Journal Title
XX(X):1–13

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Overhead of Using Spare Nodes

Atsushi Hori[1], Kazumi Yoshinaga[1], Thomas Herault[2], Aurélien Bouteiller[2], George Bosilca[2], Yutaka Ishikawa[1]

Abstract

With the increasing fault rate on high-end supercomputers, the topic of fault tolerance has been gathering attention. To cope with this situation, various fault-tolerance techniques are under investigation; these include user-level, algorithm-based fault-tolerance techniques and parallel execution environments that enable jobs to continue following node failure. Even with these techniques, some programs with static load balancing, such as stencil computation, may underperform after a failure recovery. Even when spare nodes are present, they are not always substituted for failed nodes in an effective way.

This paper considers the questions of how spare nodes should be allocated, how to substitute them for faulty nodes, and how much the communication performance is affected by such a substitution. The third question stems from the modification of the rank mapping by node substitutions, which can incur additional message collisions. In a stencil computation, rank mapping is done in a straightforward way on a Cartesian network without incurring any message collisions. However, once a substitution has occurred, the optimal node-rank mapping may be destroyed. Therefore, these questions must be answered in a way that minimizes the degradation of communication performance.

In this paper, several spare-node allocation and node-substitution methods will be proposed, analyzed, and compared in terms of communication performance following the substitution. The proposed substitution methods are named *sliding methods*. The sliding methods are analyzed by using our developed simulation program and evaluated by using the K computer, BG/Q and TSUBAME 2.5. It will be shown that when a failure occurs, the stencil communication performance on the K and BG/Q can be slowed around ten times depending on the number of node failures. The barrier performance on the K can be cut in half. On BG/Q, barrier performance can be slowed by a factor of ten. Further, it will also be shown that no communication performance degradation with failed node substitution can be seen on TSUBAME 2.5 has Fat-Tree network, while the K computer and BG/Q have Cartesian networks. Thus, the communication performance degradation depends on network topology.

Keywords

fault tolerance, fault mitigation, spare node, communication performance

Introduction

With the fault rate increasing on high-end supercomputers, the topic of fault tolerance has been gathering attention[1], and jobs are being aborted due to system errors[2]. To cope with this situation, various fault tolerance techniques have been investigated. Checkpoint and restart is a well-known technique for parallel jobs, and enabling jobs to continue execution from a previously defined checkpoint (there are many studies ans systems of checkpoint and restart, but the most notable one is CLIP[3]).

With the increase in size of parallel applications, the total amount of I/O needed for checkpoint/restart begun to be problematic. A lot of research is currently going on on techniques to reduce the checkpoint amount in order to alleviate the I/O issue (Sato[4], for example). On the other hand, user-level checkpoints, where each program implements its own checkpoint/restart strategy, have been attracting attention as a possible alternative. Since the user knows which data should be saved and which data can be lost, the amount of checkpoint data can be drastically reduced, and thus the I/O time can also be greatly reduced, at the cost of only some additional programing by the user.

Davies *et al.* presented a method that allows a user program to be fault-tolerant without using checkpointing[5]. In this technique, the parity to recover the lost data can be embedded into an LU decomposition algorithm, and the user program can recover from failure without checkpointing. Having the opportunity to address the failure at the algorithm level opens interesting perspective and new research topics. With support from the programing paradigm and the execution environment, users could write application handling faults in the most optimal way. The Message Passing Interface (MPI) is the most widely used communication library, and its specifications are well

[1]RIKEN Advanced Institute for Computational Science
[2]The University of Tennessee, Innovative Computing Laboratory

Corresponding author:
A. Hori, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo, JAPAN. 650-0047
Email: ahori@riken.jp

defined[6]. Unfortunately, in the current MPI standard, a fatal error handler is raised upon process failure, preventing any user-level fault handling to be implemented at this time.

To define the behavior of MPI when a fault occurs, User-level Failure Mitigation (ULFM) has been proposed and a prototype is being developed, capable of handling both process and node failures[7]. ULFM provides the application program interface (API) so that the modifications to the existing MPI specifications are minimized. Even with ULFM, user-level fault handling is not straightforward, and various frameworks have been proposed to simplify it. Falanx is a fault-tolerant framework for master-worker programing[8]. Local Failure Local Recovery (LFLR) is another fault-tolerant framework[9], and it covers a wider range of programing models than are supported by Falanx. Both Falanx and LFLR are implemented by using ULFM. Global View Resilience (GVR) is another user-level fault mitigation system; it is based on partitioned global address space (PGAS) programing model[10;11].

We believe that the user-level fault-handling code must be as simple as possible. It is important to avoid situations in which the code for handling the first node failure is different from the code for handling subsequent failures, because it is very hard to produce this type of situation when testing a program. This type of complexity must be hidden within the system software.

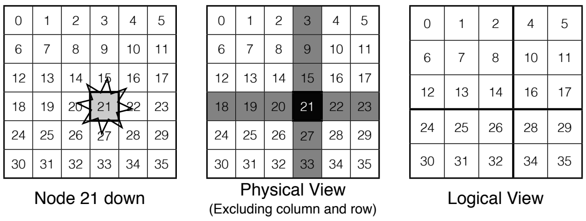

Figure 1. Example of node failure and recovery

Figure 1 shows an example of a node failure in a 2D network consisting of 36 nodes. Here, it is assumed that a job is running on this machine, and the job is written with a fail-stop-free runtime system, such as ULFM. When node 21 goes down (left panel in Figure 1), the job running on those 36 nodes can take one of the following actions:

- Abort the job and resubmit it (from a previous checkpoint, if possible), or
- Allow the remaining 35 nodes to continue to execute the job.

In the first strategy, user-level fault handling is not required. In the second strategy, the task allocated to the failed node must be equally shared by the remaining 35 nodes, otherwise, a load imbalance occurs. If the job can balance a dynamic load, which is a capability of master-worker models and particle-in-cell simulations, then the load can be rebalanced by the application itself, without the need to extensively modify the code. However, if the job is a stencil application, which, in most cases, does not have dynamic load balancing capability, then fault handling is more difficult. In most stencil applications, both the communication pattern and the load balancing are static. To preserve the communication pattern, one possibility for handling a node failure is to exclude the row and column that include the failed node (middle panel in Figure 1); this preserves the stencil communication pattern. However, the task allocated to the failed node must be shared equally by the remaining nodes (right panel in Figure 1). This load-leveling requires additional code for handling the fault, and this must be avoided if possible.

If a system software reserves a set of spare nodes in advance, and the failed node is replaced by a spare node, then the user-level handling of node failure is simplified, because the number of nodes involved in the computation remain the same. LFLR assumes the use of spare nodes, and although the detailed recovery process is hidden from users, GVR may utilize spare nodes. However, to the best of our knowledge, there has been no discussion of the best way to reserve spare nodes or of how to use them to replace failed nodes. As an evaluation index, we chose communication performance, because the use of spare nodes may introduce extra message collisions.

This paper presents the results of our investigations into these issues. As a first step to address these issues, we propose several methods for using spare nodes to replace faulty ones in addition to the spare node allocations. The proposed methods are discussed and compared from the viewpoint of communication performance degradation. The contributions of this paper are;

- spare node allocation methods are proposed,
- failure node substitution methods are proposed,
- focusing on stencil communication and some collective performance, the behaviour and characteristics of proposed spare node allocation and substitution methods are revealed by simulations, and
- evaluations results on the two supercomputers having a Cartesian network topology and one supercomputer having a FatTree network topology are shown to how a network topology affects the communication performance degradation.

Using Spare Nodes

For the remainder of this paper, we will assume that the networks being considered have a multidimensional Cartesian (mesh and/or torus) topology, otherwise noticed. We make this assumption because four of the top five machines have networks with this topology (as listed on the TOP500 Super Computer Site[12], November 2015); see Table 1.

Table 1. Network topologies in the Top500 list[12]

Rank	Name	# Cores	Topology
1	Tianhe-2	3,120K	FatTree
2	Titan (Cray XK7)	561K	3D Torus
3	Sequoia (BG/Q)	1,573K	5D Torus/Mesh
4	The K computer	705K	6D Torus/Mesh
5	Mira (BG/Q)	786K	5D Torus/Mesh
11	JUQUEEN (BG/Q)	459K	5D Torus/Mesh
25	TSUBAME 2.5	76k (+GPU)	FatTree

From the programmers point of view, it is not complicated to have spare nodes held ready, or to have them substituted in for faulty nodes. With MPI, the modification is as follows:

1) a new MPI communicator is created at the location from which the faulty node is extracted (in ULFM, the command `MPI_Comm_shrink` will do this), and a selected spare node replaces the faulty node; 2) the spare node is set up to take over the functions of the failed node. The remaining parts of the program can remain as they were. This means that the logical topology provided by the new MPI communicator can remain the same as it was before the failure; however, the actual physical topology is altered. New message collisions that would not have happened under the failure free physical topology will happen under the recovered topology (Figure 4).

