REINIT: Providing Backward Recovery by MPI Re-initialization

Ignacio Laguna, LLNL
ilaguna@llnl.gov

Contributors:

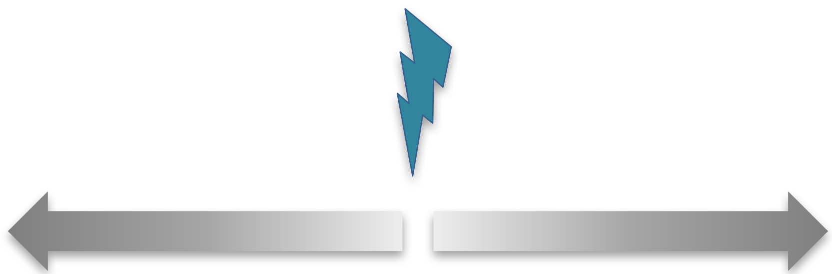
Sourav Chakraborty, Khaled Hamidouche,
Hari Subramoni, Dhabaleswar K. (DK) Panda

Bronis R. de Supinski, Murali Emani, Todd Gamblin, Tanzima Islam,
Kathryn Mohror, Adam Moody, Kento Sato, Martin Schulz

LLNL-PRES-730179
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.

Failure Recovery Models

Failure



Roll-back recovery

- Programmer assumes that MPI state is difficult to repair in a larger/complex application
- Recovery is done from “some” previous state
- Assume that that state is “safe”

Main Users

- Bulk synchronous applications
- Synchronous checkpoint/restart

Proposals

- Reinit
– Fenix (as in SC14 paper)

Roll-forward recovery

- Programmer assumes that broken MPI state can be repaired (e.g., communicators)
- Repair the state (i.e., write code to in the application to repair the state) and move on

Main Users

- Master/slave, task-based applications
- Runtime systems (e.g., Charm++?)

Proposals

- ULFM covers this model very well

Key Concepts of Reinit

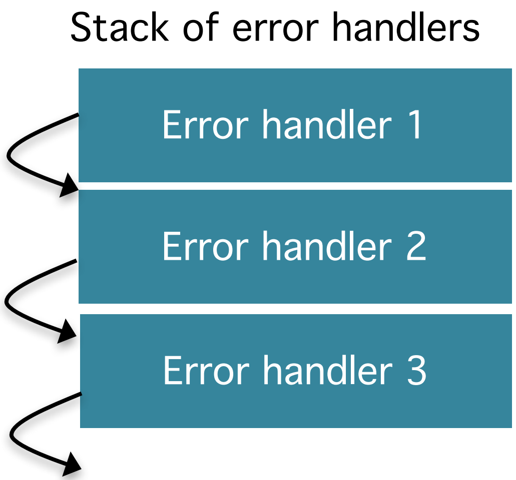
- Automatically restart MPI after a failure
  - Programmers are not involved in failure detection and/or repairing the state of MPI
- The resulting state is the same as the state after MPI_Init returns
  - All communicators (except for MPI_COMM_WORLD), requests, and messages in queue are eliminated
- Since the job is not killed, it allows various optimizations to make failure recovery faster
  - Checkpoints can be loaded from memory
  - Connections of alive processes can be re-used
  - Eliminates job startup time (in some systems it could be high)
- Few changes to the application -> low programing complexity

Description of the Reinit Interface

```c
/* Initialization routines */
typedef enum {
    MPI_START_NEW,          // Fresh process
    MPI_START_RESTARTED,    // Restarted after fault
    MPI_START_ADDED         // Replaced process
} MPI_Start_state;

/* Application entry point */
typedef void (*MPI_Restart_point)
    (int argc, char **argv, MPI_Start_state state);

int MPI_Reinit
    (int argc, char **argv, MPI_Restart_point point);
```

