Fault Tolerant MPI

Aurélien Bouteiller

ICL “Friday lunch”

Feb. 7, 2014

ICL UT
INNOVATIVE
COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

Fault Tolerant MPI, Motivation

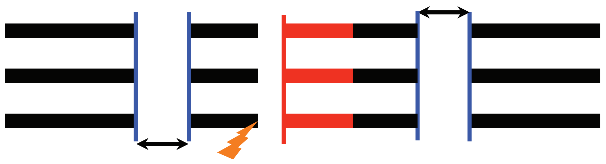
- Failures are becoming more than a distant threat
- Checkpoint/Restart (C/R) is good, but it could be better
  - Even C/R can benefit from MPI support!
- Even better FT models are available, but lack support from MPI

Standardization of MPI behavior after failures is a key missing infrastructure

Targeted Application Domains

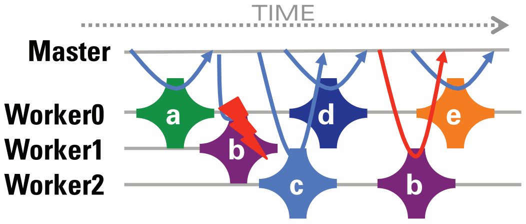
Coordinated Checkpoint/Restart, Automatic, Compiler Assisted, User-driven Checkpointing, etc.

In-place restart (i.e., without disposing of non-failed processes) accelerates recovery, permits in-memory checkpoint



Naturally Fault Tolerant Applications, Master-Worker, Domain Decomposition, etc.

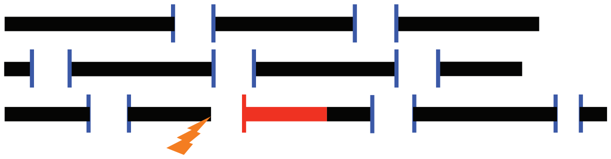
Application continues a simple communication pattern, ignoring failures



ULFM MPI Specification

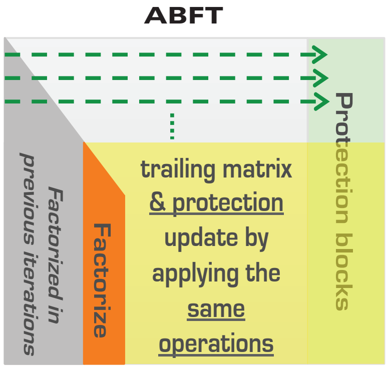
Uncoordinated Checkpoint/Restart, Transactional FT, Migration, Replication, etc.

ULFM makes these approaches portable across MPI implementations



Algorithm Fault Tolerance

ULFM allows for the deployment of ultra-scalable, algorithm specific FT techniques.



ULFM: Key Philosophy

User Level Failure Mitigation: a set of MPI interface extensions to enable MPI programs to restore MPI communication capabilities disabled by failures

- Flexibility
  - No particular recovery model imposed or favored
  - Application directs the recovery: it pays only for the level of protection it needs
  - Recovery can be restricted to subgroups for scalability
- Performance
  - Protective actions are outside of critical MPI routines
  - MPI implementors can uphold unmodified algorithms (collective, one-sided, I/O)
  - Encourages programs to be reactive to failures,
- Productivity
  - Backward compatible with legacy, fragile applications
  - Simple and familiar concepts to repair MPI
  - Provides key MPI concepts to enable FT support from Libraries, runtime, language extensions

When FT is unnecessary (small, reliable cluster, short application runtime, etc), it can be disabled completely

Failure Model

- Process Failures
  - Fail-stop failures: a process crash (dead, never comes back to life)
  - Transient (network) failures are “upgraded” to fail-stop (may be revisited later)
- Silent (memory errors) & Byzantine failures are outside of the scope
  - Memory corruptions are better addressed at the application level
  - Message corruptions can be addressed without standard modifications

