EReinit: Scalable and Efficient Fault-Tolerance for Bulk-Synchronous MPI Applications

Sourav Chakraborty[1], Ignacio Laguna[2], Murali Emani[2], Kathryn Mohror[2], Dhabaleswar K (DK) Panda[1], Martin Schulz[3], Hari Subramoni[1]

[1] Network Based Computing Laboratory, The Ohio State University
[2] Lawrence Livermore National Laboratory, USA
[3] Technische Universität München, Germany

Presented By: Murali Emani, LLNL

LLNL-PRES-XXXXXX
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

Lawrence Livermore National Laboratory

Bulk-Synchronous Processing (BSP)

- Model and simulate real world phenomena
  - Molecular Dynamics (ddcMD)
  - Weather Prediction (WRF)
  - Cosmology Simulation (Enzo)
  - Fluid Dynamics
  - Earthquake Simulation
- Highly scalable parallel applications
  - Runs on the largest machines
  - Can run for weeks
  - Very high probability of encountering faults
  - Commonly uses Checkpoint Restart

Fault-tolerance for Bulk-Synchronous applications is important!

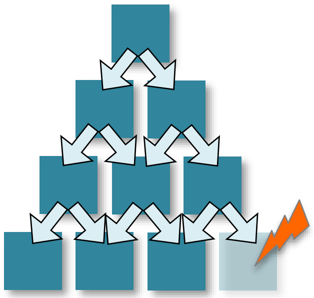
Available Fault-Tolerance (FT) Mechanisms

- ULFM (User Level Failure Mitigation)
  - An Evaluation of User-Level Failure Mitigation Support in MPI, Bland et al, Computing ‘13
- Fenix (Local Recovery)
  - Exploring Automatic, Online Failure Recovery for Scientific Applications at Extreme Scales. Gamel et al, SC ‘14
- Reinit (Global Recovery)
  - A Global Exception Fault Tolerance Model for MPI. Laguna et al, ExaMPI ‘14

Is ULFM Suitable for BSP Applications?

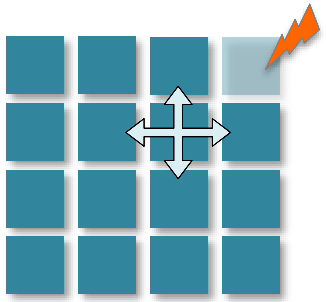
- Failure Detection
  - Checking return codes is Intrusive and impractical
  - Can be automated using PMPI
  - Prevents using other tools and libraries
  - Errors can be detected far from the operation that caused it
- Library State
  - Application may not control library communicators
  - Difficult to shrink and restore communicators
- Algorithmic Requirements
  - Many problems are impossible/impractical to recompose to arbitrary number of processes at runtime
  - Workload of failed process must be distributed

Master-slave



Some may rollback

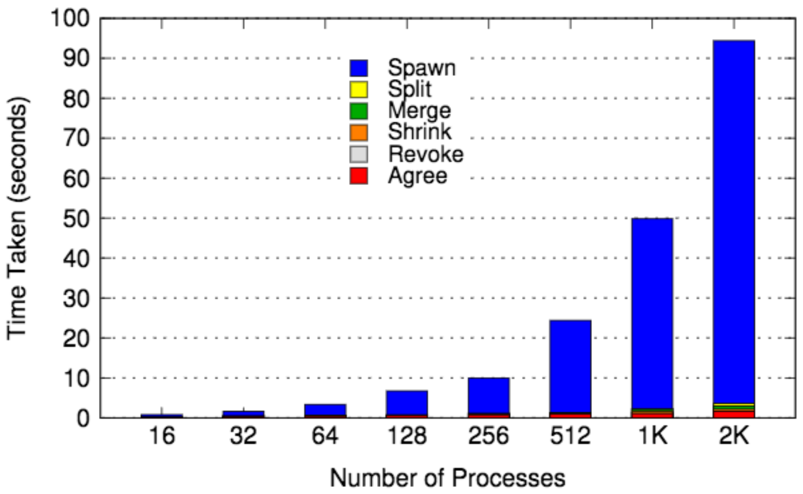
Bulk synchronous



Everyone must rollback

What about Fenix?

