How to mitigate node failures in hybrid parallel applications

Maciej Szpindler
m.szpindler@icm.edu.pl

University of Warsaw
Interdisciplinary Centre for Mathematical and Computational Modelling
http://www.icm.edu.pl

Agenda

- Introduction
- Current state
- Problem definition
- Fault-recovery model
- Detection and reconstruction
- Summary

Introduction and context

- Parallel applications are usually multi-process
  - Dominant model is message passing (MPI)
- Applications (in general) are likely multithreaded
  - Dominant models are based on threads (eg. OpenMP)
  - MPI model provides full support for threads
- In search for scalability these two models are coupled (hybrid parallelism)
  - Notable example: MPI+OpenMP – iter-node and intra-node connectivity respectively
  - Other approaches include MPI-3 shared memory model
  - **No fault tolerance is supported - must be provided on application level (as for MPI in general)**

Motivation

- Most HPC systems represent „cluster” class
  - Many nodes, each with many cpus (cores)
  - Inside one node many h-w components are shared
- Many failures have effect all node’s processes
  - In case of h-w failure all cpus in a node are affected
- Hybrid parallel applications use process-per-node allocation
  - In this case process failure means node failure
- <u>Application recovery often requires node recovery</u>

Current state
fault-tolerance in MPI applications

- Many approaches without wider adoption
- User-Level Failure Mitigation (ULFM) proposal
  - Proposal for the next MPI Standard version
  - Set of MPI primitives to apply on application level
  - Full description: https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/323
- Support for ULFM
  - MPICH: almost complete in version 3.2 (pre-release)
  - OpenMPI: dedicated ulfm-enabled branch
- More discussion on:
  - http://fault-tolerance.org

Current state
update (September 2015)

- Many approaches without wider adoption
- User-Level Failure Mitigation (ULFM) proposal
  - Proposal for the next MPI Standard version
  - Set of MPI primitives to apply on application level
  - Full description: https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/323
- Support for ULFM
  - MPICH: almost complete in version 3.2 (pre-release)
  - OpenMPI: dedicated version **ulfm-1.0 released**
- More discussion on:
  - http://fault-tolerance.org

Problem definition

- Node failure occurs with hybrid parallel application in a multi-node runtime environment
  - detect node failure,
  - exclude failed processes,
  - recover application (with additional node restarted).
- Desired approach:
  - Distributed – do not involve global operations
  - Independent of a total number of application processes (nodes used)

Fault tolerance model– ULFM

- A set of primitives for application level
  - Failure detection
  - Failure notification
  - Error propagation
  - Communication recovery
- Common usage*
  - Detect – Revoke – Shrink – **Repair**
- Repair stage
  - Need to be provided on an application level

*Bland, Wesley, et al. "An evaluation of User-Level Failure Mitigation support in MPI." Computing, 2013

Failure detection – ULFM

- MPI supports error handling with:
  - MPI_ERRORS_ARE_FATAL error handler (default approach)
    - Immediately terminates all MPI processes
  - MPI_ERRORS_RETURN handler
    - Allows proces-local operation before termination
- ULFM follows second approach
  - Communication functions may raise:
    - MPI_PROC_FAILED error code in case of participating process failure
    - MPI_COMM_REVOKE in case of communicator being revoked
- Currently ULMF is an extension
  - MPIX_[…] for MPICH, OMPI_[…] for OpenmMPI-ulfm
- Running ULMF based applications
  - MPICH: mpiexec **-disable-auto-cleanup** …
  - OpenMPI (specific builds only): mpiexec **-am ft-enable-mpi** …

2015, University of Warsaw
Interdisciplinary Centre for Mathematical and Computational Modelling

