On the Use of Cluster-Based Partial Message Logging to Improve Fault Tolerance for MPI HPC Applications

Thomas Ropars[1], Amina Guermouche[1,2], Bora Uçar[3], Esteban Meneses[4], Laxmikant V. Kalé[4], and Franck Cappello[1,4]

[1] INRIA Saclay-Île de France, France (thomas.ropars@inria.fr)
[2] Université Paris-Sud (guermou@lri.fr)
[3] CNRS and ENS Lyon, France (bora.ucar@ens-lyon.fr)
[4] University of Illinois at Urbana-Champaign, USA
({emenese2,kale,cappello}@illinois.edu)

Abstract. Fault tolerance is becoming a major concern in HPC systems. The two traditional approaches for message passing applications, coordinated checkpointing and message logging, have severe scalability issues. Coordinated checkpointing protocols make all processes roll back after a failure. Message logging protocols log a huge amount of data and can induce an overhead on communication performance. Hierarchical rollback-recovery protocols based on the combination of coordinated checkpointing and message logging are an alternative. These partial message logging protocols are based on process clustering: only messages between clusters are logged to limit the consequence of a failure to one cluster. These protocols would work efficiently only if one can find clusters of processes in the applications such that the ratio of logged messages is very low. We study the communication patterns of message passing HPC applications to show that partial message logging is suitable in most cases. We propose a partitioning algorithm to find suitable clusters of processes given the communication pattern of an application. Finally, we evaluate the efficiency of partial message logging using two state of the art protocols on a set of representative applications.

1 Introduction

The generation of HPC systems envisioned for 2018-2020 will reach Exascale from 100s of millions of cores. At such scale, failures cannot be considered as rare anymore and fault tolerance mechanisms are needed to ensure the successful termination of the applications. In this paper, we focus on message passing (MPI) HPC applications. For such applications, fault tolerance is usually provided through rollback-recovery techniques [9]: the state of the application processes is saved periodically in a checkpoint on a reliable storage to avoid restarting from the beginning in the event of a failure. In most cases, rollback-recovery protocols are used flat: the same protocol is executed for all processes.

Flat rollback-recovery protocols have several drawbacks. Coordinated checkpointing requires to restart all processes in the event of a failure leading to a massive waste of resources and energy. Message logging protocols need to log all messages contents and delivery order, which leads to high storage resource occupation, high communication overhead and high energy consumption.

One way to cope with these limitations is to use hierarchical rollback-recovery protocols [11, 12, 15, 16, 21]. Clusters of processes are defined during the execution and different protocols are used inside and between clusters, giving a hierarchical aspect to the protocol. Typically, a partial message logging protocol logs messages between clusters to confine the effects of a process failure to one cluster and a coordinated checkpointing protocol is used within clusters [21, 15]. The efficiency of these protocols depends on two conflicting requirements: i) the size of the clusters should be small to limit the rollbacks in the event of a failure; ii) the volume of inter-cluster messages should be low to limit the impact of logging on failure free performance.

This paper provides three main contributions: i) it shows that suitable process clustering can be found in most applications; ii) it proposes a bisection-based partitioning algorithm to find such clusters in an application based on its execution communication pattern; iii) it shows that partial message logging limits the amount of computing resources wasted for failure management compared to flat rollback-recovery protocols.

The paper is organized as follows. Section 2 details the context of this work, and presents the related work on MPI applications communications analysis. Section 3 analyzes a set of execution communication patterns in MPICH2. Section 4 presents our bisection-based partitioning algorithm, designed to address the partitioning problem for a partial message logging protocol. Using this algorithm, we evaluate the performance of two *state-of-the-art* partial message logging protocols [11, 15] on a set of representative MPI HPC applications. Results are presented in Section 5. Finally, conclusions are detailed in Section 6.

2 Context

In this section, we first present existing hierarchical rollback-recovery protocols. Then we present the applications studied in this paper. Finally, we detail the related work on analyzing characteristics of MPI HPC applications.

