JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 33, 183-198 (2017)
DOI: 10.6688/JISE.2017.33.1.11

An Efficient Group-based Data Backup and Recovery Scheme in Cloud Computing Systems

PO-JEN CHUANG AND CHUNG-HSING WANG
Department of Electrical and Computer Engineering
Tamkang University
Tamsui, New Taipei City, 251 Taiwan
E-mail: pjchuang@ee.tku.edu.tw

In cloud computing systems with huge volumes of data, fault tolerance is of critical importance. To enhance data fault tolerance in cloud systems, we introduce a new group-based data backup and recovery scheme in this paper. The new scheme performs efficient diskless checkpointing practices to maintain data correctness via alternative processors upon processor failure. The basic idea is to place six processors in a transmission group, with each processor sending data to only two member processors. In face of processor failure, such a practice helps reduce the needed data backup volume and recovery time, and reaches up to 3/6 fault-tolerance ratios. Our scheme attains the performance gain mainly because (1) it allows a processor to receive only two backup data from the group – each processor hence performs only one XOR during data backup, and (2) all groups work independently in parallel so that the needed data backup and recovery time is reduced to that for a single group. To compare the performance of our scheme and related schemes, we carry out extended simulation runs with results indicating improved survival counts, fault-tolerance ratios and computation overhead for our scheme.

Keywords: cloud computing systems, data fault tolerance, diskless checkpointing, exclusive-OR, data backup and recovery

1. INTRODUCTION

The internet data transfer volumes have escalated at amazing speed due to rapid advances in information technology. To facilitate the processing of big data in cloud computing systems, data fault tolerance plays an important role. In cloud systems, when a processor turns faulty during transmission, existing checkpointing practices [1-19] can achieve fault tolerance by using alternative backup processors to ensure data correctness. Checkpointing practices may involve different time lengths for data backup and recovery due to different backup modes. For example, the Mutual-Aid Checkpointing scheme [9] employs mutual assistance to enhance data fault tolerance and improve traditional neighbor-based schemes. Its practice leads to a $k/(2k+1)$ fault-tolerance ratio, where k is the number of failed processors. In [10], the Distributed Diskless Checkpointing scheme uses matrices to perform data-backup computation and matrix multiplication to recover data. It first multiplies the initial data B by a chosen matrix M to get the encoded checkpoint C, $C=M*B$, and uses formula $M^{-1}*M*B = M^{-1}*C$ to recover B when the processor storing B fails. The PSR scheme [11], on the other hand, uses processor ID distances to produce a Partial Sum Restricted (PSR) sequence whose elements cannot be duplicated and the sum of any two (or more than two) consecutive elements will not equal any sub-

Received August 17, 2015; revised April 5, 2016; accepted July 17, 2016.
Communicated by Jan-Jan Wu.

immediately when encountering a faulty processor.

When processors or tolerable faults increase in cloud systems, data recovery time for checkpointing schemes may grow significantly. To reduce the needed data backup volume and recovery time, we employ the group-based concept to form an efficient new diskless checkpointing backup scheme in this paper. We employ the group-based concept to build the new scheme because it can practically trim down the overall data recovery time. As observed, when processors are divided into groups in a system, all groups can process in parallel without interfering with each other. It indicates, when we divide processors into groups, we can engage data backup and recovery in *all* groups by the same time that a *single* group takes. That is, no matter how many groups are in the system, we can always complete data backup and recovery by the same time for a single group. After close check, we realize a group with six processors can reach the best fault tolerance for group-based checkpointing practices. We hence decide to have six processors in a group and meanwhile allow each processor in a group to get only the backup data of two other processors in the same group. The design helps each processor obtain data backup in one XOR operation and enables the system to tolerate up to three faulty processors in a group of six, successfully lifting the fault-tolerance ratio to 3/6 or 50%.

Extended simulation runs are conducted to check the performance of our group-based checkpointing scheme and related schemes. The collected results show that, among all target schemes, our new scheme yields notably improved performance in *survival counts* and *fault-tolerance ratios*. In addition to the performance gain, we meanwhile reduce *the computation overhead* for performing data backup and recovery in each processor, thanks to the adopted group-based concept.

2. BACKGROUND STUDY

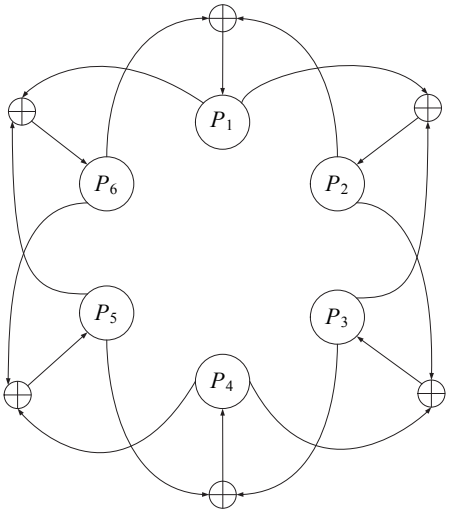
Fault tolerance is of particular importance in cloud servers with massive data flow. To tolerate faults, previous mechanisms tend to put backup data in disks at the cost of extraordinary overhead. More recent mechanisms, the so-called diskless checkpointing approaches, choose to put backup data in memory to reduce the overhead. Checkpointing, an advanced tool to enhance data fault tolerance in cloud systems, can help recover the data of failed processors whenever failures happen. Its performance – the obtained fault tolerance ratio – depends on certain key factors: the degree and limit of fault tolerance, the tradeoff between cost and efficiency, and the required recovery time. Checkpointing approaches may store a backup data item in an exclusive processor or in the buffer of another processor. Storing a backup item in an exclusive processor may cause data unrecoverable when the processor having the data and the exclusive processor with the backup data fail at the same time. By contrast, storing a backup item in the buffer of another processor may turn over more enhanced fault tolerance ratios and be a better practice.