Therefore, replacing faulty nodes with spare nodes must be done carefully in order to minimize the communication performance degradation. There are many other aspects that should be considered, such as system utilization, job turn-around time, ease of user programing, and the framework that needs to be developed. Unfortunately, almost no research has been done on this topic, so in this paper, we will focus primarily on the communication performance.

Throughout this paper, we will be concerned only with the node failure. Network failures can also occur, but we will assume that this recovery is the responsibility of the network itself[13] (see also Section). The Tofu network, which is used by the K computer, uses redundant links to detour around failed nodes[14]. We will assume that a job can survive even with the failure of one or more nodes when it is operating in a parallel computing environment that provides a user-level fault mitigation mechanism, such as ULFM, and any processes running on the failed node can be recovered from a checkpoint or by using parity with viable processes. Finally, we will assume that the processes running on a node can be migrated to any other node.

In the next subsection, we will discuss the allocation of spare nodes, and the possibility of this degrading the communication performance will be shown. Then, three methods for substituting a spare node for a faulty node will be proposed and compared.

Spare Node Allocation

In this section for simplicity, we will consider only 2D networks with static XY routing routing[15]. Figure 2 shows three different ways of allocating spare nodes. Each small square represents a node. In the left panel, the right-hand column is reserved for spare nodes; this pattern is denoted as *2D(1,1)*. In the middle panel, two sides (the right-hand column and the bottom row) are reserved for spare nodes, denoted *2D(2,1)*. In the right-hand panel, two two-node thick sides (the two right-hand columns and two bottom rows) are reserved, denoted *2D(2,2)*. In this notation, “2D” means that the allocation applies to the 2D plane, the first number in the brackets is the number of sides in which spare nodes are reserved, and the second number is the thickness, number of columns or rows, of a side of spare nodes reserved.

Spare nodes are allocated at the side(s) of a 2D grid, as shown in Figure 2; thus, a stencil application with non-periodic boundaries will not have any overhead. This will not be the case for stencil applications that have periodic boundaries or for networks that have torus topology. However, the hop count is only increased by one, so the increase in run time will be very small (100 ns per hop on the K computer).

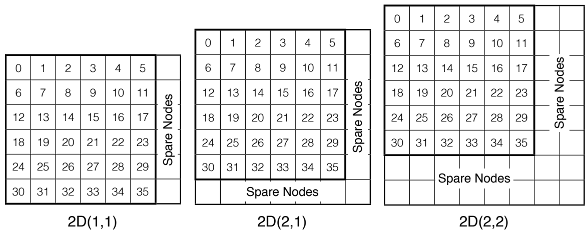

Figure 2. Patterns for allocation of spare nodes

The percentage of the nodes that are reserved as spare nodes in the 2D(2,2) case is as follows.

$$R_{2D(2,2)} = 1 - (N^{1/2} - 2)^2/N$$

Where N is the number nodes. In the more general $qD(r, s)$ case, the percentage of spare nodes can be expressed as follows.

$$R_{qD(r,s)} = 1 - \frac{(N^{1/q} - s)^r \times (N^{1/q})^{q-r}}{N}$$

Here, $r \leq q$ and $s < N^q$. Note that this expression is not precise, because the number of nodes is an integer, and the flooring effect is ignored. However, this information can be useful for determining how the spare node percentage relates to the total number of nodes used for a job.

Figure 3 shows the percentages of spare nodes for various numbers of nodes and patterns of allocation. As shown in this figure, the more dimensions the network has, the higher the percentage of spare nodes. The percentage is almost proportional to the number of sides allocated to the spare nodes. Most notably, the larger the job size, the lower the percentage. We will discuss this point in Section .

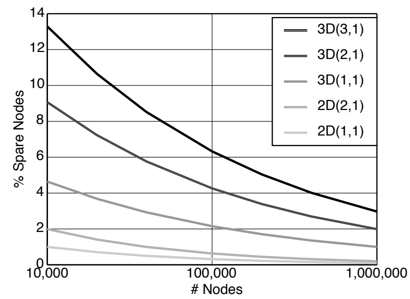

Figure 3. Percentage of spare nodes

It is possible to allocate spare nodes on four sides of a 2D grid, but on a torus network, this is almost equal to the 2D(2,2) case. In our investigation, we could not find any significant difference between 2D(4,1) and 2D(2,2), and so in this discussion, we will not further consider cases in which $r > q$. The thickness, s, does not affect the nature of the spare node substitution method described in Section , so we will investigate only cases of single-node thickness.

Having spare nodes can decrease the system utilization ratio. However, this does not always happen. On the K computer, the size of each dimension of a job must be in a *Tofu unit*, which has twelve nodes. When a user submits an 11x11x11 3D job, for example, it may be scheduled to have 12x12x12 nodes. This results in 3D(2,1) spare nodes. The same situation can be seen with the other machines that have a Cartesian topology network and are listed in Table 1. On Blue Gene/Q (BG/Q) machines, the number of nodes for a job must be a power of 2^{16}. On a Cray XK/7, jobs are allocated to 4 blocks[17]. Thus, the gap between the number of nodes required by a job and the number of nodes actually allocated can be allocated as spare nodes, without requiring additional nodes.

Substitution of a Spare Node for a Faulty Node

Communication performance degradation can be observed because when a spare node that replaces a faulty node can be located far from the original node. Figure 4 shows the 5P-stencil communication pattern (left). In 5P-stencil communication on a Cartesian topology, no messages collide, because nodes communicate only with their neighbors. Here, static XY routing is assumed. In the right-hand panel of Figure 4, when a faulty node (denoted as "F") is replaced by a spare node (denoted as "S"), the regularity of the stencil communication pattern is lost. As shown in this figure, there are five message routes crossing through the circled link, this means that up to five messages can collide.

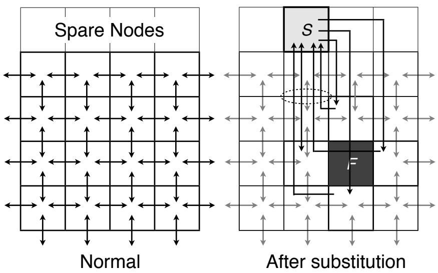

Figure 4. Message collisions

We propose three methods for substituting nodes, and these are shown in Figure 5. We call these methods the *0D*, *1D*, and *2D* sliding methods. With higher-dimension networks, those proposed methods can be augmented in a natural way, but for simplicity, we will explain them on a 2D network. We will use a 5P-stencil communication pattern, in which messages from each node are sent up, down, left, and right. In the 9P-stencil communication pattern, there are an extra four directions, since messages can be sent along the diagonals. However, in most cases, the length of those diagonal messages is much shorter than those in a 5P-stencil pattern, and so the effect on the communication performance is expected to be small.

0D sliding The 0D sliding method is the simplest. The faulty node is simply replaced by a spare node (as was shown in Figure 5). There is a big drawback to this method, however, when a node failure happens far from a spare node: the hop distance from the failed node to the spare node can be very large. This increases the possibility of message collisions and results in a higher communication latency due to the large number of hops. To minimize this, the failed node should be replaced with the spare node to which the Manhattan distance is the shortest.