Cleanup Stack Mechanisms

```
/* Cleanup routines */
typedef int (*MPI_Cleanup_handler) (
  MPI_Start_state start,
  void *state);

int MPI_Cleanup_handler_push (
  MPI_Cleanup_handler handler,
  void *state);

int MPI_Cleanup_handler_pop (
  MPI_Cleanup_handler *handler,
  void **state);
```



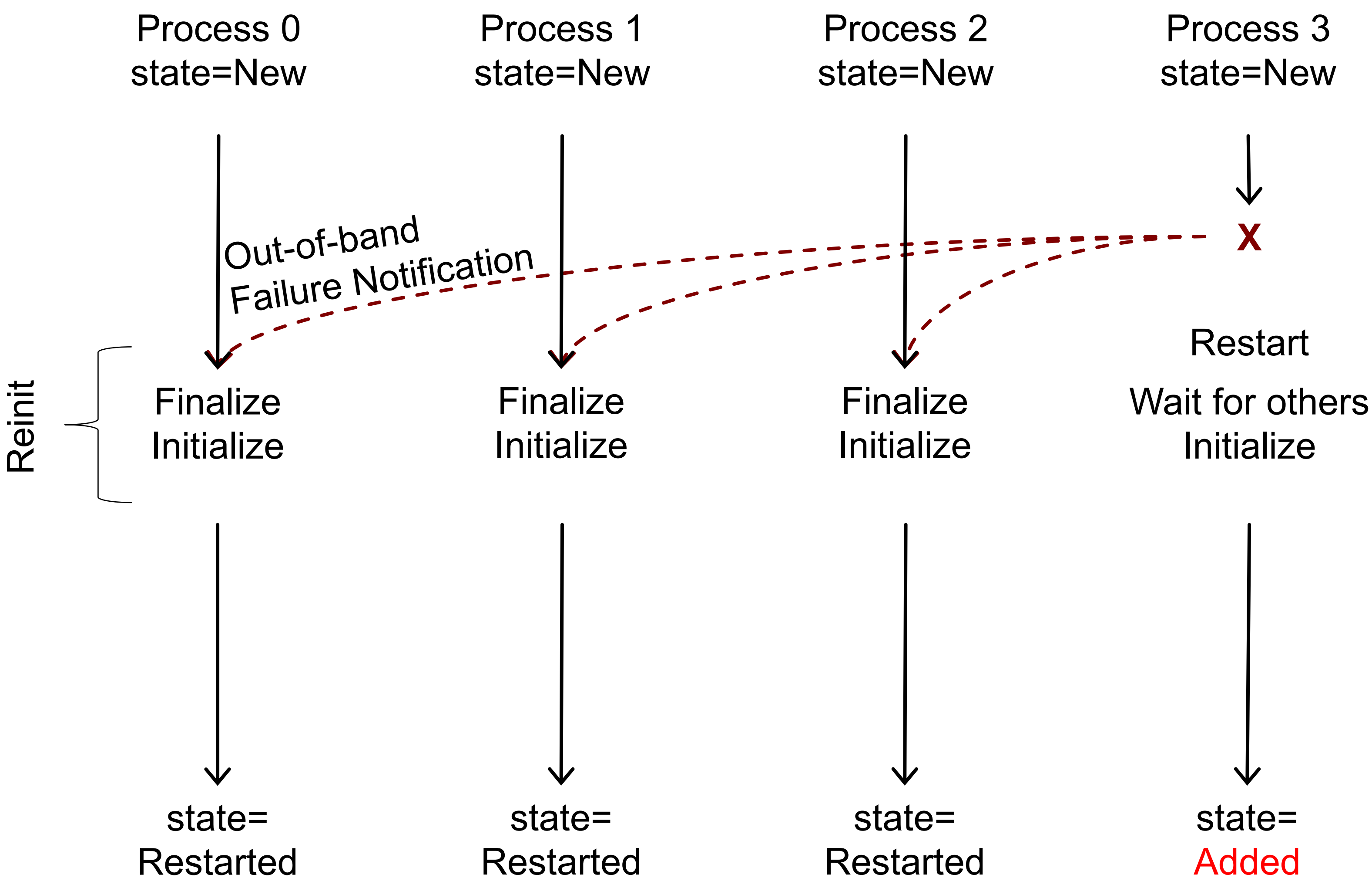
For more details:
https://github.com/tgamblin/mpi-resilience/

