How to Mitigate Node Failures in Hybrid Parallel Applications

Maciej Szpindler(✉)

Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw, Warszawa, Poland
m.szpindler@icm.edu.pl

Abstract. This paper describes approach to distributed node failure detection and communicator recovery in MPI applications with dynamic resource allocation. Failure detection is based on a recent proposal for user-level mitigation. The aim of this paper is to identify distributed and scalable approach for node failures detection and mitigation. Failed MPI communication recovery is realized with experimental implementation for MPI level resource allocation. Re-allocation of resources is used to replace failed node and enable application continuation with a full performance. Experimental results and performance of proposed techniques are discussed for schematic application scenarios.

Keywords: Message passing · Fault tolerance · Resource allocation · Dynamic execution

1 Introduction

Recent advances in HPC systems design result in increase of node level parallelism. One can expect this trend will continue up to developing substantial multi-element processing units in a form of many-core hyper-threaded computing nodes with a dozens of cores. No matter which model of software parallelism is exploited, the case of fault tolerance is significant for applications reliability and handling of hardware failures.

The most popular model that ensures both high performance and scalability on systems composed of large shared memory nodes is the hybrid parallelism. Usually the latter term refers to at least two levels of different parallelisation techniques coupled together. On the top level, preferred technique is message passing and distributed memory model such as MPI. On the lower level, different shared memory models usually provide better scalability for a range of applications classes. Popular choice there are OpenMP or other threading models. Such a combination of inter- and intra-node computing techniques is referred to as hybrid parallelism.

In the case of the MPI as a choice for the highest level of parallelisation technique, fault tolerance is widely studied area still not yet standardized. A number of approaches have been explored in this connection. Both library specific implementation [6] and MPI functionality extension approaches [7] have been proposed until now without successful adoption in a form of standardized definition. A recent proposal of fault tolerance primitives called User Level Failure Mitigation (ULFM) [3] has attracted wide recognition.

Nested parallelism on the intra-node communication level is supported within MPI model. At least two choices are possible there: either multi-process approach provided by MPI-3 shared memory windows [8] or multi-threading implemented inside MPI processes with chosen threading library. For both of these choices it is usually practical to use dedicated MPI communicator that allows intra-node synchronization. There are also advanced developments on extending this idea to a dynamic endpoints communicators [5] that will be probably included into a future version of the MPI standard.

For all the realizations of intra-node parallelism any type of failure result in a damage of associated communicator. Moreover, usually any serious hardware failure is actually resulting in whole node failure and loss of communication, no matter what the scale of the system is. It is expected that for larger systems with massive inter-node parallelism hardware failures will occur more often comparing to application lifetime. For either single multi-threaded process or multi-process execution on the failed node, the associated intra-node communicator is doomed to failure.

This paper presents basic schemes for failed communicators recovery and reconstruction that enable hybrid parallel application to mitigate node failures. Section 2 gives summary on the distributed detection of intra-node communicator failures, Sect. 3 describe reconstruction approach with a use of dynamic resource allocation. Section 4 contains an analysis of the experiments on the proposed techniques for node failure mitigation and a discussion of the performance for the proposed approach. The key contributions of the described work are the following:

- study on the distributed node failure detection using currently available implementations of the MPI user level failure mitigation approach,
- application of the dynamic resource allocation from the MPI level for failed node reconstruction,
- experiments on the performance and scalability of the proposed techniques.

2 Detecting Node Failures

2.1 User-Level Failure Mitigation Model

The MPI standard [9] defines basic abstraction for handling failures. The default approach is to use MPI_ERRORS_ARE_FATAL error handler. In this case all application processes are immediately terminated if any type of failure occurs. This is also a common choice for most of the legacy MPI codes that are in fact no fault-tolerant. Another approach, supported by MPI, is to use MPI_ERRORS_RETURN handler which gives possibility to post some process local operation before application is terminated. ULMF model is extending the latter approach, enabling application to continue its execution after the failure.

ULFM is a set of functions extending MPI API functionality with primitives for handling process failures explicitly in the application code. It is designed to provide mechanisms for failure detection, notification, propagation and communication recovery. Details of this MPI extension are described in [3]. While this enables MPI application to detect process failures and mitigate them, reconstruction and recovering application consistency are not a part of the extension and remain user responsibility.

