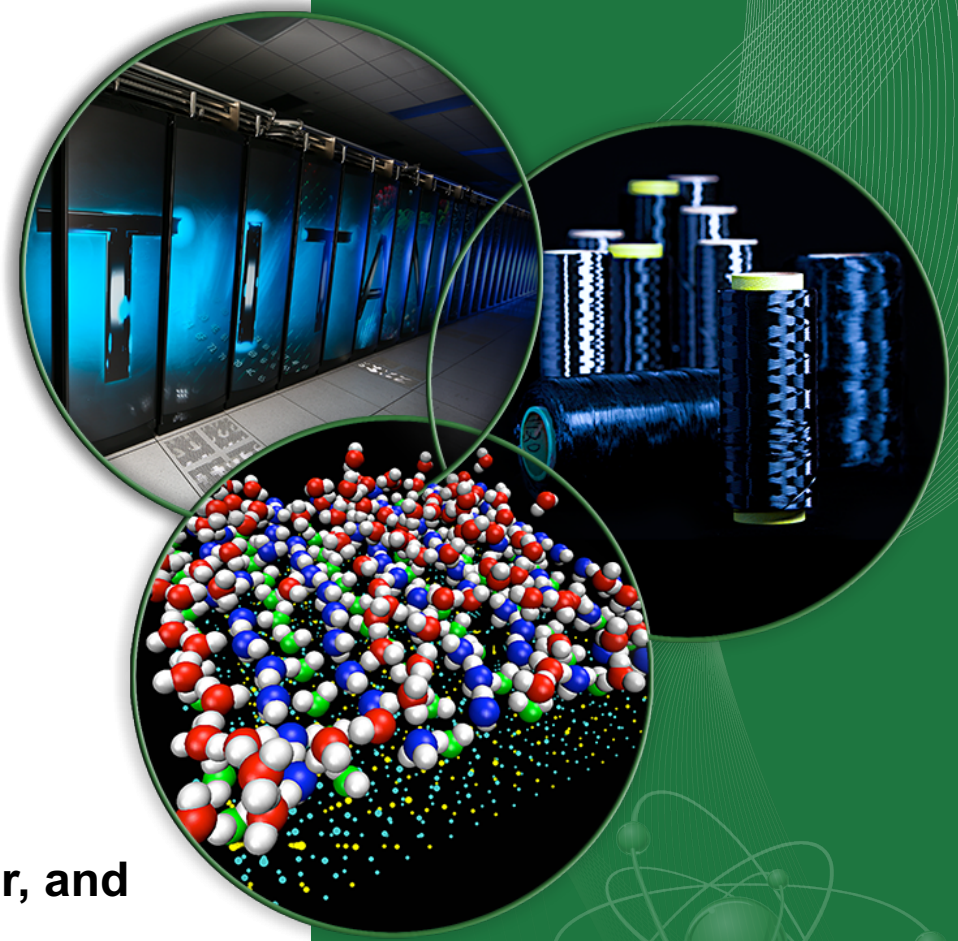
Shrink or Substitute: Handling Process Failures in HPC Systems using In-situ Recovery



PDP 2018, Cambridge, UK

March 21, 2018.

Rizwan A. Ashraf, Saurabh Hukerikar, and Christian Engelmann
Computer Science Research Group,
Computer Science and Mathematics Division,
Oak Ridge National Laboratory (ORNL), USA.

ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory

Why Resilience is Important in HPC?

- ***Resilience*** is the ability to gracefully mitigate the affects of errors and failures, prevalent in large-scale HPC systems.
- Analyses of system-logs from multiple operational large-scale HPC systems show that errors and failures are frequent.
  - Number of complex & diverse, software and hardware components,
  - Technology scaling trends in hardware components,
  - Complex compute, memory, interconnect and storage architectures,
  - Cost (design, area, power, engineering) of achieving error- and failure-free large scale systems is too high.
- Low time between failures affects long running simulations/applications by hindering progress, and causing fatal crashes.
  - Applications are expected to experience multiple failures during execution.
  - Significant wastage of resources (compute-hours, scientific time, power, etc.).
- The situation is only expected to get worse, as we move towards the goal of achieving more computational power, i.e., *exascale systems*.