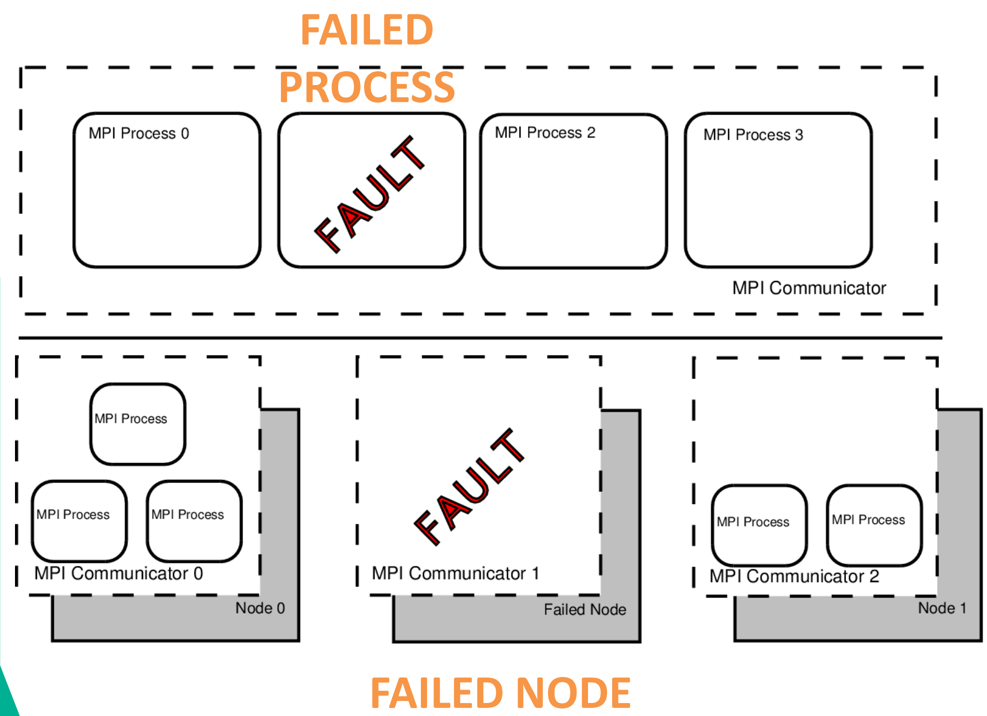
Communicator reconstruction

- Reconstruction (repair) in the ULMF model involves:
  - Excluding failed processes – shrink operation
  - Spawn new processes to replace failed ones (produce inter-communicators)
  - Merge inter-communicators
  - Optionally restore original rank order (if necessary)
- Application state recovery need additional effort
  - Recover process local state (at some point)
  - Possible approaches*: checkpoints, memory redundancy

* Ali, Md Mohsin, et al. "Application Level Fault Recovery: Using Fault-Tolerant Open MPI in a PDE Solver." *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014*