In the following, we briefly introduce a few major checkpointing schemes to facilitate later discussions.

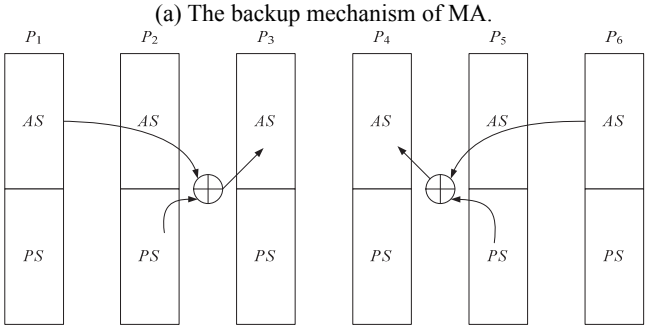
2.1 The Mutual-Aid (MA) Checkpointing Scheme [9]

The **MA** scheme is proposed for scenarios with more than five processors. It forms

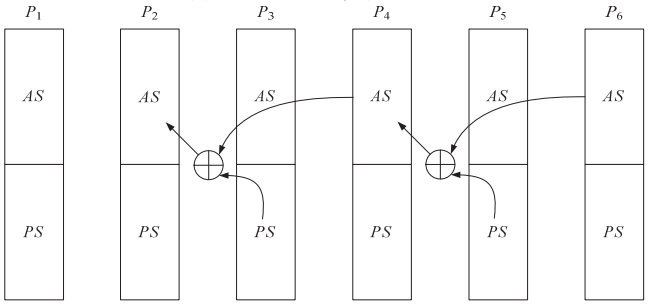
processors into a ring in which each processor P_i contains its own backup data and a buffer with the backup data of two adjacent processors. For example, in Fig. 1 (a) which depicts a scenario with six processors (P_1 to P_6), **MA** will exclusive-OR the data of P_1 and P_3, and store the result in the buffer of P_2. Hence each processor will contain its own backup data (AS) and the backup data of two neighbor processors (PS), as Fig. 1 (b) shows.



(a) The backup mechanism of MA.



(b) Data recovery (cases 1 and 2).



(c) Data recovery (case 3).

Fig. 1. Illustration of the **MA** scheme.

P_1	<P_1 checkpoint>	<P_4 encoded checkpoint>
P_2	<P_2 checkpoint>	<P_1 encoded checkpoint>
P_3	<P_3 checkpoint>	<P_2 encoded checkpoint>
P_4	<P_4 checkpoint>	<P_3 encoded checkpoint>

Fig. 2. The contents of each processor in the **DDC** scheme.

In this example, when a processor P_i fails, it can recover the data as follows.

(1) P_i (say P_3) can recover its data from XORing P_{i-1}'s (P_2's) PS and P_{i-2}'s (P_1's) AS (Fig. 1 (b)).
(2) P_i (say P_4) can recover its data from XORing P_{i+1}'s (P_5's) PS and P_{i+2}'s (P_6's) AS (Fig. 1 (b)).
(3) If P_i and P_{i+2} fail simultaneously, P_i can recover its data by other processors. For instance, when P_2 and P_4 both fail as in Fig. 1 (c), **MA** will perform indirect recovery: First XOR P_5's PS and P_6's AS to recover P_4's AS, and then XOR P_3's PS and recover P_4's AS to recover the data of P_2.
(4) **MA** will not recover the data of P_i if three consecutive processors simultaneously fail. That is, if P_2, P_3 and P_4 fail at the same time, **MA** will not recover the data of P_3 because the PS to recover the data is stored in P_2 or P_4.

2.2 The Distributed Diskless Checkpointing (DDC) Scheme [10]

The **DDC** scheme lets a processor P_i use matrix multiplication to produce the backup data and store it in the buffer of processor P_{i+1}. That is, P_i holds its own backup data and the encoded backup data of P_{i-1}, as Fig. 2 illustrates.

Based on the processor size, **DDC** will produce the matrix M with some mathematical properties to perform data backup and recovery. To generate m encoded checkpoints, it will produce a matrix with $k+m$ rows and k columns –k depends on the checkpoint data size – to recover up to m failures, as the left side of Fig. 3 shows. The matrix will be divided into $k*k$ (an identity matrix) and $m*k$ (a matrix M with some mathematical properties). The matrix size of the processor is k*1; the identity matrix I with the size $k*k$ is then generated. After performing matrix multiplication, **DDC** lets P_i store the produced encoded checkpoint data in the buffer of P_{i+1}, and maintains the original data by multiplying the identity matrix ($k*k$) by the processor matrix (k*1) to keep the original Ch_{11}~Ch_{k1} in the processor. Multiplying matrix M with the processor matrix will produce the encoded checkpoint data (C_{11}~C_{m1}).