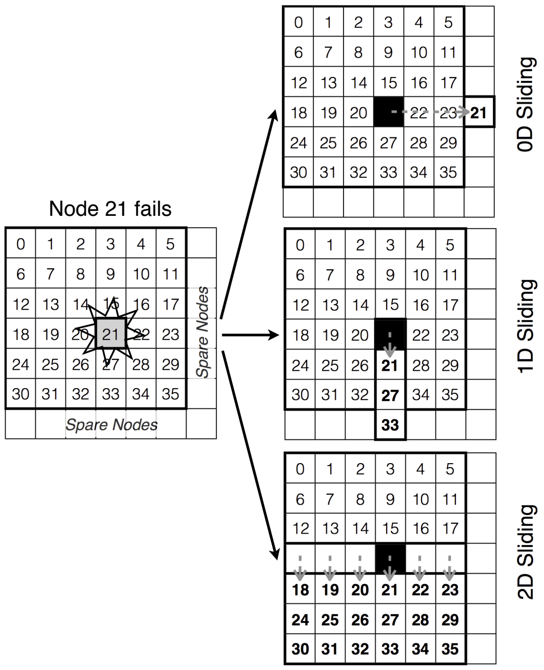

Figure 5. Substitution methods for faulty nodes

Figure 6 shows examples of the results of replacing multiple faulty nodes when using the 0D sliding method with the 2D(1,1) allocation. On the left-hand panel, nodes 1 through 5 have failed and have been replaced by spare nodes 1' through 5', respectively. The spare nodes were chosen so as to minimize the number of hop counts between each faulty node and its corresponding spare node. With non-periodic 5P-stencil communication in the XY routing algorithm, the messages from all of the spare nodes to the nodes (A through F) adjacent to the failed nodes are routed through node 1' (because of the X direction routing of the XY routing algorithm). Thus, there are eleven messages in the network links between 1' and A (these are shown in the white boxes): these ten plus the normal stencil communication message between the nodes. This is the worst-case scenario for the 0D sliding method, and the number of faulty nodes is less than or equal to six.

The right-hand panel of Figure 6 shows a case for which the network topology is a 2D mesh, spare nodes are reserved in the 2D(1,1) pattern, and the faults happen within a row or column that is close to the side of the network. Failed node 1 is replaced by spare node 1', and so on. In this case, the failures happen close to the side of the network, and it is not possible to replace the spare nodes as in the left-hand panel of Figure 6. In non-periodic 5P-stencil communication, all messages from spare nodes 4', 5', 6', and 7' to the neighbor nodes A to V go through the link between 3' and 4'. There are sixteen messages, since each node sends four messages, one to each of its neighbor nodes. This situation can happen when the number of faults is greater than or equal to seven. Below, we state the relation between the maximum number of possible message collisions (C_{max}) and the number of node failures (F_n). Note that when C_{max} is equal to one, then there is only one message on each network link, and there are no collisions.

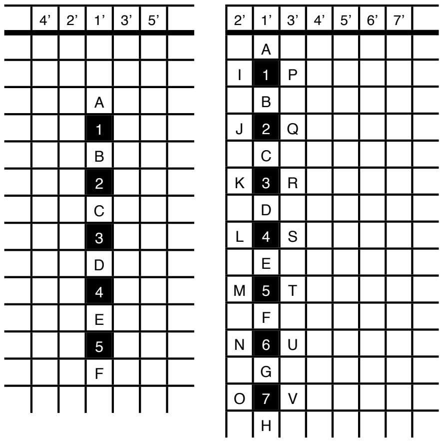

Figure 6. Worst-case scenarios for 0D sliding

$$C_{max} = \begin{cases} 2 \times F_n + 1 & F_n \leq 6 \; or \; torus \; topology \\ 4 \times (F_n - 3) & F_n \geq 7 \; and \; mesh \; topology \end{cases}$$

This worst-case scenario can be relaxed by having spare nodes allocated in the 2D(2,1) pattern. If the failures happen in the same row or column, then the spare nodes must be chosen from alternating sides. See Section .

1D sliding As described in the previous subsection, in the 0D sliding method, even if the closest spare node is chosen, the distance from the failed node is unlikely to be small. The 1D sliding method can avoid this situation, and it is shown in Figure 7. When node 21 fails, instead of replacing it with a spare node, the nodes of the column (or row) that include the failed node shift toward a spare node, as shown in the upper left-hand panel of the figure. In this way, the hop count in the 5D-stencil communication pattern is increased by only one. This is much smaller than occurs with the 0D sliding method.

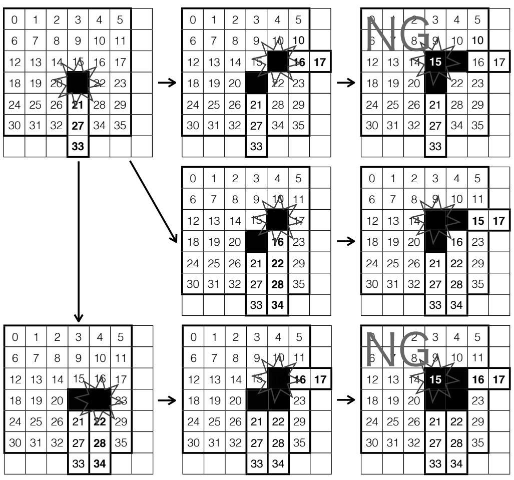

Figure 7. Example of 1D sliding

In terms of hop counts, the 1D sliding method is superior to the 0D sliding method; however, the recoverable number of faulty nodes is limited in some cases. Let us consider a case in which a second node (16) fails (again using the 2D(2,1) pattern); this is shown in Figure 7. This time, and the sliding direction is along the column. If a third node (15) fails, then there is no space left for the 1D sliding (top row of Figure 7). This situation can be avoided by sliding along the column direction after the second failure (middle row of Figure 7).

The number of nodes below which a third failure cannot be handled by the 1D sliding method is the product of the number of slidings in each direction. Thus, it is not a good idea to evenly distribute the sliding directions; instead, they should be as uneven as possible. Even when this is done, however, the 1D sliding method may be limited to three failures (bottom row in Figure 7).

The relation between the maximum number of message collisions and the number of failed nodes with the 2D(2,1) spare node allocation pattern can be expressed as shown below. Note that there may be cases in which this method cannot handle more than three node failures.

$$C_{max} = 2 + F_n$$

2D sliding, 3D sliding, ..., qD sliding In the 2D sliding method, the rows and columns of the node space are shifted by one unit to empty the row or column of the failed node (bottom panel of Figure 5). This 2D sliding method can handle only one node failure with the 2D(1,1) pattern or two node failures with the 2D(2,1) pattern.

If the network has a higher-dimensional Cartesian topology than 2D, then the 3D or higher-order sliding can take place in the same way. The highest degree of a sliding method is equal to the number of dimensions of a Cartesian network and this sliding method can handle up to the number of dimensions.

With XY routing, the messages pass orthogonally through the vacant rows or columns. All message routes are the same as they were before the failure. Thus, unlike the 0D and 1D sliding methods, although the hop counts are increased by one, message congestion can be avoided. Further, this behavior is independent of the communication pattern of the application.

Comparison of Proposed Methods

Figure 8 shows the number of possible message collisions versus the number of failed nodes for the 0D sliding method with the 2D(1,1) and 2D(2,1) spare node allocation patterns, 1D sliding with the 2D(2,1) pattern, and 2D sliding with the 2d(2,1) pattern. These numbers are obtained by our developed simulation program with which every possible combinations of node failures is simulated so that the number of message collisions in a 5P-stencil communication are counted at every link and the highest number of message collisions is reported. In this simulation, it is assumed that four messages of 5P-stencil communication are sent simultaneously.

As already described in Section , the number of possible message collisions with 0D sliding with the 2D(1,1) allocation pattern for a given number of failed nodes depends on the network topology (mesh or torus) when the number of faults is greater than six (upper left-hand panel in the

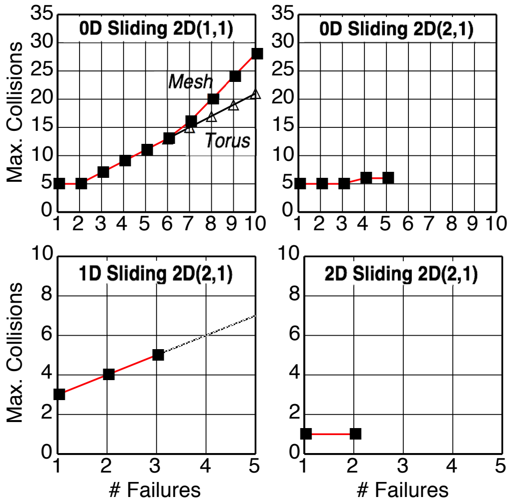

Figure 8. Comparison of 0D, 1D, and 2D sliding (5P-stencil, worst cases with exhaustive search)

figure). With 2D(2,1) case, up to 5 failures are simulated. It is possible to handle more number of failures with the 0D sliding method, however, the exponential growth of failure combinations was the obstacle for us to simulate more.

The 1D sliding method with the 2D(2,1) spare node allocation pattern can handle up to three failures perfectly. More number of failures can be handled when the failures happen at some specific locations. This is shown as a dashed line in Figure 8.

The 1D sliding method can handle no more than the number of spare nodes minus one, since the spare node at the corner of the 2D(2,1) allocation cannot be used. The 2D sliding with 2D(2,1) can handle only two failures as described before.