Process Failures in Parallel Applications

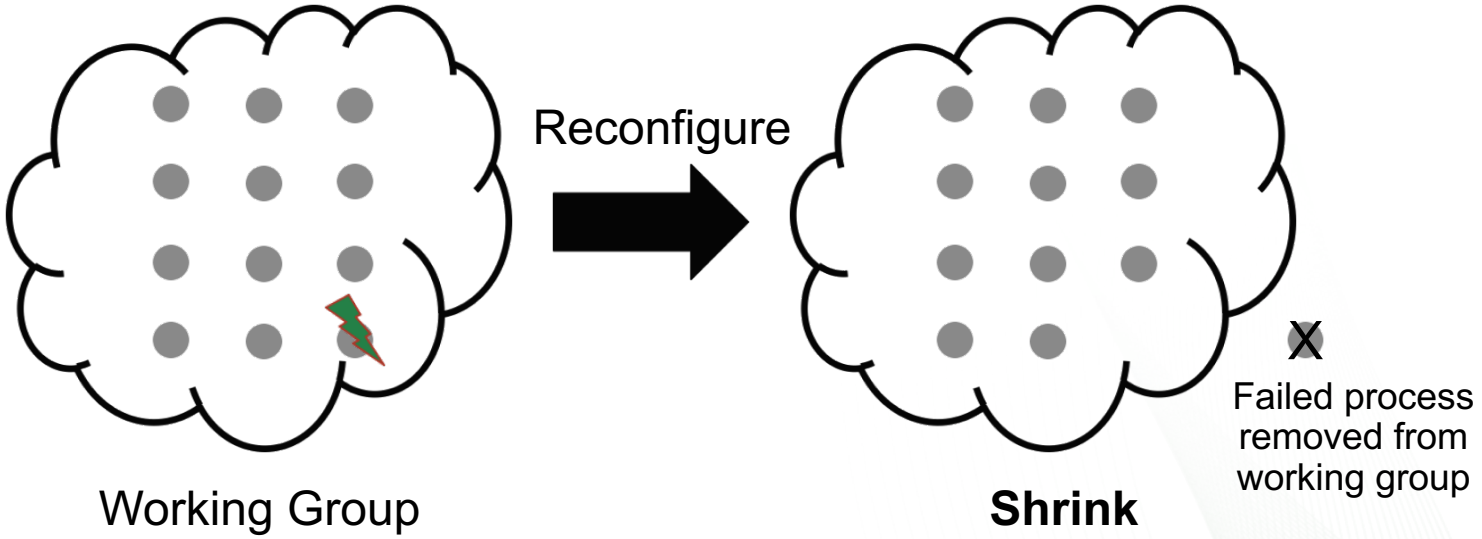
- Multiple causes of fail-stop/process failures, e.g.,
  - permanent faults in hardware (wear-out, device aging, operating conditions),
  - transient faults in hardware (radiation and manufacturing induced faults in memory, processor, interconnect),
  - segmentation or other software faults (I/O error, out-of-memory, failed allocation).
- In parallel distributed applications, failure of even one process causes an abort signal to be sent to all surviving processes.
- Explore software based mechanisms to mitigate and recover from process failures.
  - Hardware based protections are fixed and costly, and are not required by all applications.
- Need *systematic* integration and coordination of techniques from multiple layers of the HPC system to detect, contain and recover from process failures.

Checkpoint/Restart and In-situ Recovery

- In HPC, Checkpoint/Restart (C/R) is predominantly used to recover from process failures.
  - Checkpoints are performed at regular intervals,
  - When failure occurs, the job is re-scheduled from previous stable checkpoint.
- Checkpoints performed at system-level are intractable, especially as we move towards exascale systems.
- Conventional C/R re-schedules the job after failure, resulting in allocation of new resources, causing lost scientific productivity.
  - A job can be allocated on the faulty resource, again!
- How can User Level Failure Mitigation (ULFM) proposal in Message Passing Interface (MPI) standard help?
  - Detects process failures and excludes faulty processes.
  - Repairs communication objects to allow user to continue execution and recover state.
  - Eliminates the need to re-schedule job.
  - Provides the ability to do ***in-situ recovery***.