When P_i fails, **DCC** can recover the data by the backup data stored in P_{i+1}. That is, it can work even when half of the processors fail, achieving as high as 50% fault tolerance ratios. It will, however, break down when two consecutive processors (say P_i and P_{i+1}) simultaneously fail as the data of P_i is beyond recovery if the buffer of P_{i+1} is faulty. **DDC** actually performs data backup by using the inverse matrix of matrix M. Take Fig. 3 as an example. When the processor storing B turns faulty, **DDC** will multiply the initial data B (processor matrix k*1–Ch_{11}~Ch_{k1}) by matrix M (with size m*k) to get the encoded checkpoint C=M*B (with elements C_{11}~C_{m1}) and use formula M^{-1}*M*B = M^{-1}*C to recover B.

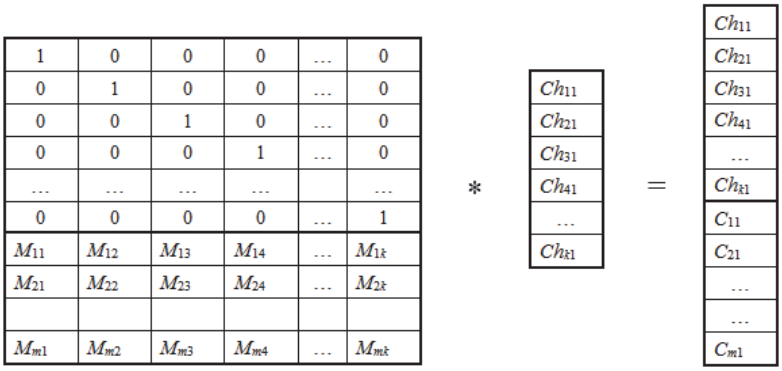


Fig. 3. Matrix multiplication for data backup in the **DDC** scheme.

2.3 The Partial Sum Restricted (PSR) Scheme [11]

The **PSR** scheme uses a partial sum restricted sequence rule to choose the interval distance for data backup and recovery. The rule restricts the interval distance number sequence in such a way that any two numbers cannot be the same and no number will equal the sum of any consecutive numbers. For example, to tolerate up to five failures, Pi will send the backup data to five specific processors which form four interval distances d_0~d_3. If d_0~d_3 is (2, 1, 9, 3), it breaks the PSR sequence rule; if (1, 3, 9, 6), it follows the rule. The purpose of the PSR sequence rule is to prevent the occurrence of duplicated data coverage. For instance, as P_2 and P_4 in Fig. 4 have the same data coverage processors P_0 and P_1, the failures of P_0 and P_1 become intolerable. That is, **PSR** cannot recover the lost data when a sequence breaks the rule.

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	$\{P_0, P_3\}$	$\{P_0, P_1\}$

P_0	P_1	P_2	P_3	P_4
P_3 $\oplus$ P_4	P_2 $\oplus$ P_3	P_0 $\oplus$ P_1	P_2 $\oplus$ P_4	P_0 $\oplus$ P_1

Fig. 4. Failed data recovery (not following the PSR sequence rule).

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

P_0	P_1	P_2	P_3	P_4
P_2 $\oplus$ P_3	P_3 $\oplus$ P_4	P_4 $\oplus$ P_0	P_0 $\oplus$ P_1	P_1 $\oplus$ P_2

Fig. 5. Successful data recovery (following the PSR sequence rule).

As mentioned, to tolerate up to k processor failures, **PSR** must have each processor (say P_i) send data to k specific processors whose IDs belong to processor ID set CS_i which includes P_i's checkpoint storage nodes and meanwhile receive data from k processors whose IDs belong to processor ID set CC_i which includes P_i's checkpoint coverage nodes. After receiving data from different processors, Pi will XOR the received data and store it in the buffer. Each processor hence has its CS and CC – formed by the PSR sequence rule – for data backup and recovery. Figs. 4 and 5 exhibit how **PSR** works or not works. Fig. 4 gives a case with five processors to tolerate up to 2 processor failures. We see that when P_0 and P_1 – both sending data to P_2 and P_4 – fail, **PSR** cannot use the data of P_1 to recover the data of P_0, *i.e.*, P_0 is unrecoverable. When the sequence rule has been observed as in the case in Fig. 5, **PSR** will succeed in recovering the data.

3. THE PROPOSED SCHEME

For existing checkpointing schemes, data recovery time may grow significantly with the increase of processor numbers or tolerable faults. To facilitate data recovery, *i.e.*, to reduce the required data backup volume and recovery time, when processor failures happen, we employ a group-based concept to set up a new and efficient diskless checkpointing scheme for cloud computing systems. In our new scheme, we put six processors – each with an ID number – in a group and let each processor (say P_i) send data to two other processors (say P_{i+2} and P_{i+3}) in the group. Each processor hence takes one XOR operation to backup its data, as Fig. 6 shows. When Pi fails, the design can always help recover its data (stored in the buffers of P_{i+2} and P_{i+3}) unless three consecutive processors P_i, P_{i+2} and P_{i+3} fail simultaneously – which is a rare case. Excluding the rare case, we can constantly restore the data lost due to processor failures even when three failures occur in the group. That is, we can tolerate up to three processor failures in a group, achieving a very satisfactory 3/6 or equivalently 50% fault tolerance ratio.