Process failures are indicated with specific return codes of MPI communication routines. Either MPI_SUCCESS or MPI_ERR_PROC_FAILED error codes are returned for completion or failure respectively. If global knowledge of failure is required, already started communication can be revoked to assert consistency, raising MPI_COMM_REVOKED error code on all active communication. Functions for local failure acknowledgement and collective agreement on the group of survived processes are provided with the ULMF model.

ULFM extensions are partly implemented in the MPICH project (as for beginning of 2015, version 3.2 pre-release) and in a specific OpenMPI branch (version 1.7ft) dedicated for fault-tolerance studies. First analyses of the ULFM model performance and limitations have already appeared in [4] and real MPI applications with fault tolerance implemented with ULFM have been studied in [1].

Since ULFM model seems likely to be adopted, it is worth to target hybrid parallel applications using this approach. Node failure mitigation is addressed in this paper.

2.2 Intra-node Communicators

MPI model uses abstract communicator construct to represent a group of processes and their interactions. It provides elegant way of separating different communication scopes for collective communication. Also it is an abstraction that allows to express different communication schemes with groups and virtual topologies. More complicated communication designs can be described with either intra-communicator for a single group of processes or inter-communicator for separating two distinct groups of processes participating in the communication (referred to as local group and remote group).

It is practical to express nested parallelism in the hybrid MPI applications with dedicated communicators. This encapsulates intra-node communication and synchronization. It allows separation between intra- and inter-node communication which may overlap. Also it enables fine-grain synchronization depending on the application design. As a result communication costs may be reduced and eventually message exchange optimized on the MPI internal level. MPI provides convenient functionality with MPI_COMM_SPIT_TYPE which partitions global communicator into a disjoint subgroups of given type. The only standardized type is MPI_COMM_TYPE_SHARED which returns groups associated with shared memory nodes. This exposes intra-node shared memory regions for local processes.

This approach is natural for two level parallelism with MPI+MPI model that is composed of MPI communication across the nodes and MPI shared memory windows within the node. It was showed that such model of hybrid parallelism is beneficial for some application classes [8].

Other hybrid parallel applications based on MPI+X approach (where X denotes some other parallel programming model, e.g. OpenMP) may also require logical separation of intra- and inter-node communication. This is quite common approach for reducing total number of MPI messages and its exchange rate.

2.3 Node Failure Detection

In this paper dedicated intra-node communicator is considered. While computing node fails, respective communicator disappears and fault-tolerant application needs to handle with corrupted communicator. With a choice of ULFM model for mitigating node failures, one must decide on detection technique. Two distinct approaches that aims distributed detection are described in this paper. Distributed method is defined as not involving all processes participating in the MPI_COMM_WORLD communicator (global communicator).

First approach relies on the MPI inter-communicators. In this case each of the intra-node communicators has its counterpart communicator acting as remote group of processes. This scheme is depicted on Fig. 1. Local group and its remote neighbour form an inter-communicator. This seemingly complicated construct allows detection of node failures locally. Broken node and associated processes group are identified with a use of ULMF detection function. Unfortunately, inter-communicators were not fully supported by ULFM implementations at the time of this research.

Latter approach does not involve inter-communicators. The most straightforward way of detecting failed processes is to test MPI_COMM_WORLD. This kind of process failures detection is not scalable while all processes are involved. More distributed attempt is proposed with a special communicators structure. Each inter-node shared memory communicator delegates one "leader" process. These processes participate in "leaders" communicator. This special communicator allows to connect processes between distinct nodes as shown on Fig. 2.

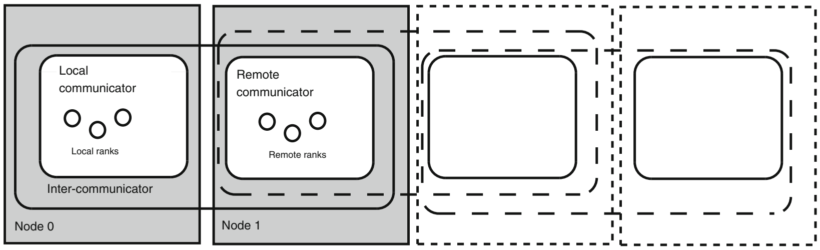


Fig. 1. Pairing local and remote node communicators in inter-communicator for local notification.

