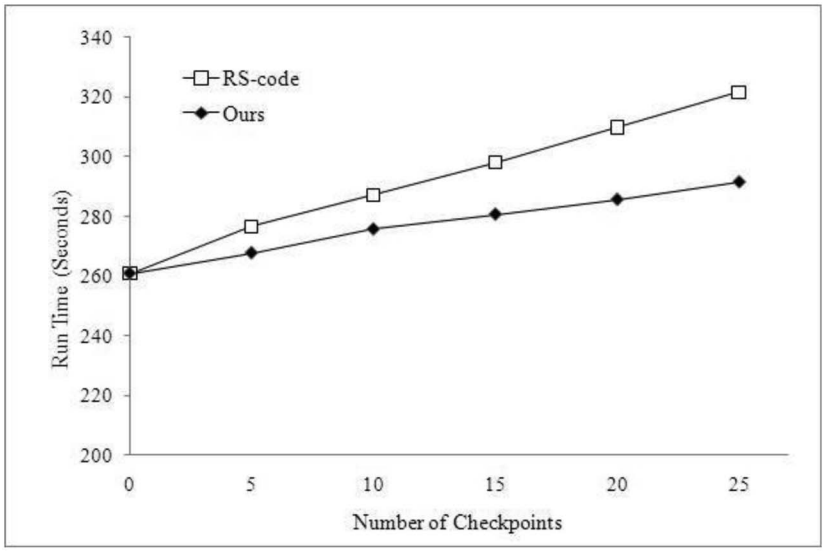


Fig. 9. Average runtime versus number of checkpoints taken in each run.

5.2 Performance Evaluation

This study includes experiments to evaluate the time overhead associated with the proposed scheme. The framework of the experiments is based on the Message Passing Interface (MPICH2-1.2.1p1) with Berkeley Lab Checkpoint/Restart (BLCR-0.8.2) [2]. The experiments in this study used 11 desktop PCs, each of which had at least 512 MB of memory and ran a Linux operating system. The nodes were connected by a Fast Ethernet switch. The value of k, the allowable number of simultaneous failures, was set to three. The application program used in the experiments was the Preconditioned Conjugate Gradient (PCG) algorithm [5]. We used the bcsstk23 matrix [17] in PCG and had made PCG run for 53,278 iterations in each run. The system took checkpoints at specified checkpoint intervals. This study compares experimental results with the Reed-Solomon coding-based method for which three extra checkpoint processors are employed and the Jerasure library [29] is used in the implementation.

This study first measures the time required to run PCG with and without checkpointing operation. Fig. 9 illustrates the average runtime for the schemes versus number of checkpoints taken during each run of the application program. In this figure, the number of checkpoints varies from 0 to 25. For both schemes, the PCG runtime grows linearly with the number of checkpoints taken by the system. This figure clearly shows that the proposed scheme incurs a smaller checkpointing time overhead than the Reed-Solomon coding-based method. To further illustrate the time overhead introduced by the checkpointing operation, Table 4 shows the average time overhead incurred by a checkpoint for the same set of experiments.

This study also measures the time it takes to recover from three failures, which is the largest number of simultaneous failures allowed by these schemes. Table 5 shows the average recovery time for the two schemes. This study defines the recovery time as the time it takes the schemes to recover previous checkpoints for the three failed processors and to have the failed processors receive the recovered checkpoints. Table 5 shows that the proposed scheme has a much shorter recovery time than the Reed-Solomon coding-based approach.

6 Conclusion and Future Research

This study addresses diskless checkpointing issues in a distributed or parallel computing environment and presents a new approach to enhancing neighbor-based schemes to tolerate multiple failures. The proposed scheme is unique in that it only uses simple XOR operations for checkpointing and failure recovery and does not require dedicated

TABLE 4
Average Time Overhead (in Seconds) for a Checkpoint

# of ckps / Schemes	5	10	15	20	25
RS-code	3.14	2.64	2.48	2.45	2.43
Ours	1.37	1.48	1.32	1.23	1.22

TABLE 5
Average Recovery Time (in Seconds) for the Schemes

Schemes	Recovery Time (seconds)
RS-code	2.13
Ours	0.61

checkpoint processors. This method allows checkpoint-related operations to be evenly distributed among all processors, achieving good load balance. This study identifies the necessary and sufficient condition for the concept of safe recovery criterion and presents a design method for constructing the checkpoint storage node sets that meet the safe recovery criterion. The proposed design method is practically useful for many existing systems.

Future research should follow several directions. One of them is seeking a tight bound on the minimum number of processors that can satisfy the safe recovery criterion for a given k, the number of failures. An approach that differs from the cyclic design is one possible solution, while another is a more sophisticated arrangement of the elements in CS_0. Future research should also investigate the possibility of slightly relaxing the single-step recovery to achieve a lower bound on n for a given k.

