Fault Tolerance Assistant (FTA): An Exception Handling Approach for MPI Programs

[Extended Abstract]

Aiman Fang
Department of Computer Science
University of Chicago
aimanf@cs.uchicago.edu

Ignacio Laguna, Kento Sato,
Tanzima Islam, Kathryn Mohror
Lawrence Livermore National Laboratory
{ilaguna, kento, tanzima, kathryn}@llnl.gov

ABSTRACT

We propose FTA, a programming model that provides failure localization and transparent recovery of process failures in MPI applications.

1. INTRODUCTION

With the expected increase of mean time between failure (MTBF) in exascale systems, along with more complex software and hardware stacks and runtimes, scientists require programming models that allow them to cope with failures efficiently and to increase productivity while developing large HPC applications. Although the MPI+X is expected to be widely used in exascale systems—where X refers to node thread parallelism models, such as OpenMP—MPI does not provide mechanisms for fault tolerance: the standard specifies that if a failure occurs, the state of MPI is undefined, thus applications can do little more than abort. One of the grand challenges for exascale computing is therefore to provide a practical fault-tolerance programming model for MPI applications.

Previous work has proposed fault-tolerant MPI libraries and interfaces [2, 6, 9]. For instance, the ULFM interface is a proposal under consideration to incorporate fault tolerance in the MPI Standard and provides functionality to deal with process failures, to repair communicators, and to propagate failures. In our attempt to make use of the programming interface that these approaches propose in large HPC applications, we have found the following limitations [7]:

- A substantial amount of code changes are required to use these interfaces. Code changes involve failure detection and fixing broken communicators; most applications do not have a central place where all communicators are created or repaired. Therefore programmers need to repeat many steps to enable resilience.
- Failure localization lacks sufficient support. Most models require checking returned error codes of MPI operations which can be cumbersome, or provide error handlers. In the latter case, when a failure is detected (which can occur in an arbitrary MPI function call) an error handler is called. However, programmers do not know where in the code the failure originated or was detected from, hence limits flexible recovery.

To address these limitations, FTA provides a try/catch model (see Figure 1), which allows transparent recovery of MPI communicators while providing failure localization guarantees—failures are detected and fixed within a user-defined code block. With FTA, programmers declare a conversation [3] (i.e., a set of MPI ranks that participate in executing a set of MPI calls) in a try code block. At the end of the conversation, all participating processes agree or disagree on a failure. If a failure is detected, FTA automatically executes recovery code, which involves repairing application-level state (e.g., by reading a checkpoint) and MPI-level state (e.g., repairing communicators).

Hassani et al. proposed FA-MPI [5], a transactional resilience scheme for MPI. This work shares some of the ideas of FTA, for example, it allows transactions, which are similar in nature to FTA conversations; however, it works only for non-blocking MPI operations. In addition, FA-MPI requires users to write code to recover MPI state, such as communicators. The goal of FTA is to perform this transparently.

Two failure recovery models can be used in MPI: shrinking recovery, in which the number of resources (i.e., MPI processes and nodes) are reduced after a failure, or nonshrinking recovery, in which failed processes and nodes are replaced so that applications can continue with the original number of resources [7]. FTA supports both shrinking and nonshrinking recovery models.

We have implemented a prototype of FTA in Open MPI and have tested it in a synthetic and in an mini HPC application, CoMD [1]. In the rest of the abstract we describe the design challenges and implementation details of FTA.

2. FTA DESIGN

The objective of FTA is to isolate the scope of failures and enable flexible shrinking/nonshrinking recovery with minor changes to applications. We deploy the try/catch mechanism to locate where failures occur. In addition, we design communicator management, which intelligently repairs all broken communicators after failures occur.

2.1 Exception Handling (Try/Catch) Model

Exception handling is a classical mechanism of program-
A challenge of exception handling in distributed systems is the complexity of asynchronous interacting activities. The idea of Coordinated Atomic Actions (CA actions) or conversations [3] has been proposed to control such complexity. In FTA, a conversation starts with a TRY statement, which sets a setjmp point for retry. The conversation encloses the interactions of a group of processes, that is, activities within one communicator and its subsets. Therefore, the granularity of failure detection is a conversation. Process failures are recognized as exceptions and raised to all members of the associated communicator.

