An Efficient Clique-Based Algorithm of Compute Nodes Allocation for In-memory Checkpoint System

Xiangke Liao(✉), Canqun Yang, Zhe Quan, Tao Tang, and Cheng Chen

College of Computer Science, National University of Defense Technology, Changsha 410073, China
{xkliao,canqun,zhequan,taotang84,chengchen}@nudt.edu.cn

Abstract. Fault-tolerant is an essential technology for high-performance computing systems. Checkpoint/Restart (C/R) is the most popular fault-tolerant technique in which the programs save their states in stable storage, typically a global file system, and recover from the last checkpoint upon a failure. Due to the high-cost of global file system, node-local storage based checkpoint techniques are now getting more and more interests, where checkpoints are saved in local storage, such as DRAM. Typically, computing nodes are divided into groups and the checkpoint data is redundantly saved on a specified another node or is distributed among all other nodes in the same group, according to different cross-node redundancy schemes, to overcome the volatility of node-local storage. As a result, multiple simultaneous failures within one group often cannot be withstood and the strategy of node grouping is consequently very important since it directly impacts the probability of multi-node-failure within one group. In this paper, we propose a novel node allocation model, which takes the topological structure of high-performance computing systems into account and can greatly reduce the probability of multi-node-failure within a group, compared with traditional architecture-neutral grouping algorithms. Experimental results obtained from a simulation system based on TianHe-2 supercomputer show that our method is very effective on random simulative instances.

Keywords: Fault-tolerance · In-memory checkpoint · Algorithm

1 Introduction

In high performance computing (HPC) systems, the probability of overall failure increases over the computing time and the number of compute nodes due to more involved components. The mean time between failures (MTBF) of toady's systems have decreased to only a few hours [5,6,14] because of hardware and/or software errors [8,15]. As a result, fault-tolerant has become a well-known issue in HPC area [11].

One commonly used fault-tolerant technique is Checkpoint/Restart (C/R) [1]. In a C/R-based method, the state of an application, known as a checkpoint, is periodically saved to stable storages, typically the global file system. Once a failure occurs, the program can be restarted from the latest saved checkpoint. The critical issue of C/R-based methods is the high-cost of checkpoint access from global file system, especially for those large-scale systems, in which the I/O bandwidth will become the performance bottleneck [7,13]. Consequently, many local-storage based C/R methods have emerged [3,4,12,16]. In this paper, we focus on one that takes host memory as the storage to save checkpoints. It should be noted that local storage based C/R method is usually adopted as a supplement to the disk based C/R, to reduce the frequency of global file system access. This is also known as multi-level checkpoint technique [10].

The performance benefit of local storage based C/R derives from the linearly increasing checkpoint access bandwidth and at least an order of magnitude lower access latency compared with disks. However, local storage is usually supposed to be unstable. For example, DRAM is volatile and the data will be lost once the power is off. Consequently, the checkpoint of one node has to be redundantly saved in other nodes, so as to recover the node failure. The most common strategy is dual-redundancy. To be more specific, local storage based C/R typically divides compute nodes into groups, and only duplicates a checkpoint onto another node in the same group (usually called *partner node*). Upon a node failure, the execution state can be recovered by the checkpoint saved on its partner node. The dual-redundancy strategy means that a given node and its partner cannot fail at the same time, otherwise the execution cannot be recovered. To reduce the data amount of checkpoint, another commonly used scheme is XOR, which calculates a parity of redundant data from all checkpoints and then distributes it among all nodes in the same group. In this case, two nodes from the same group cannot fail at the same time.

How to group the compute nodes has a direct impact on the fault-tolerant effect of these local storage based C/R techniques, since different grouping strategies often lead to different probabilities of multi-node-failure in a group. In traditional methods, grouping strategy is relatively intuitive. For instance, in the Scalable Checkpoint/Restart (SCR) library [9], a multi-level checkpointing system, the nodes can be grouped by continuous node ID or a specified stride. This strategy is straightforward to implement, while the architecture of system is ignored. In real-world large-scale parallel computing systems, multiple simultaneous failures occur with higher probability in some set of nodes than others. For instance, by omitting other factors, two nodes that share the same electricity supply module are more likely to fail simultaneously than two isolated nodes due to the possible power failure. Generally, two nodes with larger logic distance may have lower probability of failing simultaneously.

Based on these observations, we propose a new algorithm in this paper, to group the computing nodes with the topological structure of a parallel computing system taken into account. Our method transfers the computing nodes with the probability of failure into a complete weighted undirected graph and uses clique technology to improve the nodes groups. Compared with intuitive grouping strategies, our algorithm can effectively reduce the probability of multiple simultaneous intragroup failures, in which case high-cost global C/R system has to be invoked. To evaluate our method, we build a simulation system based on TianHe-2 [2], the world's fastest supercomputer in the latest TOP500 list, which has more than 16,000 nodes. The topological structure and the essential parameters of the simulation system are extracted from this real system, and can also be modified easily to simulate other systems. The experimental results obtained show that the approach is very effective on random instances, especially for hard instances.

