2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks

Grid-oriented Process Clustering System for Partial Message Logging

Hideyuki Jitsumoto[1], Yuki Todoroki[2], Yutaka Ishikawa[1,2]
Information Technology Center[1], Dept. of Information Science[2]
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8654, Japan
Email: jitumoto@cc.u-tokyo.ac.jp,
{todoroki, ishikawa}@is.s.u-tokyo.ac.jp

Mitsuhisa Sato
Center for Computational Science
University of Tsukuba
1-1-1 Tennodai, Tsukuba-City, Ibaraki 305-8577, Japan
Email: msato@cs.tsukuba.ac.jp

***Abstract*—In a computer cluster composed of many nodes, the mean time between failures becomes shorter as the number of nodes increases. This may mean that lengthy tasks cannot be performed, because they will be interrupted by failure. Therefore, fault tolerance has become an essential part of high-performance computing. Partial message logging forms clusters of processes, and coordinates a series of checkpoints to log messages between groups. Our study proposes a system of two features to improve the efficiency of partial message logging: 1) the communication log used in the clustering is recorded at runtime, and 2) a graph partitioning algorithm reduces the complexity of the system by geometrically partitioning a grid graph. The proposed system is evaluated by executing a scientific application. The results of process clustering are compared to existing methods in terms of the clustering performance and quality.**

keyword: fault tolerance, message logging, graph partition

I. Introduction

Extreme-scale high-performance computing systems encounter high failure ratios. This is because of the number of computing elements involved, as well as the fact that each computing element is constructed to be of high density and low power consumption. As a result, fault tolerance techniques are indispensable for long-running, massively parallel applications in extreme-scale environments.

Although rollback recovery with checkpoints (C/R: checkpoint/restart) is a popular fault tolerance method in high-performance computing systems, recent work suggests that naive C/R is ineffective, because the C/R cost is larger than the mean time between failures (MTBF) of the system [1].

Hence, a number of studies have attempted to reduce the cost of C/R. Broadly speaking, C/R can be classified into two approaches, coordinated or uncoordinated. Coordinated methods form one consistent checkpoint for all processes, using a coordination mechanism such as synchronization. In contrast, uncoordinated methods employ many checkpoints for each process, and select a consistent group of checkpoints on restarting. Because of complexities in the selection of consistent checkpoints and the domino effect (chained roll-backing to maintain consistency), feasible uncoordinated checkpoints use roll-forwarding from the checkpoint to the fail point with message logging.

Coordinated checkpoints and uncoordinated checkpoints with message logging, which is called log-based checkpointing, have strengths and weaknesses. Coordinated checkpoints only incur a cost at the checkpoint. However, they require forced roll-backing for all processes when a failure occurs. Forcing all processes to roll-back increases power consumption and performance overhead. With log-based checkpointing, only the failed processs has to restart, but incurs large resource and performance overheads when recording the log into stable storage or memory, even if there are no failures. In extreme-scale high-performance computing, the disadvantages of both methods cause a critical increase in the C/R cost.

Partial message logging [4] is a hybrid method with coordinated and log-based checkpointing. This method achieves partial roll-backing and reduces the logging cost. By splitting processes into a number of groups, it utilizes coordinated checkpointing within the same group, and only logs the communication between groups. Using this method, there is a trade-off between the granularity of the forced roll-backing and the size of the communication logs. Basically, this is a clustering optimization problem that can be solved using a graph partitioning method. Specifically, the processes, communication patterns, and number of communications are treated as the vertices, edges, and weights of each edge. The method then splits the graph with the minimum edge-cut. Previous work uses graph partitioning tools like METIS [5], and analyzes the communication pattern prior to execution. However, this is not realistic here, because the pre-execution step would require significant resources and a long execution time to create the same communication pattern as the performance execution.

We propose an on-the-fly analysis method with limiting area for enabling checkpoint. Our method, the Runtime Process Clustering System (RPCS), focuses on the main application loops, and analyzes the first loop execution. We consider graph partitioning tools to be too high performance, adaptive, and accurate for partial log-based checkpointing. Thus, we propose a rough partitioning method with process mapping information for the grid graph.