ACKNOWLEDGMENTS

The authors are grateful to the anonymous referees for their valuable suggestions and comments which have helped improve the quality of the paper. This work was supported by the National Science Council, Taiwan, under grants NSC 95-2221-E-011-093-MY3 and 97-2221-E-011-070-MY3.

REFERENCES

[1] N.R. Adiga et al., "An Overview of the BlueGene/L Supercomputer," *Proc. ACM/IEEE Symp. Supercomputing (SC '02)*, pp. 60-60, Nov. 2002.
[2] Berkeley Lab Checkpoint/Restart (BLCR), https://ftg.lbl.gov/CheckpointRestart, 2010.
[3] Z. Chen, G.E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra, "Fault Tolerant High Performance Computing by a Coding Approach," *Proc. ACM Symp. Principles and Practice of Parallel Programming (PPoPP '05)*, pp. 213-223, June 2005.
[4] Z. Chen and J. Dongarra, "A Scalable Checkpoint Encoding Algorithm for Diskless Checkpointing," *Proc. IEEE Symp. High Assurance Systems Eng. Symp. (HASE '08)*, pp. 71-79, Dec. 2008.
[5] Z. Chen and J. Dongarra, "Highly Scalable Self-Healing Algorithms for High Performance Scientific Computing," *IEEE Trans. Computers*, vol. 58, no. 11, pp. 1512-1524, Nov. 2009.
[6] G.-M. Chiu and C.-R. Young, "Efficient Rollback-Recovery Technique in Distributed Computing Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 7, no. 6, pp. 565-577, June 1996.
[7] J.-F. Chiu and G.-M. Chiu, "Hardware-Supported Asynchronous Checkpointing Scheme," *IEE Proc.—Computers and Digital Techniques*, vol. 145, no. 2, pp. 109-115, Mar. 1998.
[8] J.-F. Chiu and G.-M. Chiu, "Placing Forced Checkpoints in Distributed Real-Time Embedded Systems," *J. Computing and Control Eng.*, vol. 13, no. 4, pp. 197-205, Aug. 2002.
[9] J.-F. Chiu and W.-H. Hao, "Mutual-Aid: Diskless Checkpointing Scheme for Tolerating Double Faults," *Proc. IEEE Symp. High Performance Computing and Comm. (HPCC '08)*, pp. 540-547, Sept. 2008.
[10] T.-C. Chiueh and P. Deng, "Evaluation of Checkpoint Mechanisms for Massively Parallel Machines," *Proc. IEEE Symp. Fault Tolerant Computing (FTCS '96)*, pp. 370-379, June 1996.
[11] E. Elnozahy, L. Alvisi, Y. Wang, and D. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," *ACM Computing Surveys*, vol. 34, no. 3, pp. 375-408, Sept. 2002.
[12] E. Elnozahy and J. Plank, "Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Rollback-Recovery," *IEEE Trans. Dependable and Secure Computing*, vol. 1, no. 2, pp. 97-108, Apr./June 2004.
[13] E.N. Elnozahy, D.B. Johnson, and W. Zwaenepoel, "The Performance of Consistent Checkpointing," *Proc. IEEE Symp. Reliable Distributed Systems (RDS '92)*, pp. 39-47, Oct. 1992.
[14] G.H. Forman and J. Zahorjan, "The Challenges of Mobile Computing," *Computer*, vol. 27, no. 4, pp. 38-47, Apr. 1994.
[15] S.I. Feldman and C.B. Brown, "Igor: A System for Program Debugging via Reversible Execution," *ACM SIPLAN Notices, Workshop Parallel and Distributed Debugging (PADD '88)*, pp. 112-123, Jan. 1989.
[16] J.M. Hélary, A. Mostefaoui, R.H. Netzer, and M. Raynal, "Preventing Useless Checkpoints in Distributed Computations," *Proc. IEEE Symp. Reliable Distributed Systems (RDS '97)*, pp. 183-190, Oct. 1997.
[17] http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/bcsstruc3/bcsstk23.html, 2010.
[18] T.-Y. Juang and M.-C. Liu, "An Efficient Asynchronous Recovery Algorithm in Wireless Mobile Ad Hoc Networks," *J. Wireless Internet: Appl. and Systems*, vol. 3, no. 2, pp. 147-155, 2002.
[19] C.-J. Li and W.K. Fuchs, "CATCH—Compiler-Assisted Techniques for Checkpointing," *Proc. IEEE Symp. Fault Tolerant Computing (FTCS '90)*, pp. 74-81, 1990.
[20] J. Luo, L. Xu, and J.S. Plank, "An Efficient XOR-Scheduling Algorithm for Erasure Code Encoding," *Proc. IEEE Symp. Dependable Systems and Networks (DSN '09)*, pp. 504-513, June 2009.
[21] Q.M. Malluhi and W.E. Johnston, "Coding for High Availability of a Distributed-Parallel Storage System," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 12, pp. 1237-1252, Dec. 1998.
[22] J.S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-Like Systems," *Software—Practice and Experience*, vol. 27, no. 9, pp. 995-1012, Sept. 1997.
[23] J.S. Plank, "A New MDS Erasure Code for RAID-6," Technical Report CS-07-602, Dept. of Electrical Eng. and Computer Science, Univ. of Tennessee, Sept. 2007.
[24] J.S. Plank, Y. Kim, and J. Dongarra, "Algorithm-Based Diskless Checkpointing for Fault Tolerant Matrix Operations," *Proc. IEEE Symp. Fault-Tolerant Computing (FTCS '95)*, pp. 351-360, June 1995.
[25] J.S. Plank, Y. Kim, and J. Dongarra, "Fault-Tolerant Matrix Operations for Networks of Workstations Using Diskless Checkpointing," *J. Parallel Distributed Computing*, vol. 43, no. 2, pp. 125-138, 1997.
[26] J.S. Plank and K. Li, "Faster Checkpointing with N + 1 Parity," *Proc. IEEE Symp. Fault-Tolerant Computing (FTCS '94)*, pp. 288-297, June 1994.
[27] J.S. Plank, K. Li, and M.A. Puening, "Diskless Checkpointing," *IEEE Trans. Parallel Distributed Systems*, vol. 9, no. 10, pp. 972-986, Oct. 1998.
[28] J.S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent Checkpointing under Unix," *Proc. Usenix Winter 1995 Technical Conf.*, pp. 213-223, Jan. 1995.
[29] J.S. Plank, S. Simmerman, and C.D. Schuman, "Jerasure: A Library in C/C++ Facilitating Erasure Coding for Storage Applications—Version 1.2," Technical Report CS-08-627, Univ. of Tennessee, Aug. 2008.
[30] J.S. Plank and M.G. Thomason, "Processor Allocation and Checkpoint Interval Selection in Cluster Computing Systems," *J. Parallel Distributed Computing*, vol. 61, no. 11, pp. 1570-1590, Nov. 2001.
[31] R.D. Schlichting and F.B. Schneider, "Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems," *ACM Trans. Computer Systems*, vol. 1, no. 3, pp. 222-238, Aug. 1983.
[32] L.M. Silva and J.G. Silva, "An Experimental Study about Diskless Checkpointing," *Proc. Euromicro Conf. (EUROMICRO '98)*, pp. 395-402, Aug. 1998.