The remainder of this paper is organized as follows: Sect. 2 introduces the background. We propose our model and algorithm in detail in Sects. 3 and 4 respectively. Section 5 evaluates the performance of this model and conclusions are given in Sect. 6.

2 Background

In-memory checkpoint system is the most important local-storage based checkpoint technique. Generally, memory access speed is at least an order of magnitude faster than the file system. In addition, the capacity and bandwidth of memory can expand linearly with system scale from the view of the whole system. The major problem of takeing memory as checkpoint storage is its volatility. Note that in this paper we assume a fail-stop fault model, which means once an error occurs, the node stops responding and need to be replaced. Thus, we need a redundancy scheme to ensure that checkpoint data can be retrieved after the node failure. One common scheme is dual-redundancy (also called mirror scheme in some literatures), as illustrated in Fig. 1. Each node has a partner node, where its checkpoint data is stored redundantly.

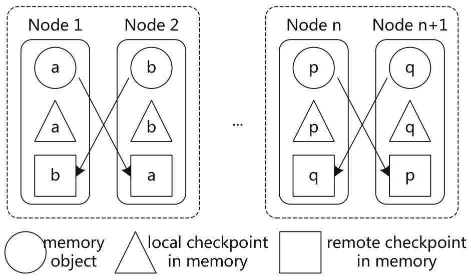

Fig. 1. Scheme of in-memory dual-redundancy checkpoint system

This dual-redundancy scheme demands that one node cannot fail simultaneously with its partner node, otherwise the checkpoint data will be lost. In practise, nodes are divided into groups and each node is assigned a partner node within the group. The strategy of node grouping is intuitive in existing checkpoint system, i.e., dividing nodes according to node's ID. Users can assign a hop distance so as to avoid adjacent nodes being allocated into the same group. Besides, a so-called XOR-scheme is another option, in which all nodes in a group collectively calculate a parity redundancy data according to their own checkpoints and then evenly distribute the parity redundancy data among all nodes in the group. Upon a node failure, other nodes in the group can recover the checkpoint according to their segments of the parity redundancy data. Compared with the dual-redundancy scheme, XOR-scheme demands less memory storage to save checkpoint data, while introducing extra computations. The XOR-scheme can withstand node failures as long as two or more nodes from the same group do not fail simultaneously.

We can see that in-memory checkpoint system is sensitive to simultaneous failures within a group. As a result, it is often taken as a complement of the global checkpoint system. That is, upon the failures that in-memory checkpoint system cannot withstand, the global checkpoint system is invoked. Notice that global checkpoint system is high-cost and thus the overall fault tolerance overhead can be reduced if we can lower the probability of simultaneous node failure within a group. This is also the object of the node allocation model we propose in this paper.

3 Node Allocation Model

3.1 Assumptions and Errors

Due to the complexity of organization structure, the fault model of high performance computing system can be very complicated, thus requiring some assumptions and simplifications when modeling the fault-tolerant system. We believe that these assumptions can cover the majority of actual situations.

- First, we assume errors follow a fail-stop model. Upon a node crash, all data on that node are supposed to be lost and we have to migrate its working state to a new node. The crashed node can be allocated again after repaired.
- We assume node failures are completely independent. In other words, a node failure does not increase or decrease the failure probability of other nodes.
- We assume that all kinds of failures have constant probabilities, including single node failure, power supply module failure, fan system failure, air condition system failure and water cooling system failure. We assume that these probabilities do not vary by time or utilization frequency.

3.2 Probability Function

As mentioned in Sect. 2, the uppermost reason that in-memory checkpoint system fails is simultaneous node failure within the same group, and the probability of that is closely related to the scheme of node grouping in a given system. So, we first calculate the simultaneous failure probability of any two given nodes before we propose the node group model in next section.

We take TianHe-2 high performance system as platform in this paper, which has a typical hierarchy architecture of large-scale parallel computing system.

As shown in Fig. 2, two nodes are integrated on a mainboard and share a power supply module. Several mainboards, then form a chassis, which is equipped with a standalone fan system. Each cabinet consists of several chassis and has its own air condition system. Finally, a row of cabinets share a water cooling system. In such a hierarchy architecture, different node grouping schemes will result in different probability of simultaneous node failure within the same group.