- Uses ULFM to detect and propagate failures
  - Comm_Agree/Revoke/Shrink
- Spawn and rewire replacement process using MPI_Comm_spawn
  - Expensive at scale
- Uses C/R to restore applications state
- Inherits the drawbacks from both approaches



Breakdown of time taken by different steps to initialize MPI in MVAPICH2

Can Reinit enable faster recovery for BSP applications?

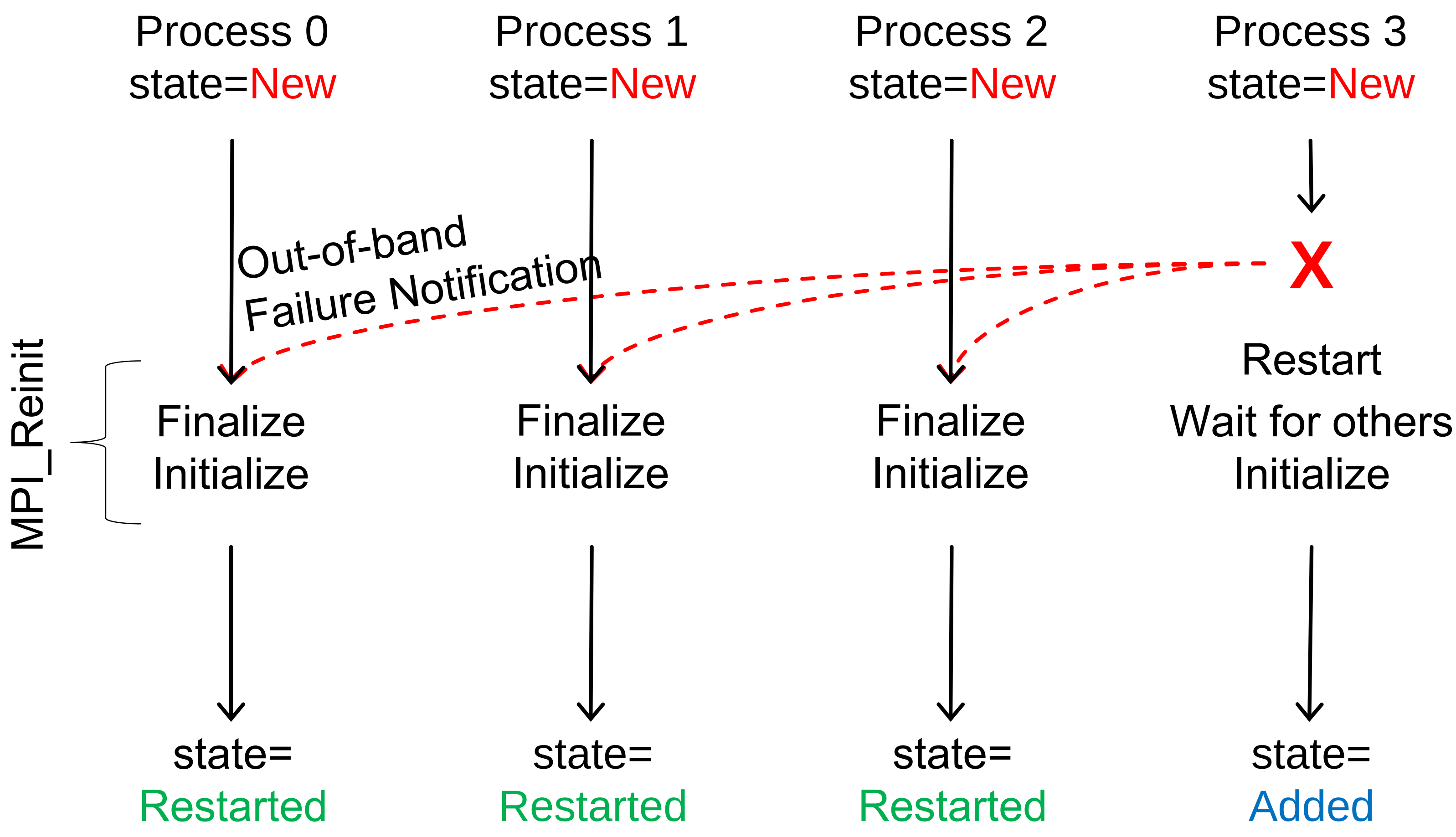
The Reinit Interface

```
//************************ API ***********************//
typedef enum {NEW, RESTARTED, ADDED}
                 MPI_Start_state;
typedef void (*MPI_Restart_point)
(int argc, char **argv, MPI_Start_state state);
int MPI_Reinit(int argc, char **argv,
                  const MPI_Restart_point point);

//******************** Application *******************//
// Real main method of the application
void resilient_main(int argc, char **argv,
      MPI_Start_state start_state)
{
  // Check if the world size is acceptable; abort otherwise
  // Find what process died, and recover based on that
  // Load checkpoint if necessary
  // Enter main computation loop (at appropriate step)
}

int main(int argc, char **argv)
{
  MPI_Init(&argc, &argv);
  // This is where the resilient MPI program starts
  MPI_Reinit(argc, argv, resilient_main);
  MPI_Finalize();
}
```