Simplified Example Program

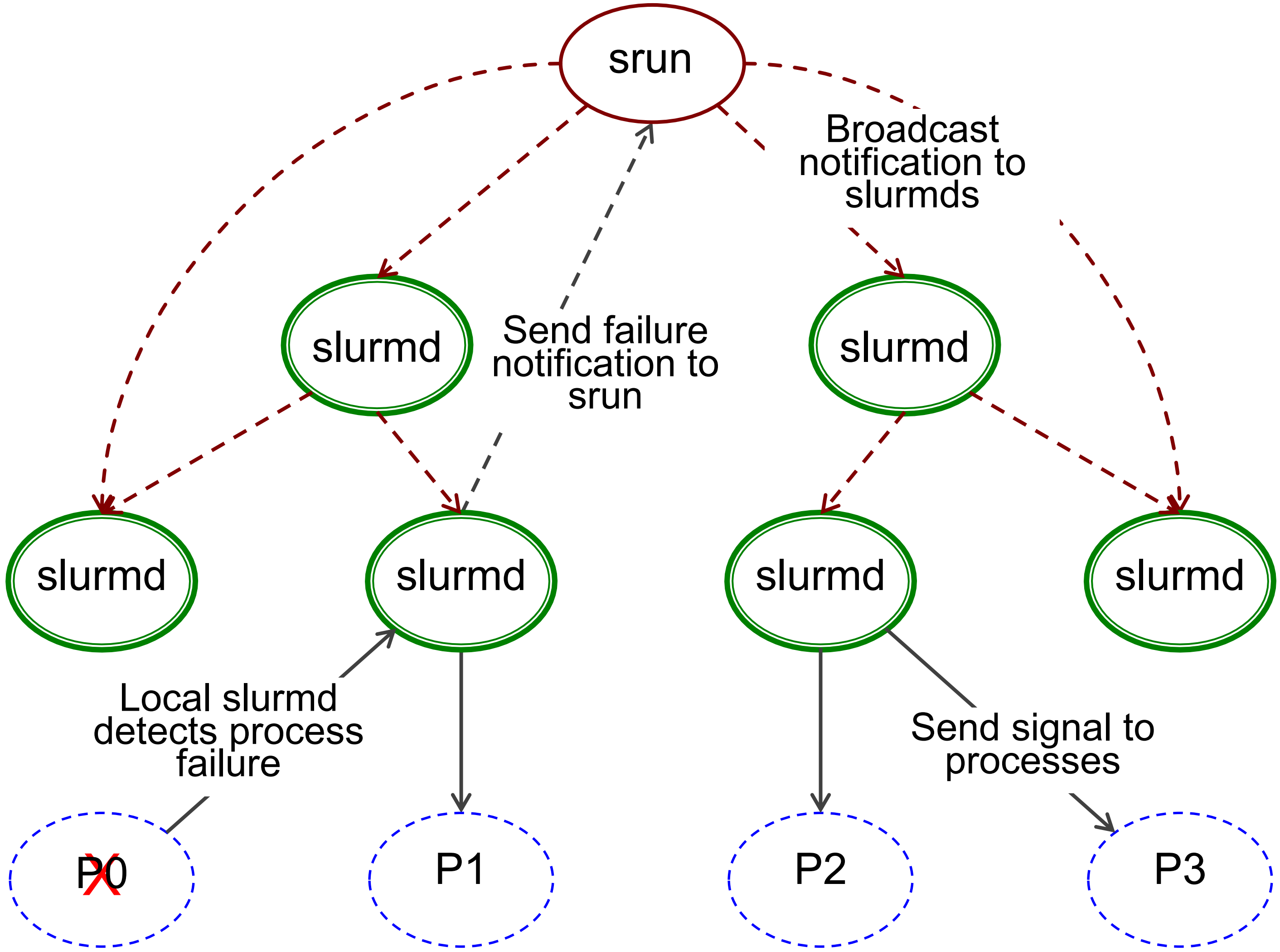
```
int resilient_main (int argc, char **argv,MPI_Start_state start_state)
{
  /* Recover using checkpoint */
  /* Do computation */
  /* Store checkpoint */
}

int main(int argc, char **argv)
{
  MPI_Init(&argc, &argv);

  MPI_Cleanup_handler_push(cleanup_handler);  // Register application cleanup handler

  MPI_Reinit(&argc, &argv, resilient_main);   // Entry point for resilient MPI program

  MPI_Finalize();
}
```