Fig. 6 gives as an example to illustrate our scheme. When P_0, P_1 and P_3 fail at the same time, we can respectively recover the data in P_0 by P_2 and P_5, the data in P_1 by P_4 and P_2, and the data in P_3 by P_5 and P_2. Note that the three recovery operations can be performed in parallel and take only one XOR, *i.e.*, they can be completed in one step. Even when the rare case of three consecutive processor failures happen, we can also restore the data – not in one but in three XORs (three times of XOR). In such a rare case, we need three steps to complete the data recovery: (1) recover the data in one failed processor, (2) recover the data in another failed processor by the recovered data in (1), and (3) recover the data in the last failed processor by the recovered data in (2). For example, if the three consecutive processors P_0, P_1 and P_2 fail at the same time, we will (1) use P_3 and P_5 to recover the data in P_2, (2) use P_2 and P_4 to recover the data in P_1, and (3) use P_1 and P_3 to recover the data in P_0. This example also reveals that, in face of three consecutive processor failures, we need at least six processors in a group to recover the lost data.

Note that we do not consider increasing the group processor amount (to over six) in order to tolerate more than three processor failures in a group because we let each processor send data to only two member processors in the group and will never recover the data of P_i when four consecutive processors P_i, P_{i+1}, P_{i+2} and P_{i+3} fail simultaneously – no matter how many processors are in the group, we always need the backup data in P_{i+2} or P_{i+3} to recover the data of P_i. The fact explicitly explains our choice of setting six

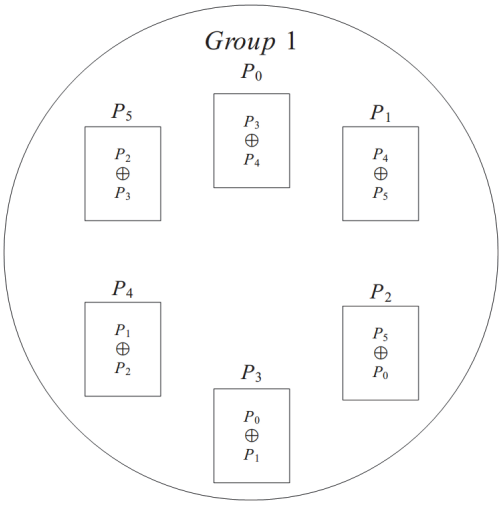


Fig. 6. An example group in our group-based scheme.

processors in a group.

Except for the specified rare case, our scheme can always recover the lost data at one XOR time. It can save significant data recovery time when the system has a large number of processors. The scheme is also advantageous in that its groups operate independently so that the failure in one group will not affect the performance of other groups. For instance, when a group in the system suffers four processor failures and is incapable of practicing data recovery, the other groups remain unaffected. The feature that all groups process in parallel and are free of mutual interference enables our scheme to yield the same performance regardless of the number of processors in the system. That is, no matter how many groups exist in the system, our scheme will constantly take the same data backup and recovery time as if there exists only one group.

For better illustration, please refer to the flowchart of our scheme in Fig. 7 and also the comparison among the schemes in Table 1.

Table 1. Comparison among various schemes.

Schemes	Backup and recovery operation	Sending the data of P_i to	P_i receives data from	Processors in groups	Parallel backup, recovery	Intolerable minimum failures
MA	XOR	P_{i-1}, P_{i+1}	P_{i-1}, P_{i+1}	No	No	Three consecutive failures
DDC	Matrix multiplication	P_{i+1} (encoded)	P_{i-1} (encoded)	No	No	Two consecutive failures
PSR	XOR	k processors (PSR rule)	k processors (PSR rule)	No	No	$k + 1$ failures
Group-based	XOR	P_{i+2}, P_{i+3}	P_{i-2}, P_{i-3}	Yes (group of 6)	Yes (All groups process in parallel)	Four consecutive failures in a group

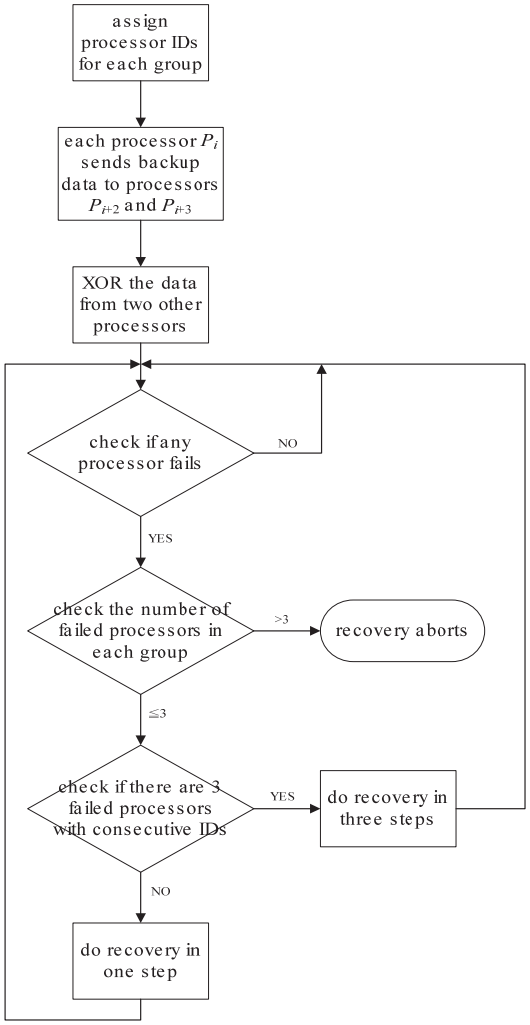


Fig. 7. The flowchart of our group-based scheme.