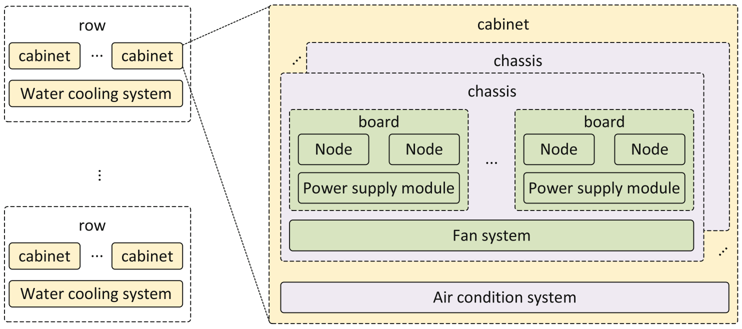

Fig. 2. Organization structure of TianHe-2

Below, we will discuss in detail the probability function of simultaneous failure of node i and j (denoted as P_i^j). The probability can be calculated according to the coordinates of the two nodes involved. We take the ratios of five kinds of failures into account when calculating the probability: single node failure, power supply failure, fan failure, air condition failure and water cooling system failure.

Single node failure ratio P_n is commonly considered as the reciprocal of the mean time between failure of node $MTBF_n$:

$$P_n = \frac{1}{MTBF_n}.$$

In the same way, the probability of power supply module failure P_m is equal to the reciprocal of the mean time between failure of power supply module $MTBF_m$:

$$P_m = \frac{1}{MTBF_m}.$$

As mentioned above, nodes on the same board share a single power supply module. In other words, the failure of the power supply module will directly result in the failure of all nodes on that board. Thus, without regard to other factors, the probability of two simultaneous node failures (on the same board) caused by power supply module failure is equal to P_m. Similarly, nodes within the same chassis share a unique fan system and will fail together due to the high temperature if the fan system stops working. Consequently, the probability of two simultaneous node failures (in the same chassis) caused by fan system failure is equal to the probability of fan failure P_f, which is equal to the reciprocal of the mean time between failure of fan $MTBF_f$:

$$P_f = \frac{1}{MTBF_f}.$$

For nodes within the same cabinet (sharing a unique air condition system) and the same row (sharing a unique water cooling system), the probabilities of two simultaneous failures caused by the air condition cooling system (P_c) and water cooling system (P_l) failures are the mean time between failure of each cooling system:

$$P_c = \frac{1}{MTBF_c},\ P_l = \frac{1}{MTBF_l}.$$

Now we consider the simultaneous failure probability of any two nodes i and j. Let symbol $m/f/c/l$ be 1 if node i and j belong to the same mainboard/chassis/cabinet/row, and 0 otherwise. First we only consider the factor of node failure. As mentioned above, all failures are assumed to be independent. So the simultaneous failure probability of i and j is ${P_n}^2$. To simplify the representation, we denote it as $P_i^j|_n$, that is,

$$P_i^j|_n = P_n^2.$$

Based on $P_i^j|_n$, we further take the power supply module into account. When i and j are on the same mainboard ($m = 1$), the simultaneous failure probability is the sum of P_m and the product of $1 - P_m$ and $P_i^j|_n$. That is because both nodes will fail definitely (with the probability of 1) if the power supply module fails (with the probability of P_m); otherwise (with the probability $1 - P_m$), the simultaneous failure probability is $P_i^j|_n$. When i and j are on different mainboards ($m = 0$), however, their failure are independent and the probability is the product of each one's failure probability, which is $P_m + (1 - P_m)P_n$. We denote the probability of single node failure considering node failure and power supply module failure as P_{nm}. Consequently, the simultaneous failure probability of i and j with node failure and power supply module failure considered (denoted as $P_i^j|_{nm}$) is

$$P_i^j|_{nm} = \begin{cases} P_m + (1 - P_m)P_i^j|_n, m = 1 \\ {P_{nm}}^2, \quad otherwise \end{cases}$$

In the same way, we take the fan system failure into account based on the equation above. When i and j are in the same chassis ($f = 1$), the simultaneous failure probability is the sum of P_f and the product of $1 - P_f$ and $P_i^j|_{nm}$, and otherwise ($f = 0$) is the product of each one's failure probability considering node failure, power supply module failure and fan system failure. We denote the latter one as P_{nmf}, which can be calculated as

$$P_{nmf} = P_f + (1 - P_f)P_{nm}.$$

So, we have

$$P_i^j|_{nmf} = \begin{cases} P_f + (1 - P_f)P_i^j|_{nm}, f = 1 \\ {P_{nmf}}^2, \quad otherwise \end{cases}$$

After all factors are involved, we can get the final probability equation as follows:

$$P_i^j|_{nmfcl} = \begin{cases} P_l + (1 - P_l)P_i^j|_{nmfc}, l = 1 \\ {P_{nmfcl}}^2, \quad otherwise \end{cases} \tag{1}$$

