MPI-4 Fault Tolerance

Chapter text available for a limited time:

http://www.mcs.anl.gov/~wbland/ft.pdf

Permanent Home:

https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/User_Level_Failure_Mitigation

Failure Notification

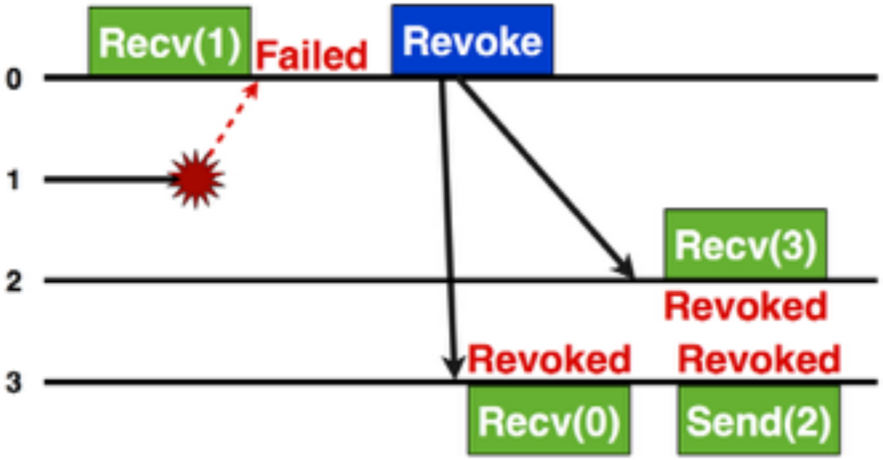
- Local failure notification only
  - Global notification can be built on top of these semantics
- Return error class to indicate process failure
  - **MPI_ERR_PROC_FAILED**
- Errors are only returned if the result of the operation would be impacted by the error
  - i.e. Point-to-point with non-failed processes should work unless routing is broken
- Some processes in an operation will receive MPI_SUCCESS while others will receive MPI_ERR_PROC_FAILED
  - i.e. Collective communication will sometimes work after a failure depending on the communication topology
    - Broadcast might succeed for the top of the tree, but fail for some children
    - MPI_ALLREDUCE would always fail if the error occurred before the start of the operation
- Wildcard operations must return an error because the failed process might have been sending the message that would have matched the MPI_ANY_SOURCE.
  - Return MPI_ERR_PENDING for MPI_IRECV.
  - If application determines that it's ok, the request can be continued after re-enabling wildcards

Failure Notification (cont.)

- To find out which processes have failed, use the two-phase functions:
  - **MPI_COMM_FAILURE_ACK(MPI_Comm comm)**
  - Internally “marks” the group of processes which are currently locally know to have failed
    - Useful for MPI_COMM_AGREE later
  - Re-enables wildcard operations on a communicator now that the user knows about the failures
    - Could be continuing old wildcard requests or new ones
  - **MPI_COMM_FAILURE_GET_ACKED(MPI_Comm comm, MPI_Group *failed_grp)**
    - Returns an MPI_GROUP with the processes which were marked by the previous call to MPI_COMM_FAILURE_ACK
    - Will always return the same set of processes until FAILURE_ACK is called again
- Must be careful to check that wildcards should continue before starting/restarting a wildcard operation
  - Don’t enter a deadlock because the failed process was supposed to send a message
- Future wildcard operations will not return errors unless a new failure occurs.