Hierarchical Rollback-Recovery Protocols Rollback-recovery techniques are based on saving information during the execution of an application to avoid restarting it from the beginning in the event of a failure. One of the main concerns is the amount of resources wasted with respect to computing power or energy. Several factors are contributing to this waste of resources: i) the overhead on performance during failure free execution; ii) the amount of storage resources used to save data; iii) the amount of computation rolled back after a failure.

Rollback-recovery protocols are usually divided into two categories: checkpointing-based and logging-based protocols [9]. In checkpointing protocols, processes checkpoints can be either coordinated at checkpoint time, taken independently or induced by the communications. For all these protocols, a single failure implies the rollback of all processes in most cases, which is a big waste of resources. On the other hand, message logging protocols log the content as well as the delivery order (determinant) of the messages exchanged during the execution of the application to be able to restart only the failed processes after a failure. However, logging all messages during a failure free execution can be very wasteful regarding communications and storage resources.

Hierarchical rollback-recovery protocols divide the processes of the applications into clusters and apply different protocols for the communications inside a cluster and for the communications among clusters. Our work focuses on partial message logging protocols [11, 12, 15, 21] which apply a checkpointing protocol inside the clusters and a message logging protocol among the clusters. They are attractive at large scale because only one cluster has to rollback in the event of a single failure. These protocols can work efficiently if the volume of inter-cluster messages is very low in which case the cost of message logging is small.

In this paper, we use two of these protocols for evaluations. Meneses et al. [15] use a coordinated checkpointing protocol inside clusters. Intra-cluster messages determinants are logged to be able to replay these messages in the same order after the failure and reach a consistent state. Considering a single failure, determinants can be logged in memory. In this study [15] is comparable to [12] and [21]. Guermouche et al. [11] propose an uncoordinated checkpointing protocol without domino effect, relying on the send-determinism of MPI HPC applications. Using this protocol, an ordered set of p clusters can be defined. Only messages going from one cluster to a *higher* cluster are logged, limiting the number of clusters to roll back after a failure to $(p+1)/2$ on average. Thanks to send-determinism, this protocol does not require any determinant to be logged and can tolerate multiple concurrent failures.

Applications Studied We study a representative set of MPI HPC applications to see if partial message logging could be used. Thirteen dwarfs have been defined and seven of them represent seven main classes of computation and communication patterns corresponding to numerical methods for high-end simulation in the physical sciences [2]: dense linear algebra, sparse linear algebra, spectral methods, N-body methods, structured grids, unstructured grids, and MapReduce. This paper does not consider MapReduce applications because rollback-recovery is not adapted in this case. Our set of applications includes five of the NAS Parallel Benchmarks (NPB) [3] containing BT, LU, CG, FT, and MG; three of NERSC-6 Benchmarks [1] (GTC, MAESTRO, and PARATEC); one of the Sequoia Benchmarks, http://asc.llnl.gov/sequoia/benchmarks/, (LAMMPS); and an Nbody kernel. Table 1 summarizes the dwarfs covered.

HPC Applications Communications Characteristics The communication patterns of most of the applications considered in this paper have already been published [1, 18]. Previous studies highlighted some properties. First, most MPI

Dense linear algebra	Sparse linear algebra	Spectral methods	N-body simulations	Structured grids	Unstructured grids
BT, LU, PARATEC	CG, MAESTRO	FT, PARATEC	GTC, LAMMPS, Nbody	MG, GTC, MAESTRO, PARATEC	MAESTRO

Table 1. Dwarfs covered by the studied applications

applications make use of collective communications, but in general with a very small payload size that remains invariant with respect to the problem size [20]. Second, the communication graphs of many MPI HPC applications have a low degree of connectivity [13], which might indicate that processes can be partitioned into clusters.