For more details:
https://github.com/tgamblin/mpi-resilience/

Execution Flow of Reinit



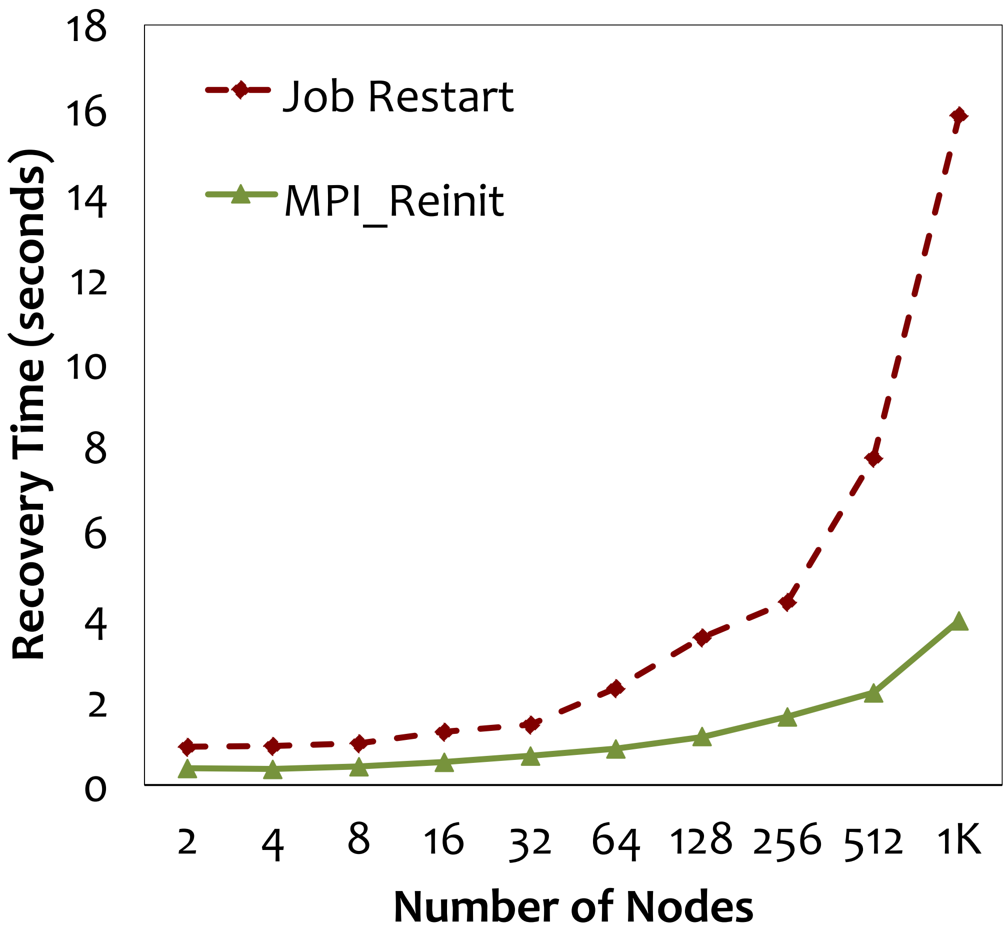
Failure Detection and Notification in SLURM