The sliding method has a good characteristic where node migration can take place in a pipeline fashion. Therefor the time to migrate nodes can be independent ($O(1)$, assuming the amount of data to be migrated are the same over nodes) from the number of migrating nodes.

Hybrid method The substitution methods described so far are independent and can be applied in a combined way. Figure 9 shows an example of a hybrid method. The first and second failures are handled by using the 2D sliding method (left-hand and middle panels), and the third failure is handled by using the 1D sliding method (right-hand panel). In this way, message collisions can be avoided even up to two failures, and the job can survive even with a greater number of failures.

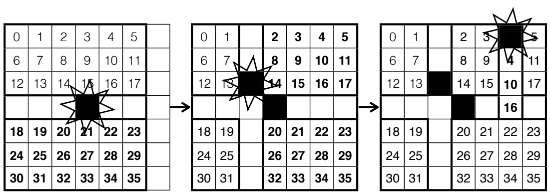

Figure 9. Example of Hybrid(2D+1D) Sliding

Thus, a hybrid sliding method can be expressed with the set of sliding methods and the order of their applications. As described in Subsection, the higher order sliding methods incur lower message collisions but the number of failure able to handle is smaller. Thus, the order of sliding methods to be applied in a hybrid method should be a descending order of the degree of sliding method. Hereinafter, a hybrid method will be expressed as “hybrid(2D+1D+0D),” for example, meaning 2D sliding is applied whenever it is possible, then 1D sliding method is applied, and finally 0D sliding method is applied. Since 0D sliding method can be applied in any circumstances, any hybrid method should have 0D sliding method as a last resort. In this paper, a hybrid sliding method combining all possible sliding methods with the descending order of degrees is also denoted as “hybrid(all)” for short. Another hybrid method combining all possible methods except X is denoted as “hybrid(-X).”

In the next section, some simulation results will be shown followed by an evaluation section on the K computer, BG/Q and TSUBAME 2.5[18]. One may argue that the numbers (percentages) of failed nodes simulated and evaluated in this paper are too many and not realistic. However, those simulations and evaluations are done to reveal the behaviour and characteristics of the proposed substitution methods. We believe that the research on how to utilize spare nodes is a new frontier of fault mitigation.

Simulation

Since the number of combination of failed nodes is the factorial of the number of failed nodes, it is impossible to simulate all possible cases especially when the number of nodes is large. Instead of having the exhaustive search, the simulation results shown in this section are obtained with random sampling.

Figure 10, Figure 11 and Figure 12 show this simulation results on 2D network (5P-stencil, 100x100, 2D(2,1) spare node allocation, 3,340,000 random cases), 3D network (7P-stencil, 12x12x12, 3D(2,1) spare node allocation, 3,686,400 random cases) and 3D network (7P-stencil, 24x24x24, 3D(2,1) spare node allocation, 3,686,400 random cases), respectively. Each graph compares hybrid sliding method (3 thick and grey lines, worst, average and best from top to down) and 0D sliding (3 thin and black lines, similarly, worst, average and best from top to down) method. “Best” in the legend means the lowest number of message collisions, “Worst” means the largest number of collisions and “Average” means the average of all cases.

In the graphs in Figure 10 and Figure 11, hybrid sliding method outperforms 0D sliding method in terms of best, average and worst cases in the smaller number of node failures. In Figure 12, however, the hybrid method outperforms 0D sliding not so much as in the cases of 100x100 and 12x12x12. Comparing the worst numbers, 0D sliding is better in the range of the number of node failures between 10 to 155. Comparing the average numbers, 0D sliding is better in the range of the number of node failures between 248 to 740.

It is obvious that stencil communication matches with Cartesian topology and no message collisions happen if the degree of the network is equal to or higher than the degree of the stencil communication. When a sliding method higher than 0D takes place, the network links connecting the

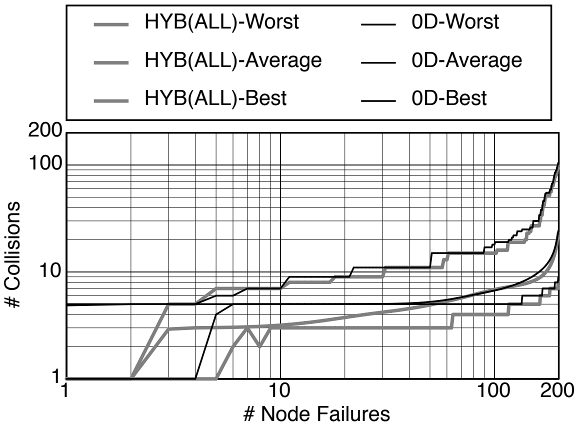

Figure 10. Hybrid(all) vs. 0D, 2D Network (100x100), 2D(2,1) Spare Nodes

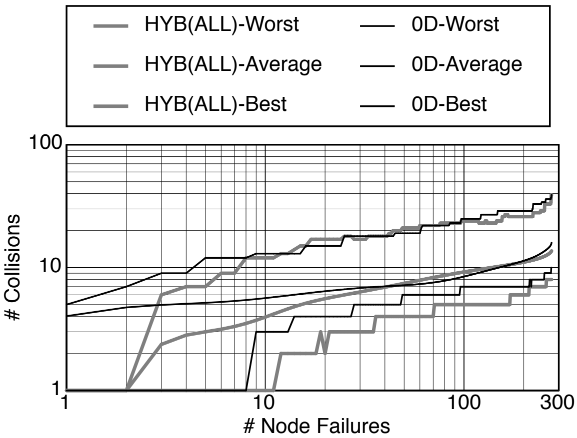

Figure 11. Hybrid(all) vs. 0D, 3D Network (12x12x12), 3D(2,1) Spare Nodes

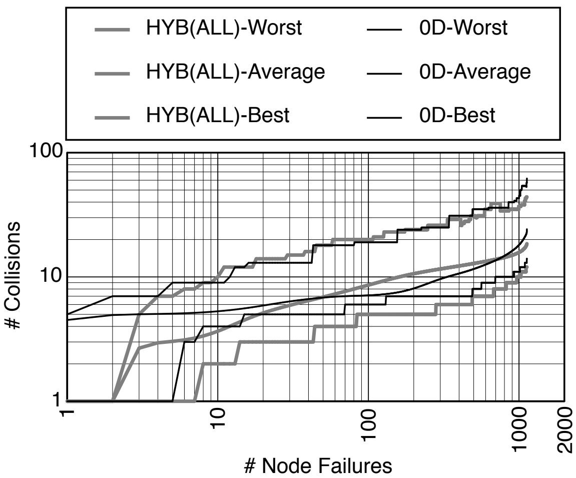

Figure 12. Hybrid(all) vs. 0D, 3D Network (24x24x24), 3D(2,1) Spare Nodes

nodes adjacent to the *sliding planes* can result in message congestion. Here, *sliding plane* is defined as the planes which surrounds the sliding nodes, except 0D sliding. The number of nodes (or the size of area of the plane) adjacent to the 1D sliding plane is smaller than that of 2D sliding. Although the maximum number of message collisions of 1D sliding and 2D sliding are the same, the number of network links having collisions caused by the 2D sliding is bigger than that of 1D. So, the possibility of adding more message collision(s) to the link(s) gets higher. From this viewpoint, 1D sliding might be better than 2D sliding. Based on this idea, hybrid(-2D) or hybrid(3D+1D+0D) might outperform hybrid(all).

Figure 13 shows the simulation results of hybrid(all), already shown in Figure 12, and hybrid(-2D) to compare. In this figure, the lines of hybrid(-2D) are thick. When attentions is paid to the average lines, hybrid(all) outperforms hybrid(-2D) at the rages at the left most part, from 1 to 18, and the right most part, from 993 to 1128 which is the number of spare nodes. Middle part excepting those ranges, however, hybrid(-2D) performs very well.