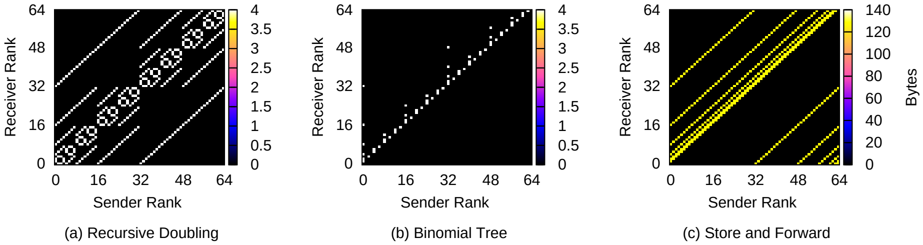
3 Communication Patterns

In this section, we first study the communication patterns of some collective operations in MPICH2. Then, we present communication patterns we collected by running some applications. To get the communication patterns of MPI applications execution, we modified the code of MPICH2[1] to collect data on communications. The applications run on Rennes Grid'5000 cluster over TCP.

We focus on collective communications because they could generate patterns that are hard to cluster (they involve all processes in the application). Figure 1 presents the set of communication patterns used for MPICH2 collective communications, for a power-of-two number of processes (64 processes) and short messages (4 bytes). Details on the implementation of collective communications in MPICH2 can be found in [19]. The recursive doubling algorithm, Fig. 1(a), is used to implement *MPI_Allgather* and *MPI_Allreduce* operations. The recursive halving algorithm, used in *MPI_Reduce_scatter*, has the same communication pattern. A binomial tree, Fig. 1(b), is used in *MPI_Bcast*, *MPI_Reduce*, *MPI_Gather* and *MPI_Scatter*. These two patterns are easily clusterizable using for instance clusters of size 16. The last pattern, corresponding to a store-and-forward algorithm, is the one used for *MPI_Alltoall*. This pattern is more difficult to cluster, but the use of *MPI_Alltoall* should be limited in applications targeting very large scale. For large messages, many of the collective operations do involve communications between all application processes. Clustering is also difficult to apply to such patterns. However, as mentioned in Section 2, the payload for collective communications is usually small: on the set of applications we tested, we only found this pattern in NPB FT.

Although some of the collective communication patterns define natural clusters (see Fig. 1(a)), some others do not (see Fig. 1(c)). Furthermore, the point-to-point communications can render the patterns more sophisticated, so much so that the size or the number of clusters cannot be known a priori. Figure 2 presents the communication pattern of MAESTRO and GTC executed on 512 processes. As exemplified by these two applications, an automated means of clustering is strictly required to identify clusters of processes in an application.

[1] http://svn.mcs.anl.gov/repos/mpi/mpich2/trunk:r7592

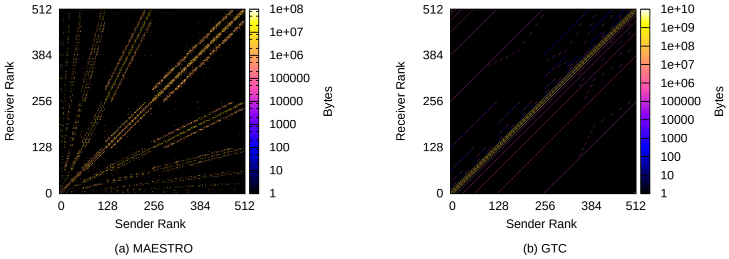


(a) Recursive Doubling

(b) Binomial Tree

(c) Store and Forward

Fig. 1. Communication patterns inherent in collective communications of MPICH2



(a) MAESTRO

(b) GTC

Fig. 2. Communications patterns of two applications

4 Partitioning for Partial Message Logging Protocols

Here, we propose a bisection-based partitioning algorithm to automate clustering for partial message logging protocols. The objectives of the clustering are to reduce the inter-cluster communications, to increase the number of clusters, and to limit the maximum size of a cluster. We first illustrate the limits of existing tools and then present our new method.

4.1 Two Possible Approaches

A simplified version of the partitioning problem in which the objectives are to minimize the size of the logged messages and the maximum size of a part corresponds to the NP-complete graph partitioning problem (see the problem ND14 in [10]). This can be easily seen by considering a graph whose vertices represent the processes and whose edges represent the communication between the corresponding two processes. Using heuristics for the graph partitioning problem would require knowing the maximum part size. This can be done but requires an insight into the application and the target machine architecture.

