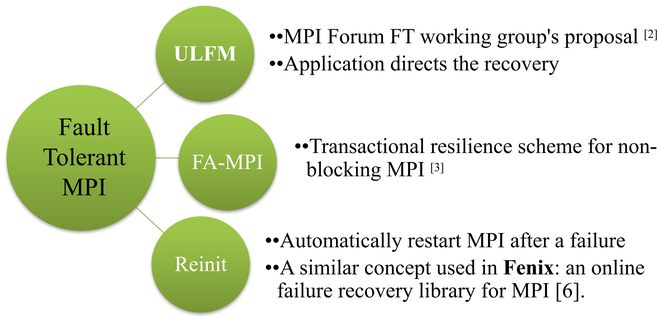
Designing a Reinitializable and Fault Tolerant MPI Library

Nawrin Sultana[1] Shane Farmer[1] Anthony Skjellum[2] Ignacio Laguna[3] Kathryn Mohror[3] Murali Emani[3]
[1] Auburn University, Auburn, AL
[2] University of Tennessee at Chattanooga, TN
[3] Lawrence Livermore National Laboratory, Livermore, CA

Motivation

- MTBF reduces as HPC systems grow larger with 1000's to 1M+ processors/cores [1]
- ***Checkpoint/Restart*** is the standard way of surviving failures
- We want ***Checkpoint/Restart*** to be faster
- We need to be able to reinitialize MPI rapidly

Current Fault-Tolerance Proposals for MPI



Objectives

- Checkpoint the state of the MPI library
  - Remove application state dependency
  - Allow applications to tell MPI to checkpoint itself
- Reinitialize the state of the MPI library from a previously taken checkpoint
  - Running MPI job will continue without being suspended from the job queue
- Define portable checkpoint

Considerations and Challenges

- Synchronous checkpoints
  - Checkpoint when an application is in a consistent global state
- No pending requests when the checkpoint is taken
- No one-side communication
- Define checkpoints of MPI state
  - Identify what MPI state needs to be checkpointed (e.g., communicator, datatypes, etc.)
  - No implementation-dependent aspects
- Reinitialize low-level transport components to known consistent state

Implementation/Architecture

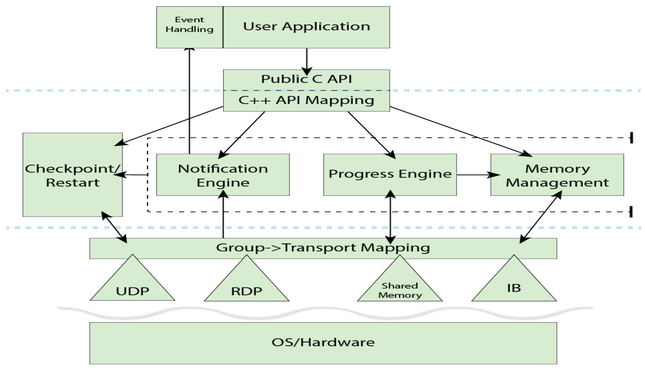


Figure-1: Module structure of new MPI implementation (exaMPI)

Detect Process Failure

- Fault Detector
  - Runtime processes will detect process failure
  - No restrictions on failure detection
  - Also includes any local/global failure detector such as Heartbeat [4]

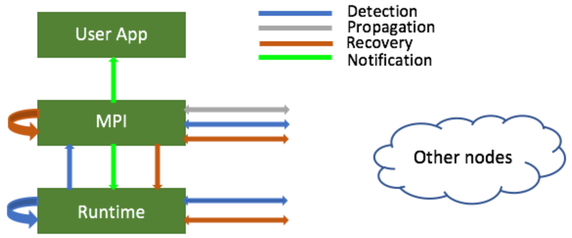


Figure-2: Failure detection and recovery in different levels of system

Recovering From Failure

```
#include <mpi.h>

int main(int argc, char **argv) {
  // variable initialization

  while (!abort && !done) {
    switch(code) {
      case MPI_SUCCESS:
      case MPIX_TRY_REINIT: // code indicating reinitialization
        /* Reads last checkpointed MPI state or starts fresh as needed */
        recovery_code = MPIX_New_reinitialize(. . .);
        MPIX_Get_fault_epoch(. . ., &fault_epoch);
      . . .
    }
    switch(recovery_code) {
      case MPI_SUCCESS:
      case MPIX_SUCCESS_RECOVERY: // Reloaded MPI checkpoint
      case MPIX_SUCCESS_RESTART:  // Process restarted from scratch
        . . .
      code = App_Checkpoint_Read(. . ., &restart_iteration);
        . . .
      code = main_loop(restart_iteration, &done);
        . . .
    }
  }
  MPI_Finalize();
  return 0;
}

int main_loop(int restart_iteration, int *done) {
  for (int i = restart_iteration; i < MAX_ITERATIONS; ++i) {
    . . .
    Application_Checkpoint_Write(. . .)
    . . .
    MPIX_Checkpoint_write(. . .);
    . . .
  }
  return code;
}
```

Step1	Detect the failure of process K;
Step2	Notify all other processes through asynchronous notification;
Step3	Spawn a new process on the node where rank K ran on;
Step4	Spawning entity loads CP0 from checkpoint;
Step5	Notify that the newly spawned process of the modified table made from CP0; it is now in a consistent state;
Step6	We notify all the other ranks of the modified table by sending just the new process tuple;
Step7	A new checkpoint of the modified process table is made; Call it CP1;
Step8	The MPI_COMM_WORLD and group underneath is restored to a consistent state; computation can continue

References

1. G. Gibson, B. Schroeder, and J. Digney, "Failure tolerance in petascale computers," *Software Enabling Technologies for Petascale Science*, vol. 3, no. 4, November 2007, pp. 4–10.
2. Bland, Wesley, et al. "An evaluation of user-level failure mitigation support in MPI." *European MPI Users' Group Meeting*. Springer Berlin Heidelberg, 2012.
3. Hassani, Amin, Anthony Skjellum, and Ron Brightwell. "Design and evaluation of FA-MPI, a transactional resilience scheme for non-blocking MPI." *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 2014.
4. Kawazoe Aguilera, et al. "Heartbeat: A timeout-free failure detector for quiescent reliable communication." *Distributed algorithms* (1997).
5. Moody, Adam, et al. "Design, modeling, and evaluation of a scalable multi-level checkpointing system." *Proceedings of the 2010 ACM/IEEE international conference for high performance computing, networking, storage and analysis*. IEEE Computer Society.
6. Teranishi, Keita, et al. Fenix: An Online Failure Recovery Library for MPI applications. No. SAND2016-1907PE. SNL-CA, Livermore, CA;SNL, Albuquerque, NM, 2016.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC (LLNL-POST-734163) and partial support from NSF under grants 1562659 and 1229282 acknowledged. The ideas expressed are those of the authors and not necessarily LLNL or NSF.