II. Related Work

A. Previous Work on Partial Message Logging

Partial message logging or hybrid checkpointing was first mentioned in [4], [6] and others. In [3], a clustering method for effective partial message logging was studied, and this

978-1-4799-2233-8/14 $31.00 © 2014 IEEE
DOI 10.1109/DSN.2014.72

is closely related to our work. This study used graph partitioning tools that require the maximum partition size, then applied a bisection-based method to satisfy this constraint. Until minimizing evaluation formula which proposed on [3], the communication graph is repeatedly partitioned as two parts. The following formula based on [3] evaluates the wasted computer resources:

$$Cost = \alpha \times L + \beta \times R \quad (1)$$

In this formula, α and β are the costs associated with message logging and restarting processes after failure, respectively. They are determined by the execution environment. L is the ratio of logged data, and R is the proportion of processes that must be restarted after a failure. These are determined by the application characteristics. RPCS uses the same method, but employs a different algorithm for partitioning.

B. Graph Partitioning Tools

METIS is a widely used graph partitioning tool. It provides two algorithms that can be applied to general graph partitioning problems. These algorithms combine vertices that are closely connected, and construct graphs that are smaller than the original. These smaller graphs are then partitioned into subgraphs using a simple sorting algorithm based on the edge weights. Both phases depend on the number of edges E—hence, METIS has $O(|E|)$ computational complexity. The SCOTCH[7] tool can provide a mapping between two graphs. It can also consider graph partitioning problems, and uses almost the same method as METIS by default.

The graph partitioning problem is also used in the field of region segmentation. Recursive graph bisection (RGB) and recursive coordinate bisection (RCB) achieve graph partitioning by sorting the coordinates of the vertices. Their computational complexity is $O(NlogN)$ for a graph consisting of N vertices. RPCS also uses the coordinates of vertices, but only requires information about vertices on the boundary. The computational complexity of our method is $O(\sqrt{N})$ for a 2D grid graph, and $O(\sqrt[3]{N})$ for a 3D grid graph. Therefore, our algorithm is more scalable than the other simple partitioning algorithms discussed here.

III. RPCS: Runtime Process Clustering System for Partial Message Logging

The proposed RPCS incorporates the runtime analysis of communication between nodes, and geometrical graph partitioning with a fast/scalable algorithm.

A. Motivation

Previous studies[2] assumed that the communication pattern of an application is given. However, it is difficult to obtain the communication pattern without executing the application. Although advance execution can obtain the complete communication pattern, this requires significant resources that do not contribute to the results of the application. In addition, it is difficult for application users to estimate the communication pattern of execution from a small dataset, and it is unrealistic to expect an application programmer to extract communication code from an application as a micro-benchmark. Therefore, we propose a dynamic analysis method that can easily be employed by application users.

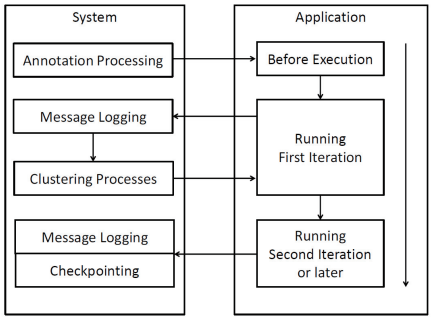

Fig. 1. Scenario of partial message logging

Secondly, previous studies use graph partitioning tools such as METIS and SCOTCH. These tools can treat various applications and are highly focused on accuracy. For dynamic analysis, because the performance execution includes the analysis cost, the execution time is important. We thus reduce the accuracy and adaptability to reduce the graph partitioning time.

B. Runtime Analysis of Communication and Partial Message Logging

Figure 1 illustrates the analysis of communication at runtime. High-performance computing applications usually have loops that repeat the same calculations and communications on each iteration. These loops typically take up most of the execution time of an application. On such a loop, the communication pattern can be determined from the first iteration. Because RPCS only logs the complete communication pattern on the first iteration, the cost of determining the communication pattern is less than that using the advance execution method.

