A Proposal for Process Fault Tolerance in the MPI-3 Standard

Wesley Bland George Bosilca Aurelien Bouteiller
Thomas Herault
Jack Dongarra
{bland, bosilca, bouteill, herault, dongarra } @ eecs.utk.edu
Innovative Computing Laboratory,
University of Tennessee, Knoxville

February 6, 2012

Abstract

In this document we propose a flexible approach providing fail-stop process fault tolerance by allowing the application to react to failures while maintaining a minimal execution path in failure-free executions. Our proposal focuses on returning control to the application by avoiding deadlocks due to failures within the MPI library. No implicit, asynchronous error notification is required. Instead, functions are provided to allow processes to invalidate any communication object, thus preventing any process from waiting indefinitely on calls involving the invalidated objects. We consider the proposed set of functions to constitute a minimal basis which allows libraries and applications to increase the fault tolerance capabilities by supporting additional types of failures, and to build other desired strategies and consistency models to tolerate faults.

Chapter 17

Process Fault Tolerance

17.1 Introduction

MPI processes may fail at any time during execution. Long running and large scale applications are at increased risk of encountering process failures during normal execution. This chapter introduces the MPI features that support the development of applications and libraries that can tolerate process failures. The approach described in this chapter is intended to prevent the deadlock of processes while avoiding any impact on the failure-free execution of an application.

The expected behavior of MPI in case of a process failure is defined by the following statements: any MPI call that involves a failed process must not block indefinitely, but either succeed or return an MPI error (see Section 17.2); asynchronous failure propagation is not required by the MPI standard, an MPI call that does not involve the failed process must not return an error. If an application needs global knowledge of failures, it can use the interfaces defined in Section 17.3 to explicitly propagate locally detected failures.

Advice to users. Many of the operations and semantics described in this chapter are only applicable when the MPI application has replaced the default error handler MPI_ERRORS_ARE_FATAL on, at least, MPI_COMM_WORLD. (End of advice to users.)

17.2 Failure Notification

In this section, we specify the behavior of an MPI communication call when failures happened on processes involved in the communication. A process is considered as involved in a communication if either:

1. the operation is a collective call and the process appears in one of the groups on which the operation is applied;
2. the process is a named or matched destination or source in a point-to-point communication;

3. the operation is an MPI_ANY_SOURCE reception and the process belongs to the source group.

Therefore, if an operation does not involve a failed process (such as a point to point message between two non-failed processes), it must not return an error.

Advice to implementers. It is a legitimate implementation to provide failure detection only for processes involved in an ongoing operation and postpone detection of other failures until necessary. Moreover, as long as an implementation can complete operations, it may choose to delay returning an error. Another valid implementation might choose to return an error to the user as quickly as possible. (End of advice to implementers.)

When a failure prevents the MPI implementation from completing a point-to-point communication, the communication is marked as completed with an error of class MPI_ERR_FAILED. Further point-to-point communication with the same process on this communicator must also return MPI_ERR_FAILED.

In case of a failure of involved processes, the completion of an unmatched MPI reception from MPI_ANY_SOURCE is undecidable. Such communication is marked with an error of class MPI_ERR_PENDING and the completion operation returns. If the operation worked on a request, and the request was allocated by a nonblocking communication call, then the request is still valid and pending. To acknowledge this failure and discover which processes failed, the user should call MPI_COMM_FAILURE_ACK.

Non-blocking operations must not return an error about failures during initialization. All failure errors are postponed until the corresponding completion function is called.

When a collective operation cannot be completed because of the failure of an involved process, the collective operation eventually returns an error of class MPI_ERR_FAILED. The content of the output buffers is undefined.

Advice to users. Collective operations involve all processes in the communicator. However, some processes may succesfully complete the collective despite failures. For example, in MPI_Bcast, the root process is likely to succeed before a failed process disrupts the topology, resulting in some other processes returning an error. However, it is noteworthy that non-rooted collectives always return MPI_ERR_FAILED on all ranks, if the failure occurred before the failed process entered the collective operation.(End of advice to users.)

Advice to users. Note that communicator creation functions (like MPI_COMM_DUP or MPI_COMM_SPLIT) are collective operations. As such, if a failure happened during the call, an error might be returned to some processes while others succeed and obtain a new communicator. It is the responsibility of the user to ensure that all involved processes have a consistent view of the newly created communicator. A successfull barrier following the communicator creation function will ensure that all processes created

the new communicator and it can be used normally. If the barrier fails, the conservative approach is to invalidate the created communicator. (End of advice to users.)

17.3 Failure Handling Functions

MPI provides no guarantee of global knowledge of a process failure. Only processes involved in a communication with the failed process are guaranteed to eventually detect its failure. If global knowledge is required, MPI provides a function to globally invalidate a communicator.

MPI_COMM_INVALIDATE(comm)

IN	comm	communicator (handle)

This function eventually notifies all ranks within the communicator *comm* that this communicator is now considered invalid. An invalid communicator preempts any non-local MPI calls on *comm*, with the exception of MPI_COMM_SHRINK. Once a communicator has been invalidated, all subsequent non-local calls on that communicator, with the exception of MPI_COMM_SHRINK, must fail with an error of class MPI_ERR_INVALIDATED.

MPI_COMM_SHRINK(comm, newcomm)

IN	comm	communicator (handle)
OUT	newcomm	communicator (handle)

This function partitions the group associated with *comm* into two disjoint subgroups: the group of failed processes and the group of alive processes. A new communicator is created for the group of alive processes and returned as *newcomm*. This function is illegal on a communicator which has not been invalidated and will return an error of class MPI_ERR_ARG. This call returns the same value on all ranks, even if failures happen during the call. If the return is MPI_SUCCESS, the call is semantically equivalent to MPI_COMM_SPLIT where living processes participate with the same color and a key equal to their rank in *comm*, and an agreement is made among living processes to determine the group of failed processes whose implicit contribution is MPI_UNDEFINED.

Advice to users. This call does not guarantee that all processes in *newcomm* are alive, but that all processes in *newcomm* agreed on a consistent set that includes at least the union of processes locally known to have failed before the call. Any new failure will be detected in subsequent MPI calls. (End of advice to users.)

MPI_COMM_FAILURE_ACK(comm)

IN	comm	communicator (handle)

This local function gives the users a way to acknowledge all locally notified failures on *comm*. After the call, operations that would have returned MPI_ERR_PENDING proceed without further reporting acknowledged failures.

Advice to users. It is an incorrect MPI code to call a collective communication on a communicator with acknowledged failures. To reliably use collective operations on a communicator with failed processes, the communicator should first be invalidated using MPI_COMM_INVALIDATE and then a new communicator should be created using MPI_COMM_SHRINK. (End advice to users.)

MPI_COMM_FAILURE_GET_ACKED(comm, failedgroup)

IN	comm	communicator (handle)
OUT	failedgroup	group (handle)

This local function returns the group *failedgroup* of processes from the communicator *comm* which were locally acknowledged as failed by preceding calls to MPI_COMM_FAILURE_ACK.

17.4 Error Codes and Classes

MPI_ERR_FAILED	A process in the operation has failed (a fail-stop failure).
MPI_ERR_INVALIDATED	The communicator used in the operation was invalidated.