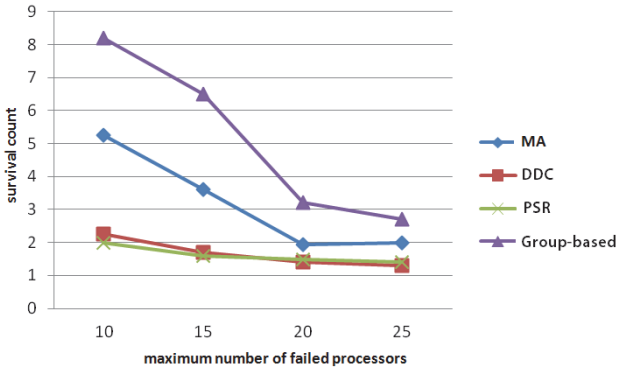


Fig. 8. *Survival counts* for various schemes.

4. SIMULATION RESULTS

As mentioned, we build our group-based data backup and recovery scheme to pursue more desirable performance than related schemes. In this section, we carry out extensive simulation runs to collect the results of *survival counts*, *fault-tolerance ratios* and *recovery time* for performance comparisons between our group-based scheme and other checkpointing schemes, including **MA**, **DDC** and **PSR**. The three indexes actually reveal the performance and cost of the four target schemes and also the pros and cons of their designs. To be more specific, *survival counts* and *fault tolerance ratios* are proper indicators for their data backup and recovery practices, and *recovery time* (which exhibits how fast the schemes recover the data) indicates the required cost.

4.1 *Survival Counts* vs. Maximum Numbers of Failures

In this simulation, we evaluate and compare *the survival counts* for our scheme and related schemes, including **MA**, **DDC** and **PSR**, in a network of 50 processors with uniformly distributed failed processors. The numbers of failed processors are uniformly distributed random numbers between 1 and the assumed maximum numbers 10, 15, 20 and 25 in the x-axis. Considering the bandwidth, operation speed and data size involved in the recovery, we assume a one-second delay for each scheme to recover the data of a failed processor. In the special case when one processor P_1 fails, needs another processor P_2 to recover the data but finds P_2 turns faulty in the next second, we assume P_1's data is unrecoverable, *i.e.*, no *survival count* will be added. Fig. 8 depicts *the survival counts* collected in the simulation for all schemes. As it shows, our group-based scheme yields more *survival counts* than other schemes, thanks to its feature design which can tolerate as many as three processor failures in a group of six processors to reach up to 3/6 fault tolerance ratios.

4.2 *Fault Tolerance Ratios* vs. Processor MTBF

Mean Time between Failures (MTBF) and reliability are often taken to depict processor performance. MTBF can actually be taken as an attribute or measure of hardware and software reliability [20]. It decides processor durability. For a repairable system, it indicates the average time between two consecutive processor failures, which is a proper performance parameter for system reliability and maintainability because the longer MTBF is, the more durable processors will be or the less likely they will fail. We therefore use MTBF to estimate *fault tolerance ratios* in this simulation. Fig. 9 gives the results of *fault-tolerance ratios* vs. processor MTBF. We assume the system has 100 processors and set the range of MTBF between 100 and 450 to see how it affects *the fault tolerance ratios* of different schemes (**PSR** is excluded from the evaluation because it cannot tolerate any failure under the assumed MTBF range.) The result in Fig. 9 exhibits clearly that *fault tolerance ratios* of each scheme grow with increased MTBF values. Among the schemes, **DDC** yields constantly the lowest ratios due to its "more stringent" conditions for fault tolerance. Our group-based scheme, by contrast, has "looser" conditions for fault tolerance and thus generates much higher ratios at any MTBF value.

4.3 Recovery Time Comparisons

Processor failure rates tend to grow after certain execution time. If we can restore the data lost upon processor failures as soon as possible, we may enhance system reliability significantly. This makes the following recovery time comparisons between schemes interesting and meaningful.

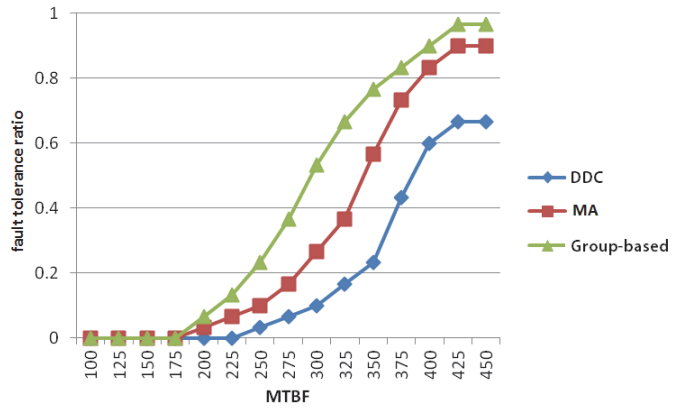


Fig. 9. *Fault-tolerance ratios* vs. processor MTBF for various schemes.