Minimal Feature Set for FT

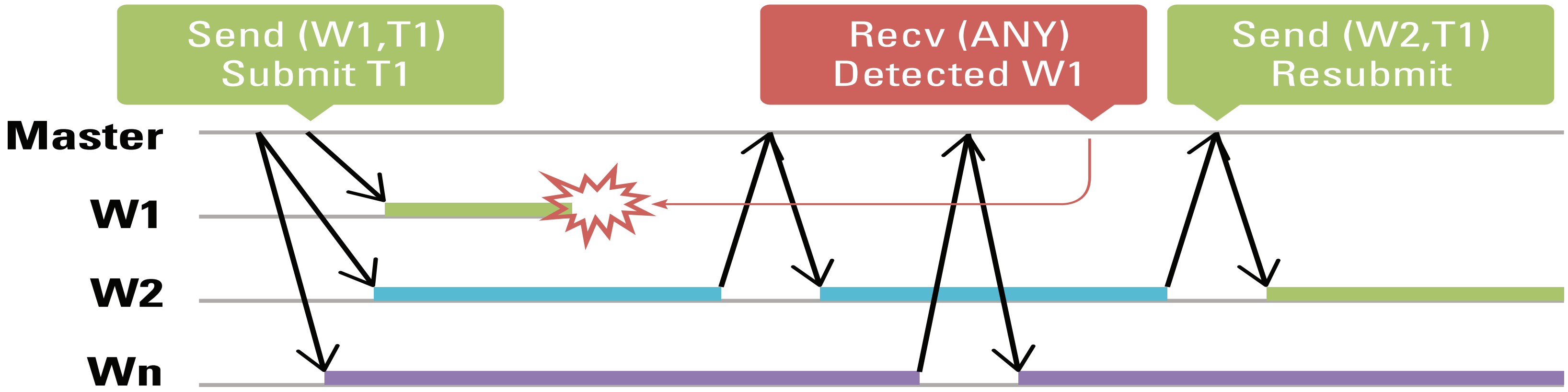
- Failure Notification
- Error Propagation
- Error Recovery

Not all recovery strategies require all of these features, that's why the interface splits notification, propagation and recovery

Failure Notification

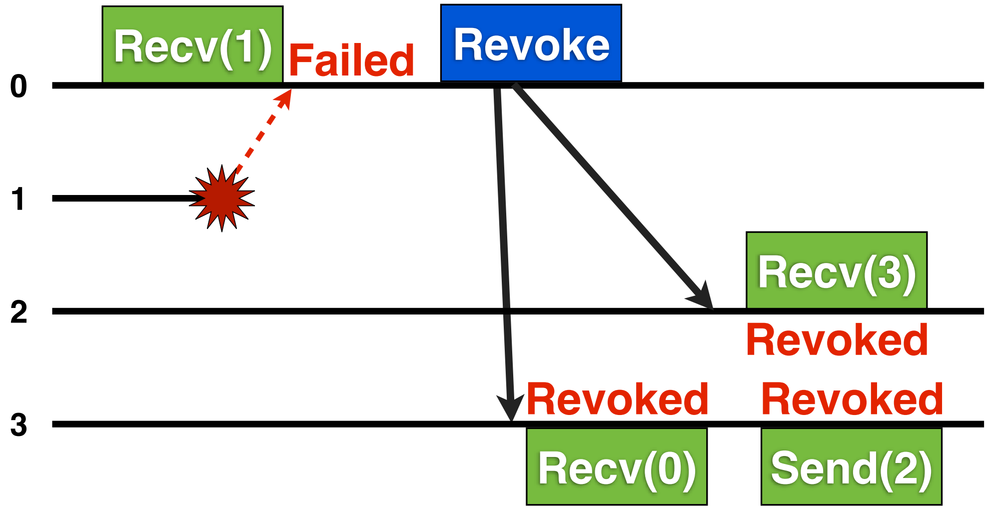
- Notification of failures is local only
  - New error MPI_ERR_PROC_FAILED Raised when a communication with a targeted process fails
- In an operation (collective), some process may succeed while other raise an error
  - Bcast might succeed for the top of the tree, but fail for some subtree rooted on a failed process
- ANY_SOURCE must raise an exception
  - the dead could be the expected sender
  - Raise error MPI_ERR_PROC_FAILED_PENDING, preserve matching order
  - The application can complete the recv later (MPI_COMM_FAILURE_ACK())
- Exceptions indicate an operation failed
  - To know what process failed, apps call MPI_COMM_FAILURE_ACK(), MPI_COMM_FAILURE_GET_ACKED()