Execution Flow in Reinit



Fault Tolerance Primitives

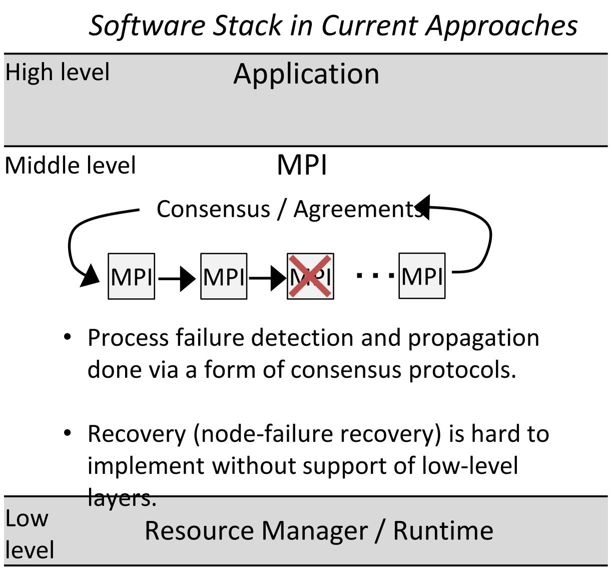
A Fault Tolerance mechanism must provide:

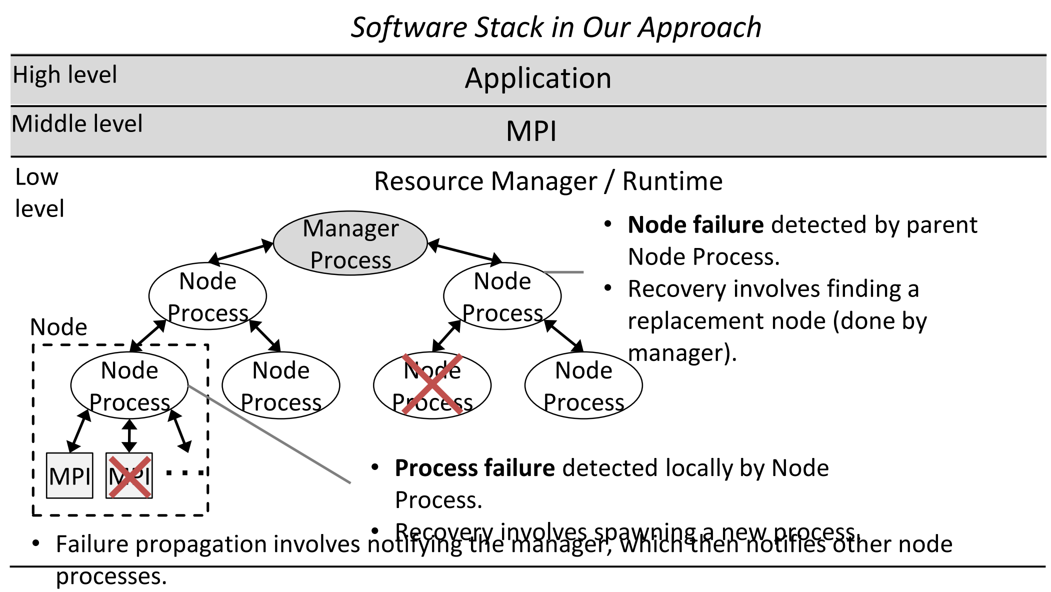
1. Failure Detection
2. Failure Propagation / Notification
3. Recovery