We can see that Eq. 1 is a recursion function and can be easily extended to a failure model with more organization hierarchies. Generally, for an S-level model, the simultaneous failure probability of i and j considering all S kinds of failures (denoted as $P_i^j|_{1\sim S}$) is

$$P_i^j|_{1\sim S} = \begin{cases} P_S + (1 - P_S)P_i^j|_{1\sim(S-1)}, T_S = 1 \\ {P_{1\sim S}}^2, \quad otherwise \end{cases}$$

where T_S represents whether the two nodes are in the same set at level S and $P_{1\sim S} = P_S + (1 - P_S)P_{1\sim(S-1)}$. $P_i^j|_{1\sim 1}$ means the probability of simultaneous failure considering the factor of level 1 failure (node failure) only, which is ${P_1}^2$. Given P_k and T_k $(1 \leq k \leq S)$, we can get the simultaneous failure probability of any two nodes.

3.3 Model Overview

Based on the probability function, we propose a node allocation model, to find the optimal node grouping scheme for a given node set, a given probability function and a given group size, so that the probability of simultaneous node failure within the same group is minimal.

In the paper, we abstract the allocation model as a weighted undirected graph, where vertices represent the computer nodes and the weight on the edge indicates the probability that the two connected nodes fail simultaneously. Figure 3 shows a partial view of a basic model with 3 individual computing nodes 1,2 and 3. The position of Node in the system is denoted by its coordinate x_i, y_i, z_i, k_i. The value P on the edge is the weight.

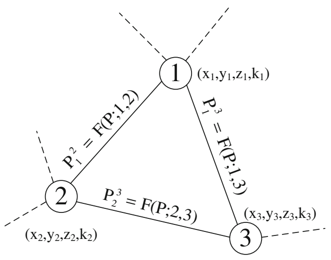

Fig. 3. A weighted undirected graph

It should be explained that (x_i, y_i, z_i, k_i) indicates the specific position of node i in the system, where x_i, y_i, z_i and k_i denote the number of board, the number of chassis, the number of cabinet and the number of row where node i is located in respectively.

Consequently, we abstract the node allocation model as a graph problem. For a given system, we use a graph to represent any given set of nodes. According to the probability function proposed above, the weight of each edge in the graph can be calculated. Then, the problem is to find a graph partition scheme with a given group size so that the probability of system failure due to two simultaneous node failures in a group is minimal. In the next section, we propose a novel algorithm to solve this problem.

4 Node Allocation Algorithm Based on Clique

SCR uses hop algorithm to divide compute nodes to groups, and hops are generally selected to be 1 in many systems. Our model, however, transfers the compute nodes into a weighted undirected graph, and tries to find an optimal combination checkpoint sets of nodes with the minimal weight. Given the positions of nodes, we can use Eq. 1 to calculate the simultaneous failure probability of every two nodes in the node set.

Table 1. Probabilities of simultaneous failure

Node	1	2	3	4	5	6
1	-	0.18	0.1	0.18	0.18	0.18
2	0.18	-	0.85	0.18	0.1	0.1
3	0.1	0.85	-	0.1	0.1	0.36
4	0.18	0.18	0.1	-	0.1	0.25
5	0.18	0.1	0.1	0.1	-	0.85
6	0.18	0.1	0.36	0.25	0.85	-

For instance, we assume a task that occupies 6 compute nodes: {1,2,3,4,5,6}, and the simultaneous failure probabilities are listed in Table 1:

As shown in Fig. 4, these nodes can be transferred into a complete weighted undirected graph, where the weight of edge denotes the simultaneous failure probability of these two nodes.

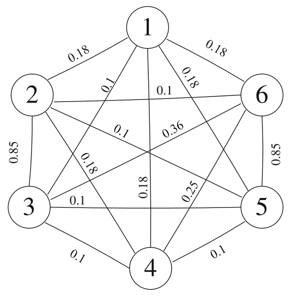

Fig. 4. A complete weighted undirected graph.

Consider a weighted undirected graph $G = (N, E, W)$, where N is a set of nodes $\{n_1, n_2, ..., n_n\}$, and E and W are the edge and weight sets respectively, we have:

Definition 1. *Given a node n in G, the number of its neighbor nodes is called the degree of n.*

Definition 2. *Given a graph G, a subset of N is called a clique if every two nodes in the subset are connected by an edge in G.*

The problem is then attributed to find a clique partition of the graph with specified size, so as to minimize the probability of system failure due to two simultaneous node failures in a clique.