Under RPCS, an application user need only insert some annotation in the application source code as directives that function as I/O to the graph partitioning tools. The most important annotation is the declaration of an analysis loop that includes mainly calculation and communication (Figure 2). On the first iteration of a declared loop, RPCS logs the communication and analyzes its graph. Then, on the second (or later) iteration, RPCS only logs the communication between different groups. In this study, RPCS does not consider a process checkpointer; therefore, we need some checkpointing tool to achieve fault tolerance.

C. Fast and Scalable Geometrical Graph Partitioning

To reduce the graph partitioning cost, we focus on the geometrical mapping of processes on physical nodes. Many modern supercomputers support a 2D or 3D torus network topology. Thus, application programmers distribute processes to physical nodes according to their connectivity. Therefore, processes allocated to nearby nodes can be grouped in the same cluster of partial message logging. RPCS uses a grid network topology and the geometry of process mapping on the physical nodes. This grouping method has three advantages in terms of process based grouping: 1) node-based methods group

```
.....
  // declaration of main loop
  #pragma RPCS MAIN_LOOP
  for (i ···) {
    ......
    //calculations
    Calc_or_solver();
    //communications
    MPI_XXXX();
  }
.....
```

Fig. 2. Declaration of main loops with directive method

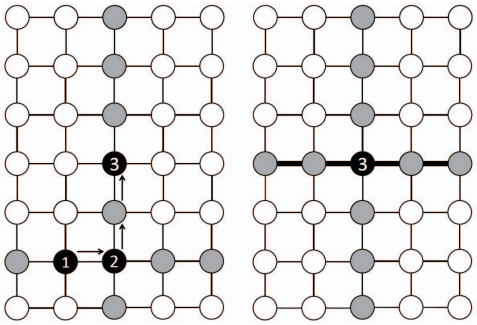

Fig. 3. Flow of partitioning algorithm; Finding center vertex (left): Node1 is result of Step2, Node2 is result of Step3 to Step4 and Node3 is result of Step5 to Step6. And measureing width along each axis and bisection (right)

processes faster than process-based methods, because there are fewer vertices and edges in the communication graph; 2) all applications executed on a 2D or 3D torus can be treated by a grid topology model, even if the application does not use a grid topology (a limitation will be mentioned later); and 3) fewer processes need to be rolled back when node failure occurs, because processes on failed nodes are included in the same group. The accuracy of node-based methods depends on the quality of process allocation on the physical nodes. We can assume the quality is high, because the optimization of process allocation is important in improving application performance.

A flowchart of our algorithm to divide the physical node topology into a grid graph is shown in Figure 3. This figure uses a 2D grid topology as an example to simplify the explanation of our algorithm. In most high-performance computing applications, each process only knows its neighbor processes, but is aware of the whole process mapping of the application. RPCS detects (roughly) the whole process mapping, and divides it using the following bisectional method:

1) Detect nodes executing processes using the hostname table, such as "machinefile" of MPI[8], and convert process communication log to node communication log.
2) Determine a temporal reference vertex (in our implementation, the temporal start vertex corresponds to the physical node running process rank 0).
3) Measure the width along the X-axis.
4) Move reference vertex to the middle vertex of the line along the X-axis determined in step 3.
5) Measure the width along the Y-axis from the new reference point.
6) Move reference vertex to the middle vertex of the line along the Y-axis determined in step 5.
7) Measure the width along the X-axis again from the new reference point.
8) Bisect the graph by crossing the axis that has the narrower width.

Some region splitting methods, such as RGB and RCB, use a concept that is similar to our method. However, these methods have $O(NlogN)$ computational complexity for a graph of N nodes, because all of the nodes must be sorted. The method used in RPCS has $O(\sqrt{N})$ computational complexity under an ideal process mapping on a 2D grid topology, because the proposed method checks the vertices as it crosses the whole graph object. The basic idea of this algorithm can easily be applied on a 3D grid topology. Specifically, we add a Z-axis, measure its width, and use the surface as a border between partitions, instead of the border line used in 2D. In this case, the computational complexity is $O(\sqrt[3]{N})$.

