A Transactional Model for Fault-Tolerant MPI for Petascale and Exascale systems

[Extended Abstract]

Amin Hassani, Anthony Skjellum
University of Alabama at Birmingham
CH 115, 1300 University Blvd
Birmingham, AL, 35205
{ahassani,tony}@cis.uab.edu

Ron Brightwell
Sandia National Laboratories
PO Box 5800
Albuquerque, NM 87185-1319
rbbrigh@sandia.gov

Abstract

Fault-Aware MPI (FA-MPI) is a novel approach to provide fault-tolerance through a set of extensions to the MPI Standard. It employs a transactional model to address failure detection, isolation, mitigation, and recovery via application-driven policies. This approach allows applications to employ different fault-tolerance techniques, such as algorithm-based fault tolerance (ABFT) and multi-level checkpoint/restart methods. The goal of FA-MPI is to support fault-awareness in MPI objects and enable applications to run to completion with higher probability than running on a non-fault-aware MPI. FA-MPI leverages non-blocking communication operations combined with a set of TryBlock API extensions that can be nested to support multi-level failure detection and recovery. Managing fault-free overhead is a key concern as well.

1. INTRODUCTION

Adding fault-tolerance support to MPI standard [4] has been an important topic in the past few years. MPI Forum's Fault-Tolerant Working Group (FTWG) has been developing and considering proposals [1, 2] for a fault-tolerant MPI. The most recent proposal under consideration is User-Level Failure Mitigation [1]. One of the main differences between FA-MPI and the ULFM is that FA-MPI does not restrict itself to only "process failures." Instead, any types of failure that can be detected and disseminated is supported in FA-MPI. The goal of FA-MPI is not to replace any other approach for fault-tolerant MPI, but rather to extend MPI minimally to support a transactional model for fault awareness. Also FA-MPI, is restricted to using only non-blocking operations, so it is not intended to support all legacy applications. We expect that tools can be developed to enhance applications that use blocking operations with FA-MPI's transactional fault tolerance capabilities.

The path toward exascale and the need for scalable and dependable applications and libraries motivates the use of non-blocking communication calls in message passing systems to achieve higher performance through overlapping computation, communication, and I/O. Non-blocking semantics can be used to achieve fault-tolerance in MPI applications. As an example, a communication operation should not hang because of a failure in a remote rank in a blocking operation. MPI can use the request handle from a non-blocking operation to uncover errors and propagate them to other ranks. This motivates FA-MPI [5] to support only non-blocking communication calls.

FA-MPI detects failures, broadcast them to other ranks through fault-tolerant collective calls, and notifies the application of all failures. Application can choose to isolate and mitigate the failures with FA-MPI's help by creating smaller communicators, replacing broken communicators with new ones, and then trying to recover from the failed state. The transactional model allows applications to do a soft retry, rollback, roll-forward, or perform a restart of application from a checkpoint if continuing execution is not possible.

2. FAILURE DETECTION

Unlike the ULFM's proposal, FA-MPI does not check for error state after each operation. The goal of FA-MPI is not to ensure the healthy state of MPI in order to provide continuous operation. The goal is to provide the ability to execute a series of operations, to wait for them to complete, and to detect and broadcast any failures in operations involved at a variable granularity. This approach is similar to BSP [6] where asynchrony is reached in "epochs" and blocking barrier fence completes all outstanding operations.

TryBlocks. TryBlock operations are the fundamental building blocks upon which the FA-MPI model is based. TryBlock operations model a transaction block inside which several non-blocking communication, computation and I/O operations are executed. Each TryBlock starts with an `MPI_TryBlock_start` function, which binds a communicator to its request handle. Any communicators (including the communicators associated window and file objects) used inside a TryBlock should be a proper subset of TryBlock's communicator's group. TryBlocks are completed by a call to `MPI_TryBlock_finish`. This function is a synchronizing collective operation that broadcast failures in a fault-tolerant allreduce/allgather over all ranks in the TryBlock's communicator's group. At the end of transaction, ranks decide consistently to accept or reject the transaction by checking returned failures. TryBlock completion allows determination of faulty or failed objects, requests, ranks, and failures associated with each rank. This global knowledge lets the application define policies to achieve resiliency with the help of FA-MPI.