Experimental Evaluation

- Implementation of Reinit in SLURM-2.6.5 + MVAPICH2-2.1
- System
  - Sierra cluster @ LLNL
  - Intel Xeon 6-core EP X5660
  - 12 Cores per Node
- Single process failure scenario

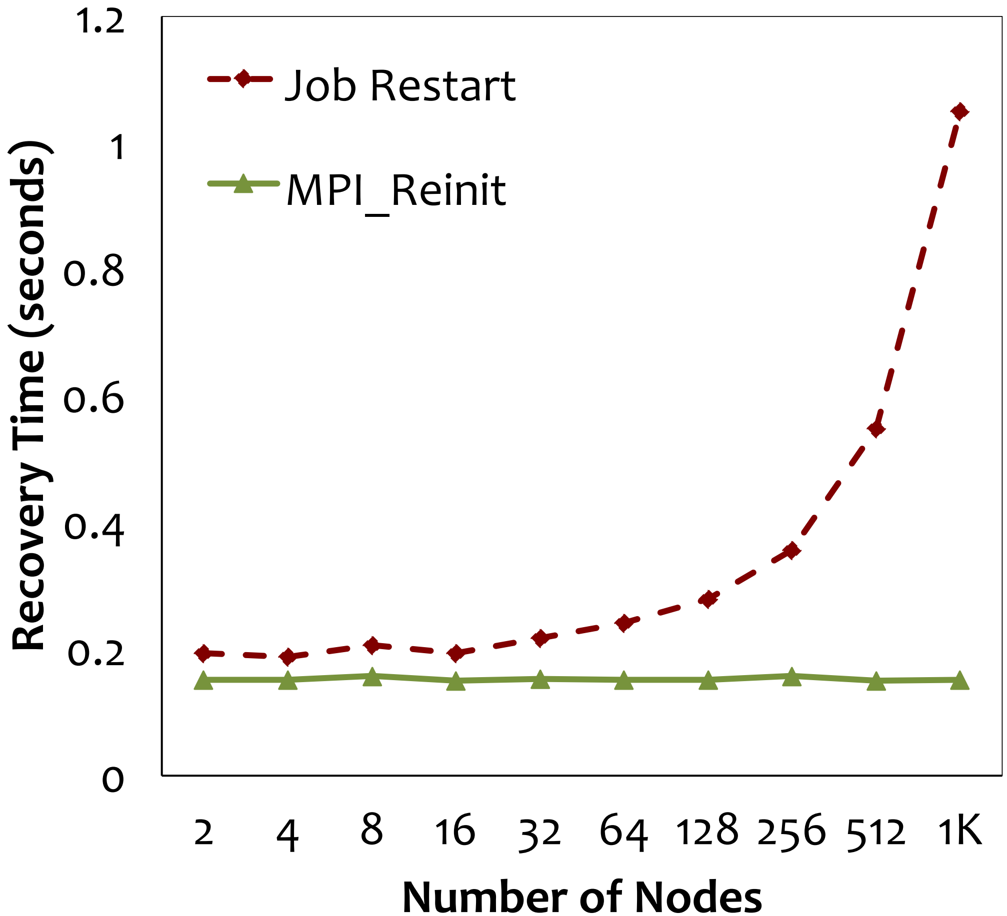
Recovery Time of Reinitializing MPI



Less than 4 seconds to recover with 1K nodes, 12K processes

Recovery with REINIT is 4 times faster than Job restart

Time to Restore a 100 MB Checkpoint



Job restart forces each process to load checkpoints from persistent storage

Only the failed processes need to reload for REINIT

REINIT is 7 times faster than Job restart with 1K nodes, 12K processes