Common scenarios

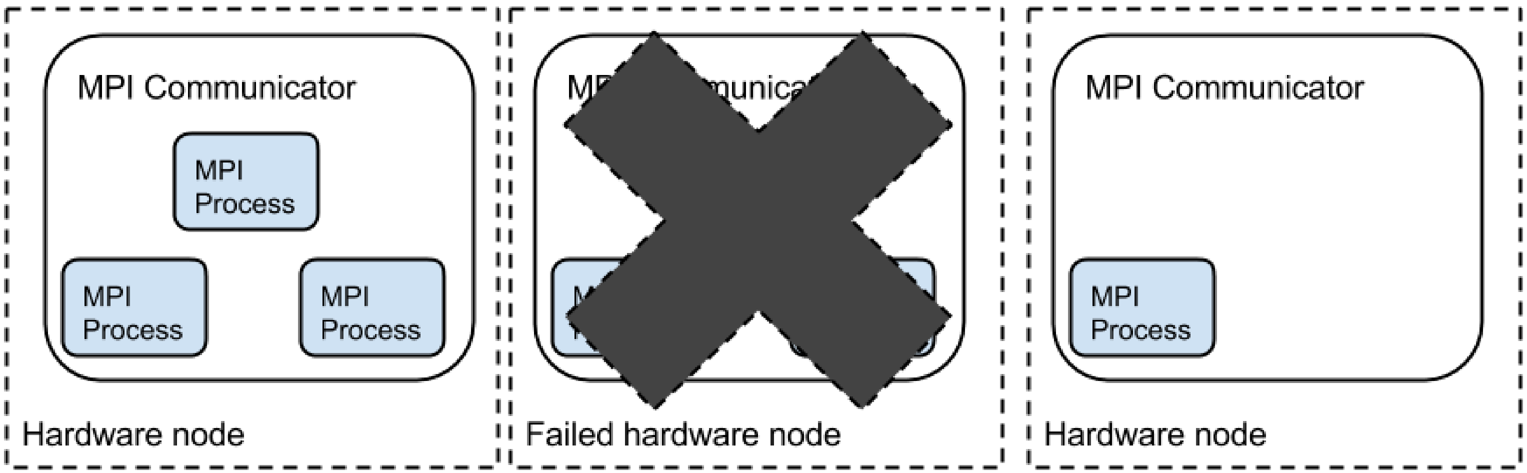
- Single process failure (top)
  - One of the communicator's members fails
  - Process local memory vanishes
- Node failure (bottom)
  - All node's processes are lost
  - Node memory is lost
  - Node communicator(if used) is unreachable
  - Common case for MPI+OpenMP choice



CASE OF INTEREST

Intra-node communitors

- Intra-node communicator is a reasonable choice
  - Logical separation of intra-node synchronization and inter-node communication
  - Enable shared memory islands with MPI_COMM_SPLIT_TYPE (and MPI-3 shared memory windows)*
- In case of node failure associated communicator is lost



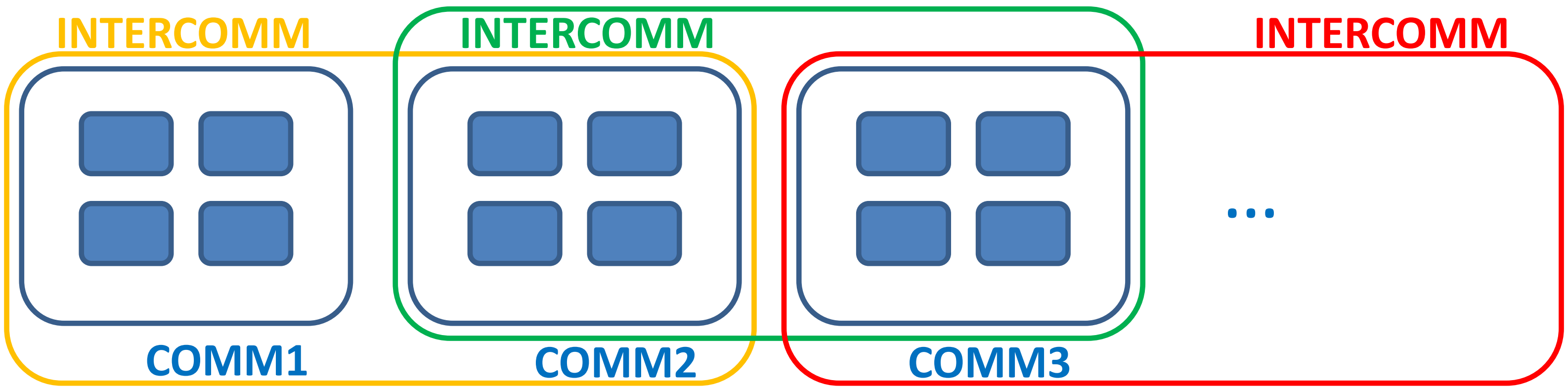
*Hoefler, Torsten, et al. "MPI+ MPI: a new hybrid approach to parallel programming with MPI plus shared memory." *Computing* 95.12 (2013)

How to detect node failure?