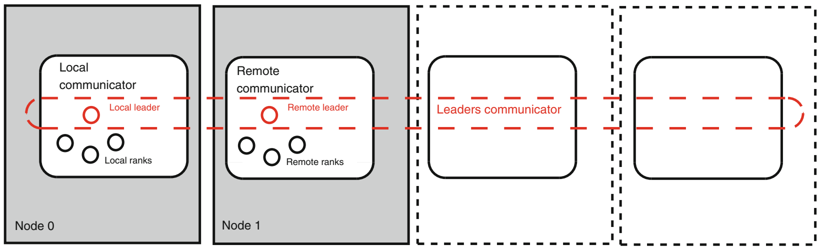


Fig. 2. Leaders communicator for inter-node communication and notification.

Members of the "leaders" communicator notify failures locally. Node failure is detected and group of failed processes is identified without involving global operations on the MPI_COMM_WORLD level.

3 Dynamic Reconstruction

3.1 Communicator Reconstruction

When the failed communicator is identified, reconstruction is possible. The choice of reconstruction approach depends on application type and user requirements. If communicator needs to be recreated to continue execution, then new processes are spawned. Spawning means dynamic creation of processes in the MPI terminology. Spawned processes eventually build new communicator to swap and restore failed one. Such approach introduces significant overheads due to process spawning as discussed in [3]. It is also required that application would support restore of lost data of the failed communicator member processes. At least two choices are considered in previous studies: either using checkpoints to dump application state in a selected points of execution or to replicate node private data on different remote node. Both choices require significant changes on the application level and these are discussed in [1].

3.2 Dynamic Resource Allocation

Another essential issue concerning node-communicator reconstruction is resource utilization. If performance degradation or increased node memory load are not acceptable, over-subscription of processes on the remaining set of nodes is a bad solution. Recreated processes need to be spawned on a new node. New resource need to be granted to application. This is usually not immediate nor possible with a general purpose HPC systems that execute many user jobs simultaneously.

One of the possible solution is to use dynamic resource allocation. It was showed that resizing node allocation is possible and basic implementation was presented for the hydra process manager of the MPICH library and Slurm resource management infrastructure. The details of this work are described in [10].

Proposed approach allows resizing Slurm allocation directly from the MPI spawn call. This is available from application code as depicted on diagram Fig. 3. It is implemented with hydra process manager (part of the MPICH library) and the Slurm allocation techniques using the Process Management Interface (PMI) API. PMI is the interim layer that provides MPI processes control [2]. In the case of modern implementations, MPI process spawning model is implemented with PMI infrastructure. For two common MPI implementations, MPICH is providing PMI layer implementation tightly integrated within its own process manager called *hydra* while OpenMPI has similar approach with closely related project called *PMIx*.

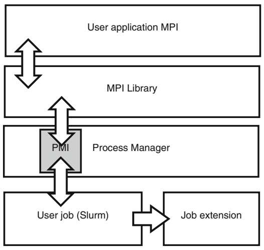


Fig. 3. Dynamic allocation scheme with process and resource managing layers.

Three modes of resource allocation were implemented and provided support for different applications requirements. Immediate allocation mode provides access to the resources only if currently available. It raises an error in the other case. Immediate mode was implemented using native Slurm request features. Non-blocking mode gives immediate return to execution after the allocation request. It was intended to use a helper thread to track allocation status. Blocking mode returns only if resources are successfully allocated. It is using Slurm blocking request. While blocking and non-blocking modes depend on external conditions and availability of the resources were not addressed in experiments discussed in the next section.

4 Experimental Results

In this section experimental results are described. Synthetic application was implemented to test the performance of proposed node failure detection and reconstruction approach. It focuses on node failures in case of hybrid parallelism. Application kernel is a two level reduction with a local operation over node's shared memory and a global MPI reduce operation across nodes. If global reduction raises fault error, failed node is detected and associated communicator is re-created. This schematic kernel aims to reproduce nested parallelism and it's typical communication pattern. Reconstruction of the failed communicator allocates new node dynamically with a use of described resource allocation technique. Experiments were performed using beta release if the MPI library which was the only choice available supporting ULFM extension. More complex communication schemes were considered to behave unstable and schematic application was selected as reliable test at this stage.