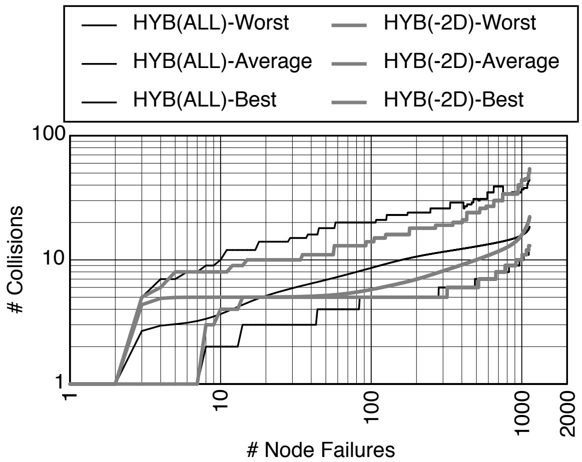

Figure 13. Hybrid(all) vs. Hybrid(-2D), 3D Network (24x24x24), 3D(2,1) Spare Nodes

Figure 14 shows the rates of which sliding methods are used in the sampling set. Figure 15 shows the accumulated selection rate. As shown in these figures, the first two node failures are substituted by using the 3D sliding method. Because spare node allocation is 3D(2,1), two is the maximum number of 3D sliding methods. 2D sliding method dominates until having 38 node failures. Then, 1D sliding method take over until 915 node failures. Finally 0D sliding method takes the rest. As shown in Figure 15, when all the spare nodes are substituted, 0D, 1D and 2D sliding methods occupy 25%, 71%, and 5% respectively.

Evaluations on K, BG/Q and TSUBAME 2.5

In this section, the sliding methods described so far are evaluated by using the actual supercomputers; the K computer JUQUEEN[19], a BG/Q machine, and TSUBAME 2.5 listed in Table 1. This experimental campaign will characterize the difference between the theoretical analysis and observed practical consequences.

The proposed sliding methods have been explained and discussed by using a 2D Cartesian network, however, the actual physical network can be more complex, having 5 or more number of dimensions, as shown in Table 1. Even if users require their jobs to run in 2D node spaces, those 2D node spaces are folded to fit in the actual network topologies. On the K computer, any 2D Cartesian node

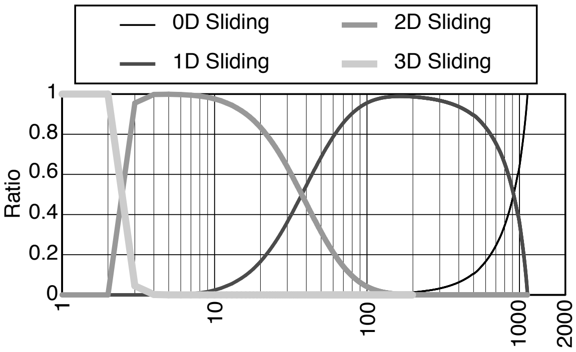

Figure 14. Selection Rate of Hybrid(all) Sliding, 3D Network (24x24x24), 3D(2,1) Spare Node Allocation

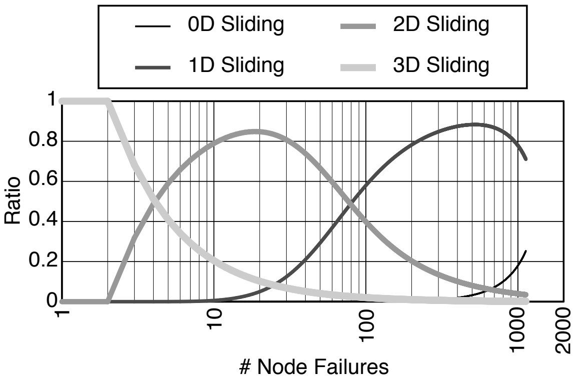

Figure 15. Accumulated Selection Rate of Hybrid(all) Sliding, 3D Network (24x24x24), 3D(2,1) Spare Node Allocation

planes are mapped to the 6D Tofu network so that the neighbor relationship of the 2D or 3D Cartesian topology can be preserved. On the BG/Q system, the node-rank mapping is the user's responsibility. To preserve the neighbor relationship of the 2D or 3D Cartesian topology, "snake-like pattern" is recommended[20]. On TSUBAME 2.5, the physical node space is one dimensional, dare to say. Anyhow, the mapping or folding of users' topologies to fit into a physical network topologies may affect the communication performance in different ways discussed so far.

Also in this paper, we have focused our analytical effort on the maximum number of message collisions, which has the implicit assumption that all messages are sent from nodes simultaneously, thereby always resulting in collisions if their path follows the same link. However, the number of message that can be sent simultaneously is dependent on network hardware features (i.e. the number of DMAs). When the maximum number of simultaneous sends is one, for example, the number of collisions is reduced.

Table 2 lists some characteristics of the K computer, BG/Q and TSUBAME 2.5. The K computer has 4 DMA engines and up to 4 messages can be sent simultaneously. BG/Q has 11 FIFOs and up to 10 messages can be sent simultaneously. On the other hand, the network of TSUBAME 2.5 is Infiniband[22]. TSUBAME has two Infiniband HCAs on a node, one of them is used in this paper to avoids the interference with the other jobs. So, TSUBAME 2.5, in this evaluation, can sent only one message at a time.

Table 2. K, BG/Q and TSUBAME 2.5

Machine Name	Topo.	# DMAs	Ratio to 1 Msg. Sending	
			5P-Stencil	7P-Stencil
K	6D Cart.	4	1.7	3.7
BG/Q[21]	5D Cart.	10	1.1	1.3
TSUBAME	FatTree	1†	4‡	6‡

† TSUBAME 2.5 has 2 IB networks but only one of them is used to avoid interference with the other jobs.
‡ Estimated values.

The rightmost two columns of this table show the ratios to send multiple messages simultaneously, four messages with 5P-stencil, six messages for 7P-stencil, based on the time to sent one message. These values, except TSUBAME 2.5, are measured by our program. In theory, 5 message collisions, for example, means the communication time gets slower 5 times. On the K computer, only 3 times slower communication time was observed because simultaneous 4 message sending takes 1.7 times of the time of sending one message ($3 \approx 5/1.7$)[23]. One possible reason to explain this slowness (1.7 with 5P-stencil and 3.7 with 7P-stencil) is the insufficient bandwidth between the memory and the network controller chip.

In the following subsection, the evaluation results of the K computer and BG/Q are shown, followed by the evaluation results of TSUBAME 2.5. This is because they have Cartesian network topologies while TSUBAME 2.5 has FatTree network topology. And the behaviour of the K and BG/Q is very different from that of TSUBAME 2.5.

Evaluations on K and BG/Q

Stencil Communication Figure 16 shows the simulation results on the 3D(12x12x12) network and the evaluation results of the K computer with the 12x12x12 node allocation. Spare node are allocated in the way of 2D(2,1). Message size of 4MiB is used on the K computer evaluations. The upper graph in this figure shows and compares the results using hybrid(all) sliding method, and the lower graph shows the results of using hybrid(-2D) sliding method. The 768 failure patterns (the set of failed nodes) are chosen from the worst cases in the simulation.

As shown in both graphs, the average lines observed on the K computer is almost always better than the result of the simulations. This is considered as that the actual network degree of the K computer (6D) is higher than the degree of the simulation (3D). The links provided by the additional dimensions give the paths to bypass resulting lower message collisions. Comparing the best lines, the simulation outperforms in the rage of less than or equal to 10 node failures. This is because of the sampling effect, the number of simulation cases are much bigger than that of evaluation and the best cases found in the simulation could not be found in the evaluations. The same situation happens on the worst cases.

To compare hybrid(all) and hybrid(-2D), Figure 17 shows the evaluation results of them (the same data used in Figure 16). The effect of hybrid(-2D) shown in this figure is very similar to the one found in Figure 13.

Figure 18 shows the simulation results on the 3D(16x8x8) network and the evaluation on BG/Q computer with the 16x8x8 node allocation. Spare nodes are allocated in the way