- ULFM detection function operates on a given communicator
  - Failed processes are eventually identified
- The most simple method is to test for process failures on MPI_COMM_WORLD communicator
  - This is not distributed approach (involves all processes in the worst case)
  - This will introduce huge overhead if called frequently

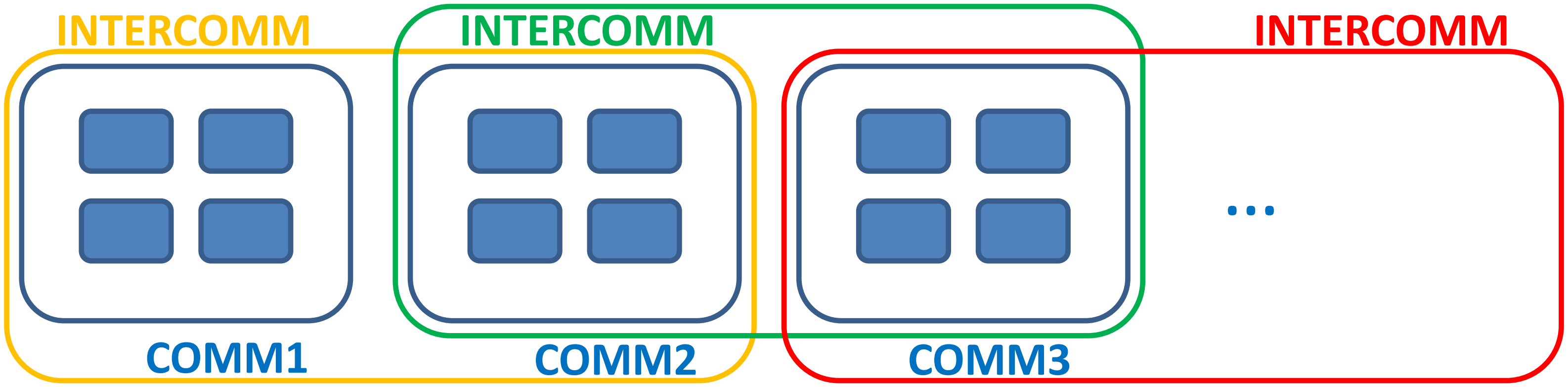
Node failure detection – 1st approach

- First approach uses intercommunicators
- Intercommunicator connects two independent communicators: local and remote group
- Proposed scheme



- Node intra-communicators (COMM1,COMM2,…) are arranged in a ring
- Each comm pair forms inter-communicator
- Each node can test its neighbor for failure using remote group of inter-comm