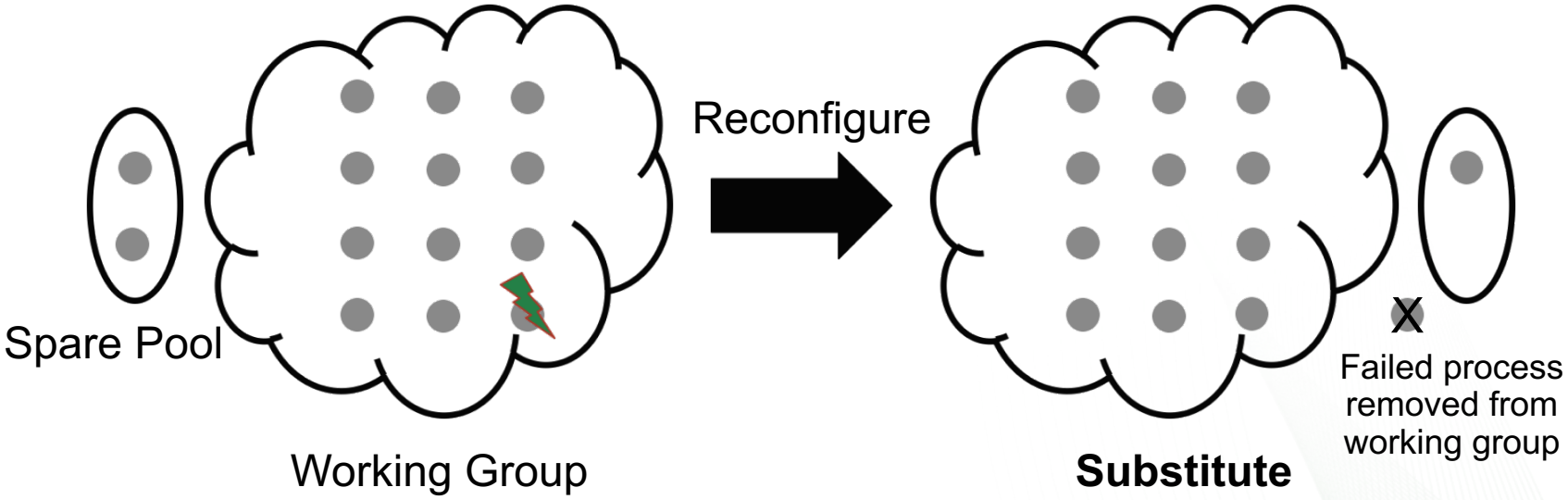
Shrink or Substitute (1/2)

- ULFM does not provide a concrete mitigation strategy, nor recovers application state.
  - User decides since every application is different.
  - Two distinct options to continue forward progress, after a process failure.



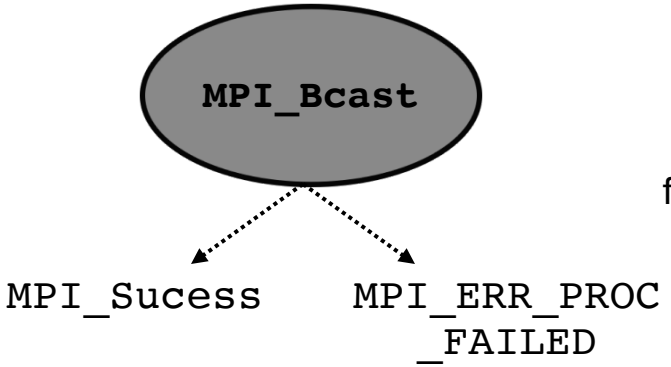
Shrink or Substitute (2/2)

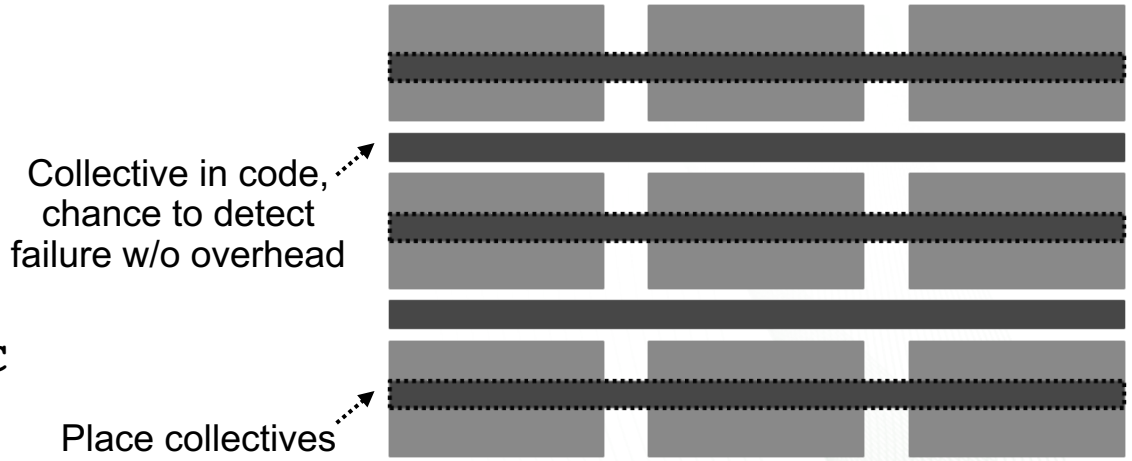
- ULFM does not provide a concrete mitigation strategy, nor recovers application state.
  - User decides since every application is different.
  - Two distinct options to continue forward progress, after a process failure.