A common variant of the above graph partitioning problem specifies the number of parts (in other words, specifies the average size of a part) and requires parts to have similar sizes (thusly reducing the maximum size of a partition). The problem remains NP-complete [4]. Tools such as MeTiS [14] and Scotch [17] can be used to solve this problem. A possible but not an economical way to use those tools in our problem is to partition the processes for different number of parts (say 2, 4, 8, ...) and try to select the best partition encountered.

4.2 Bisection-Based Partitioning

Bisection based algorithms are used recursively in graph and hypergraph partitioning tools, including PaToH [6], MeTiS and Scotch, to partition the input graph or hypergraph into a given number of parts. Simply put, for a given number K of parts, this approach divides the original graph or hypergraph into two almost equally sized parts and then recursively partitions each of the two parts into $K/2$ parts until the desired number of parts is obtained.

We adapt the bisection based approach and propose a few add-ons to address our partitioning problem. The proposed algorithm is seen in Algorithm 1. The algorithm accepts a set of P processes and a matrix M of size $P \times P$ representing the communications between the processes where $M(u, v)$ is the volume of messages sent from the process u to the process v. The algorithm returns the number of parts $K^\star$ and the partition $\Pi^\star = \langle P_1, P_2, \ldots, P^\star_K \rangle$. In the algorithm, the operation $\Pi \leftarrow \Pi \ominus P_k \oplus P_{k_1} \oplus P_{k_2}$ removes the part P_k from the partition Π and adds P_{k_1} and P_{k_2} as new parts, where $P_k = P_{k_1} \cup P_{k_2}$. We use $M(U, V)$ to represent the messages from processes in the set U to those in the set V.

Algorithm 1 The proposed partitioning algorithm

```
Input: The set of $P$ processes = $\{p_1, p_2, \ldots, p_P\}$; the $P \times P$ matrix $M$ representing
    the communications between the processes
Output: $K^\star$: the number of process parts; $\Pi^\star = \langle P_1, P_2, \ldots, P^\star_K \rangle$: a partition of
    processes
 1: $K^\star \leftarrow K \leftarrow 1$;   $B^\star \leftarrow B \leftarrow 0$
 2: $\Pi^\star \leftarrow \Pi \leftarrow \langle P_1 = \{p_1, p_2, \ldots, p_P\} \rangle$   ▶ a single part
 3: while there is a part to consider do
 4:   Let $P_k$ be the largest part
 5:   if ShouldPartition($P_k$) then
 6:     $\langle P_{k_1}, P_{k_2} \rangle \leftarrow$Bisect($P_k, M(P_k, P_k)$)
 7:     if AcceptBisection($P_{k_1}, P_{k_2}$) then
 8:       $K \leftarrow K + 1$
 9:       $\Pi \leftarrow \Pi \ominus P_k \oplus P_{k_1} \oplus P_{k_2}$   ▶ replace $P_k$
10:       $B \leftarrow B + \sum M(P_{k_1}, P_{k_2})$   ▶ Update volume of inter-parts messages
11:       if BestSoFar($\Pi$) then
12:         $\Pi^\star \leftarrow \Pi$;   $B^\star \leftarrow B$;   $K^\star \leftarrow K$   ▶ save for output
13:   else
14:     mark $P_k$ in order not to consider it again at lines 3 and 4.
15: return $\Pi^\star$
```

The algorithm uses bisection to partition the given number of processes into an unknown number of parts. Initially, all the processes are in a single part. At every step, the largest part is partitioned, if it should be, into two (by the subroutine BISECT), and then if the bisection is acceptable (determined by the subroutine ACCEPTBISECTION), that largest part is replaced by the two parts resulting from the bisection. If this new partition is the best one seen so far (tested in subroutine BESTSOFAR), it is saved as a possible output. Then, the algorithm proceeds to another step to pick the largest part. We now add a few details. Let P_k be the largest part, then if P_k should be partitioned (tested in the subroutine SHOULDPARTITION), it is bisected into two P_{k_1} and P_{k_2} and tested for acceptance; if not accepted then the bisection is discarded and P_k remains as is throughout the algorithm. Notice that when a bisection of P_k is accepted, then P_k is partitioned for good; if any BESTSOFAR test after this bisection returns true, then P_k will not be in the output $\Pi^\star$.