Fault Tolerance Primitives

Which part of the system should provide these functionalities?

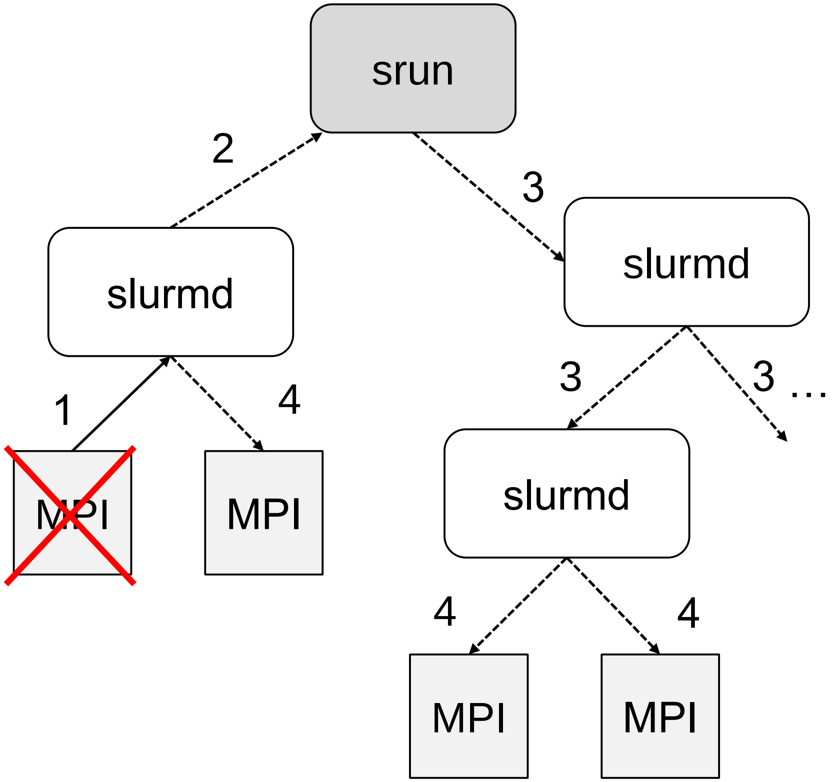
Placement of Fault Tolerance Primitives





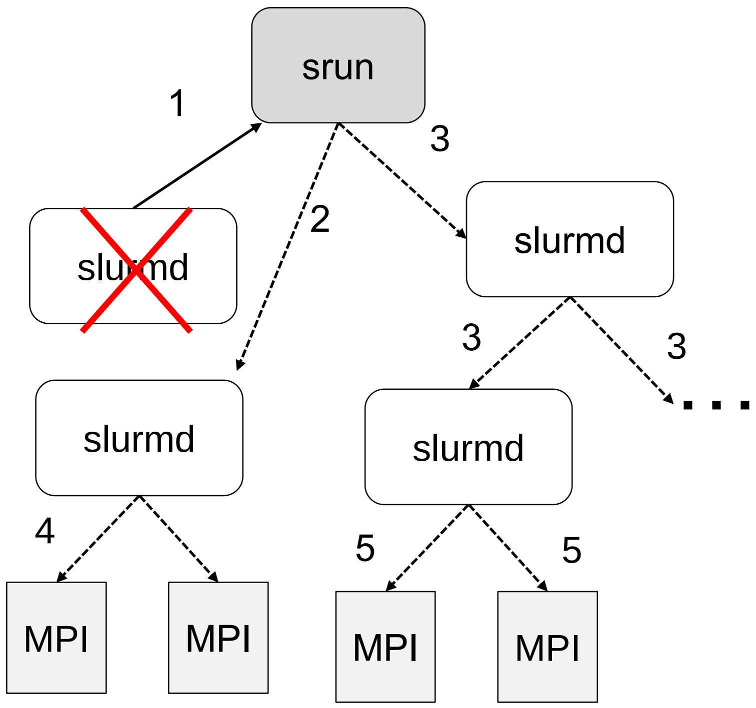
- Key observations:
  - MPI libraries have difficulty distinguishing different types of failures and recovering from them
  - Resource managers have a more global view, offers more flexibility for recovery
- EReinit provides a scalable, high-performance FT solution by placing Fault-tolerance primitives in the Resource Manager