Algorithm 1 shows the pseudo-code of a basic algorithm for node allocation. The algorithm based on clique(*CB algorithm* for short) finds all cliques of specified size in a set of compute nodes N. Given a set N, clique size s and the probability function of simultaneous failure P, the algorithm will find a clique sets $C : \{C_1, C_2, C_3, ..., C_n\}$. In line 5, the function $BuildProbMatrix(N, P)$ calculates the simultaneous failure probability of every two nodes in N based on function P (i.e., Formula 1). The function *AddEdges* in line 12 adds new edges with minimal weight for nodes in the latest graph G. Note that there may be multiple edges added at one time since they have the same weight. We start with minimal weight edges to make the weight of clique as small as possible. Lines 13–24 search all cliques in the current graph. We travel the node set N in ascending order of node degree since node with small degree has less opportunity to form cliques with other nodes, so as to get as more cliques with minimal weight as possible.

Algorithm 1. Find all cliques

```
Input: nodes set $N$, clique size $s$, probability function $P$
Output: clique set $C$
 1. $C \leftarrow \emptyset$
 2. $G \leftarrow \{N, \emptyset, \emptyset\}$
 3. $W \leftarrow BuildProbMatrix(N, P)$
 4. while $N \neq \emptyset$ do
 5.    if $(\#N) \leq s$ then
 6.       $c \leftarrow N$
 7.       $C \leftarrow C + \{c\}$
 8.       return $C$
 9.    end if
10.    $G \leftarrow AddEdges(G, W)$ // edges with minimal weight for nodes in current $G$
11.    $N' \leftarrow N$
12.    while $N' \neq \emptyset$ do
13.       Let $v \in N'$ be the node with minimal degree in current $G$
14.       $c \leftarrow FindAClique(G, v, s)$
15.       if $c \neq \emptyset$ then
16.          $C \leftarrow C + \{c\}$
17.          $N \leftarrow N - c$ // also remove all edges connected to $c$ in $G$
18.          $N' \leftarrow N' - c$
19.       else
20.          $N' \leftarrow N' - \{v\}$
21.       end if
22.    end while
23. end while
24. return $C$
```

The function *FindAClique* in line 16 is used to find a clique that contains node v with size s in graph G. It uses a basic branch-and-bound algorithm to search for a clique. Once a clique is found, we add it into the clique set C (line 18), remove the nodes from original graph G (line 19), and also remove all edges connected to these nodes. After the while-loop finishes (line 24), all possible cliques are generated and removed from the current graph. Then some new edges with minimal weight should be added and the search is redone until the graph is empty. Note that if the number of nodes in current graph is no more than s, i.e., the clique size, we directly output it as the last clique and quit (lines 7–11). This works for the situations that the number of nodes is not divisible by s.

For the instance with 6 compute nodes given in Fig. 4, traditional hop algorithm will divide them into two groups: $\{1, 2, 3\}$ and $\{4, 5, 6\}$ (assume that hop distance is 1 and group size is 3). Based on the probabilities in Table 1, the failure probability of the system will be 0.9888.

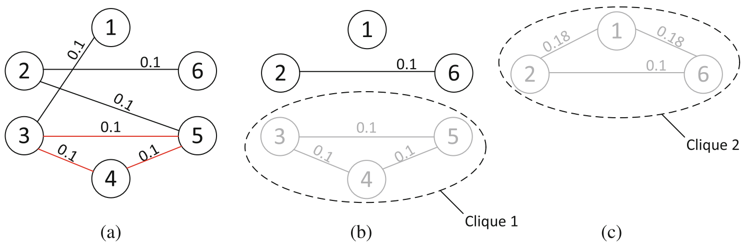

Fig. 5. An example of the algorithm.

The CB algorithm, however, tries to find all cliques of size 3 with minimal weight. First, all edges with weight 0.1 are added to the graph, as shown in Fig. 5(a), and we travel the graph in the order $1 \to 6 \to 2 \to 4 \to 3 \to 5$ according to the node degree. The function *FindAClique* will find the first clique ($\{3, 4, 5\}$) when $v = 4$. Then, the clique and all related edges are removed, as shown in Fig. 5(b). New edges with weight 0.18 are added after that because no more clique of size 3 can be found in the left graph. Actually, in this example, ($\{1, 2, 6\}$) can be directly denoted as a clique without search since the number of left nodes is 3, which is equal to the clique size. The failure probability of whole system in this solution is 0.5588, which is only 56.51 % of that in the hop algorithm.

As mentioned before, the "short-plate" of in-memory checkpoint system is probability of failure of any two nodes from the same group. The CB algorithm takes probabilistic model as a guide, initiatively avoids the allocation of checkpoint set with high simultaneous failure probability, and makes each "short-plate" as long as possible. Consequently, it reduces the frequency of using file system checkpoints, which means the cost of fault tolerance will be decreased.

5 Experimental Results

We have compared the performance of our algorithm with hop algorithms from the state-of-the-art fault-tolerant library. In practice, the execution time of application could be very long (up to days or months), and the overhead of grouping algorithm is negligible; so we only compare the probability of simultaneous failure, in which case the in-memory checkpoint system cannot recover the execution and higher level fault-tolerant system with much higher cost has to be involved.