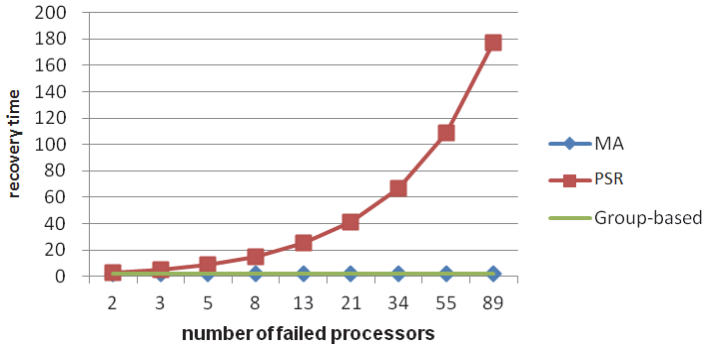


Fig. 10. Recovery time comparisons among schemes.

In **PSR**, the maximum tolerable failures will decide the number of XOR operations. In general, if the maximum tolerable failures are k, each processor buffer must store the data from k processors which will be combined into one by k–1 XOR operations. For example, when the maximum tolerable failures are two, **PSR** needs one XOR because each processor buffer stores the data from two other processors which can be combined into one by one XOR operation. For our new scheme and **MA**, each processor buffer will store only the data of two other processors, regardless of the maximum tolerable failures. That is, in all situations, both schemes will take only one XOR to combine the data.

Fig. 10 gives recovery time for different schemes. The numbers of failed processors in the x-axis are randomly generated. We assume that the data size of each processor is 1T bytes, the XOR rate is 10G Hz and the bandwidth is 10G bytes.

When the number of failed processors grows in the system, the result depicts significant rising recovery time for **PSR** but negligible rising time for our scheme and **MA**. The result is expectable because, no matter how many processor failures the system endures, our scheme and **MA** need only one XOR to achieve data recovery, whereas **PSR** needs k–1 XOR operations to accomplish the same job when facing k failed processors. It is also observed that, if the number of failed processors is fixed and we list increasing data sizes (instead of failed processors) in the x-axis, the schemes will yield similar trends of recovery time as in Fig. 10. That is, when data sizes grow, **PSR** also needs more XOR operations and more recovery time to achieve data recovery, in contrast to our scheme and **MA**. Note that **MA** performs as well as our group-based scheme in recovery time evaluation, but it trails behind us in *fault tolerance ratios* and *survival counts*.

4.4 The "Leftover" Processors Which Fail to Form a Group

By "leftover" processors, we indicate those processors whose number is below six and therefore insufficient to form a new group. To cover these processors in our operation, we put them into a special group and use the following counter measures to practice data backup and recovery.

(1) If there is only one leftover processor, the processor will be unrecoverable if it fails.
(2) If there are two leftover processors, they will back up each other's data. When one fails, we can recover the data by the other; if both fail, they become unrecoverable.
(3) If there are three leftover processors, the data of each processor will be stored in the other two. When one processor fails, the data can be recovered; when two fail simultaneously, they are unrecoverable.
(4) With four leftover processors, the data of each processor (say P_i) can be stored in two other processors (say P_{i+2} and P_{i+3}). If two processors with no consecutive ID numbers fail, we can still recover the data.
(5) With five leftover processors, we can back up the data of P_i in two other processors P_{i+2} and P_{i+3} and, when any two processors fail, we can always recover the data.

We carry out the simulation under the same environment specified above, except that the reliability is set at 0.8 and the numbers of processors are not multiples of six. Fig. 11 plots the results. It shows that, in systems with 12+n, 18+n, … and 54+n processors (n is a randomly generated number between 1 and 5), our group-based scheme also outperforms **DDC** and **MA** in *fault-tolerance ratios*.

As mentioned, with six processors in a group, our scheme can obtain up to 3/6 *fault tolerance ratios*. With five processors in a group, it is easy to derive that *the fault tolerance ratio* will be 2/5. It makes *the total fault tolerance ratio* (for the two groups) = (3+2)/(6+5) = 5/11. Instead of putting the leftover processors into a new group, we can merge them into any existing group of six processors to form a special group of more than six processors. For example, we can merge 5 leftover processors into an existing group to form a group of 11 processors and derive *the fault tolerance ratio* to be 6/11 (in one merged group), which is better than 5/11 (in two individual groups). In fact, we have analyzed and simulated *the fault tolerance ratios* with different numbers of leftover pro-

cessors for two situations: forming an *individual* new group or *merged* into an existing group. The results in Table 2 and Fig. 12 indicate overall better *fault tolerance ratios* for *merged* groups, regardless of the leftover numbers. It reveals that merging the leftover processors into an existing group is a better counter measure and right choice for our scheme.