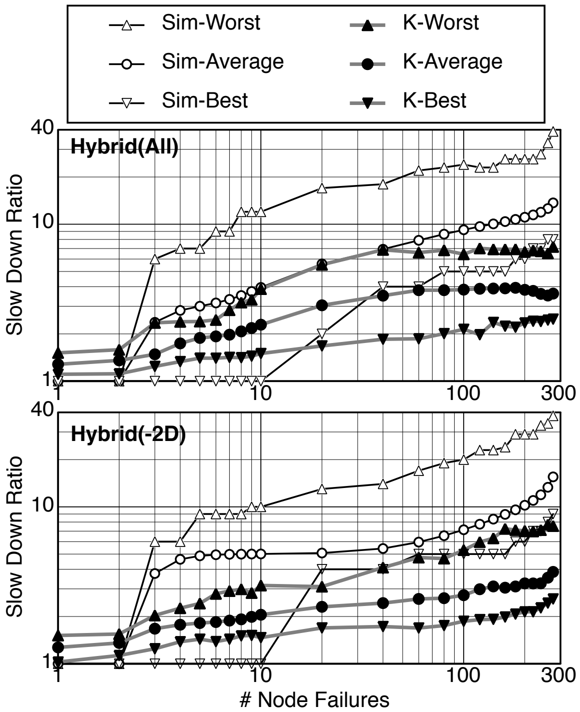

Figure 16. Stencil Communication on K (12x12x12), 3D(2,1) Spare Node Allocation

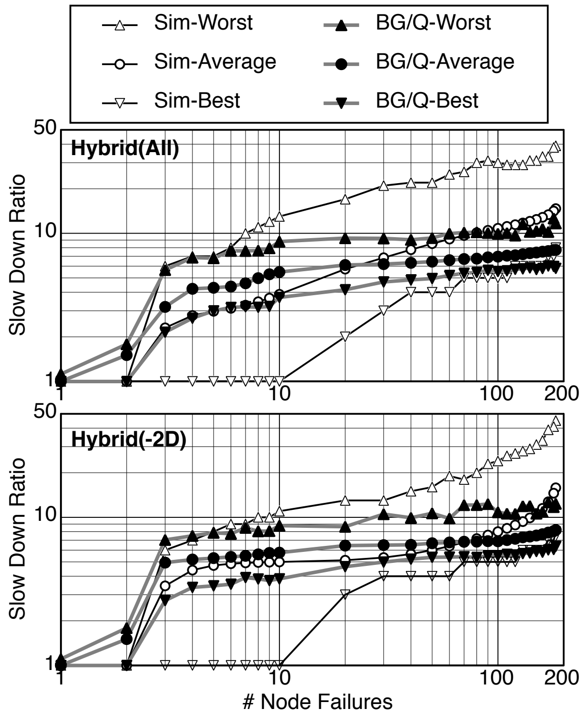

Figure 18. Stencil Communication on BG/Q (16x8x8), 3D(2,1) Spare Node Allocation

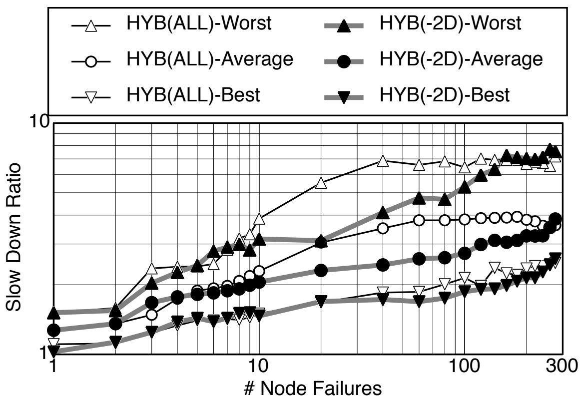

Figure 17. Hybrid(all) vs. Hybrid(-2D) on K (12x12x12), 3D(2,1) Spare Node Allocation

of 2D(2,1). Message size is 4MiB, the same as above cases on the K computer. As in the previous graphs, the upper graph in this figure shows and compares the results using hybrid(all) sliding method, and the lower graph shows the results of using hybrid(-2D) sliding method. Here again, the 768 failure patterns (the set of failed nodes) are chosen from the worst cases in the simulation.

Comparing the BG/Q results and the K computer results (Figure 16), the differences between the evaluation and the simulation in the range of less than or equal to 10 node failures are very small. This might come from the fact of that the network degree of the BG/Q (5) is less than the degree of the K computer (6) and/or that the network topology of BG/Q and the K computer are different. The BG/Q network topology is full 5 dimensions, while the K computer, the last three dimensions (A,B,C links out of X,Y,Z,A,B,C) are limited more frequently, and to form a 2x3x2 subnetwork unit (Tofu unit).

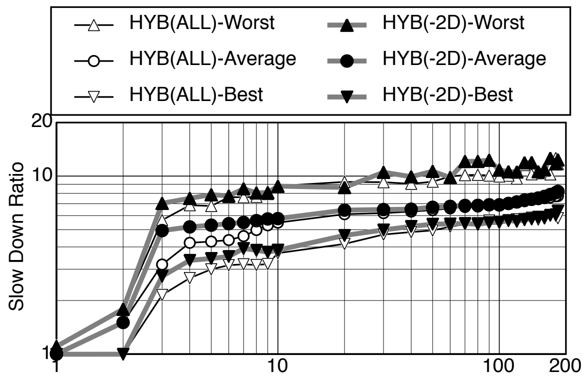

Figure 19. Hybrid(all) vs. Hybrid(-2D) on BG/Q (16x8x8), 3D(2,1) Spare Node Allocation

Figure 19 shows the evaluations of hybrid(all) and hybrid(-2D) (the same data used in Figure 18). Unlike the case on the K computer (Figure 17), the difference between hybrid(all) and hybrid(-2D) is very small.

Collective Communication Up to now, the peer-to-peer (P2P) communication performance in 5P-stencil communication pattern has been the primary focus. In this subsection, we will extend to the case of collective communication performance. The communication patterns of collective communications are more varied that the stencil pattern, thereby providing a wider insight about less regular P2P communication patterns as well.

On the K computer, the Tofu network supports hardware barrier. The other various collective communications are tuned so that the best performance can be obtained based on the Tofu network topology and characteristics[14]. The tuning of collective protocols is also very important for the Cray's Gemini network[17]. However, it is very difficult to predetermine optimized collective protocols for any possible set of node failures.

In order to use the tuned collective protocol for the Tofu netwotk, each MPI collective communication has some conditions for the physical shape of the communicator. Some of the conditions come from the special protocol tuned for the Tofu network, and the others come from implementation issues. When a substitution is made for a failed node, one or more of these conditions cannot be met and generic algorithms are used. Thus, the performance of the collective communication can degrade much more than that of the stencil communication, because the special tuned protocols cannot be applied in addition to the collision issue.

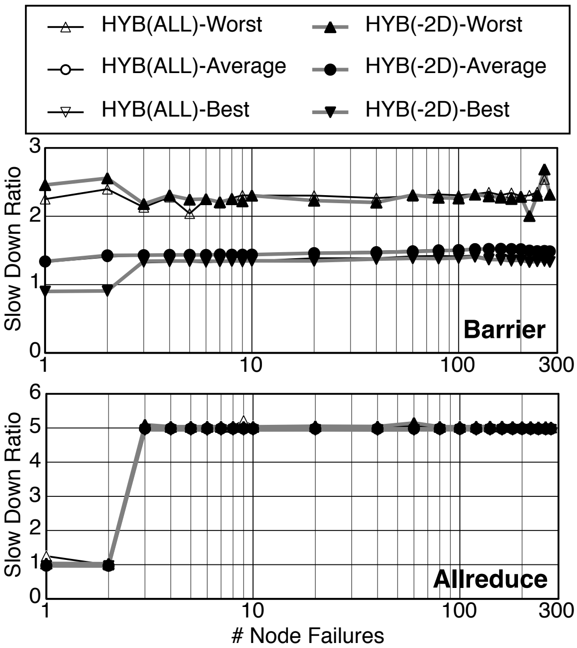

Figure 20. Barrier and Allreduce on K (12x12x12), 3D(2,1) Spare Node Allocation

Figure 20 shows the relative performance of barrier (upper graph) and allreduce (lower graph) collective communications based on the performance of no substitutions are made. Nodes of the evaluation job are allocated in 3D(12x12x12) and the sampling set is the same as the evaluation of stencil on the K computer in the previous subsection. The message size of allreduce is 64KiB.

On the allreduce case, the slowdown cannot be seen with one node failure and two node failures. In this evaluation, hybrid sliding methods are used and spare nodes are allocated in 2D(2,1), so 3D sliding method can be applied up to two node failures. The sliding method having the largest degree happens no message collision as described in Section . And this property of the sliding method with the largest degree does not break the conditions where the optimized allreduce protocol on the K computer can be applied.