App using notification only



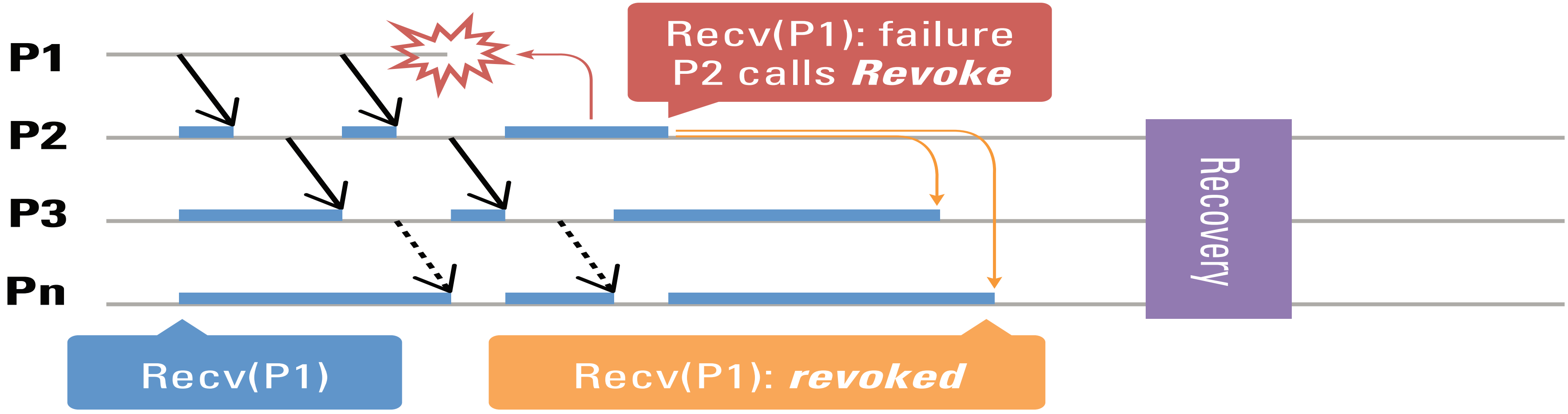
- Error notifications do not break MPI
  - App can continue to communicate on the communicator
  - More errors may be raised if the op cannot complete (typically, most collective ops are expected to fail), but p2p between non-failed processes works
- In this Master-Worker example, we can continue w/o recovery!
  - Master sees a worker failed
  - Resubmit the lost work unit onto another worker
  - Quietly continue

Error Propagation



- Errors are local, processes have a different view of failures
  - We need a tool to resolve potential inconsistent behavior
- When necessary, app can manually propagate an error
  - MPI_COMM_REVOKE(comm)
  - Interrupts all non-local MPI calls at all ranks on comm
  - Once revoked, any non-local operation on comm raises MPI_ERR_REVOKED (except recovery functions, duh)