The computational core of the algorithm is the BISECT routine. This routine accepts a set of processes and the communication between them and tries to partition the given set of processes into two almost equally sized parts by using existing tools straightforwardly. One can use MeTiS or Scotch quite effectively if the communications are bidirectional ($M(u,v) \neq 0 \implies M(v,u) \neq 0$), in which case $\frac{M+M^T}{2}$ can be used. Alternatively one can use PaToH as each communication (bidirectional or not) can be uniquely represented as an edge.

The routines SHOULDPARTITION, ACCEPTBISECTION, and BESTSOFAR form the essence of the algorithm. The routine BESTSOFAR requires a cost function to evaluate a partition. The cost function should be defined based on the metric the user wants to optimize, e.g., the performance overhead, and the characteristics of the targeted partial message logging protocol. It is function of the part sizes and of the volume of inter-parts messages. We define a cost function for both of the mentioned rollback-recovery protocols later in Section 5.1.

SHOULDPARTITION is used to stop partitioning very small parts. It returns false if the size of the part in question is smaller than a threshold. Although we mostly used 1 as threshold in our experiments, using a larger threshold will make the algorithm faster. One could select this threshold based on a minimal part size considering the properties of the target machine architecture.

The routine ACCEPTBISECTION returns true in two cases. The first one is simply that for Π and $\Pi' = \Pi \ominus P_k \oplus P_{k_1} \oplus P_{k_2}$, we have $\text{cost}(\Pi') \leq \text{cost}(\Pi)$ according to the cost function. If this does not hold, we may still get better partitions with further bisections. To this end, we have adapted the graph strength formula [7]. For a partition $\Pi = \langle P_1, P_2, \ldots, P_K \rangle$, we use $\text{strength}(\Pi) = \frac{B}{K-1}$ as its strength, B being the volume of inter-parts data. We accept the bisection if $\text{strength}(\Pi') \leq \text{strength}(\Pi)$. If the bisection reduces the strength, then it can be beneficial, as it increases the size of the logged messages only a little with respect to the number of parts.

5 Evaluation

This section presents our experimental results. For the evaluations, we consider the two partial message logging protocols described in Section 2. We start by defining a cost function for each of them, evaluating the amount of wasted computing resources for failure management. Then we present the results obtained by running our implementation of the proposed partitioning algorithm using these cost functions, on the set of applications executions data we collected. We first show that these results overcome coordinated checkpointing and message logging protocols. Then we validate our partitioning algorithm by comparing our results to the results obtained by the existing graph partitioning tools.

5.1 Defining a Cost Function

We define a cost function to evaluate the amount of computing resources wasted by failure management, for the two partial message logging protocols. We would like to stress that the cost functions we define in this section do not give a very precise evaluation of the protocols, because a lot of parameters would have to be taken into account. We use some parameters, that we consider realistic, to provide an insight of the protocols cost. The main goal in this section, is to show that we manage to find clustering configurations with good trade-off between the clusters size and the amount of data logged.

We model the cost of a partial message logging protocol for a given clustering configuration as a function of the total volume of logged messages and the size of the clusters. We use a formula of the form

$$\mathrm{cost}(\Pi) = \alpha \times L + \beta \times R \tag{1}$$

where L is the ratio of logged data and R is the ratio of processes to restart after a failure. The multipliers α and β are the cost associated with message logging and with restarting processes after a failure, respectively.