The RPCS method cannot be easily applied to general graph partitioning, which is often represented as a process communication graph. This is because our algorithm cannot deal with edge weights, i.e., the number and volume of communications. Our method can be used in applications that have similar edge weights between neighbor nodes. The most common application that can use this method is the stencil computation of a simply shaped object. However, the stencil model is a very basic method for high-performance applications because of the physical node topology. In addition, if the process mapping of an application has a well-optimized communication pattern, our method can be effective.

IV. Prototype Implementation

A. Message Logging for Graph Partitioning

On the first iteration of the main loop, RPCS accesses the communication log by hooking MPIs through PMPI. Therefore, the application user need only set the environmental value, LD_PRELOAD, or links with the hooking library. RPCS only includes a wrapper for point-to-point MPI communication, because logging collective communication with PMPI is difficult, and does not have the capability of MPI implementation. However, communication in an application that has a stencil model is almost point-to-point because of its algorithm. Thus, we can currently tolerate this limitation. Furthermore, the message contents are not logged (though logging the contents of point-to-point communication is possible), because that information is not necessary for process clustering. For overall fault tolerance with partial message logging, we will combine a log-based checkpointer that includes a fault/recovery model and content logging that satisfies fault/recovery model of combined checkpointer.

B. Communication Between RPCS and Application

Tentatively, we communicate information between the system and application through files. We assume a shared parallel file system for this communication. However, our design is such that this method of communication is replaceable.

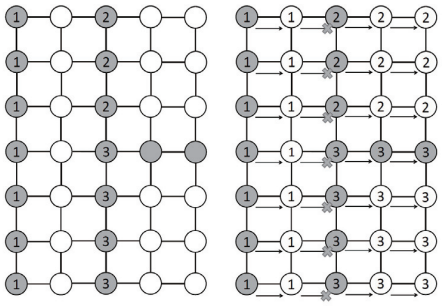

Fig. 4. Flow of broadcasting group ID: Border vertices receive group ID (left), and group ID is propagated to another border (right)

C. Broadcasting the Grouping Result

When using some common graph partitioning tools, the processes need only broadcast a group number corresponding to the processes, but this approach presents a problem in our method. Because our algorithm only determines the border between groups, RPCS must convert border information to group information for each process.

The flow of 2D broadcasting for RPCS is shown in Figure 4, and proceeds as follows:

1) On partitioning, RPCS saves group ID 1 for the Y-axis that was calculated as the edge of the grid.
2) RPCS also saves later group IDs appropriately for each bisectional partitioning.
3) Processes that have a group ID send their ID to the next process along the X-axis until the information reaches a process that already has a group ID.

In 3D grouping, the same broadcasting technique can be used by applying a similar modification as for the clustering algorithm for a grid graph.

V. Evaluation

We evaluate the performance of the proposed RPCS by three methods. First, we measure the partitioning time required for a 2D grid graph. Second, we confirm the size of the edge-cut message to evaluate the accuracy of RPCS with an actual application. Third, we estimate the cost of partial logging-based fault tolerance using the evaluation method of a previous study. We use GEOFEM[9] as the actual application. This provides several common functionalities of FEM applications. In this evaluation, we run an elastic analysis of a rectangular solid on GEOFEM, and evaluate the partitioning given by RPCS. We also use CGPOP[11] that is a conjugate gradient (CG) solver that was developed as a miniature performance-tuning application for the Parallel Ocean Program (POP). GEOFEM has a 3D grid graph topology, and CGPOP has a 2D grid graph topology.

To form the communication pattern graph, we use a parallel implementation of the FX10 supercomputer, which has an architecture based on the K computer[10]. The specification of a node is shown in Table I. Graph partitioning is performed serially by one calculation node of FX10. As mentioned above, the communication between application and graph partitioner is achieved using a file system.