Figure 21 shows the barrier and allreduce performance on BG/Q. We found that the collective performance on the node

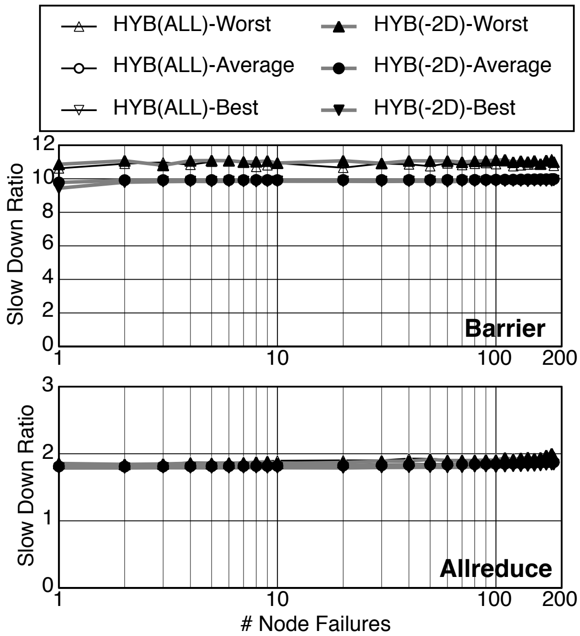

Figure 21. Barrier and Allreduce on BG/Q (16x8x8), 3D(2,10 Spare Node Allocation

set having a spare node set is slower than the cases without having any spare node. 8.2 times slower with the barrier operation and 1.8 times slower with the allreduce operation on BG/Q. Such slowdown cannot be seen on the K computer. This graph shows the slowdown ratio based on the time of the cases having spare nodes.

To make sure, we evaluated the barrier performance without the snake-like pattern mapping. When the spare nodes were allocated on one specific physical dimension out of the 5 dimensions of the BG/Q network, such barrier performance degradation could not be seen. However, when one node was excluded from `MPI_COMM_WORLD`, then the barrier performance was slowed down to one tenth. Thus, the best way is to allocate spare nodes according to the network topology and then apply the snake-like pattern to the node space without spare nodes.

Unlike the K computer collective cases, the barrier and allreduce performance of BG/Q is quite stable over the number of node failures independent from hybrid method. The 3D sliding method did not help to improve the situation.

Evaluations on TSUBAME 2.5

We ran the same evaluation programs used in the previous subsection on TSUBAME 2.5, 7x7x7 node space with 2D(2,1) spare node allocation. Unlike the K and BG/Q cases, we could not see any significant slowdown. The communication performance is almost constant within the range of 1.0 to 1.2 over the sliding methods and the number of node failures, without obvious correlation. To make sure of this phenomenon, we additionally evaluated the sliding methods with random node-rank mapping. As far as we tried, no obvious performance degradation can be seen.

One possible reason of this phenomenon is that the network topology of TSUBAME 2.5 is 2-stage FatTree. The node space is expressed in one dimensional way for the sake of convenience. However, there is no significance on the node numbers of the nodes connecting to the same "edge" switch. Any changes on node-rank mapping on those nodes have no effect.

The other factor of this phenomenon is that the network of TSUBAME 2.5 is Infiniband. Indeed, TSUBAME 2.5 has two network sets so that the communication bandwidth can be doubled by utilizing them in the multi-rail way[24]. One of the network sets is also used for Lustre file system and it is likely to have I/O traffics of the other jobs. So we decided to use the other network set to avoid the interference with the other jobs. Each Infiniband HCA has one DMA engine and, unlike the K computer and BG/Q, only one message can be sent from an HCA at a time. In a stencil communication, in theory, multiple messages can be sent simultaneously, however, TSUBAME 2.5 has the capability of sending only one message in this case. This situation is very different from the K computer able to send up to 4 messages and BG/Q able to send up to 10 messages simultaneously (Table 2). This leads to have less messages in the network and to have less chance to have message collisions. This may happen also with the collective communications.

Related Work

Ferreira *et al.* indicated that dual hardware redundancy while utilizing only 50% of the hardware resource, might be under some assumptions more efficient than the traditional checkpoint and restart method in Exascale systems This redundancies can be thought of as spare nodes. The difference is that the redundant nodes are *hotter*-standby than the hot-standby nodes waiting for the intermediate computational results. The spare nodes can be substituted for the failed nodes, and they can almost immediately take over the computations.

Domke *et al.* showed the difference in communication performance between the presence or absence of network failure (link or switch) over different network topologies and routing algorithms[13]. They analyzed the communication performance degradation when network links or switches failed; this was done by simulation using TSUBAME 2.0. In the K computer, the Tofu direct network has redundant routes to bypass failed nodes. However, a job is aborted and resubmitted by the operating system if it uses a failed part. In this work we focus on node failures rather than network failures. There is a long way to go until we reach the goal where any kind of failures, node and/or network, can be mitigated.

Brown *et al.* proposed a visualizing system of message traffics in a communication network[25] and they succeeded to identify hot-spots. In their case study using the samplesort program running on TSUBAME 2.5, 5% performance gain was obtained by avoiding the hot-spots which they discovered by using their tool. Conversely speaking, their paper reveals that finding an optimal node-rank mapping to level hot-spots according to network topology and communication pattern of an application is not an easy task.

Discussion

Node Utilization in a Multijob Environment

Most supercomputers use a batch scheduling system, in which many jobs run simultaneously. The spare node percentages shown in Figure 3 are for individuals jobs, not systems. If a machine has one million nodes and 100 jobs are running (for simplicity, assume that these jobs each require 10,000 nodes), then the overhead cost of spare nodes can exceed 10%.

The possibility that a job has a failed node is proportional to the number of nodes assigned to the job and execution time. Thus, the number of spare nodes must also be proportional to the number of nodes assigned and execution time. Therefore, the number of spare nodes allocated by the proposed method may be excessive when only a small number of nodes are required by a given job. Ideally, the curves shown in Figure 3 would be a horizontal line at the height determined by node failure rate, if the execution times are the same.

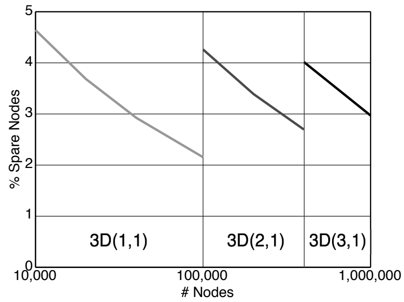

Figure 22. Combinations of spare node allocation methods

Figure 22 shows a countermeasure for this. Large jobs should have a higher-order spare node allocation method, and smaller jobs should have a lower-order method; this will allow the spare node percentage to approximate a horizontal line. In the example shown in Figure 22, the spare node percentage is kept in the range from 2% to 5% by using a combination of the 3D(2,1), 3D(2,1), and 3D(1,1) methods.

User-level vs. System-level Substitutions

So far in this paper, we have considered methods in which the spare nodes are allocated by the job. We would like to develop a framework that uses something like ULFM and that framework replaces faulty nodes with spare nodes, so that users do not need to be concerned with how failures are handled. When spare nodes are allocated and substitutions are determined by user programs, this is called as *user-level substitution*; when this is done at a lower system software level, it is called *system-level substitution*.

With user-level substitution, the user program is also invoked at each spare node, and it waits in hot-standby mode for the data to migrate from the failed node. This means that calling `MPI_Comm_spawn` is not required. On the other hand, system-level substitution can reduce the percentage of spare nodes, because spare nodes can be shared by several jobs. For example, spare nodes can be allocated at the boundaries of jobs, and these can be used to replace failed nodes on both sides of the boundary. However, it is not possible to have spare nodes on hot standby, as with user-level substitution. If the spare nodes are not adjacent to the job in which they are needed, this can result in uncontrollable message collisions with other jobs, and unexpected communication performance degradation.

Job Resubmission vs. Fault Mitigation

One may argue that a job can be aborted and then resubmitted using a checkpoint, instead of mitigating the fault. In this way, the problem of utilizing spare nodes and the degradation of communication performance, described above, can be avoided. Job resubmission, however, may incur a long turnaround time, especially when the system is heavily loaded, and user-level fault mitigation techniques, such as those described in[26], cannot be utilized. When considering which is better, there are many aspects to be considered. In this paper, we considered only the effect on communication performance. It is still an open question if it is better to resubmit a job or mitigate the fault.

Worm Eaten Node Space

So far in this paper, a job is allocated with a node space where there is no node failure at the beginning. Usually, when a node failure happens, the failed node is to be replaced with a new healthy node. To replace the failed node, firstly this node is unplugged from a rack and or chassis and then sanity node is plugged in. When a node is unplugged from a rack or chassis and if its network is a direct network, then the network switch (router) associated with the node is also gone. Thus, unplugging a node unintentionally simulates a switch failure, even with the assumption of no network failure. When a switch failure happens, network routing must be changed to bypass the failed switch and this may affects the other running jobs. So, the physical node replacement cannot take place soon after the node failure happens.