TryBlocks can be nested to support multi-level failure detection and recovery. The nested property of TryBlocks is required to achieve high scalability through application of data and task parallelism for smaller communicator groups and multiple user threads. An outer TryBlock can provides global application progress while several nested TryBlocks inside can be run in parallel (in different user threads) or serial. The outer TryBlock can sift out success and failed

TryBlocks and progress forward or backward based on the failure types. A basic pseudo code for an application using FA-MPI is shown in Algorithm 1.

```
communication initialization;
if restarted then
  | load data from last checkpoint (optional);
end
repeat
  | while more_work_to_do do
  |   | MPI_TryBlock_start();
  |   | computation, communication and/or I/O;
  |   | wait for operations to finish;
  |   | inject local errors;
  |   | MPI_TryBlock_finish();
  |   | if failure happened then
  |   |   | isolate and mitigate the failure;
  |   |   | if recovery_needed then break;
  |   | end
  |   | periodically checkpoint;
  | end
  | if recovery_needed then
  |   | do recovery procedure;
  | end
until more_work_to_do or restart_needed;
```

Algorithm 1: A basic application using FA-MPI

FA-MPI doesn't restrict mechanisms used to implement the semantics of TryBlocks. Any implementation may use a consensus algorithm through piggybacking, gossiping, collective, a hybrid algorithm, and/or other methods for broadcasting failure information to alive ranks. Some recent publications [3] on implementation of consensus problem can be used to synchronize failures in TryBlock completion.

Failure Injection. FA-MPI proposes a fault-injection mechanism to allow both applications and the MPI implementation to collaborate consistently to detect and notify failures and resolve them with each other's help. The coordination can be done by allowing both the application and MPI library to detect and "inject" errors on requests, objects, and state of the MPI inside a TryBlock and retrieve the errors at the completion of the TryBlock. For example, This approach allows the application to use an ABFT approach, such as a checksum calculation on the result of a computation or communication and simply notify the other ranks of the failure in TryBlock completion and make a decision from there.

Local Completion. TryBlock completion calls need communication operation request handles to perform error detection and notification, but local completion functions like `MPI_Wait` destroys request handles on successful return. This behavior is insufficient if the application needs completion of a communication request before TryBlock completion call, or if it needs to check the request's failure state. To be able to take advantage of error notification to MPI implementation, request handles should not be freed until the TryBlock completion call. To solve this problem we introduced a few local completion functions that do not destroy requests after completion.

Timeout. A timeout is an effective mechanism to handle exceptional behaviors, such as delay in response or remote failure. FA-MPI uses timeout semantic to allow applications variable granularity for trying (and failing) a transaction.

3. ISOLATION, MITIGATION, RECOVERY

Sometimes continuing work with a sick communicator is impossible. FA-MPI provides API calls to shrink a faulty communicator (and continue work with the new smaller communicator)[1] and possibly regrow it later by spawning new processes and merge all ranks into a new communicator. FA-MPI maintains single-assignment properties of MPI objects (communicators, windows, and files) and repairing or modifying any of these objects is not implied.

Recovery comprises another block of computation and communication and should be handled in a TryBlock even in the presence of faults. Any failure during recovery can result in retry or rollback to the last checkpoint. All of these potential scenarios can be policies decided by an application using FA-MPI.

4. FAULT-FREE OVERHEAD

We expect that applications using FA-MPI will be able to run longer on larger machines in compare to a non-fault-tolerant version of the application. In order to achieve resiliency, sacrifice in performance cannot be avoided. We allow applications to run slightly slower but with enough forward progress to reach the completion of execution. FA-MPI allows the application to control the fault-free overhead by setting the granularity of synchronization.

5. CONCLUSION AND FUTURE WORK

FA-MPI is a set of extension APIs for MPI standard to allow fault-awareness using a transactional model. FA-MPI detects and propagates failures in non-blocking communication calls, and notifies application of the failures. We expect applications using FA-MPI run to completion with higher probability than the non-fault-aware versions. We are currently developing the proposed API and we will publish further results in near future publications.F

Acknowledgments

This work was supported in part by the United States Department of Energy under contract DE-AC04-94AL85000, the National Science Foundation under grant CCF-1239962 and Sandia National Laboratories under grant PO-1330625.

References

[1] Wesley Bland. User level failure mitigation in MPI. In *Euro-Par 2012: Parallel Processing Workshops*, pages 499–504, 2013.

[2] Joshua Hursey, Richard L. Graham, Greg Bronevetsky, Darius Buntinas, Howard Pritchard, and David G. Solt. Run-through stabilization: An MPI proposal for process fault tolerance. In *Recent Advances in the Message Passing Interface*, pages 329–332. Springer, 2011.

[3] Joshua Hursey, Thomas Naughton, Geoffroy Vallee, and Richard L. Graham. A log-scaling fault tolerant agreement algorithm for a fault tolerant MPI. In *EuroMPI*, pages 255–263, 2011.

[4] Message Passing Interface Forum. MPI: a message passing interface standard version 3.0. Technical report, September 2012.

[5] Anthony Skjellum and Purushotham V. Bangalore. FA-MPI: fault-aware MPI specification and concept of operations. Technical Report UABCIS-TR-2012-011912, May 2012.

[6] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[1] This is a similar approach to the ULFM proposal.