To evaluate the overhead of message logging, we use results from [11], where message logging impact on communications (latency and bandwidth) on a high performance network results in a 23% performance drop on average. Therefore, $\alpha = 23\%$ can be considered as a maximum theoretical overhead induced by message logging for a communication-bounded application. For the sake of simplicity, we do not consider in this study the cost of logging intra-cluster messages reception order in Meneses et al. protocol [15].

To evaluate the amount of computing resources wasted during recovery after a failure, we consider the global performance of the system. While a subset of the processes are recovering from a failure, the other processes usually have to wait for them to progress. We assume that the resources of the hanging processes could be temporarily allocated for other computations, until all application processes are ready to resume normal execution. So β includes only the amount of resources wasted by rolling back processes after a failure.

To compute β, we consider an execution scenario with the following parameters [5]: a Mean Time Between Failure ($MTBF$) of 1 day; 30 minutes to checkpoint the whole application (C); 30 minutes to restart the whole application (Rs). The optimum checkpoint interval (I) can be computed using Daly's formula [8]: $I = \sqrt{(}2 \times C \times (MTBF + Rs)) = 297min$. This formula was originally used in coordinated checkpointing protocols, but we think we can safely apply it to our case. Assuming that failures are evenly distributed a failure occurs at time $\frac{I}{2}$ on average. The total time lost per MTBF period can be approximated by $\frac{I}{2} + Rs = 179min$, that is $12,4\%$ of the period, giving $\beta = 12.4\%$.

The two other parameters, L and R, are protocol-dependent. In the protocol proposed by Meneses et al. [15], all inter-cluster messages are logged and only the cluster where the failure occurs roll back. Considering a partition $\mathit{\Pi} = \langle P_1, P_2, \ldots, P_K \rangle$ which entails a volume of B inter-cluster messages over a total volume of messages D,

$$\mathrm{cost}(\mathit{\Pi}) = 23\% \times \frac{B}{D} + 12.4\% \times \frac{\sum_k |P_k|^2}{P^2} \tag{2}$$

where $\frac{\sum_k |P_k|^2}{P^2}$ is the average number of processes restarting after a failure if the failures are evenly distributed among the P application processes. As described in Section 2, the protocol proposed by Guermouche et al. [11] only logs half of the inter-cluster messages but requires to roll back $\frac{K+1}{2}$ clusters on average after a failure:

$$\mathrm{cost}(\mathit{\Pi}) = 23\% \times \frac{B}{2 \times D} + 12.4\% \times \frac{K+1}{2} \times \frac{\sum_k |P_k|^2}{P^2} \quad . \tag{3}$$

5.2 Results

We first evaluate the cost of partial message logging. Table 2 presents the results obtained by running our partitioning algorithm with PaToH and cost function (2). The results show that for all applications except FT and MAESTRO, our tool was able to find a clustering configuration where less than 15% of the processes have to roll back on average after a failure, while logging less than 20% of the data exchanged during the execution.

To get an insight on the quality of the costs obtained, they might be compared to the cost of a coordinated checkpointing and a message logging protocol. With a coordinated checkpointing ($L = 0$, $R = 1$), the amount of wasted resources would be 12.4%. With a message logging protocol ($L = 1$, $R = \frac{1}{P}$), this amount would be 23%. The results show that for all applications except FT, using the partial message logging protocol minimizes the cost. For applications based on dense linear algebra (BT, LU, SP, and PARATEC) or N-body methods (GTC, LAMMPS, and Nbody), the cost obtained is always below 5%.

Table 3 presents the results obtained by running our partitioning algorithm with PaToH and cost function (3). We evaluate it only with the applications having a symmetric communication pattern, because in this protocol, messages logging is not bidirectional and our partitioning tool does not take this in account yet. In all the tests except MAESTRO, we managed to find clusters such that the ratio of rolled back processes is around 55% while logging less than 5.3% of the messages. The cost function of the two partial message logging protocols cannot be compared because the first one is only valid for a single failure case while the second one can handle multiple concurrent failures.