App using propagation only

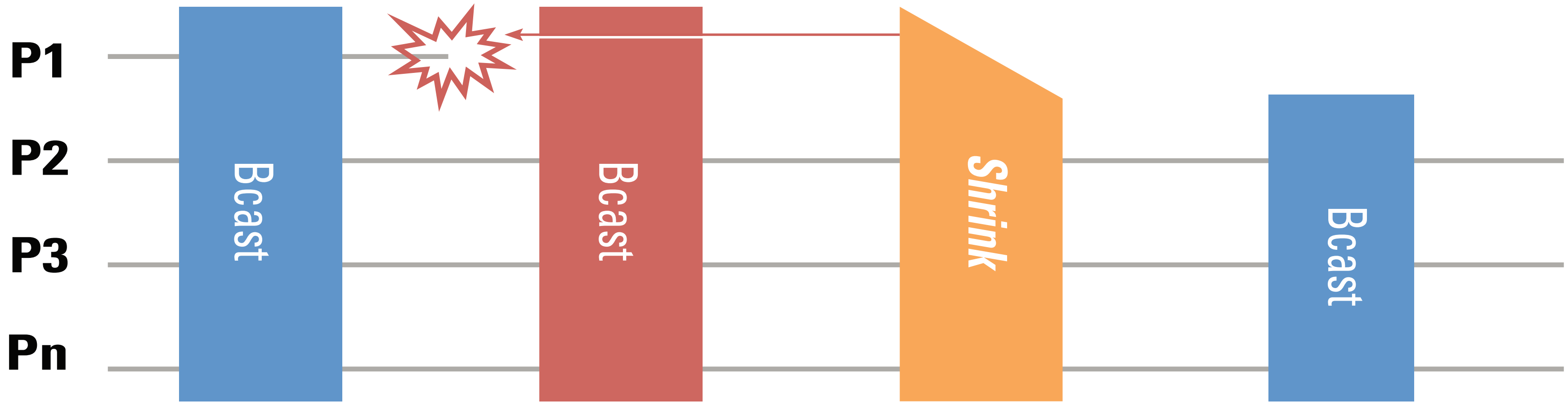


- Application does only p2p communications
- P1 fails, P2 raises an error and wants to change comm pattern to do application recovery
- but P3..Pn are stuck in their posted recv
- P2 unlocks them with Revoke
- P3..Pn join P2 in the new recovery p2p communication pattern

Error Agreement

- When in need to decide if there is a failure and if the condition is recoverable (collectively)
  - MPI_COMM_AGREE(comm, flag)
    - Fault tolerant agreement over boolean flag
    - Unexpected failures (not acknowledged before the call) raise MPI_ERR_PROC_FAILED
    - The flag can be used to compute a user condition, even when there are failures in comm
- Can be used as a global failure detector

Error Recovery



- Restores full communication capability (all collective ops, etc).
- MPI_COMM_SHRINK(comm, newcomm)
  - Creates a new communicator excluding failed processes
  - New failures are absorbed during the operation

Also supported

- Remote Memory Access Window objects
  - The window becomes unusable after a failure, but
  - State of memory window is defined after an error (except for write regions)
  - The window can be recreated (by repairing the parent communicator)
- Files
  - The file pointer is scrambled after a failure, but
  - It can be reset by the application, and resume
  - The file can be recreated (by repairing the parent communicator)

Standardization progress

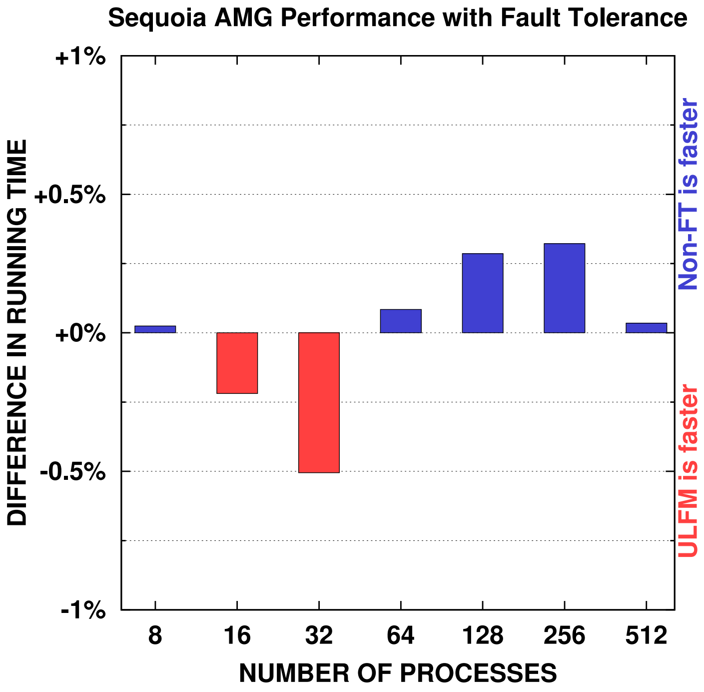
- Draft document is complete
- First reading in 3 weeks from now (San Jose MPI forum meeting)
- Probably first vote in june