TABLE I. Experimental Environment

SYSTEM	Oakleaf FX-10
CPU	SPARC64TM IXfx (1.848 GHz, 16 cores, 12 MB L2 cache)
Memory	32 GB/Node
OS	RedHat Enterprise Linux for FX10 Kernel Linux 2.6.25.8
Network	Tofu interconnect (5 $GB/sx2$)

A. Graph Partitioning Performance

We compared the time required for the clustering process with that from two METIS algorithms (multilevel recursive bisection (pmetis) and multilevel K-way partitioning (kmetis)) and a naive implementation of RCB. The computational complexities of RPCS, pmetis, kmetis, and RCB are $O(k\sqrt{n}), O(kE), O(E)$, and $O(knlogn)$, where k is the number of partitions, n is the number of vertices, and E is the number of edges. For this evaluation, we used a large 2D grid graph whose vertices are connected to four adjacent vertices by edges of the same weight, because the number of vertices can easily be scaled. The number of groups was fixed to 100 because the computational complexity of several algorithms scales linearly with the number of groups.

Figures 5 and 6 show the results of this simulation. In a grid graph, the number of edges is a constant multiple of the number of vertices, so E is equivalent to n when considering computational complexity. RPCS is much faster than the other algorithms. In addition, RPCS is scalable, even if the number of vertices increases. The time required by the proposed algorithm increases in proportion to $\sqrt{n}$, whereas the time complexity of the other algorithm scales linearly with n. These results also show that RPCS is faster than other widely used algorithms when applied to a grid graph, and can be easily scaled to larger graphs.

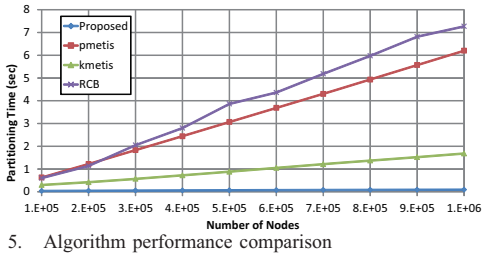

Fig. 5. Algorithm performance comparison

B. Edge-cut of Partitioning

Our method decreases the accuracy to improve the grouping time. Thus, we should confirm the quality of grouping. One criterion for evaluating the grouping of processes is the edge-cut, which is defined as the amount of communication between processes belonging to different groups. We executed CGPOP with 22500 (150 x 150) processes on 1440 (36 x 40) physical nodes and GEOFEM with 21942 (28 x 28 x 28) processes on 1440 (12 x 12 x 10) physical nodes. We fixed the

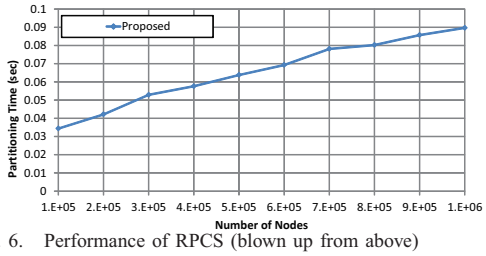

Fig. 6. Performance of RPCS (blown up from above)

number of groups to 16. The results for the optimized node allocation are shown in Figure 7 and Figure 8, where each bar represents the edge-cuts produced by each method. These show that our partitioning method obtains a better solution than both METIS algorithms. The algorithms except RPCS got disadvantage for dealing with various types of graphs. With complicated graph, our approach will get degradation of quality of partitioning. However, we consider our approach will not become worse than RCB because our approach get advantage from node allocation optimization that is done on application execution. We expect this optimization transforms complicated graph to simple graph, because most users allocate application process to be suitable for the network topology between physical nodes.

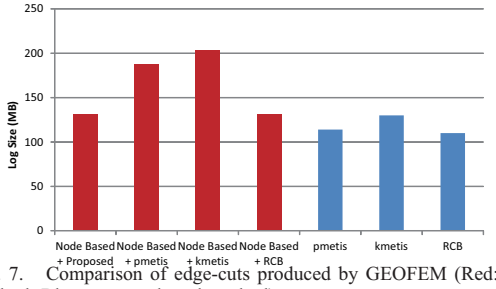

Fig. 7. Comparison of edge-cuts produced by GEOFEM (Red: node-based method, Blue: process-based method)

