Run-Through Stabilization: An MPI Proposal for Process Fault Tolerance[⋆]

Joshua Hursey[1], Richard L. Graham[1], Greg Bronevetsky[2], Darius Buntinas[3], Howard Pritchard[4], and David G. Solt[5]

[1] Oak Ridge National Laboratory {hurseyjj,rlgraham}@ornl.gov
[2] Lawrence Livermore National Laboratory greg@bronevetsky.com
[3] Argonne National Laboratory buntinas@mcs.anl.gov
[4] Cray, Inc. howardp@cray.com
[5] Hewlett-Packard david.solt@hp.com

Abstract. The MPI standard lacks semantics and interfaces for sustained application execution in the presence of process failures. Exascale HPC systems may require scalable, fault resilient MPI applications. The mission of the MPI Forum's Fault Tolerance Working Group is to enhance the standard to enable the development of scalable, fault tolerant HPC applications. This paper presents an overview of the Run-Through Stabilization proposal. This proposal allows an application to continue execution even if MPI processes fail during execution. The discussion introduces the implications on point-to-point and collective operations over communicators, though the full proposal addresses all aspects of the MPI standard.

Keywords: MPI, Fault Tolerance, Run-through Stabilization, Algorithm Based Fault Tolerance, Fail-Stop Process Failure

1 Introduction

High Performance Computing (HPC) applications, particularly those running in fault-prone environments, use fault tolerance techniques to ensure successful completion of their computational objectives. As HPC systems push toward exascale, projections indicate that these large-scale systems will become more fault-prone, posing a greater threat to the existing HPC applications [2]. In preparation for such fault-prone computing environments, applications are investigating Algorithm Based Fault Tolerance (ABFT) [5] techniques to improve

[⋆] Special thanks to the MPI Forum and Fault Tolerance Working Group members that contributed to the run-through stabilization proposal. Their comments and insights continue to help strengthen the developing proposals targeted for inclusion in the Message Passing Interface (MPI) standard. Research sponsored by the Office of Advanced Scientific Computing Research; Office of Science; Mathematical, Information, and Computational Sciences Division at Oak Ridge National Laboratory; U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC; U.S. Department of Energy, under Contract No. DE-AC02-06CH11357; U.S. Department of Energy, under Contract No. DE-AC52-07NA27344 by Lawrence Livermore National Laboratory; The ARRA / DoE - Early Career Research Program; and by award #CCF-0816909 from the National Science Foundation.

```
MPI_Comm_validate{_all}(MPI_Comm c, int *newfailures)
MPI_Comm_validate_get_num_state{_all}(MPI_Comm c, int type, int *count)
MPI_Comm_validate_get_state{_all}(MPI_Comm c, int type, int incount,
                                 int *outcount, MPI_Rank_info rank_infos[])
MPI_Comm_validate_get_state_rank{_all}(MPI_Comm c, int rank,
                                       MPI_Rank_info *rank_info)
MPI_Comm_validate_set_state_null(MPI_Comm c, int incount,
                                 MPI_Rank_info rank_infos[])
```

Fig. 1. Validation Interfaces for Communications (C interface shown)

the efficiency of application recovery after process failure beyond that which checkpoint/restart solutions alone can provide.

The lack of standardized fault tolerance semantics and interfaces prevents HPC applications from portably exploring ABFT techniques using the Message Passing Interface (MPI) standard. The MPI Forum created the Fault Tolerance Working Group in response to the growing need for portable, fault tolerant semantics and interfaces in the MPI standard to support application level fault tolerance development.

The Fault Tolerance Working Group (FTWG)'s run-through stabilization (RTS) proposal enables an MPI application to continue execution even if one or more MPI processes fail. The discussion focuses on the central themes of the proposal in the context of a communicator though all aspects of MPI are addressed in the proposal under consideration for the MPI-3.0 version of the MPI standard [4]. Various MPI implementations are currently exploring implementations of the RTS proposal. The complementary process recovery proposal is being actively developed by the FTWG.

2 Process Fault Tolerance Model

Under the RTS proposal, the primary role of the MPI implementation is to (i) inform the application of process failures, and (ii) allow the application to continue running and communicating with unaffected processes. The application is guaranteed to be eventually informed, via error handlers, of all process failures and that no process will be reported as failed before it actually fails. Therefore the MPI implementation must provide a *perfect* failure detector for *fail-stop* process failure (i.e., a process is permanently stopped, often due to a crash) [3].

From the perspective of one process, other processes can be in one of the following states (prefixed with MPI_RANK_STATE_): OK, FAILED or NULL. Processes with state OK are executing normally. Processes with state FAILED have been detected by MPI as failed-stop. Processes with state NULL are failed processes treated as if their ranks are MPI_PROC_NULL.

2.1 Validation of Process State

The RTS proposal focuses on high scalability by treating process failures differently from the perspective of point-to-point and collective communication. This is because point-to-point communication between a given pair of processes is rarely affected by the failure of another process, while collective communication implies dependance upon the participation of the entire group. As such, the proposal provides two scopes of application fault recognition: *local* and *global*.

A process uses the validation functions in Figure 1 to update, access, and modify the known state of a process in a communicator. Local recognition is implemented by the variants of the MPI_Comm_validate operation, and are designed to support point-to-point communication. Global recognition is implemented by variants of the MPI_Comm_validate_all operation, and are designed to support collective communication.

A fault tolerant agreement algorithm is provided by the MPI_Comm_validate_all collective operation [1]. This operation synchronizes the fault detectors, re-enables collective operations, globally recognizes known failed processes, and provides a uniform return value across the collective group.

The failure of a process must be recognized on each communicator of which it is a member. This allows libraries, that create their own communicators, to be able to receive notification of the failure even if another library or the main application has already recognized the failure on another communicator.

2.2 Semantic Modifications

Point-to-Point Communication between two active processes is unaffected by the failure of other non-participating processes. For example, if process A fails, process B can still send messages to process C, and vice versa. Communication with process A returns an error (MPI_ERR_RANK_FAIL_STOP) until process B *recognizes* the failed process.

Collectives Collective operations must be *fault-aware*, meaning that they will not hang in the presence of failures. To preserve failure-free performance, collective operations are not required to provide uniform return codes. For example, using MPI_Bcast it is possible for a process to fail inside the collective such that those processes that left early returned success while the remainder will return an error. When a process fails, all collective operations are disabled in communicators that contain that process. Collective communication can be re-enabled by calling MPI_Comm_validate_all.

Communicator Management All failed processes must be globally recognized in the participating communicator(s) before calling any communicator construction operation. If a globally recognized failed process is represented in a communicator passed to a communicator construction operation other than MPI_COMM_SPLIT, then it is represented in the new communicator. In the presence of failures, the communicator construction operations ensure uniformly consistent creation of the communicator handle and return codes.

References

1. Barborak, M., Dahbura, A., Malek, M.: The consensus problem in fault-tolerant computing. ACM Computing Surveys 25, 171–220 (June 1993)
2. Cappello, F., Geist, A., Gropp, B., Kale, L., Kramer, B., Snir, M.: Toward exascale resilience. International Journal of High Performance Computing Applications 23(4), 374–388 (2009)
3. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM 43, 225–267 (March 1996)
4. Fault Tolerance Working Group: Run-though stabilization proposal, `svn.mpi-forum.org/trac/mpi-forum-web/wiki/ft/run_through_stabilization`
5. Huang, K.H., Abraham, J.A.: Algorithm-based fault tolerance for matrix operations. IEEE Transactions on Computers 33(6), 518–528 (1984)