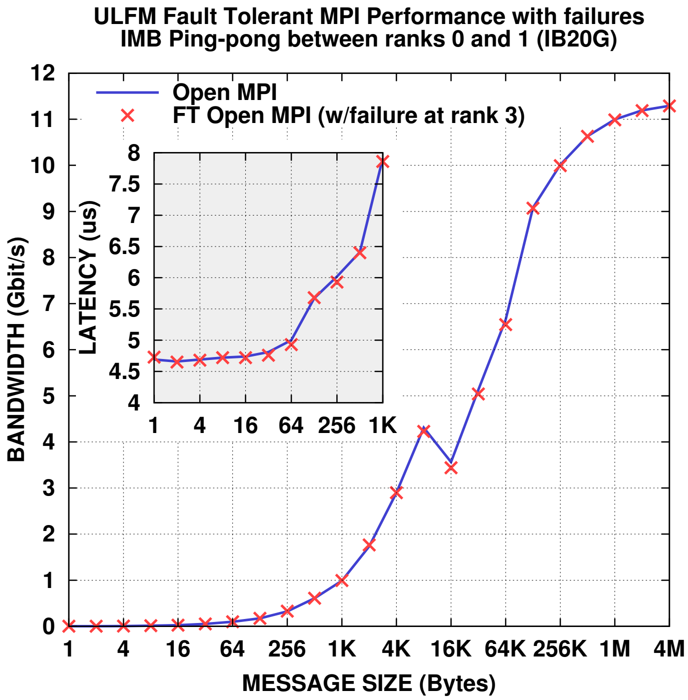
Document design participants:

UTK (lead), Argonne (Wesley), Intel (J. Dinan), ORNL (T. Naughton & friends), other friendly reviewers

Implementation in Open MPI

- It works! Performance is good!





Sequoia AMG is an unstructured physics mesh application with a complex communication pattern that employs both point-to-point and collective operations. Its failure free performance is unchanged whether it is deployed with ULFM or normal Open MPI.

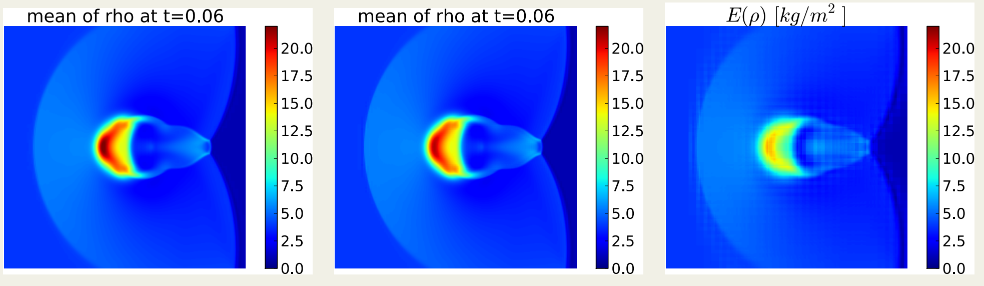
The failure of rank 3 is detected and managed by rank 2 during the 512 bytes message test. The connectivity and bandwidth between rank 0 and rank 1 are unaffected by failure handling activities at rank 2.

ICL UT Thanks for CREST, Rinken support

User activities

- ORNL: Molecular Dynamic simulation
  - Employs coordinated user-level C/R, in place restart with Shrink
- UAB: transactional FT programming model
- Tsukuba: Phalanx Master-worker framework
- Georgia University: Wang Landau Polymer Freezing and Collapse
  - Employs two-level communication scheme with group checkpoints
  - Upon failure, the tightly coupled group restarts from checkpoint, the other distant groups continue undisturbed
- Sandia: Sparse solver
  - ???
- Others…

- Cray: CREST miniapps, PDE solver Schwartz, PPStee (Mesh, automotive), HemeLB (Lattice Boltzmann)
- UTK: FTLA (dense Linear Algebra)
  - Employs ABFT
  - FTQR returns an error to the app, App calls new BLACS repair constructs (spawn new processes with MPI_COMM_SPAWN), and re-enters FTQR to resume (ABFT recovery embedded)
- ETH Zurich: Monte-Carlo
  - Upon failure, shrink the global communicator (that contains spares) to recreate the same domain decomposition, restart MC with same rank mapping as before



(a) failure-free (b) few failures (c) many failures

Figure 5. Results of the FT-MLMC implementation for three different failure scenarios.

Credits: ETH Zurich

Thank you

To know more…

http://fault-tolerance.org/ulfm/

ICL UT
INNOVATIVE
COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE