2012 IEEE 14th International Conference on High Performance Computing and Communications

Double Mutual-Aid Checkpointing for Fast Recovery

Jane-Ferng Chiu
Department of Information Technology and Communication,
Tungnan University,
New Taipei 22202, Taiwan, R.O.C.
E-mail: jfchiu@mail.tnu.edu.tw

***Abstract*—Because of the enlarging system size and the increasing number of processors, the probability of errors and multiple simultaneously failures become the norm rather than the exception. Therefore, to tolerate multiple failures is indispensable. Normally, most diskless checkpointing need the maximum recovery overhead no mater how many failures happen at the same time. However, a small number of processors' failures happen more frequently than the worse case. This study resolves the dilemma between more fault tolerance and fast recovery by presenting a novel diskless checkpointing which makes use of double mutual-aid checkpoints. It not only gives the necessary and sufficient condition but also proposes a method for determination the setting of double mutual-aid checkpoints.**

Keywords-diskless checkpointing, tolerate multiple failures, fast recovery

I. INTRODUCTION

It is fundamental that cloud computing systems can provide high performance and dependable services. Thus, the cloud platforms composed from a pool of computer resources typically deploying into clusters of a massive number of servers hosted in data centers. As high performance clusters continue to grow in size and the mean time between failures shrinks. With the increasing probability of fault occurrences for a large scale cloud computing, to tolerate multiple failures is essential. Checkpoint/restart [8] is the ability to save the state of a running application. Because of that it can later resume its execution from the time when it was checkpointed on the same or a different machine. However, taking checkpoint into disks needs unacceptable time for large number nodes.

Diskless checkpoint [18] saves its checkpoint to the memory for recovery since memory accessing is much faster than disk accessing. The potentially low checkpoint overhead and faster restart should allow us to achieve better performance than traditional disk-based checkpoint schemes. However, the encoding time, the dedicated resources (the spares) and the memory overhead, which imposed by diskless checkpoint, are significant obstacles against its adoption. Using the results from [1], the overhead and the recovery time for a diskless checkpoint scheme increase linearly with the number of failures the scheme is capable of handling.

There are many ways [5],[8] to reduce checkpoint overhead. For example, a processor takes a checkpoint by selecting idle time or by minimizing checkpoint size. However, failure time is random. In [13] experience, most failures only disable one or two nodes at a time, and multi-node failures often disable nodes in a predictable pattern. [13] shows the probability of multiple failures on Tsubame, as we can see 95% of them are single failures.

However, both tolerating multiple faults and fast recovery are two important features for availability. These work address one of the basic questions in diskless checkpointing studies: How can systems cope effectively for tolerating number failures and recover rapidly while a single or double node failure?

The rest of the paper is organized as follows. Section II discusses some background and related work. The basics of our scheme are presented in Section III and details of its implementation in Section IV. Overhead analysis is provided in Section V. Finally, Section VI summarizes the contribution of our approach and thoughts for future work.

II. RELATED WORK

Checkpointing using stable storage has been studied extensively in the literature [8]. Since stable storage accessing is a considerable overhead, the number of checkpoints for taking is restricted.

Diskless checkpointing approach was first proposed in [17] to avoid the excessive overhead associated with stable storage operation. The various diskless checkpointing can be broadly classified into three categories: neighbor-based [20], parity-based [2],[14],[16], and Reed-Solomon code-based[18]. In the neighbor-based approach, each processor saves its checkpoints in the memory of another processor. 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 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. Hence, the amount of diskless checkpoint data that is stored is small in the parity-based technique. However, the time overhead depends on the number of application processors. The Reed-Solomon code-based diskless checkpointing 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 is no more than k.

978-0-7695-4749-7/12 $26.00 © 2012 IEEE
DOI 10.1109/HPCC.2012.148

Our previous study [6] 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. In [3], we have proposed a diskless checkpointing for tolerating multiple faults without spare nodes and demonstrated the necessary and sufficient condition.

III. BASICS OF THE PROPOSED SCHEME

A. System Model

Consider a high performance distributed computing system consisting of a collection of n processors (or nodes) P_0 , P_1 , P_2 ,…, P_{n-1} that are interconnected by a 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 , $0 \le i \le n-1$ 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 [19]. However, communication channels are assumed to be reliable. Each processor in the system takes checkpoints according to some criteria.