Scenario: Process Failure



1. Local Slurm daemon (slurmd) detects process failure
   – SIGCHLD is raised at the parent process when child exits
2. Send failure notification to Slurm controller (srun)
3. srun broadcasts failure notification to slurmds
4. slurmds send predefined signal to MPI processes

Scenario: Node Failure



1. srun detects node failure (no response from slurmd)
2. Find replacement node (preallocated or on-demand)
3. Broadcast notification to slurmds (includes info about replacement node)
4. Launch replacement processes
5. Send signal to processes

Recovering from Failure

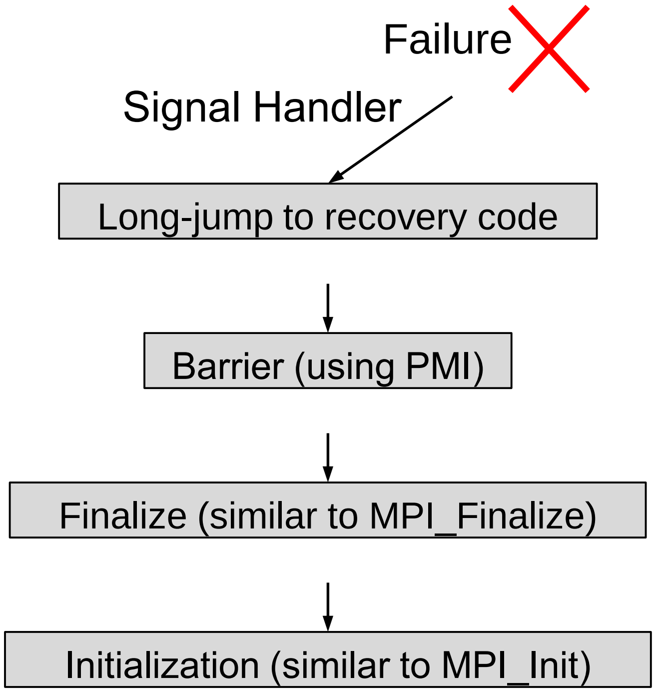
Process Failure

- Local Resource Manager daemon spawns replacement MPI process
- New MPI process determines its state (ADDED) using environment variables
- Fetches the connection information about other processes using PMI (cached at local Slurmd)
- Publishes the new connection information through PMI