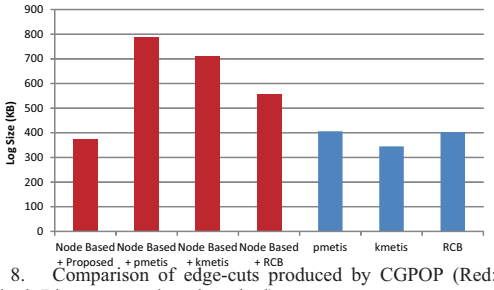

Fig. 8. Comparison of edge-cuts produced by CGPOP (Red: node-based method, Blue: process-based method)

C. Cost of Fault Tolerance

The number of restarted processes following a fault occurrence is an important measure of the quality of grid partitioning. We can estimate the fault tolerance cost using the evaluation formula proposed in [3]. We assume the same conditions as in [3] based on [2], whose execution environment set the message logging impact on performance $\alpha = 23\%$ and the checkpoint/restart impact on MTBF $\beta = 12.4\%$. This value of β was calculated from an MTBF of 1 day, and checkpoint/restart times of 30 min. Thus, the evaluation formula can be written as follows:

$$process\_fault\_cost = 23\% \times \frac{B}{D} + 12.4\% \times \frac{\sum_k |P_k|^2}{|P|^2} \quad (2)$$

In this formula, $|P|$ is the total number of processes, $|P_k|$ is the number of processes belonging to group k, and B, D are the size of the logged messages and the total size of messages. Then, $\frac{B}{D}$ represents the proportion of logged communication, and $\frac{\sum_k |P_k|^2}{|P|^2}$ represents the expected proportion of processes to be restarted. For this evaluation, we extend this formula to node faults as follows:

$$node\_fault\_cost = 23\% \times \frac{B}{D} + 12.4\% \times \frac{\sum_{n=0}^{N} \sum_k^{k \in A_n} |P_k|}{|P| \times N} \quad (3)$$

Two new variables, N and A_n, represent the number of physical nodes and the set of group IDs belonging to processes running on node n. Hence, $\sum_k^{k \in A_n} |P_k|$ represents the expected number of rolled-back processes when node n fails.

We estimated these costs for GEOFEM and CGPOP. We used the same configuration as in the previous section for GEOFEM, and ran 22500 (150 x 150) processes on 1440 (36 x 40) physical nodes with CGPOP. Both applications used SCOTCH for node allocation. The estimation results are shown in Figures 9 and 10. The left bar of each column represents the process fault cost, and the right bar represents the node fault cost. Our method gives a process fault cost that is almost equal to the cost score of other approaches. This implies that our method does not have problems in attaining a grouping balance or with excessive edge-cuts. The cost of node faults follows a similar trend to the other node-based methods, because $\frac{\sum_{n=0}^{N} \sum_k^{k \in A_n} |P_k|}{|P| \times N}$ and $\frac{\sum_k |P_k|^2}{|P|^2}$ tend to the same value. Comparing process- and node-based methods, we can observe an increased cost for pmetis, kmetis, and RCB. Especially, the score of CGPOP become much worse than the GEOFEM. This is because CGPOP includes two domain, ocean and land, and processes allocated domain of land do not communicate with other processes. They can be placed on any physical node, and the graph partitioning tool does similarly. As a result, graph partitioning tends to distribute processes running on the same node to other nodes in CGPOP. Contrary to CGPOP, GEOFEM does not produce non-communication processes, so A_n remains low. However, in all evaluations, our method does not suffer from any large degradation in quality when compared with the other methods. This shows that our node-based approach is not inferior in terms of either edge-cuts or grouping balance. In addition, our approach is able to tolerate node failures.

VI. Conclusion

We designed and implemented a system supporting partial message logging. RPCS has three main features: the runtime analysis of communication, physical node-based partitioning, and a scalable algorithm for 2D or 3D grid graphs. Annotation