B. Basic operations

The basic idea is to add memory spaces bolster reliability in cloud computing. A processor sends some copies of its checkpoint to other processors for backup. When a processor fails, it could be recovered by retrieving its checkpoint from other processors. A system is k-fault tolerant if it can withstand k faults, so each processor has $k+1$ copies with fail-stop. Conversely, each node would be received many checkpoints from other processors and can lead to occupy much memory space. For reducing memory space, each processor exclusive OR those received checkpoints to an encoded result as the size of a single checkpoint. Since this parity result could help one of those engaged nodes for recovering, we call the result is *mutual checkpoint* [3],[6]. In other words, the checkpoint of failure node could be decoded by those checkpoints from other nodes and mutual checkpoint. From the probability calculation, single or double node failure is much higher than the other numbers of nodes failure. Therefore, the recovery cost of tolerating more failures will increase and reduce the overall system performance. To separate the two orientations will be easier to satisfy both considerations. The rapid recovery orientation is less engaged members for encoding and decoding. But, to tolerating more faults orientation is needed more engaged members for encoding and decoding.

To tolerate up to k arbitrary failures in a system, each processor P_i , $0 \le i \le n-1$, must send its checkpoint to a set of at least k other processors for storage. These k processors are called the *checkpoint storage nodes* (CS_i) of P_i . Meanwhile, P_i receives checkpoint data from other k processors and stores these checkpoints into P_i 's memory since no dedicated processors. The k processors who send and store checkpoint to P_i are called the *checkpoint coverage nodes* (CC_i) of P_i .

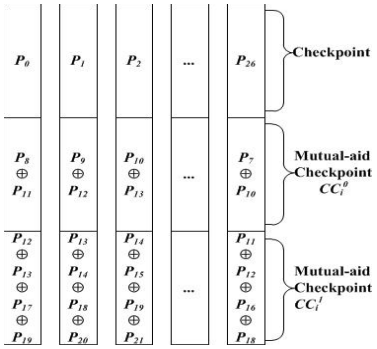

Figure 1. Double mutual-aid checkpoints

To overcome the recovery overhead is increased by k, we separate two section. In order to recover rapidly after single or double faults, we divided a checkpoint storage set to two subsets, CC_i^0 and CC_i^1 as Fig. 1. Thus, $P_r \in CS_i$ if and only if $P_i \in CC_r^0$ or $P_i \in CC_r^1$.

Consider fast recovery, our proposed failure recovery mechanism is single-step in the sense that each failed processor can be directly recovered by existing surviving processors. We adopt a coordinated checkpointing strategy. The checkpointing process involves two concurrent steps: (a) each processor packs up its system state and sends it to its checkpoint storage nodes. (b) each processor receives checkpoints from its checkpoint coverage nodes and parities those to double mutual-aid checkpoints.

To guarantee direct and fast recovery, the design of checkpoint storage node subset must ensure the following direct recovery criterion. The decoded work for recovery needs not only the mutual-aid checkpoint encoded the failure node but also other engaged members are alive. Therefore, the necessary condition for retrieved the checkpoint of failure node is the mutual-aid checkpoint encoded the failure node and other engaged members are alive.

Direct recovery criterion

For any processor P_f , given any subset of processors F such that $|F| \le k$ and $P_f \in F$, there always exists at least one processor $P_f \in CC_r^i; 0 \le i \le 1$ such that $P_r \notin F$ and, for any processor $P_l \in CC_r^i$, $P_l \ne P_f$, we have $P_l \notin F$.

C. Necessary condition

Three necessary conditions are for a failure processor P_f .

Theorem 1. If the direct recovery criterion is met, then for all processor P_x , $CC_x^0 \cap CC_x^1 = \phi$.

Proof. We prove the theorem by contradiction. Assume $CC_x^0 \cap CC_x^1 = P_z$. If P_x and P_z fail, the system can not tolerate k faults since there only $k-2$ copies of P_z in other nodes but P_x. Thus contradiction arises. ∎

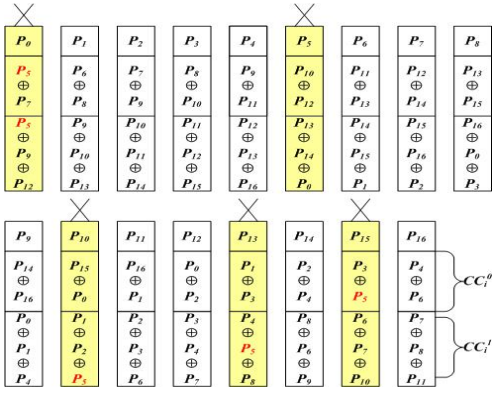

Figure 2. $CC_0^0 \cap CC_0^1 = P_5$

For example, as Fig. 2, a system has 17 processors and installs double mutual-aid checkpoints. Since $CC_0^0 \cap CC_0^1 = P_5$, P_5 could not recover when P_0, P_5, P_{10}, P_{13} and P_{15} fail. Therefore, the tolerate failure number is not five. When P_0, P_5, P_{10}, P_{13} and P_{15} fail, there are no recovery information for P_5. As result, the allocation could not tolerate five number failures.