Two types of experiments was executed. One type addressed absolute performance of the proposed node failure detection and application reconstruction. Node failure detection overhead was analysed using high precision timer. Performance of the dynamic reconstruction of failed nodes was measured with a focus on dynamic allocation time and process spawning time. Other type of experiments tested relative cost of application reconstruction against the cost of application restart including the cost of resource allocation and application re-initialization.

4.1 Absolute Performance

Performance of the described node failure detection without usage of inter-communicators was addressed. Time overhead introduced by the proposed detection scheme was measured. Time cost versus a number of participating nodes was studied. Averaged results are shown on Fig. 4. Time measurements was based on a CPU cycles. The choice of the time measure was motivated by insufficient precision of the MPI_Wtime function.

Detection scheme was tested for up to 24 nodes running from 4 to 16 local processes using MPI_COMM_TYPE_SHARED sub-communicators. It was found that scalability is limited more by a number of processes per node that by the actual number of nodes. This exposes limitations of the remote process interactions used in a detection scheme.

Cost of the dynamic process allocation used in reconstruction is shown on Fig. 5. Time spend waiting for reallocation of failed nodes was compared to the process spawning cost. As expected spawning new processes were associated with overheads [3]. Dynamic allocation implementation was corrected and time was

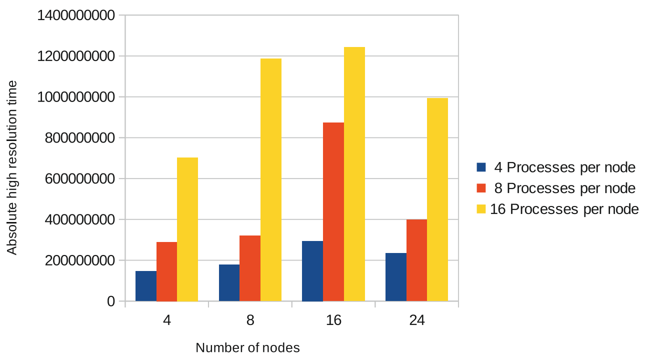
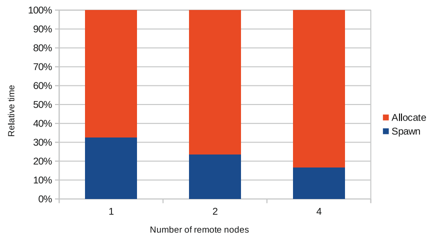


Fig. 4. Relative time spent in detection phase in the case of schematic hybrid parallel application.



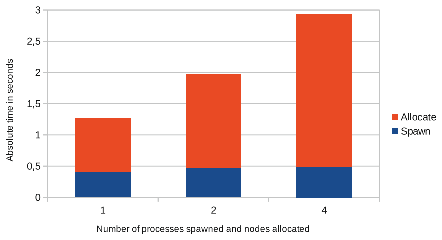


Fig. 5. Left: relative cost of the spawn and allocate operations for increasing number of nodes. Right: Time cost in seconds of spawning processes and allocating additional nodes.

greatly improved comparing to the previous results [10]. Nevertheless significant overheads were observed for dynamic allocation of nodes, due to user job resizing which involves many, possibly slow, system components. Experimental results were collected using "immediate" allocation mode. Presented time measurements are averaged over the series of experimental runs. Despite using a pool of reserved nodes for experiments, results still tend to be highly biased by the internal Slurm allocation procedures.

4.2 Relative Performance

To demonstrate practicality of the discussed approach, cost of the detection and dynamic reconstruction of failed nodes was compared to cost of re-scheduling and re-initialization of the schematic mini-application. Overall approach should also contain full application state recovery, including state of failed node's memory. It can use memory image cached on the remote node that is periodically updated which obviously introduces significant memory footprint and synchronization overhead. Other choices are possible but were not addressed in the described work. Instead of studying application specific state recovery that is discussed in [4], neglected costs of job re-scheduling and MPI related re-initialization were addressed.

Experiment tested average time needed to detect and dynamically re-allocate resources in case when half of nodes used failed. Collected results show that despite its obstacles, reconstruction with dynamic node allocation is practical approach. It still needs less time to recover than complete re-initialization of application including resource re-allocation. This test does not take into account time required to recover application to a state before the failure. Obviously re-scheduling of application also require new job creation, in case of scheduling system. Moreover additional waiting is required if nodes are no longer available to the user. Results of these relative performance comparison are summarized on Fig. 6.