Table 2 lists the probabilities of simultaneous failure of our algorithm and hop algorithm. P_{CB} represents the probability of our clique algorithm, P_{HOP} represents the best result of the hop algorithm, and *ratio* represents the difference between them. As mentioned in Sect. 3.2, the system has a total of 16,000 nodes, and we choose a random subset of nodes in this experiment. It can be seen obviously in Table 2 that clique allocation algorithm is very efficient when the size of XOR set is small, especially with size of 2. The ratio between the two algorithms becomes smaller as the size of XOR set increases, and the solutions of two algorithms will become identical when the size is equal to (or larger than) the number of computing nodes. This tendency is more clearly illustrated in Fig. 6.

To be more clear, Fig. 7 gives the simultaneous failure probabilities of two algorithms when #Node is 2048. We can see from the figure that the probability of the

Table 2. Probabilities of simultaneous failure with different node numbers and group sizes

#Node	$Size_{XOR} = 2$			$Size_{XOR} = 4$			$Size_{XOR} = 8$			$Size_{XOR} = 16$		
	P_{CB}	P_{HOP}	ratio	P_{CB}	P_{HOP}	ratio	P_{CB}	P_{HOP}	ratio	P_{CB}	P_{HOP}	Ratio
64	1E-09	0.00031	**114959**	3.97E-05	0.00113	**67.87**	0.00034	0.00296	**13.50**	0.00465	0.00684	**1.52**
128	1E-08	0.00090	**167483**	3.19E-05	0.00245	**88.53**	0.00032	0.00589	**18.90**	0.00595	0.01295	**2.27**
256	1E-08	0.00170	**145818**	6.64E-05	0.00526	**117.40**	0.00059	0.01204	**20.30**	0.00912	0.02551	**2.90**
512	2E-08	0.00327	**139793**	0.00015	0.00981	**148.50**	0.00120	0.02288	**19.10**	0.01211	0.04923	**4.21**
1024	5E-08	0.00668	**145621**	0.00017	0.02010	**207.60**	0.00242	0.04596	**19.10**	0.02112	0.09744	**4.81**
2048	9E-08	0.01401	**152651**	0.00016	0.04020	**411.70**	0.00480	0.09059	**18.90**	0.03186	0.18509	**6.06**
4096	1.8E-07	0.00730	**39570**	0.00040	0.03976	**287.50**	0.00962	0.13239	**13.80**	0.04554	0.30447	**6.73**
8192	3.7E-07	0.01442	**39092**	0.00025	0.03938	**154.50**	0.01913	0.16271	**8.51**	0.07993	0.45199	**5.65**
16384	7.4E-07	0.12184	**165302**	0.00177	0.29667	**167.20**	0.03818	0.57240	**15.00**	0.71965	0.80620	**1.12**
#Node	$Size_{XOR} = 32$			$Size_{XOR} = 64$			$Size_{XOR} = 128$			$Size_{XOR} = 256$		
	P_{CB}	P_{HOP}	ratio	P_{CB}	P_{HOP}	ratio	P_{CB}	P_{HOP}	ratio	P_{CB}	P_{HOP}	Ratio
64	0.01114	0.01392	**1.25**	0.02705	0.02705	**1.00**	0.02705	0.02705	**1.00**	0.02705	0.02705	**1.00**
128	0.01989	0.02613	**1.32**	0.05203	0.05263	**1.02**	0.10208	0.10208	**1.00**	0.10208	0.10208	**1.00**
256	0.03645	0.05126	**1.41**	0.09059	0.10132	**1.12**	0.19173	0.19381	**1.01**	0.35117	0.35117	**1.00**
512	0.06925	0.10048	**1.45**	0.19105	0.19369	**1.02**	0.32208	0.34899	**1.08**	0.53657	0.57689	**1.08**
1024	0.13777	0.19352	**1.41**	0.30183	0.35450	**1.18**	0.56614	0.58503	**1.03**	0.74841	0.82544	**1.10**
2048	0.25198	0.34635	**1.37**	0.47044	0.58109	**1.24**	0.77218	0.82778	**1.07**	0.87476	0.97030	**1.11**
4096	0.43440	0.55524	**1.28**	0.67615	0.81967	**1.21**	0.87961	0.96930	**1.10**	0.89930	0.99908	**1.11**
8192	0.65864	0.77372	**1.17**	0.84162	0.96272	**1.14**	0.89956	0.99894	**1.11**	0.89999	0.99999	**1.11**
16384	0.86865	0.96454	**1.11**	0.89906	0.99890	**1.11**	0.89999	0.99999	**1.11**	0.90000	0.99999	**1.11**

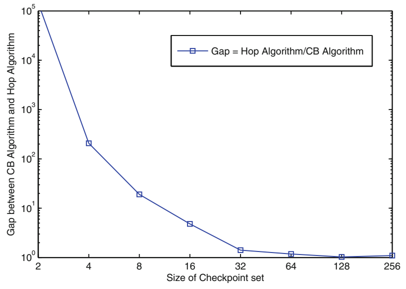

Fig. 6. Gap between two algorithms when #Node=1024.