It is expected that MTBF is increasing in the future. And there will be the case where node failure happens much more frequently. Thus, the number of node failures will increase with in the interval of repairing nodes. This may result in the situation where large jobs might be allocated with a “worm-eaten” node space, instead of having all healthy nodes. In this case, node substitution methods described in this paper and/or algorithms to find an optimal node-rank mapping must be applied at the beginning of job execution and every time node failure happens. Therefore we believe the research on node substitution and the algorithm to find (sub)optimal node-rank mapping will be very important. This is left for our future work.

Summary and Future Work

In this paper, we considered methods for allocating spare nodes and replacing failed nodes in jobs whose rank-node mapping is critical to performance. We compared these methods in terms of communication performance following substitutions. The substitution methods are 0D, 1D, 2D, higher sliding methods. In the Stencil communication, the higher the order of the sliding method, the fewer message collisions but more failure distributions are unrecoverable for lack of spares. Thus, a combination of these methods would seem to be the best strategy. We also extended the evaluation to widely used collective operations.

Utilizing spare nodes influences various fields in high-performance computer design, hardware and software. As shown in the evaluations, a network topology plays an important role. It is expected that the communication performance degradation found in the substitutions can be relaxed by having a dynamic routing. The mapping of applications’ communication patterns and optimizations of collective communication patterns to fit in the available network topology becomes very difficult with the presence of failed node substitutions. Because there is no regular pattern where node failure happens and the number of node failure patterns are explosive. Therefore the assumption to have spare nodes has a significant impact on hardware and software design.

The research on this failed node substitution with spare nodes has just begun. We will continue investigating on this research topic.

Acknowledgements

We thank Dr. Norbert Attig at Jülich Supercomputing Center for allowing us access to the JUQUEEN platform. We also thank Dr. Franck Cappello, at Argonne National Laboratory, for his useful comments.

A part of the results in this paper is obtained by using the K computer at the RIKEN Advanced Institute for Computational Science (HPCI Project ID `hp150240`).

Funding

This research is partially supported by the CREST project of the Japan Science and Technology Agency (JST).

References

1. Cappello F, Geist A, Gropp WD et al. Toward Exascale Resilience: 2014 Update. *Supercomputing Frontiers and Innovations* 2014; 1: 1–28. URL `http://superfri.org/superfri/article/view/14/7`.
2. Di Martino C, Kalbarczyk Z, Iyer R et al. Lessons learned from the analysis of system failures at petascale: The case of blue waters. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. pp. 610–621. DOI:10.1109/DSN.2014.62.
3. Chen Y, Plank JS and Li K. CLIP: A checkpointing tool for message-passing parallel programs. In *Proceedings of the 1997 ACM/IEEE Conference on SuperComputing (SC’97)*. pp. 1–11.
4. Sato K, Maruyama N, Mohror K et al. Design and Modeling of a Non-blocking Checkpointing System. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC ’12, Los Alamitos, CA, USA: IEEE Computer Society Press. ISBN 978-1-4673-0804-5, pp. 19:1–19:10. URL `http://dl.acm.org/citation.cfm?id=2388996.2389022`.
5. Chen Z and Dongarra J. Algorithm-based fault tolerance for fail-stop failures. *IEEE Trans Parallel Distrib Syst* 2008; 19(12): 1628–1641. DOI:10.1109/TPDS.2008.58. URL `http://dx.doi.org/10.1109/TPDS.2008.58`.
6. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard Version 3.0, 2012. URL

http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf.

7. Bland W, Bouteiller A, Herault T et al. Post-failure recovery of MPI communication capability: Design and rationale. *International Journal of High Performance Computing Applications* 2013; 27(3): 244–254. DOI: 10.1177/1094342013488238. URL http://hpc.sagepub.com/content/27/3/244.abstract. http://hpc.sagepub.com/content/27/3/244.full.pdf+html.
8. Takefusa A, Ikegami T, Nakada H et al. Scalable and Highly Available Fault Resilient Programming Middleware for Exascale Computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC'14.
9. Teranishi K and Heroux MA. Toward local failure local recovery resilience model using mpi-ulfm. In *Proceedings of the 21st European MPI Users' Group Meeting*. EuroMPI/ASIA '14, New York, NY, USA: ACM. ISBN 978-1-4503-2875-3, pp. 51:51–51:56. DOI:10.1145/2642769.2642774. URL http://doi.acm.org/10.1145/2642769.2642774.
10. Fujita H, Dun N, Fang A et al. Using Global View Resilience (GVR) to add Resilience to Exascale Applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC'14.
11. Zheng Z, Chien A and Teranishi K. Fault Tolerance in an Inner-outer Solver: A GVR-enabled Case Study, 2014. http://www.vecpar.org/papers/vecpar2014_submission_4.pdf.
12. Strohmaier E, Dongarra J, Simon H et al. TOP500 Supermputer Site. URL http://www.top500.org/.
13. Domke J, Hoefler T and Matsuoka S. Fail-in-place Network Design: Interaction Between Topology, Routing Algorithm and Failures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '14, Piscataway, NJ, USA: IEEE Press. ISBN 978-1-4799-5500-8, pp. 597–608. DOI:10.1109/SC.2014.54. URL http://dx.doi.org/10.1109/SC.2014.54.
14. Sumimoto S. The MPI Communication Library for K computer: Its Design and Implementation. URL http://www.par.univie.ac.at/conference/eurompi2012/docs/s9t1.pdf. Invited talk at EuroMPI 2012 in Vienna.
15. Zhang W, Hou L, Wang J et al. Comparison Research Between XY and Odd-Even Routing Algorithm of a 2-Dimension 3X3 Mesh Topology Network-on-Chip. In *Proceedings of the 2009 WRI Global Congress on Intelligent Systems - Volume 03*. GCIS '09, Washington, DC, USA: IEEE Computer Society. ISBN 978-0-7695-3571-5, pp. 329–333. DOI:10.1109/GCIS.2009.110. URL http://dx.doi.org/10.1109/GCIS.2009.110.
16. IBM. *IBM System Blue Gene Solution: Blue Gene/Q System Administration*, Second Edition ed, 2013.
17. Peña AJ, Carvalho RGC, Dinan J et al. Analysis of Topology-dependent MPI Performance on Gemini Networks. In *Proceedings of the 20th European MPI Users' Group Meeting*. EuroMPI '13, New York, NY, USA: ACM. ISBN 978-1-4503-1903-4, pp. 61–66. DOI:10.1145/2488551.2488564. URL http://doi.acm.org/10.1145/2488551.2488564.
18. Endo T, Nukada A and Matsuoka S. Tsubame-kfc: A modern liquid submersion cooling prototype towards exascale becoming the greenest supercomputer in the world. In *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*. pp. 360–367. DOI:10.1109/PADSW.2014.7097829.
19. Stephan M. JUQUEEN: Blue Gene/Q - System Architecture, 2012. http://www.training.prace-ri.eu/uploads/tx_pracetmo/JUQUEENSystemArchitecture.pdf.
20. IBM. *IBM System Blue Gene Solution: Blue Gene/Q Application Development*, Second Edition ed, 2013.
21. Chen D, Eisley NA, Heidelberger P et al. The IBM Blue Gene/Q Interconnection Network and Message Unit. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '11, New York, NY, USA: ACM. ISBN 978-1-4503-0771-0, pp. 26:1–26:10. DOI:10.1145/2063384.2063419. URL http://doi.acm.org/10.1145/2063384.2063419.
22. Infiniband Trade Association. InfiniBand, http://www.infinibandta.org/. URL http://www.infinibandta.org/.
23. Hori A, Yoshinaga K, Herault T et al. Sliding substitution of failed nodes. In *Proceedings of the 22Nd European MPI Users' Group Meeting*. EuroMPI '15, New York, NY, USA: ACM. ISBN 978-1-4503-3795-3, pp. 14:1–14:10. DOI:10.1145/2802658.2802670. URL http://doi.acm.org/10.1145/2802658.2802670.
24. Liu J, Vishnu A and Panda DK. Building multirail infiniband clusters: Mpi-level design and performance evaluation. In *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference*. pp. 33–33. DOI:10.1109/SC.2004.15.
25. Brown K, Domke J and Matsuoka S. Hardware-centric analysis of network performance for mpi applications. In *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*. pp. 692–699. DOI:10.1109/ICPADS.2015.92.
26. Davies T, Karlsson C, Liu H et al. High Performance Linpack Benchmark: A Fault Tolerant Implementation Without Checkpointing. In *Proceedings of the International Conference on Supercomputing*. ICS '11, New York, NY, USA: ACM. ISBN 978-1-4503-0102-2, pp. 162–171. DOI:10.1145/1995896.1995923. URL http://doi.acm.org/10.1145/1995896.1995923.