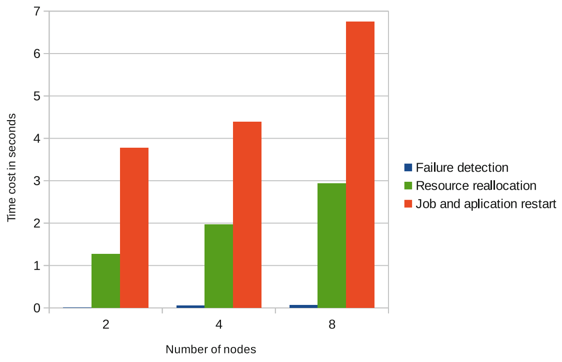


Fig. 6. Relative costs of detection and dynamic allocation versus cost of complete job re-initialization.

5 Summary and Future Work

Node failure detection and associated application reconstruction is important issue in case of hybrid parallelism. In this paper distributed approach for node failure detection is proposed and possible implementation choices with ULFM extension of the MPI standard discussed. This is the attempt to enable scalable and fault-tolerant applications with a hybrid parallelism. Performance of the proposed approach was invalided and experimental tests are discussed. Limitations and performance issues were identified. This work is related to unstable and experimental implementation of ULFM extension and other more scalable approaches are still available. Possible choices are described in this paper and are easy to apply with more refined and stable implementations.

Another contribution of this paper to the discussion on fault-tolerant MPI applications is proposal for communicator reconstruction involving dynamic resources allocation. It is demonstrated as practical alternative for application restart in case of node failures. Implementation of the proposed mechanism is described and experimental results included. Identified limitations are related to immediate allocation and need to be addressed with better Slurm integration. The case of non-blocking allocation requests and pending for resources still need to be refined to provide more capabilities and integrity.

References

1. Ali, M.M., Southern, J., Strazdins, P., Harding, B.: Application level fault recovery: using fault-tolerant open MPI in a PDE solver. In: 2014 IEEE International, Parallel & Distributed Processing Symposium Workshops (IPDPSW), pp. 1169–1178. IEEE (2014)
2. Balaji, P., Buntinas, D., Goodell, D., Gropp, W., Krishna, J., Lusk, E., Thakur, R.: PMI: a scalable parallel process-management interface for extreme-scale systems. In: Keller, R., Gabriel, E., Resch, M., Dongarra, J. (eds.) EuroMPI 2010. LNCS, vol. 6305, pp. 31–41. Springer, Heidelberg (2010)

3. Bland, W., Bouteiller, A., Herault, T., Hursey, J., Bosilca, G., Dongarra, J.J.: An evaluation of user-level failure mitigation support in MPI. Computing **95**(12), 1171–1184 (2013)
4. Bland, W., Raffenetti, K., Balaji, P.: Simplifying the recovery model of user-level failure mitigation. In: Proceedings of the 2014 Workshop on Exascale MPI, pp. 20–25. IEEE Press (2014)
5. Dinan, J., Balaji, P., Goodell, D., Miller, D., Snir, M., Thakur, R.: Enabling MPI interoperability through flexible communication endpoints. In: Proceedings of the 20th European MPI Users' Group Meeting, pp. 13–18. ACM (2013)
6. Fagg, G.E., Dongarra, J.: FT-MPI: fault tolerant MPI, supporting dynamic applications in a dynamic world. In: Dongarra, J., Kacsuk, P., Podhorszki, N. (eds.) PVM/MPI 2000. LNCS, vol. 1908, pp. 346–353. Springer, Heidelberg (2000)
7. Gropp, W., Lusk, E.: Fault tolerance in message passing interface programs. Int. J. High Perform. Comput. Appl. **18**(3), 363–372 (2004)
8. Hoefler, T., Dinan, J., Buntinas, D., Balaji, P., Barrett, B., Brightwell, R., Gropp, W., Kale, V., Thakur, R.: MPI+MPI: a new hybrid approach to parallel programming with MPI plus shared memory. Computing **95**(12), 1121–1136 (2013)
9. MPI Forum: MPI: A Message-Passing Interface Standard. Version 3.0, 21 September 2012. http://www.mpi-forum.org
10. Szpindler, M.: Enabling adaptive, fault-tolerant MPI applications with dynamic resource allocation. In: Proceedings of the 3rd International Conference on Exascale Applications and Software (2015)