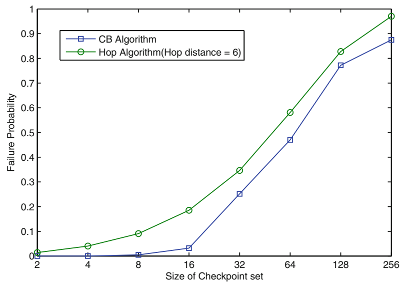

Fig. 7. Simultaneous failure probabilities of the two algorithms when #Node=2048.

simultaneous failure is always larger in hop algorithm than in our algorithm with all set sizes.

Tables 3 and 4 compare the simultaneous failure probabilities of two algorithms when $size_{XOR} = 3$ and $size_{XOR} = 4$ respectively, where we vary the hop distance in the hop algorithm within different ranges. The results show that the probability in the hop algorithm is not affected by the hop distance evidently. We can also see that our algorithm obtains much lower simultaneous failure probabilities than the hop algorithm in all cases. Figure 8 illustrates this result more clearly and intuitively.

Table 3. Probabilities of simultaneous failure with small hops in hop algorithm when $size_{XOR} = 3$.

#Node	P_{HOP}										P_{CB}
	hop=1	hop=2	hop=3	hop=4	hop=5	hop=6	hop=7	hop=8	hop=9	hop=10	
1000	0.01058	0.01574	0.01511	0.01430	0.01120	0.01414	0.01405	0.01167	0.01414	0.01272	**1.01E-05**
6000	0.13404	0.02085	0.02418	0.02445	0.02534	0.02562	0.02605	0.01772	0.01866	0.03975	**5.99E-07**
10000	0.31091	0.03400	0.03196	0.03063	0.03286	0.04901	0.07563	0.11029	0.10535	0.07420	**1.1E-05**
15000	0.54986	0.05346	0.05140	0.04826	0.04553	0.04403	0.04140	0.04100	0.04892	0.08605	**1.5E-06**

Table 4. Probabilities of simultaneous failure with large hops in hop algorithm when $size_{XOR} = 4$.

#Node	P_{HOP}										P_{CB}
	hop=1	hop=17	hop=33	hop=49	hop=65	hop=81	hop=97	hop=113	hop=121	hop=136	
1024	0.01853	0.02186	0.01971	0.02012	0.01971	0.02070	0.01971	0.01925	0.01981	0.01930	**0.00018**
2048	0.03534	0.04087	0.04171	0.04143	0.03893	0.04063	0.04042	0.04023	0.04054	0.04002	**0.00017**
8192	0.06354	0.04198	0.04047	0.04210	0.03984	0.03963	0.04264	0.03842	0.03951	0.03889	**0.00026**
16384	0.66958	0.15221	0.08228	0.33279	0.21335	0.38715	0.21852	0.28729	0.20119	0.33215	**0.00177**

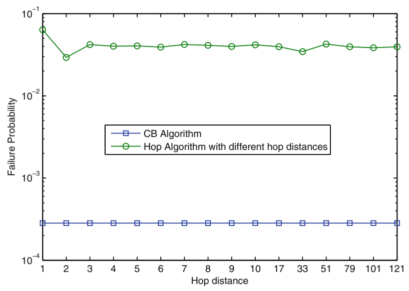

Fig. 8. Simultaneous failure probabilities of hop algorithm with different hop distances when #Node=8192 and $size_{XOR} = 4$.

Figure 9 shows the simultaneous failure probabilities with different hop distances and checkpoint set sizes. As concluded above, hop distance has little influence on probability in the hop algorithm. Those curves represent hop algorithm almost coincide in this figure. Also, the blue curve, which represents our algorithm, shows better results with all checkpoint set sizes compared to the hop algorithm.

Table 5 collects the times that our algorithm outperforms the hop algorithm in 1,000 random experiments. Since our algorithm is heuristic, the search result is not necessarily the optimum solution, and the hop algorithm gets chance to obtain better

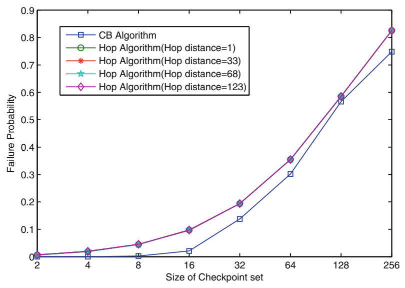

Fig. 9. Simultaneous failure probabilities of the two algorithms with different hop distances and $size_{XOR}$s when #Node=1024

result due to the randomness of the node set we choose. However, we can notice in the table that for most situations, our method can outperform the hop algorithm in all 1,000 random tests. We can also see that with a fixed checkpoint set size, the times that our method win decrease mildly when the number of nodes gets larger. That is because when almost all nodes in the system are involved, the topology of these nodes is also pretty much fixed, in which case, the hop algorithm can decrease simultaneous node failure probability easily by assigning a hop distance large enough.