Figure 3. $CC_0^0 \cap CC_{10}^1 = \{P_5, P_7\}$

Theorem 2. If the direct recovery criterion is met, then for all processor P_x and P_y, $\left|CC_x^i \cap CC_y^j\right| \le 1$ for $x \ne y$ and $i \ne j$.

Proof. We prove the theorem by contradiction. Assume $CC_x^i \cap CC_y^j = \{P_a, P_b\}$, other $k-2$ copies of P_a in P_f; $P_a \in CC_f^i$ and $f \ne a$ or $f \ne b$, if all P_f, P_a and P_b fail, P_a can not be decoded from CC_x^i and CC_y^j. ∎

For example, in Fig. 3, $CC_0^0 \cap CC_{10}^1 = \{P_5, P_7\}$. When P_5, P_7, P_{13}, P_{14} and P_{15} fail, P_5 can not be decoded from P_0 or P_{10} although both P_0 and P_{10} alive, since both P_5 and P_7 are not exit.

Theorem 3. If the direct recovery criterion is met, then P_x and P_y, $CC_x^i \cap CC_y^j = \phi$; for $x \ne y$, $P_y \in CC_x^i$ and $0 \le i \le 1; 0 \le j \le 1$

Proof. We prove the theorem by contradiction. Assume $CC_x^i \cap CC_y^j = P_z$. Therefore, since $P_y \in CC_x^i$, we have $\{P_y, P_z\} \subset CC_x^i$. When all P_f where $P_z \in CC_f^i$ and $f \ne x$ fail, if P_z also fail, P_z can not be decoded from CC_x^i since both P_z and P_y are not alive. ∎

For example, as Fig. 4, $CC_0^0 \cap CC_7^1 = P_5$; $P_7 \in CC_0^0$. When P_5, P_7, P_{13}, P_{14} and P_{15} fail, P_5 can not be decoded although P_0 alive since both P_5 and P_7 are not exit.

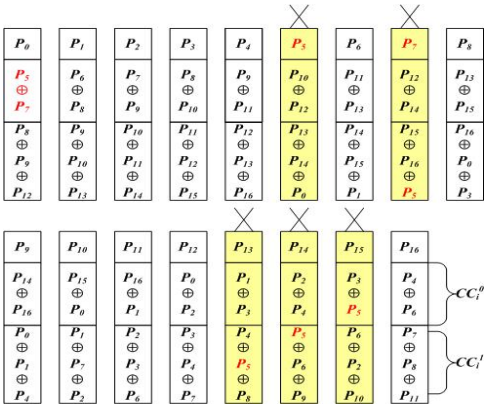

Figure 4. $CC_0^0 \cap CC_7^1 = P_5$; $P_7 \in CC_0^0$

D. Sufficient condition

Theorem 4. If the conditions specified in Theorems 1, 2 and 3 are all satisfied, then the direct recovery criterion is met.

Proof. Three cases are considered here.

Case 1: There exists no processor P_x such that $P_f \in CC_x^i$; $0 \le i \le 1$ such that $P_x \notin F$.

We prove the theorem by contradiction. Since $|F| = k$, there are *k-1* failure processors without P_f, it means $\left|CS_f\right| = k-1$ there are two $CC_x^0 \cap CC_x^1 = P_f$ and $P_x \in F$. Hence, to satisfy the direct recovery criterion for all processor P_x, $CC_x^0 \cap CC_x^1 = \phi$ is true.

Case 2: There exists only one processor P_r such that $P_f \in CC_r^i$; $0 \le i \le 1$ such that $P_r \notin F$.