Node Failure

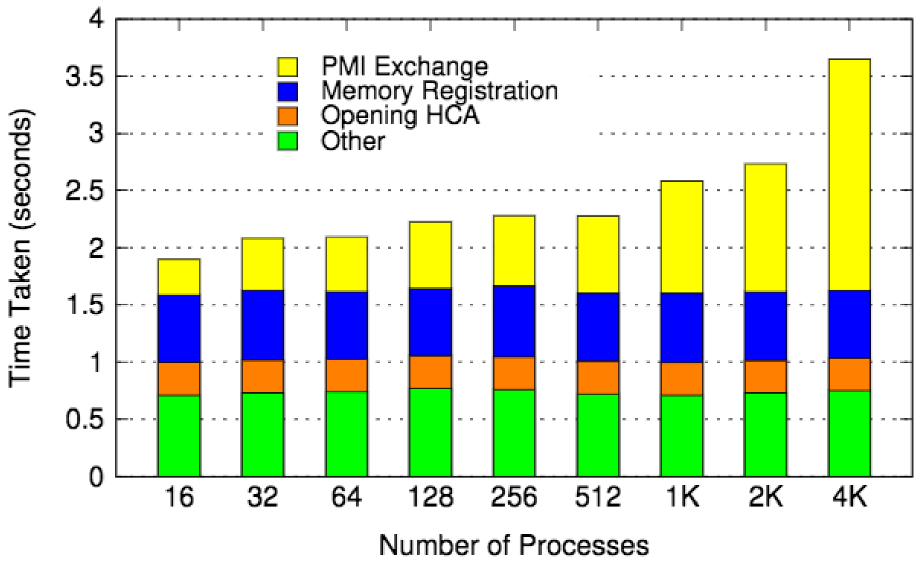
- Controller (srun) allocates replacement node
- Can be pre-allocated, taken from a spare pool, or selected on-demand
- All slurmds notified of replacement
- Surviving slurmds notify local processes
- Slurmd on replacement node spawns new processes
- Recovery similar to process failure

MPI Library Reinitialization



1. Invoke Recovery function registered during init
2. PMI barrier is used to ensure all survivors are ready
   - (Replacement processes need an extra barrier)
3. Internal state is reset during Finalize
4. Initialization is similar to regular MPI_Init

Is Multiple Init + Finalize Good Enough?



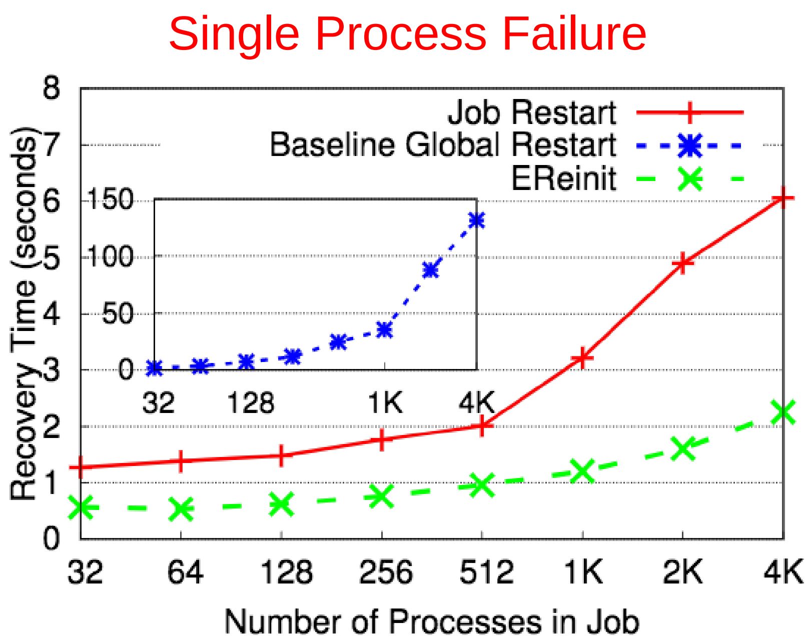
Breakdown of time taken by different steps to initialize MPI in MVAPICH2