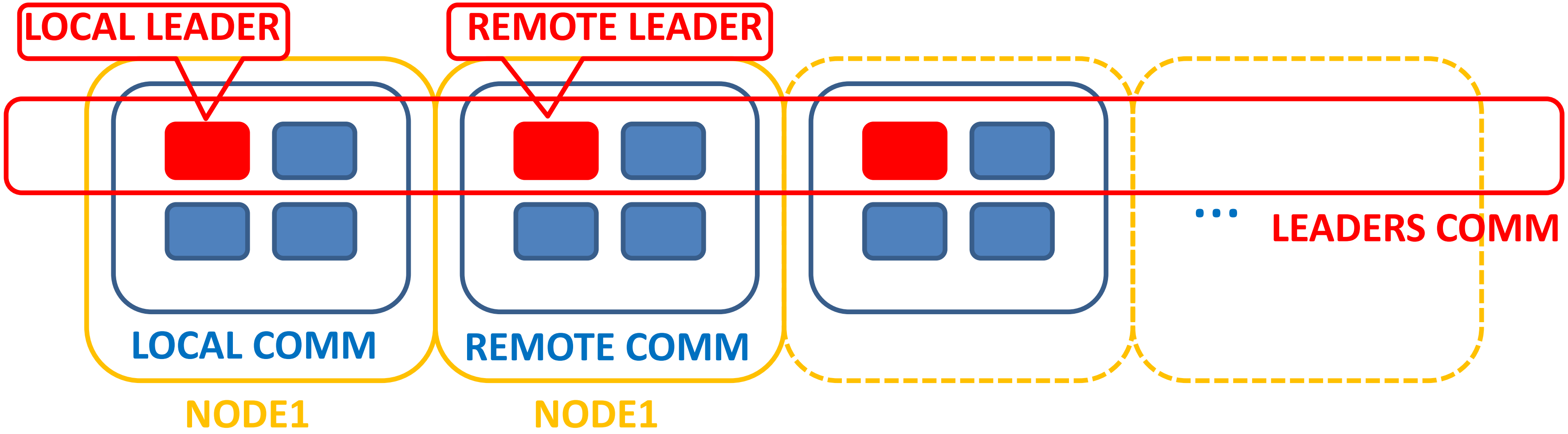
Node failure detection – 1st approach



- Each node probes failure only with remote group – no global operations
- Remarks
  - Not supported by MPICH ULFM prototype (as for version 3.2 beta)
  - Seems to be broken in OpenMPI ULFM (as for 1.7 branch, ulfm-1.0 not tested yet)

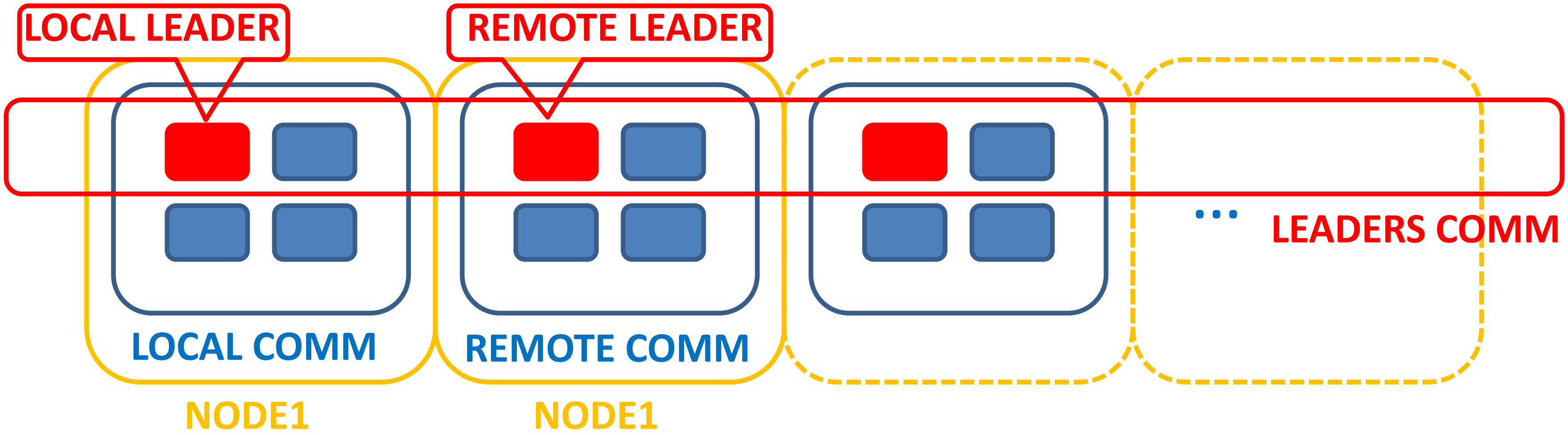
Node failure detection – 2nd approach

- More complicated scheme without intercommunicators
- Still aims to behave in a distributed way
- Additional communicators required



- Each inter-node communicator delegates „leader” process
- Leaders form dedicated communicator
- Leaders communicator is probed for failures and propagates notification

Node failure detection – 2nd approach



- Remarks
  - More sophisticated scheme
  - Additional communication between „leaders" processes
  - If local leader fails (only) remaining local processes must agree on new leader
  - Works with existing ULFM implementations

2015, University of Warsaw
Interdisciplinary Centre for Mathematical and Computational Modelling

How to recover failed node?