To detect process failures within a conversation, FTA sets the error handler of designated communicator for the conversation. When MPI operations fail due to process failures, a PROC_FAIL_EXCEPTION is raised by the error handler. Processes that detect the failure will skip the rest of work in the conversation and jump to CATCH block, while other processes continue execution. At the end of a conversation, all processes within the conversation reach consensus on whether or not a failure occurred. Therefore, failures are guaranteed to be acknowledged by all the non-failed processes of the communicator at the end of a conversation. In our prototype implementation, we use the MPI_Comm_agree function provided by ULFM to perform the consensus. We expect that if the FTA programming model is incorporated in the MPI standard, other consensus protocol implementations can be provided by ULFM to perform the consensus. We expect that if the FTA programming model is incorporated in the MPI standard, other consensus protocol implementations can be used in the MPI library.

The recovery procedure is located in a CATCH block which usually includes code for cleaning up states, repairing communicators (transparently facilitated by FTA), and data redistribution (if needed). A RETRY statement—essentially a longjmp function—can be used to restart the TRY block. Finally, the TRY/CATCH pair is completed with the statement ENDTRY. Our initial prototype uses macros to implement language statements, such as TRY and CATCH.

### 2.2 Communicator Management

A challenge in FTA is repairing communicators. Applications may have a set of communicators with various relationships, such as parent/child, overlapping (i.e., they have mutual MPI ranks), and isolated, depending on how communicators are created. In large and complex MPI applications, identifying and repairing broken communicators is a demanding task since applications do not usually have a central place where all communicators are created. Moreover, using the repaired communicator (which usually comes with a different handler) in the right code location is difficult. For instance, a shrinking operation on the broken communicator COMM_1 will return COMM_2, which should be semantically used as COMM_1 by applications.

FTA provides the ability to automatically repair all communicators through communicator management, requiring no effort from users. Programmers only need to specify the recovery mode of communicators—either shrinking or non-shrinking. In shrinking mode, all communicators exclude the failed processes, while in non-shrinking mode, pre-allocated spare processes substitute failed ones.

The design of FTA communicator management and automatic communicator repair is illustrated in Figure 2. FTA first creates the FTA_COMM_WORLD, which excludes the pre-allocated spare processes from MPI_COMM_WORLD. Without spare processes, FTA_COMM_WORLD is essentially a duplicate. Applications are required to use FTA_COMM_WORLD instead of MPI_COMM_WORLD. FTA then tracks the creation of any communicator by trapping all communicator creation operations, and recording the corresponding information, which includes a pointer to the communicator handler and associated ranks. This information incurs negligible memory overhead. During recovery, FTA first recovers FTA_COMM_WORLD through the interface FTA_Comm_repair that is provided to users and it takes recovery mode as the parameter. Given the stored communicator and failed rank information, FTA finds all broken communicators and revokes them. Second, FTA constructs new communicators with the same set of process ranks. By assigning the pointer of communicator handler to the new communicator, FTA allows applications to use the original communicator handler in the application.

### 2.3 Recovery

In distributed parallel programs, processes assigned in different communicators perform work concurrently. Our model guarantees that processes in the same conversation will be notified of failures synchronously at the end of the TRY block. However, failure notification may occur asynchronously in different communicators, and processes in the communicator that encounter the failure first need to wait for other processes to perform global repair before continuing.
Applications needs to recover. In the for checkpoint at the beginning, which has all the state the application might use our model. The application writes a checkpoint.

3. EXAMPLES

Figure 4 shows how a bulk synchronous parallel (BSP) program might use our model. The application writes a checkpoint at the beginning, which has all the state the application might use our model. The application writes a checkpoint.

4. SUMMARY AND FUTURE WORK

In this work, we describe our initial design and prototype of FTA, a try/catch model for failure localization and automatic failure recovery in MPI applications. FTA is work-in-progress—the following are some of the challenges that we plan to address as future work.

Efficient global failure detection. The FTA model assumes that at the end of a conversation, all the participating processes must agree on whether or not a failure occurred. This type of agreement can be implemented using distributed consensus protocols at the cost of communication overhead. Our FTA prototype uses the ULFM agreement function calls for failure detection but it can use other methods. In the future we plan to investigate alternative failure detection and agreement protocols to reduce the overhead that is involved in each try/catch block.

Arbitrary non-shrinking recovery. Currently, FTA allows non-shrinking recovery only when a try/catch block is placed in the main function. This is because when a failure occurs at an arbitrary location in the call stack, there is no system-independent way to replace the failed process with a spare one. Most solutions to create an identical process (with arbitrary call-path depth) require system-level checkpointing. We will investigate how to provide a system-independent solution for this problem.

Nested error handling. A try/catch model should support nested failure handling, just as traditional exception handling models do in object-oriented programming. We have not explored fully the semantics and applications of nested failure handling in FTA; however, when provided, it would allow users to handle a variety of failures, including silent data corruption detected in the application inside a code block. This would allow to apply concepts, such as containment domains [4], in MPI applications.

5. REFERENCES