Opposite to direct recovery criterion, there is at least one processor $P_l \in CC_r^i$. $P_l \ne P_f$; $P_l \in F$ and $l \ne r \ne f$. Since $|CS_f| = k$ and k failure nodes, there are two possible conditions; $P_f \notin CC_l^x$ or $P_f \in CC_l^x$; $0 \le x \le 1$. If $P_f \notin CC_l^x$, there exits the other processor $P_{r'}$ such that $P_f \in CC_{r'}^y$; $0 \le y \le 1$ such that $P_{r'} \notin F$. However, $P_{r'}$ is not in Case 2. Hence $P_f \in CC_l^x$, we have $CC_l^i \cap CC_r^j = P_f$. To satisfy the direct recovery criterion must be $CC_l^i \cap CC_r^j = \phi$.

Case 3: There exists more than one processor P_r such that $P_f \in CC_r^i$; $0 \le i \le 1$ such that $P_r \notin F$.

Assume there exists two processors P_r and P_l, $P_f \in CC_r^i$; $0 \le i \le 1$ and $P_f \in CC_l^j$; $0 \le j \le 1$ such that $P_r \notin F$ and $P_l \notin F$ .Therefore, $P_f \in CC_r^i \cap CC_l^j$. Contrary to direct recovery criterion, there exists least one processor $P_a \in CC_r^i$, $P_a \in F$, $a \ne f$ and at least one processor $P_b \in CC_l^j$, $P_b \in F$, $b \ne f$.

If $a = b$, then $|CC_r^i \cap CC_l^j| \ge 2$, this is contrary to all processor P_x and P_y, $|CC_x^i \cap CC_y^j| \le 1$ for $x \ne y$.

If $a \ne b$, there are two possible conditions, $\{P_a, P_b\} \cap CS_f \ne \phi$ or $\{P_a, P_b\} \cap CS_f = \phi$.

If $\{P_a, P_b\} \cap CS_f \ne \phi$, then $CC_r^i \cap CC_a^x = P_f$ or $CC_l^j \cap CC_b^y = P_f$; $0 \le x \le 1$; $0 \le y \le 1$. It is not true for all processor P_x and P_y, $|CC_x^i \cap CC_y^j| \le 1$ for $x \ne y$.

If $\{P_a, P_b\} \cap CS_f = \phi$, let $T = CS_f \cap F$ and $t = |T|$ then there $k - t$ processors $P_z \in CS_f$ such that $P_z \notin F$ .For each P_z, $P_f \in CC_z^i$, $0 \le i \le 1$ there exists at least one processor $P_c \in CC_z^i$; $c \ne f$ $P_c \in F$ and $P_c \cap CS_f = \phi$. However, $|P_c| \ge |P_z| = k - t$ is not true because $|F| = t + |P_z| + |P_f| > k$ which contradicts the assumption of $|F| \le k$. ∎

Corollary 1: The direct criterion is met, if and only if the conditions specified in Theorem 1, 2, 3 are both true.

IV. METHOD

Our approach of assigning checkpoint coverage node sets to processors in the system is based on the notion of cyclic design. That is, for any processor P_x, $0 \le x \le n-1$, $0 \le i \le 1$, CC_x^i can be derived from CC_0^0 as follows: $CC_x^i = \{P_r | r = (a+i) \bmod n, \forall P_a \in CC_0^0\}$

In other words, checkpoint coverage node sets can all be cyclically generated from CC_0^0. Once CC_x^i's are designated for all processors, their counterparts 's are also determined. The fact that all CC_x^i's and CS_i's are of the same size implies that the property of load balance can be achieved for all the processors in the system. Another advantage of the approach is that the task of checkpoint operation is evenly distributed among all processors. This property avoids possible bottleneck and is positive in terms of network scalability. With this approach, we only need to focus on the design of CC_0^0. The checkpoint storage node sets of both Fig. 5 are generated in this manner.

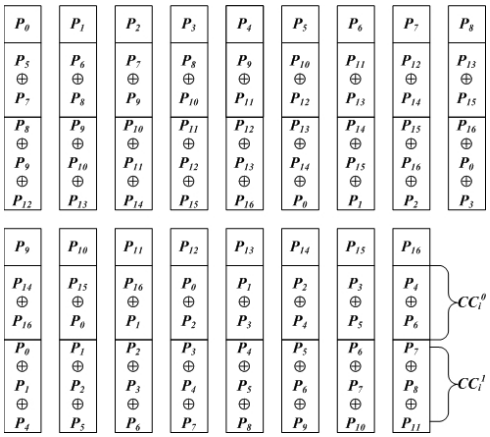

Figure 5. A design for N=17 and k=5

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

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., $k = 5$. First construct a PSR sequence of three (i.e. $k - 2$) positive integers.