Table 5. Times that our algorithm outperforms the hop algorithm in 1,000 random experiments.

#Node	$size_{XOR}$				#Node	$size_{XOR}$			
	2	4	64	256		3	5	65	257
512	1000	1000	1000	1000	500	1000	1000	1000	1000
1024	1000	1000	1000	999	1000	1000	1000	1000	1000
8192	1000	1000	998	992	10000	1000	1000	996	988
16384	1000	1000	990	978	15000	1000	1000	991	976

Experimental results above show that our clique-based algorithm is very efficient, especially for small size of checkpoint set. The probability of simultaneous node failure is far below the hop algorithms, which means we can greatly reduce the chance to invoke the high-cost global checkpoint system.

6 Conclusion and Future Work

We build a new node allocation model based on the architecture of TianHe-2 and propose a new algorithm to decrease the probability that in-memory checkpoint system cannot work. We calculate the probability of simultaneous failure of any two nodes, transfer it into a complete weighted undirected graph, use a heuristic algorithm to find clique in the graph, and then rationally divide the compute nodes into groups to decrease the in-group simultaneous failure probability. The experimental results performed based on the probability model abstracted from TianHe-2 show that, compared to the traditional node distribution scheme, our model can find near optimal combination of nodes with lower simultaneous failure probability. This also means that we can greatly reduce the cost of recovery in multi-level checkpoint system. In the future, we will take the communication cost into account when grouping the nodes based on the topology of the interconnect network.

Acknowledgment. This work is supported by National High Technology Research and Development Program of China (863 Program) No.2012AA01A301 and 2012A A01A309.

References

1. http://source-forge.net/projects/scalablecr/scalable-checkpoint/restart-library
2. http://www.netlib.org/utk/people/jackdongarra/papers/tianhe-2-dongarra-report.pdf
3. Daly, J.: A higher order estimate of the optimum checkpoint interval for restart dumps. Future Gener. Comput. Syst. **22**(3), 303–312 (2006)
4. Duda, A.: The effects of checkpointing on program execution time. Inf. Process. Lett. **16**(5), 221–229 (1983)
5. Vivek Sarkar, E.: Exascale software study: Software challenges in exascale systems (2009)
6. Glosli, J.N., Caspersen, K.J., Gunnels, J.A., Rudd, D.F.R.A.E., Streitz, F.H.: Extending stability beyond cpu millennium: a micron-scale atomistic simulation of kelvin-helmholtz instability. In: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC), pp. 1–11 (2007)
7. Iskra, K., Romein, J.W., Yoshii, K., Beckman, P.: Zoid: I/o-forwarding infrastructure for petascale architectures. In: PPoPP 2008: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 153–162 (2008)
8. Michalak, S.E., Harris, K.W., Hengartner, N.W., Takala, B.E., Wender, S.A.: Predicting the number of fatal soft errors in los alamos national laboratory's ASC Q supercomputer. IEEE Trans. Device Mater. Reliab. **5**(3), 329–335 (2005)
9. Moody, A.: The scalable checkpoint/restart (scr) library, user manual version 1.1-6 (2010)
10. Moody, A., Bronevetsky, G., Mohror, K., de Supinski, B.R.: Design, modeling, and evaluation of a scalable multi-level checkpointing system. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis(SC), pp. 13–29, November 2010
11. Naksinehaboon, N., Liu, Y., Leangsuksun, C.B., Nassar, R., Paun, M., Scott, S.L.: Reliability-aware approach: an incremental checkpoint/restart model in hpc environments. In: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID), pp. 783–788 (2008)
12. Plank, J.S., Thomason, M.G.: Processor allocation and checkpoint interval selection in cluster computing systems. J. Parallel Distrib. Comput. **61**(11), 1570–1590 (2001)
13. Ross, R., Moreira, J., Cupps, K., Pfeiffer, W.: Parallel i/o on the ibm blue gene/l system. Blue Gene/L Consortium Quarterly Newsletter. Technical report (2006)

14. Schroeder, B., Gibson, G.: Understanding failure in petascale computers. J. Phys. Conf. Series: SciDAC **78**, 012–022 (2007)
15. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN), pp. 249–258 (2006)
16. Young, J.W.: A first order approximation to the optimum checkpoint interval. Commun. ACM **17**(9), 530–531 (1974)