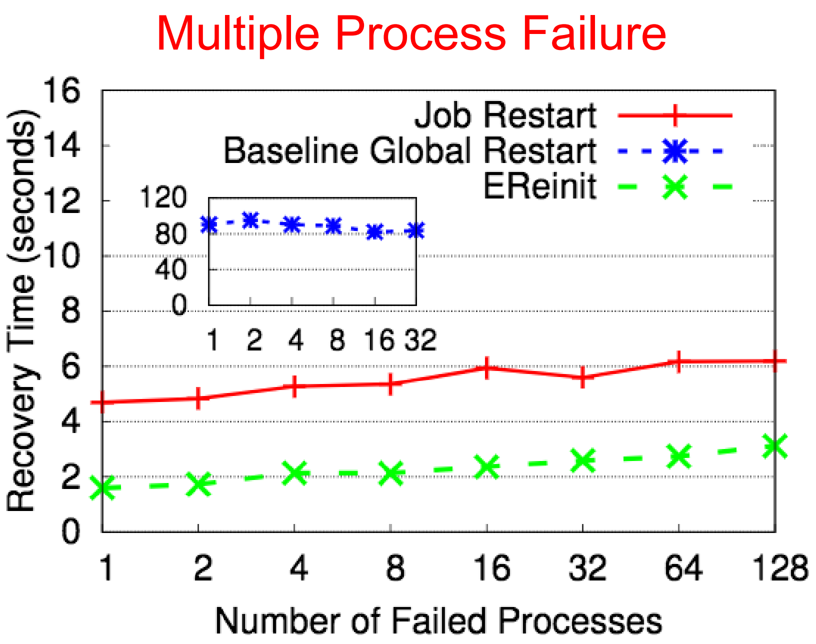
- Reinit allows partial finalization and initialization
- Avoid redundant and expensive steps
  - Close/Reopen HCA
  - Dereigster/Register Memory
- Reduces PMI exchange cost
  - Only replacement processes broadcast new information
  - Fetch information cached by local slurmd

Experimental Setup

- 1,296 node cluster (1,118 compute nodes)
- 2x Intel Xeon E5-2670 CPU (16 cores per node)
- QLogic InfiniBand QDR (40Gbps), Gigabit Ethernet
- MVAPICH2-2.2b, SLURM-15.08.1
- GCC4.9.2, RHEL 6.8, ULFM v1.1, r5433807

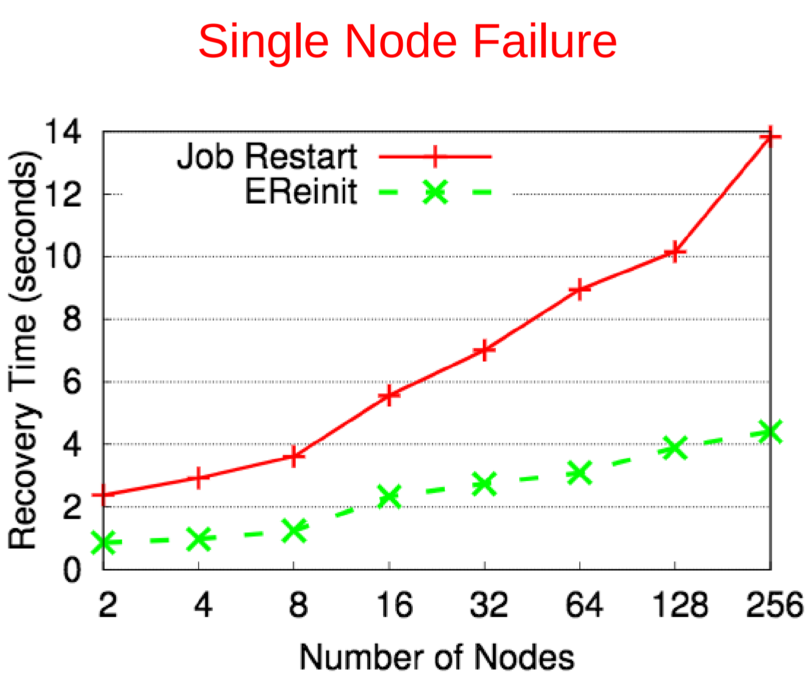
Recovery Time: Process Failure





- EReinit shows good scalability
  - Avoids Shrink/Spawn and reduces PMI exchange cost
  - Recovery time at 4,096 processes (256 nodes) is 2.25 seconds
  - Gracefully handles multi-process failures