Although there are many such PSR sequences, use $d_0 = 2$, $d_1 = 1$, and $d_2 = 3$, because this sequence gives the minimum sum, d, of six. This sequence defines the

spacing between two consecutive checkpoint coverage nodes of a processor. CC_0^0 is constructed as follows. The first checkpoint coverage node in CC_0^0 is P_5, whose subscript is one more than the value of Max_d. The other checkpoint coverage nodes in CC_0^0 is obtained by d_0. That is, the other checkpoint coverage node in CC_0^0 is P_7. The first checkpoint coverage nodes in CC_0^1 is one more than the value of the maximum of two checkpoint coverage node in CC_0^0 is P_8. The second is obtained by d_1, and d_2 as respective increments. That is, the second checkpoint coverage node in CC_0^1 is P_9, and the last one is P_{12}. Eventually, we have $CC_0^0 = \{P_5, P_7\}$ and $CC_0^1 = \{P_8, P_9, P_{12}\}$, 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 $3Max\_d + Min\_d + 3 = 27$. The design is illustrates in Fig. 5, which meets the safe recovery criterion. Formally, this design is based on the following theorem. Here, we only consider $k \geq 4$.

Theorem 5. Suppose that d_0, d_1, d_2, d_{k-3} is a PSR sequence of positive integers. Let $d\_CC_x^1 = \sum_{i=1}^{k-3} d_i$; $Max\_d = Max\left(d_0, d\_CC_x^1\right)$; $mid\_d = \min\left(d_0, d\_CC_x^1\right)$. If $n \geq 3Max\_d + \min\_d + 3$, the direct recovery criterion is met if $CC_0^0 = \{P_u, P_v | u = Max\_d + 1; v = u + d_0\}$ and $CC_0^1 = \{P_{m_i} \mid 0 \leq i \leq k-3; m_0 = v+1; m_i = m_0 + \sum_{i=1}^{k-3} d_i\}$

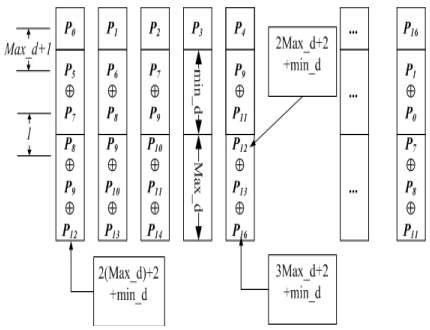

Figure 6. The design of Theorem 5

Proof. Fig. 6 shows the design of Theorem 5. Because PSR sequence, $Max\_d + 1$ and $n \geq 3Max\_d + \min\_d + 3$, this method is to meet the direct recovery criterion. It is not hard to show. ∎

For pragmatic purpose, we list the minimum *n*, along with one possible sequence of d_i 's, for k = 4 to 10 in TABLE I.

TABLE I. MINIMUN N AND ONE POSSIBLE PSR SEQUENCE

k	$d\_CC_x^0$	$d\_CC_x^1$	Max_d	Min_d	Min_N
4	1	2	2	1	10
5	2	1,3	4	2	17
6	3	1,4,2	7	3	27
7	6	1,3,5,2	11	6	42
8	14	1,7,3,2,4	17	14	68
9	11	1,3,6,8,5,2	25	11	89
10	16	1,3,5,6,7,10,2	34	16	121

V. OVERHEAD ANALYSIS

Double mutual-aid checkpointing can drastically increase fast recovery and overall reliability with little extra investment in memory. 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 our scheme, each processor must send a local checkpoint to *k* checkpoint storage nodes, and receives *k* checkpoints from P_i 's respective coverage nodes. The entire process takes *k* steps. 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 steps. 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

$$T_{ckp} = k\frac{S_{ckp}}{R_{net}} + (k-3)\frac{S_{ckp}}{R_{xor}}$$

where R_{net} represents the bandwidth of the network and R_{xor} represents the rate of performing XOR operation. Note that each P_i can perform the transmission of a checkpoint and XOR operation in parallel.