[33] L.M. Silva and J.G. Silva, "Using Two-Level Stable Storage for Efficient Checkpointing," *IEE Proc.—Software,* vol. 145, no. 6, pp. 198-202, Dec. 1998.
[34] J. Tsai, S.Y. Kuo, and Y.M. Wang, "More Properties of Communication-Induced Checkpointing Protocols with Rollback-Dependency Trackability," *J. Information Science and Eng.,* vol. 21, no. 2, pp. 239-257, Mar. 2005.
[35] J. Tsai, C.Y. Lin, and S.Y. Kuo, "Adaptive Communication-Induced Checkpointing Protocols with Domino-Effect Freedom," *J. Information Science and Eng.,* vol. 20, no. 5, pp. 885-901, Sept. 2004.
[36] N.H. Vaidya, "A Case for Two-Level Recovery Schemes," *IEEE Trans. Computers,* vol. 47, no. 6, pp. 656-666, June 1998.
[37] Y.M. Wang, "Consistent Global Checkpoints that Contain a Set of Local Checkpoints," *IEEE Trans. Computers,* vol. 46, no. 4, pp. 456-468, Apr. 1997.
[38] S. Yi, J. Heo, Y. Cho, and J. Hong, "Adaptive Mobile Checkpointing Facility for Wireless Sensor Networks," *Lecture Notes in Computer Science,* pp. 701-709, Springer-Verlag, Apr. 2006.
[39] J.W. Young, "A First Order Approximation to the Optimum Checkpoint Interval," *Comm. ACM,* vol. 17, no. 9, pp. 530-531, Sept. 1974.

Ge-Ming Chiu received the BS degree from the National Cheng Kung University, Taiwan, in 1976, the MS degree from the Texas Tech University, in 1981, and the PhD degree from the University of Southern California, in 1991, all in electrical engineering. He is currently a professor in the Department of Computer Science and Information Engineering at the National Taiwan University of Science and Technology, Taipei, Taiwan. While most of his previous research was focused on parallel architecture, distributed computing, and fault tolerance, his current research interests also include mobile computing, mobile ad hoc networks, and sensor networks. He is a member of the IEEE Computer Society.

Jane-Ferng Chiu received the BS degree in industrial education from the National Kaohsiung Normal University, in 1979, and the MA and PhD degrees in electrical engineering from the National Taiwan University of Science and Technology, in 1994 and 2002, respectively. His research work focuses on dependable distributed computing. He is currently an assistant professor in the Department of Information Technology and Communication, Tungnan University, Taipei, Taiwan.