Table 2. *Fault-tolerance ratios* with different leftover processors.

leftover processors	in an individual group	merged into another group
1	3/7	4/7
2	4/8	4/8
3	4/9	5/9
4	5/10	5/10
5	5/11	6/11

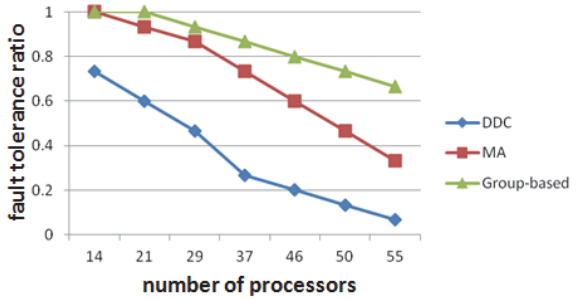


Fig. 11. *Fault-tolerance ratios* vs. numbers of processors not multiples of six.

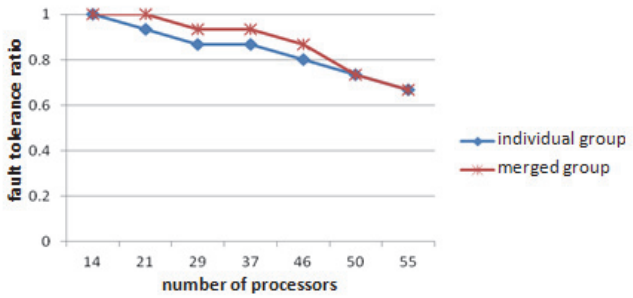


Fig. 12. *Fault tolerance ratios* vs. numbers of processors not multiples of six.

4.5 Other Discussions

In developing a new checkpointing scheme to outperform existing schemes, we put our focus on finding a better way to backup processor data and then use the backup data to recover the data lost due to processors failures. The above simulation results show that we have successfully built such a desired data backup and recovery scheme which advances related schemes in both performance and cost. More specifically, the obtained

higher *survival counts* and *fault-tolerance ratios* exhibit our performance gain in data backup and recovery practices, and the lower *recovery time* indicates we can recover the lost data faster, *i.e.*, with less cost. Our simulation does not include memory cost or communication cost because they are hardly involved in the evaluation of related schemes. If the two costs are included, we can expect they will be less for our scheme than for others due to the feature that we let each processor send data to only two other processors in the group.

As mentioned previously, data backup and recovery remains a critical issue in cloud computing systems. We observe that a number of new data backup and recovery schemes, such as [21-24], have been recently proposed to enhance the performance and efficiency of data backup and recovery in cloud systems. Among the schemes, some performs user authentication based on attributes and secret keys to secure data backup and recovery from the remote cloud server [21]. Some adopt cost effective filtration schemes to further reduce the required complexity and overheads, and meanwhile achieve faster remote recovery [22, 23]. They achieve the goal by eliminating redundancy and compressing similarity of data to transmit massive data in a more efficient way between long-distance data centers. There are also schemes (*e.g.*, [24]) which employ optimization data scheduling strategies to schedule data backup in order to balance such data recovery objectives as high data reliability, low backup cost and short recovery time. By doing so, they can reduce data backup cost and recovery time as much as possible. The above schemes, as we can see, attempt to tackle the data backup and recovery issue in cloud computing by different approaches. The fact reveals the importance of the topic and also the rich variety of possible solutions. We believe that, if these schemes are brought to work together with our group-based scheme, we will be able to further enhance both the performance and efficiency of data backup and recovery in cloud computing systems.

5. CONCLUSIONS

Data fault tolerance is critical for cloud computing systems with massive data volumes. When processor failures occur in the system, a checkpointing recovery mechanism can restore the lost data by alternative backup processors, to maintain data correctness and system performance. However, when processors or tolerable faults grow in the system, checkpointing approaches may require significant time to accomplish the data recovery task. Our research goal is to build an efficient group-based data backup and recovery scheme able to reduce data backup volumes and recovery time. The new scheme is built over two observations: (1) a group of six processors can attain optimal fault tolerance for group-based checkpointing practices and (2) all groups in a system can process in parallel without inter-influence. We hence involve a group-based concept to form the new diskless checkpointing scheme, in which each group contains six processors and each processor sends data to two member processors. By allowing a processor to receive only two backup data from two member processors, we enable each processor to attain backup data by a single XOR operation. The design helps us tolerate up to three faulty processors in a group of six processors, elevating fault tolerance ratios to as high as 3/6 or 50%. Extensive simulation runs have been conducted to evaluate the performance of our group-based data recovery scheme and related schemes. The results show that, in

contrast to other schemes, ours yields significantly better performance in *survival counts* and *fault tolerance ratios*. When performing the efficient data backup and recovery, we also succeed in reducing *the computation overhead* (which is big in existing schemes) in each processor. We notice that each data backup and recovery scheme may face the memory competition problem, such as the impact of memory competition from diskless checkpoints on the execution of applications. We consider memory competition a significant and interesting issue worth our further investigation in the future.