	Size	Nb Clusters	Min/Max cluster size	Processes to roll back	Log/Total Amount of data (in GB)	Cost
NPB BT	1024	8	123/133	12.5%	201/1635 (12.3%)	4.37
NPB CG	1024	32	32/32	3.1%	910/5606 (16.2%)	4.12
NPB FT	1024	2	502/522	50%	432/864 (50%)	17.7
NPB LU	1024	16	64/64	6.25%	67/700 (9.7%)	3.0
NPB MG	1024	8	128/128	12.5%	20/107 (18.5%)	5.8
NPB SP	1024	8	123/133	12.5%	366/2989 (12.2%)	4.4
GTC	512	16	32/32	6.25%	240/3654 (6.6%)	2.3
MAESTRO	1024	4	252/259	25%	55/309 (17.7%)	7.17
PARATEC	1024	13	64/128	8.5%	2262/23914 (9.4%)	3.23
LAMMPS	1024	8	127/129	12.5%	0.3/4 (7.6%)	3.3
Nbody	1024	30	31/61	3.5%	80/2733 (2.9%)	1.1

Table 2. Partitioning for the protocol of Meneses et al.

	Size	Nb Clusters	Min/Max cluster size	Processes to roll back	Log/Total Amount of data (in GB)	Cost
NPB LU	1024	16	64/64	53.1%	34/700 (4.8%)	7.7
MAESTRO	1024	4	250/262	62.5%	27/309 (8.9%)	9.78
PARATEC	1024	16	61/68	53.1%	1285/23914 (5.3%)	7.8
LAMMPS	1024	8	127/129	56.2%	0.15/4 (3.8%)	7.9

Table 3. Partitioning for the protocol of Guermouche et al.

To validate our bisection-based partitioning algorithm, we used PaToH, MeTiS and Scotch as outlined in Section 4.1. Table 4 presents the result of running the three tools on PARATEC with the cost function (2). We ran the experiment with $K = 2, 4, 8, 16, 32, 64$, and also with $K = 13$ which is the result provided by our tool (see table 2). First, it has to be noticed that the number of clusters found by our tool is close to the number of clusters that minimizes the cost function with PaToH and Scotch. Second, only Scotch manages to slightly improve the cost compared to our tool. However, if we use our tool with Scotch instead of PaToH, we obtain exactly the same cost. This is mostly due to the fact that Scotch obtains well balanced partitions for any given K, and that the β in (2) is relatively high. In cases where β is smaller, the partitioner has a higher degree of freedom. Whereas the proposed method automatically exploits this leeway, it is hard to specify the imbalance parameter for the three existing tools we have used (we have not reported these experiments, but this was observed for β around 5). We conclude from these results that the proposed algorithm manages to find good clusters without taking a number of clusters as input. The row "run time" below PaToH contains the running time of PaToH with the given K compared to that of the proposed algorithm (which finds $K = 13$). As is seen, two runs of PaToH take more time than a single run of the proposed algorithm (despite the overheads associated with repeated calls to the library, including converting the data structure).

	2	4	8	13	16	32	64
PaToH	7.07	4.52	3.42	3.47	3.36	3.95	5.04
run time	0.60	0.57	0.60	0.59	0.60	0.61	0.60
MeTiS	7.47	5.36	5.38	5.47	5.92	3.82	4.96
Scotch	7.09	4.56	3.44	3.37	3.22	3.67	4.94

Table 4. Evaluating three tools on PARATEC with the cost function (2).

6 Conclusion

Partial message logging protocols, combining a checkpointing and a message logging protocol, are an attractive rollback-recovery solution at very large scale because they can provide failure containment by logging only a subset of the application messages during the execution. To work efficiently, such protocols require to form clusters of processes in the application, such that inter-cluster communications are minimized. In this paper, we showed that such clustering can be done in many MPI HPC applications. To do so, we analyzed the communication patterns we gathered from the execution of a representative set of HPC MPI applications. To find clusters, we proposed a bisection-based partitioning algorithm that makes use of a cost function evaluating the efficiency of a partial message logging protocol for a given clustering configuration. Contrary to existing graph partitioning tools, this algorithm does not require the number of clusters as input. We defined a cost function for two *state-of-the-art* partial message logging protocols and ran tests on our set of execution data. With both protocols, results show that we were able to get a good trade-off between the size of the clusters and the amount of logged messages. Furthermore, with Meneses et al. protocol, percentage of processes to rollback and of message to log is in many cases around 10%, which is an order of magnitude improvement compared to flat rollback-recovery protocols. This result encourages us to continue our work on partial message logging protocols.

Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr). This work was supported by INRIA-Illinois Joint Laboratory for Petascale Computing and the ANR RESCUE project.

References

[1] K. Antypas, J. Shalf, and H. Wasserman. NERSC-6 Workload Analysis and Benchmark Selection Process. Technical Report LBNL-1014E, Lawrence Berkeley National Laboratory, Berkeley, 2008.

[2] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, University of California, Berkeley, 2006.

[3] D. Bailey, T. Harris, W. Saphir, R. van der Wilngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. Technical Report NAS-95-020, NASA Ames Research Center, 1995.

[4] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42:153–159, 1992.

[5] F. Cappello. Fault tolerance in petascale/ exascale systems: Current knowledge, challenges and research opportunities. *International Journal of High Performance Computing Applications*, 23:212–226, August 2009.

[6] Ü. V. Çatalyürek and C. Aykanat. PaToH: A multilevel hypergraph partitioning tool, version 3.0. Technical Report BU-CE-9915, Bilkent Univ., 1999.

[7] W. H. Cunningham. Optimal attack and reinforcement of a network. *J. ACM*, 32:549–561, July 1985.

[8] J. Daly. A model for predicting the optimum checkpoint interval for restart dumps. In *Proceedings of the 2003 international conference on Computational science*, ICCS'03, pages 3–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[9] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. *ACM Computing Surveys*, 34(3):375–408, 2002.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[11] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello. Uncoordinated Checkpointing Without Domino Effect for Send-Deterministic Message Passing Applications. In *25th IEEE International Parallel & Distributed Processing Symposium (IPDPS2011)*, Anchorage, USA, 2011.

[12] J. C. Y. Ho, C.-L. Wang, and F. C. M. Lau. Scalable Group-Based Checkpoint/Restart for Large-Scale Message-Passing Systems. In *22nd IEEE International Parallel and Distributed Processing Symposium*, Miami, USA, 2008.

[13] S. Kamil, J. Shalf, L. Oliker, and D. Skinner. Understanding ultra-scale application communication requirements. In *Proceedings of the 2005 IEEE International Symposium on Workload Characterization*, pages 178–187, 2005.

[14] G. Karypis and V. Kumar. *MeTiS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0.* Univ. Minnesota, Minneapolis, 1998.

[15] E. Meneses, C. L. Mendes, and L. V. Kale. Team-based Message Logging: Preliminary Results. In *3rd Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids (CCGRID 2010).*, May 2010.

[16] S. Monnet, C. Morin, and R. Badrinath. Hybrid Checkpointing for Parallel Applications in Cluster Federations. In *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid (CCGRID'04)*, pages 773–782, Washington, DC, USA, 2004. IEEE Computer Society.

[17] F. Pellegrini. *SCOTCH 5.1 User's Guide.* LaBRI, 2008.

[18] R. Riesen. Communication Patterns. In *Workshop on Communication Architecture for Clusters CAC'06*, Rhodes Island, Greece, Apr. 2006. IEEE.

[19] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of Collective Communication Operations in MPICH. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[20] J. S. Vetter and F. Mueller. Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures. *Journal of Parallel and Distributed Computing*, 63:853–865, September 2003.

[21] J.-M. Yang, K. F. Li, W.-W. Li, and D.-F. Zhang. Trading Off Logging Overhead and Coordinating Overhead to Achieve Efficient Rollback Recovery. *Concurrency and Computation : Practice and Experience*, 21:819–853, April 2009.