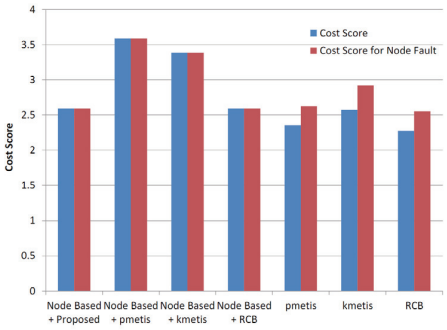

Fig. 9. Cost score of GEOFEM

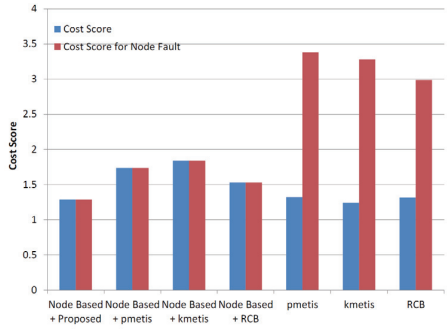

Fig. 10. Cost score of CGPOP

of RPCS processes can be added in advance, allowing us to insert the code necessary for fault tolerance. This inserted code calls an actual grouping procedure. The graph partitioning component divides processes into several groups using a new algorithm with a node-based topological approach. The node-based method reduces the size of the graph to be partitioned, and decreases the number of processes that must be rolled back in case of node failure. In addition, our topological approach can partition the grid graph without checking all vertices. It has a computational complexity of O($\sqrt[d]{N}$), where d and N are the dimension of the topology and the number of vertices. The characteristics of our method were demonstrated by a series of evaluations. These showed that RPCS reduces the number of processes rolled back when a node fault occurs, and achieves faster process grouping without any large degradation in quality. In future work, we will integrate a full fault tolerance framework by combining RPCS with a log-based checkpointer. In addition, we will expand the evaluation scope to several applications that have more complicated communication topologies.

ACKNOWLEDGMENT

The part of this work was supported by ANR-JST FP3C: Collaborative Project between Japan and France ”Framework and Programing for Post Petascale Computing” and JST Crest ppOpen-HPC: ”Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning”.

REFERENCES

[1] Franck Cappello, Fault Tolerance in Petascale/Exascale Systems: Current Knowledge, Challenges and Research Opportunities, INRIA, IJHPCA 23(3): 212-226, 2009

[2] Amina Guermouche, Thomas Ropars, Elisabeth Brunet, Marc Snir and Franck Cappello, Uncoordinated Checkpointing Without Domino Effect for Send-Deterministic Message Passing Applications, Proceedings of IPDPS 2011

[3] Thomas Ropars, Amina Guermouche, Bora Ucar, Esteban Meneses, Laxmikant V. Kale, and Frank Cappello, On the use of cluster-based partial message logging to improve fault tolerance for MPI HPC applications, In proceedings of the 17th international conference on Parallel processing, Vol. Part I, Euro-par ’11, pp. 567-578, Berlin, Heidelberg, 2011.

[4] Meneses, E., Mendes, C. L. and Kale, L. V., Team-Based Message Logging: Preliminary Results, Cluster, Cloud and Grid Computing (CCGrid), 2010.

[5] Karypis, G. and Kumar, V., A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, SIAM J. Sci. Comput., Vol.20, No.1, pp. 359-392, 1998

[6] Jin-Min Yang, Kim Fun Li, Wen-Wei Li, and Da-Fang Zhang, Trading off logging overhead and coordinating overhead to achieve efficient rollback recovery. Concurr. Comput.: Pract. Exper., 21(6):819-853, 2009

[7] F. Pellegrini and J. Roman., Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs., In High-Performance Computing and Networking, pp. 493-498, Springer, 1996

[8] The MPI Forum, MPI: A Message Passing Interface Standard. Version 3.0, available at http://www.mpi-forum.org

[9] H. Okuda, K. Nakajima, M. Iizuka, L. Chen, and H. Nakamura, Parallel finite element analysis platform for the earth simulator: Geofem. Computational Science (ICCS) 2003, p. 700, 2003

[10] M. Yokokawa, F. Shoji, A. Uni, M. kurokawa and T. Watanabe, The K computer: Japanese next-generation supercomputer development project, In International Symposium on Low Power Electronics and Design (ISLPED) 2011, pp. 371-372, IEEE, 2011

[11] Andrew I. Stone, John M. Dennis, and Michelle Mills Strout, The CGPOP Miniapp, Version 1.0, Technical Repoert CS-11-103 July, 2011