Next, consider the failure recovery operation. Suppose that *k* failures occur simultaneously in the system. Since the proposed scheme satisfies the direct recovery criterion, each of the *k* failed processors can be recovered by a distinct surviving checkpoint storage node. Let P_f be a failed processor and assume that $P_r \in CS_f$ is to help P_f recover its previous checkpoint. P_r will ask all the processors, except P_f, in CC_r^0 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_f for recovery operation. Assuming that computation and transmission do not overlap, the time it takes to recover the previous checkpoint for P_f , denoted by T_{rcv} , is

$$T_{rcv} = 2\frac{S_{ckp}}{R_{net}} + \frac{S_{ckp}}{R_{xor}} \text{ for } |P_f| \le 2$$

$$T_{rcv} \le (k-2)\frac{S_{ckp}}{R_{net}} + (k-3)\frac{S_{ckp}}{R_{xor}} \text{ for } 2 < |P_f| \le k$$

The overhead of double mutual-aid checkpointing is less than those of our previous work [3]. Since a small number of processors' failures happen more frequently than the worse case, the recovery overhead cut more obviously. Assume the chance of one or double processors' failure ten times other number failure at the same time. The MTTR is reduced by about 80% where $k = 8$. The diskless checkpointing scheme can reduce the total fault tolerance increase in runtime significantly compared with a one-level scheme [3].

This minimum n in TABLE I. is smaller than the number in [3]. For example, the minimum n is *104* for tolerating $k = 8$ failures simultaneously in [3], but n is 68 in this study. That is, double mutual-aid checkpointing is enabled to tolerate more faults.

VI. Conclusion and Future Research

This study addresses fast recovery issues for tolerating multiple faults in high performance computing environment. Double mutual-aid checkpointing enhances fast recovery as well as to tolerate multiple failures. The proposed scheme is only uses simple XOR operations for checkpointing and failure recovery, which does not require dedicated checkpoint processors. This study identifies the necessary and sufficient condition for the concept of direct recovery criterion, and presents a design method for constructing the checkpoint coverage node sets that meet the direct recovery criterion. The proposed design method is practically useful for many existing systems with fast recovery and tolerating multiple faults.

For our future work, we want to propose a multiple mutual-aid diskless checkpoint library and evaluate it with petascale applications in petascale machines.

Acknowledgment

This work is supported by the National Science Council, Taiwan, under grants NSC 100-2221-E-236 -008. We are grateful to the National Center for High-performance Computing for computer time and facilities.

References

[1] L. Bautista-Gomez, N. Maruyama, F. Cappello, S. Matsuoka, " Distributed Diskless Checkpoint for large scale systems", (CCGrid2010), Melbourne, Australia, May2010.

[2] 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.

[3] G.-M. Chiu and Jane-Ferng Chiu, "A New Diskless Checkpointing Approach for Multiple Processor Failures," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 4, pp. 481 – 493,July 2011.

[4] Jane-Feng Chiu and G.-M. Chiu, "Hardware-Supported Asynchronous Checkpointing Scheme," *IEE Proceedings - Computers and Digital Techniques*, vol. 145, no. 2, pp. 109-115, Mar. 1998.

[5] Jane-Feng Chiu and G.-M. Chiu, "Placing Forced Checkpoints in Distributed Real-Time Embedded Systems," *J. Computing & Control Engineering*, vol. 13, no. 4, pp. 197-205, Aug. 2002.

[6] J.-F. Chiu and W.-H. Hao, "Mutual-Aid: Diskless Checkpointing Scheme for Tolerating Double Faults," *Proc. IEEE Symp. High Performance Computing and Communications* (*HPCC'08)*, pp. 540-547, Sep. 2008.

[7] 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.

[8] 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, Sep. 2002.

[9] D. Hakkarinen and Z. Chen, "N-level diskless checkpointing," High Performance Computing and Communications, *10th IEEE International Conference*, vol. 0, pp. 384–391, 2009.

[10] 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.

[11] S. I. Feldman and C. B. Brown. "Igor: A System for Program Debugging via Reversible Execution," *ACM SIPLAN Notices, Workshop on Parallel and Distributed Debugging* (*PADD'88*), pp. 112-123, Jan. 1989.

[12] 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'2009*), pp. 504-513, June 2009.

[13] A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," *Proc. ACM/IEEE Symp. Supercomputing* (*SC'10*),, pp. 1–11, Nov. 2010.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] L. M. Silva and J. G. Silva, "An Experimental Study about Diskless Checkpointing," *Proc. Euromicro Conference (EUROMICRO'98)*, pp. 395-402, Aug. 1998.

[21] N. H. Vaidya, "A Case for Two-Level Recovery Schemes," *IEEE Trans. Computers*, vol. 47, no. 6, pp. 656-666, June 1998.