Recovery Time: Node Failure

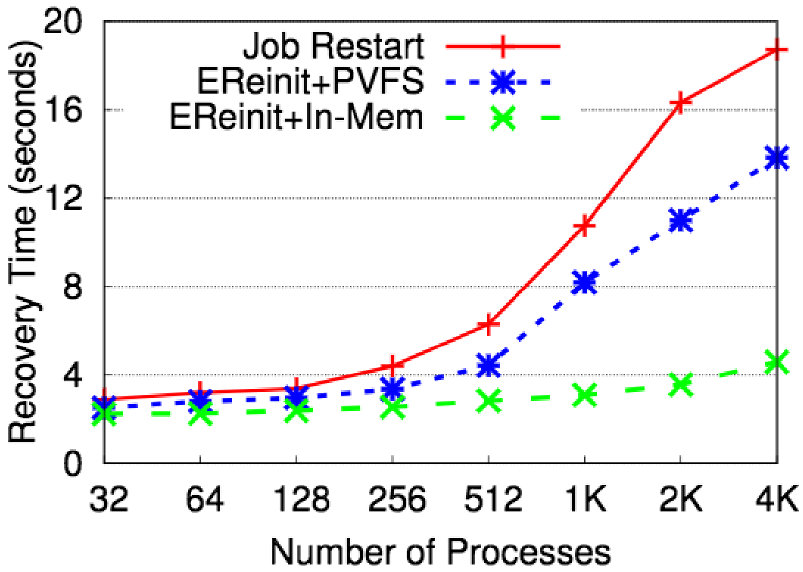


- Node failure is simulated by killing all Application processes and slurm daemons on victim node
- EReinit recovers from single node failure in 4.41 seconds
- 3.14 times faster than basic Job Restart

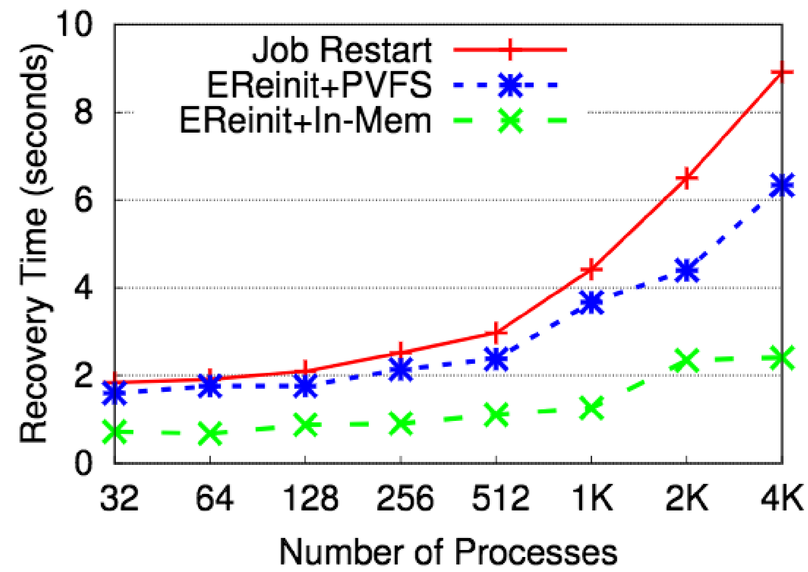
* Publicly available version of ULFM was not able to recover from node failure

Recovery Time: Applications

Enzo



LULESH



- Baseline: Job Restart + Read Checkpoint from Parallel File System
- EReinit + PVFS benefits from faster recovery time (up to 25% faster)
- EReinit enables loading in-memory checkpoints from surviving nodes
- EReinit + In-Mem significantly reduces the load on PVFS (up to 4x faster)

See paper for more application results!

Summary & Future Work

- Global-Restart model can simplify recovery of BSP applications
- EReinit proposes a co-design between Resource Manager and MPI library to achieve better scalability and performance
  - Efficacy of proposed designs demonstrated experimentally
    - Fast recovery at large scale
    - Scalable handling of multiple process failures
  - Enables efficient checkpoint-restart schemes for applications
    - In-memory checkpoints can reduce load on parallel file systems
- More work in progress on efficient checkpointing iterative applications

Questions?

Lawrence Livermore National Laboratory