SAND2013-9179P

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

Amin Hassani[1] Anthony Skjellum[1] Ron Brightwell[2]

[1] University of Alabama at Birmingham, Birmingham, Alabama

[2] Sandia National Laboratories, Albuquerque, New Mexico

THE UNIVERSITY OF ALABAMA AT BIRMINGHAM

Knowledge that will change your world

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. Scalability and Management of fault-free overhead are the key concerns. Failure models supported by FA-MPI include but are not limited just to "process failures" unlike other proposed systems [2][3].

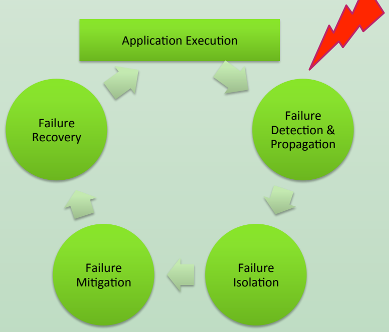
Fault-Free Overhead

We expect that applications using FA-MPI to run longer on larger machines as compare to non-fault-tolerant versions of the application. In order to achieve resiliency, some sacrifice in instantaneous performance cannot be avoided. We expect to have slightly less performance because of the synchronization call at the end of TryBlocks. 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.

Four Phases of Fault-Tolerance

FA-MPI allows failure detection and propagation through the TryBlock mechanism. But after this first phase, failed components should be isolated from other parts and then failure mitigation phase alleviate the severity of failure. Finally recovery phase brings back the state of the system to the healthy state before failure.

Sometimes continuing work with a failed communicator is impossible. FA-MPI can provide API calls similar to the approach in [2] to shrink a sick communicator to smaller size (and continue work with the new smaller communicator) and possibly regrow it later by spawning new processes and merge all 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 are not implied. Recovery comprises another block of computation and communication and should can be handled in a TryBlock even in the presence of faults. Any failures during recovery can result in retry or rollback to the last checkpoint. These all can be policies decided by the application with the help of FA-MPI.



FA-MPI's API

TryBlock

TryBlocks are the fundamental extensions enabling applications to behave transactional using FA-MPI. It allows applications to implement multiple levels of recovery.

int MPI_TryBlock_start(**MPI_Comm** comm, **int** flag, **MPI_Request*** try_request);

- Collective but not necessarily synchronizing
- *flag* defines the need for global error propagation

int MPI_TryBlock_ifinish(**MPI_Request** try_request, **MPI_Timeout** timeout, **int** count, **MPI_Request**[] array_of_requests, **MPI_Status**[] array_of_statuses);

- Synchronizing collective propagates global errors
- Can be nested

Failure Injection

Failure injection is a mechanism that allows both an MPI implementation and user application to inject errors into MPI consistently and to faciliate different methods of ABFT recoveries.

int MPI_Request_raise_error(**MPI_Request** request, **int** errcode);

- Define error codes:
  - MPI_ERR_PROCESS_FAILED
  - MPI_ERR_REQUEST_FAILED
  - implementations can add additional error codes

Timeout

Timeout is an effective mechanism to handle exceptional behaviors properly, such as unexpected delay in response or remote failure.

int MPI_Timeout_set_ticks(**MPI_Timeout*** timeout, **int** ticks);

int MPI_Timeout_get_ticks(**MPI_Timeout** timeout, **int*** ticks);

- In units of MPI_Wtick();

Local Completion

Local completion functions do not destroy request handle upon success. They add the timeout mechanism.

int MPI_Wait_local(**MPI_Request** request, **MPI_Status** *status, **MPI_Timeout** timeout);

int MPI_Waitany_local(**int** count, **MPI_Request** array_of_requests[], **int** *index, **MPI_Status** *status, **MPI_Timeout** timeout);

int MPI_Waitsome_local(**int** incount, **MPI_Request** array_of_requests[], **int** *outcount, **int** array_of_indices[], **MPI_Status** array_of_statuses[], **MPI_Timeout** timeout);

int MPI_Waitall_local(**int** count, **MPI_Request** array_of_requests[], **MPI_Status** array_of_statuses[], **MPI_Timeout** timeout);

Or

Chane the semantics of MPI to not delete request handle. Timeout can be added with:

int MPI_Request_timeout_set(**MPI_Request** request, **Timeout** timeout)

Failure Notification

Failures can be revealed to the user after TryBlock's finish call. Querying for failure is a mechanism for user to retrieve information about local and global failures in the system.

int MPI_Get_failed_requests(**MPI_Request try_request**, **int** max, int* count, int[] array_of_index);

int MPI_Get_failed_ranks(**MPI_Request** try_request, **MPI_Group*** fgroup);

int MPI_Get_failed_objects(**MPI_Request** try_request, **int** max, **int*** count, **MPI_Comm**[] array_of_communicators);

Composing an Applications with FA-MPI

Data Parallel

```
begin program
initialization;
if (restarted) then
    load data from last checkpoint (optional);
end if;

repeat
    while (more_work_to_do) do
        MPI_TryBlock_start(comm,global,req);
        computation, communication and/or I/O;
        local wait for operations to finish;
        inject local errors;
        MPI_TryBlock_ifinish(req);
        MPI_Wait_local(req, status, timeout);

        if (failure_happened) then
            isolate and mitigate the failure;
            if (recovery_needed) then
                break;
            end if;
        end if;
        periodically checkpoint;
    end while;
    if (recovery_needed) then
        do recovery procedure;
    end if;
until (more_work_to_do or restart_needed);
end program;
```

Master

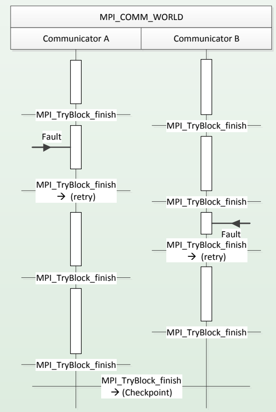
```
function submit_job(comm, tryreq, job)
    MPI_TryBlock_start(comm,global,tryreq);
    non-blocking send job;
    non-blocking receive results;
    MPI_TryBlock_ifinish(tryreq);
end function;

begin program
initialization;
create a communicator for each worker;
for (i from 1 to number_of_workers) do
    submit_job(comm[i], tryreqs[i], jobs[i]);
end for;
while (more_work_to_do and still_have_workers) do
    MPI_Waitany_local(req,idx,timeout);
    if (error_occured in tryreqs[idx]) then
        recover jobs[idx];
        free tryreqs[idx];
        submit_job(comm[idx], tryreqs[idx], jobs[idx]);
    else
        free tryreqs[idx];
        create new jobs[idx];
        submit_job(comm[idx], tryreqs[idx], jobs[idx]);
    end if;
end while;
end program;
```

Worker

```
begin program
initialization;
create a communicator with master;

while (more_work_to_do) do
start:
    MPI_TryBlock_start(comm,global,req);
    non-blocking receive job;
    if (not_more_work_to_do) then
        goto finish;
    end if;
    compute results;
    non-blocking send results;
finish:
    MPI_TryBlock_finish(req);
    if (error_happened) then
        do recovery;
        goto start;
    end if;
end while;
end program;
```



Conclusions and Future Work

FA-MPI is a set of extension APIs for the MPI standard to allow fault-awareness using a transactional model. FA-MPI detects and propagates failures in non-blocking communication calls, and notifies global failures to the application. 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. The subset of MPI we support are non-blocking APIs in MPI-3 and logical extensions thereof in MPI-4 (*e.g.*, complete support for non-blocking collectives is expected in MPI-4).

References

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

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

[3] 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.

[4] 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.

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

[6] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103-111, August 1990. ISSN 0001-0782.

Acknowledgments

This work was supported in part by the National Science Foundation under grant CCF-1239962. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.