- Having failed (node) communicator identified one may need to recover failed node
- Application processes need to be either restarted or recovered
- Backup (checkpoint) of node-internal memory need to be provided
- How to provide additional resources to swap failed nodes – associated work (next slides)

Dynamic Resource Allocation

- Idea based on the Slurm job resize method
  - Active allocation (already started job) is extended using additional, dependent allocation
  - Accessible from Slurm API
- Somehow inspired by dynalloc Slurm plugin (for hadoop??)
  - PMI calls Slurm API functions to extend allocation
  - It is a legal Slurm operation (no hacks)
- Eliminates need of pre-allocation or over-subscription of processes in case of dynamic MPI application

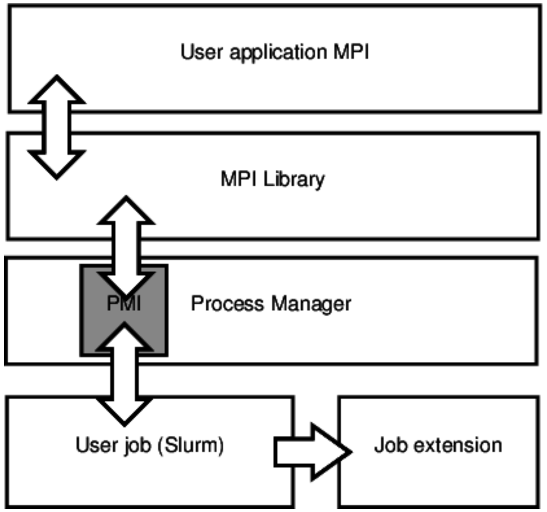
MPI and PMI

- MPI (standard) do not define how to create new processes
- Process Management Interface* (PMI) – quasi standard
  - v1 and v2, PMI3 (?)
  - Abstract layer for inter-node process management
- Different implementations
  - Hydra – MPICH process manager (pm)
  - PMIx – OpenMPI effort (in development)
  - Slurm
  - Not necessarily compatible

* Balaji, Pavan, et al. "PMI: A scalable parallel process-management interface for extreme-scale systems." *Recent Advances in the Message Passing Interface,* 2010

Dynamic process creation cascade

- MPI: MPI_Comm_spawn[_multiple]
  - API function, creates new processes
  - No control over exact startup parameters
  - `Info` argument theoretically passes additional requirements
- PMI: MPI – PMI interaction
  - With KVS pairs (key, value)
  - Parsed from MPI_Info structure
  - Comm_spawn is realized by PMI_Spawn
- Slurm:
  - Initializes slurm step and actually start process



2015, University of Warsaw
Interdisciplinary Centre for Mathematical and Computational Modelling

Resource allocation modes

- In reality, additional resources are not immediately available
  - Might be not practical for a range of applications
  - Blocking and non-blocking resource allocation if waiting is acceptable
  - Immediate allocation mode in the other case

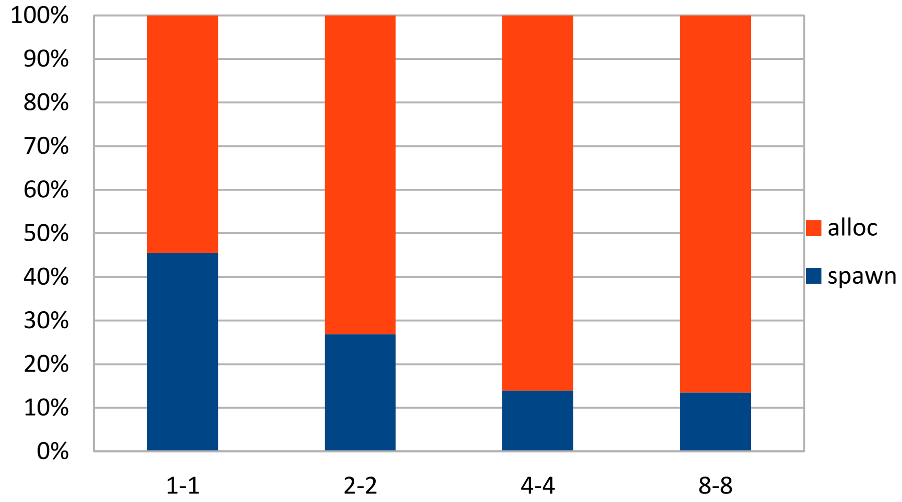
Resource allocation modes cont.