REFERENCES

1. 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," in *Proceedings of ACM Symposium on Principles and Practice of Parallel Programming*, 2005, pp. 213-223.
2. T.-C. Chiueh and P. Deng, "Evaluation of checkpoint mechanisms for massively parallel machines," in *Proceedings of IEEE Symposium on Fault Tolerant Computing*, 1996, pp. 370-379.
3. L. M. Silva and J. G. Silva, "An experimental study about diskless checkpointing," in *Proceedings of Euromicro Conference*, 1998, pp. 395-402.
4. J. S. Plank, Y. Kim, and J. Dongarra, "Algorithm-based diskless checkpointing for fault tolerant Matrix operations," in *Proceedings of IEEE Symposium on Fault-Tolerant Computing*, 1995, pp. 351-360.
5. J. S. Plank and K. Li, "Faster checkpointing with *N*+1 parity," in *Proceedings of IEEE Symposium on Fault-Tolerant Computing*, 1994, pp. 288-297.
6. J. S. Plank, K. Li, and M. A. Puening, "Diskless checkpointing," *IEEE Transactions on Parallel Distributed Systems*, Vol. 9, 1998, pp. 972-986.
7. Z. Chen and J. Dongarra, "A scalable checkpoint encoding algorithm for diskless checkpointing," in *Proceedings of IEEE Symposium on High Assurance Systems Engineering Symposium*, 2008, pp. 71-79.
8. Z. Chen and J. Dongarra, "Highly scalable self-healing algorithms for high performance scientific computing," *IEEE Transactions on Computers*, Vol. 58, 2009, pp. 1512-1524.
9. J.-F. Chiu and W.-H. Hao, "Mutual-aid: Diskless checkpointing scheme for tolerating double faults," in *Proceedings of IEEE Symposium on High Performance Computing and Communications*, 2008, pp. 540-547.
10. L. B. Gomez, N. Maruyama, F. Cappello, and S. Matsuoka, "Distributed diskless checkpoint for large scale systems," in *Proceedings of IEEE/ACM International Symposium on Cluster*, *Cloud and Grid Computing*, 2010, pp. 63-72.
11. G. M. Chiu and J. F. Chiu, "A new diskless checkpointing approach for multiple processor failures," *IEEE Transactions on Dependable and Secure Computing*, Vol. 8, 2011, pp. 481-493.
12. J. Plank, K. Li, and M. A. Puening, "Diskless checkpointing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, 1998, pp. 972-986.
13. Z. Chen, and *et al.*, "Testing and fault tolerance: Fault tolerant high performance computing by a coding approach," in *Proceedings of the 10th ACM SIGPLAN Sym-*

posium on PPoPP, 2011, pp. 93-103.
14. T.-C. Chiueh and P. Deng, "Evaluation of checkpoint mechanisms for massively parallel machines," in *Proceedings of Annual Symposium on Fault Tolerant Computing*, 1996, pp. 370-379.
15. J.-F. Chiu and G.-M. Chiu, "Hardware-supported asynchronous checkpointting scheme," *IEE Proceedings – Computers and Digital Techniques*, Vol. 145, 1998, pp. 109-115.
16. J. S. Plank, "A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems," *Software – Practice and Experience*, Vol. 27, 1997, pp. 995-1012.
17. W. Chen and J. Tsai, "Fault-tolerance implementation in typical distributed stream processing systems," *Journal of Information Science and Engineering*, Vol. 30, 2014, pp. 1167-1186.
18. J. Liao, "A transparent, incremental, concurrent checkpoint mechanism for real-time and interactive applications," *Journal of Information Science and Engineering*, Vol. 29, 2013, pp. 1285-1298.
19. P. K. Jaggi and A. K. Singh, "Log based recovery with low overhead for large mobile computing systems," *Journal of Information Science and Engineering*, Vol. 29, 2013, pp. 969-984.
20. M. Krasich, "How to estimate and Use MTTF/MTBF would the real MTBF please stand up?" in *Proceedings of Annual Reliability and Maintainability Symposium*, 2009, pp. 353-359.
21. M. Raje and D. Mukhopadhyay, "Algorithm for back-up and recovery of data stored on cloud along with authentication of the user," in *Proceedings of International Conference on Information Technology*, 2015, pp. 175-180.
22. Y. Hua, X. Liu, and D. Feng, "Neptune: Efficient remote communication services for cloud backups," in *Proceedings of IEEE Conference on Computer Communications*, 2014, pp. 844-852.
23. Y. Hua, X. Liu, and D. Feng, "Cost-efficient remote backup services for enterprise clouds," *IEEE Transactions on Industrial Informatics*, 2016, pp. 1650-1657.
24. Y. Gu, D. Wang, and C. Liu, "DR-Cloud: Multi-cloud based disaster recovery service," *Tsinghua Science and Technology*, Vol. 19, 2014, pp. 13-23.

Po-Jen Chuang (莊博任) received the B.S. degree from National Chiao Tung University, Taiwan, in 1978, the M.S. degree in Computer Science from the University of Missouri at Columbia, U.S.A., in 1988, and the Ph.D. degree in Computer Science from the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, U.S.A. (now the University of Louisiana at Lafayette), in 1992. Since 1992, he has been with the Department of Electrical and Computer Engineering, Tamkang University, Taiwan, where he is currently a Professor. He was the department chairman from 1996 to 2000. His main areas of interest include parallel and distributed processing, fault-tolerant computing, mobile computing, network security, cloud computing, software defined networking, virtualization and internet of things.

Chung-Hsing Wang (王竣星) received his B.S. and M.S. degrees in Electrical Engineering in 2010 and 2013 from Tamkang University, Taiwan. He is currently with Vision Advance Technology Inc. in Taiwan. His research interests include parallel and distributed processing, fault-tolerant computing and cloud computing.