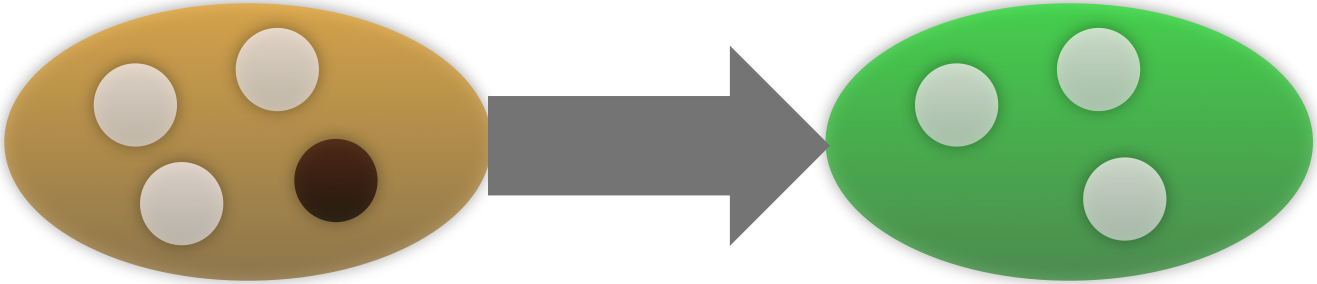
Failure Propagation

- Often unnecessary
  - Let the application discover the error as it impacts correct completion of an operation.
- When necessary, manual propagation is available.
  - **MPI_COMM_REVOKE(MPI_Comm comm)**
    - Interrupts all non-local MPI calls on all processes in comm.
    - Once revoked, all non-local MPI calls on all processes in comm will return MPI_ERR_REVOKED.
      - Exceptions are MPI_COMM_SHRINK and MPI_COMM_AGREE (later)
  - Necessary for deadlock prevention
    - Example on right



Failure Recovery

- Some applications will not need recovery.
  - Point-to-point applications can keep working and ignore the failed processes.
- If collective communications are required, a new communicator must be created.
  - **MPI_Comm_shrink(MPI_Comm *comm, MPI_Comm *newcomm)**
    - Creates a new communicator from the old communicator excluding failed processes
    - If a failure occurs during the shrink, it is also excluded.
    - No requirement that comm has a failure. In this case, it will act identically to MPI_Comm_dup.
- Can also be used to validate knowledge of all failures in a communicator.
  - Shrink the communicator, compare the new group to the old one, free the new communicator (if not needed).
  - Same cost as querying all processes to learn about all failures



Fault Tolerant Consensus

- Sometimes it is necessary to decide if an algorithm is done.
  - **MPI_COMM_AGREE(MPI_comm comm, int *flag);**
    - Performs fault tolerant agreement over boolean flag
    - Non-acknowledged, failed processes cause MPI_ERR_PROC_FAILED.
    - Will work correctly over a revoked communicator.
  - Expensive operation. Should be used sparingly.
  - Can also pair with collectives to provide global return codes if necessary.
- Can also be used as a global failure detector
  - Very expensive way of doing this, but possible.
- Also includes a non-blocking version

MPI RMA

- **MPI_WIN_REVOKE**
  - Provides same functionality as MPI_COMM_REVOKE
- The state of memory targeted by any process in an epoch in which operations raised an error related to process failure is undefined.
  - Local memory targeted by remote read operations is still valid.
  - It's possible that an implementation can provide stronger semantics.
  - If so, it should do so and provide a description.
  - We may revisit this in the future if a portable solution emerges.
- MPI_WIN_FREE has the same semantics as MPI_COMM_FREE

Passive Target Locks

- Without restricting lock implementations, it's difficult to define the status of a lock after a failure
  - With some lock implementations, the library doesn't know who is holding the lock at any given time.
  - If the process holding the lock fails, the implementation might not be able to recover that lock portably.
    - Some other process should be notified of the failure and recovery can continue from there (probably with MPI_WIN_REVOKE).
    - If the implementation can get around the failure, it should try to do so and mask the failure.

MPI I/O

- When an error is returned, the file pointer associated with the call is undefined.
  - Local file pointers can be set manually
    - Application can use MPI_COMM_AGREE to determine the position of the pointer
  - Shared file pointers are broken
- **MPI_FILE_REVOKE**
  - Provides same functionality as MPI_COMM_REVOKE
- MPI_FILE_CLOSE has similar to semantics to MPI_COMM_FREE