- Blocking mode
  - Blocking mode returns control if resources are already allocated or given timeout reached (uses Slurm blocking API)
- Non-blocking mode
  - Most elegant option: MPI_Icomm_spawn + Wait (with Slurm callbacks)
  - Considered in the past by the MPI Forum as MPI extension*, but eventually dropped
  - Implementation is hard due to complicated MPI progress engine
  - Most practical: use helper thread for allocation, easy to implement
- Immediate mode
  - Returns extended allocation if resources are available immediately or raise an error
  - Uses Slurm allocation constraints

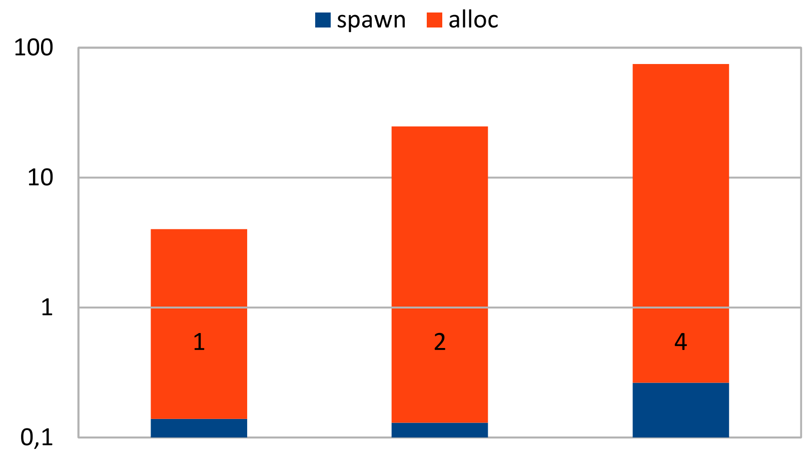
* Nonblocking Process Creation and Management Operations, MPI-Forum ticket
https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/Async-proc-mgmt

Results

- All experiments in immediate mode
- Single process failure
  - Dynamic allocation on the local node (upper figure)
- **Node failure**
  - **Dynamic allocation of the remote node (lower)**
  - **Detection costs negligible (comparing to reconstruction costs)**
- Significant costs
  - MPI_Comm_Spawn is costly*
  - Order of magnitude slower is remote allocation



Relative cost of the spawn and allocate operations for increasing number of processes (N-M: number of parents and children)



Time cost in seconds of the spawn with remote node allocation. Note the logarithmic scale.

* Bland, Wesley, et al. "An evaluation of User-Level Failure Mitigation support in MPI." Computing , 2013

Results - technicalities

- Resource allocation time with Slurm highly dependent on the machine state (at least on the computer tested)
- Technical details
  - MPICH – easier PMI integration but lack of ULFM support for inter-communicators (v. 3.2a2)
  - OpenMPI – more ULFM supported but not in the mainline code, specific process manager, PMIx not integrated with distribution
  - Slurm memory management caused problems with hydra (MPICH)
- PMIx integration in progress

Summary

- Node failure mitigation
  - Common approach with ULFM model described
  - Node failure detection addressed
  - Efficient (scalable) approaches for detection described
- Dynamic resource allocation
  - Dynamic resource allocation for MPI with Slurm shown
  - Basic integration with MPICH implemented
  - Practical usage for ULFM based application recovery demonstrated

2015, University of Warsaw
Interdisciplinary Centre for Mathematical and Computational Modelling