Synopsis: Process Failure Detection

- ULFM extends MPI routines to report any anomalies in a group of communicating processes.
  - Utilizes consensus and background liveliness checks to detect failures.
- Notification and propagation of failures to all other surviving processes, e.g., by revoking communicators once failure occurs.
- Fix communication objects by removing dead processes, e.g., using `MPI_COMM_SHRINK()`.

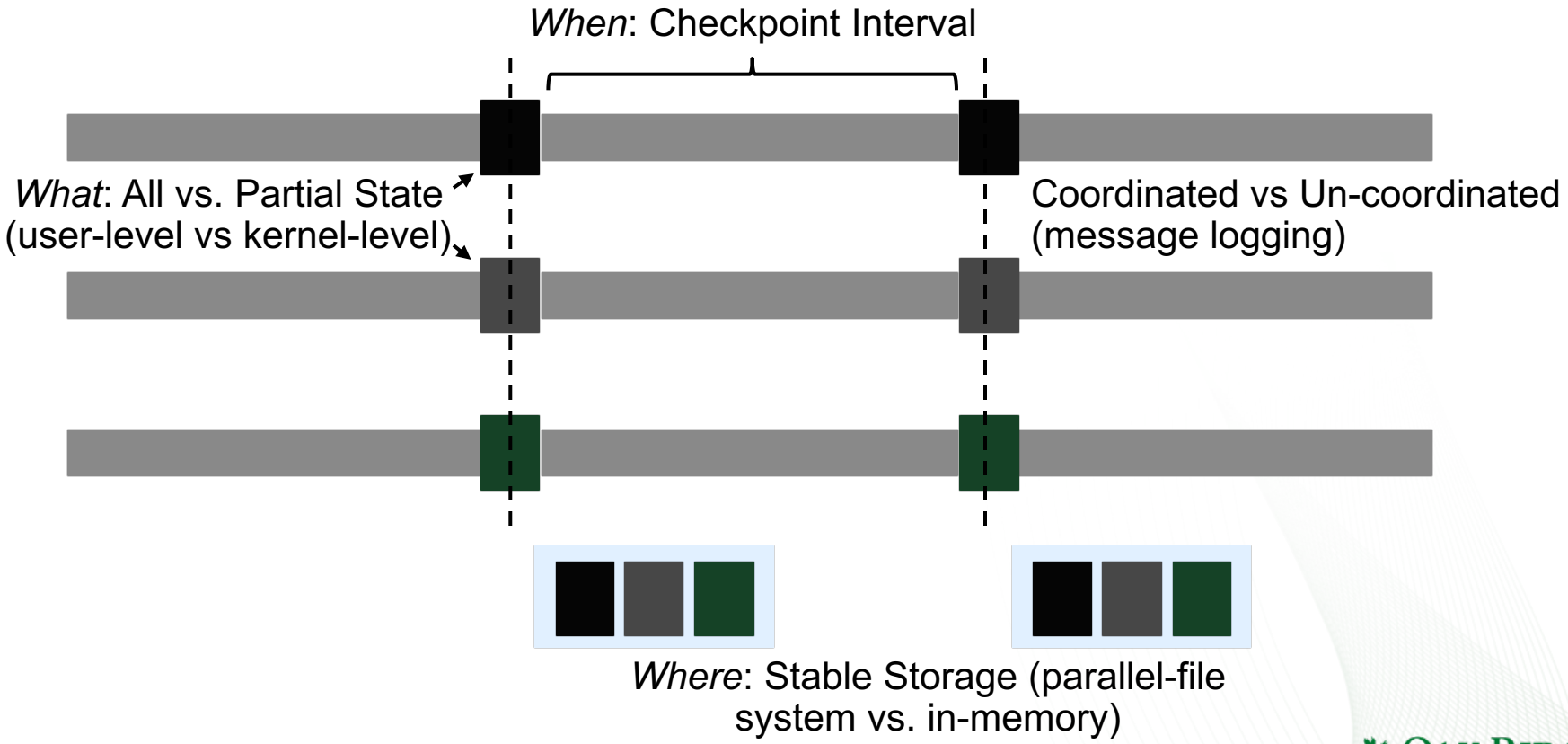




Reactive vs. Proactive Failure Detection

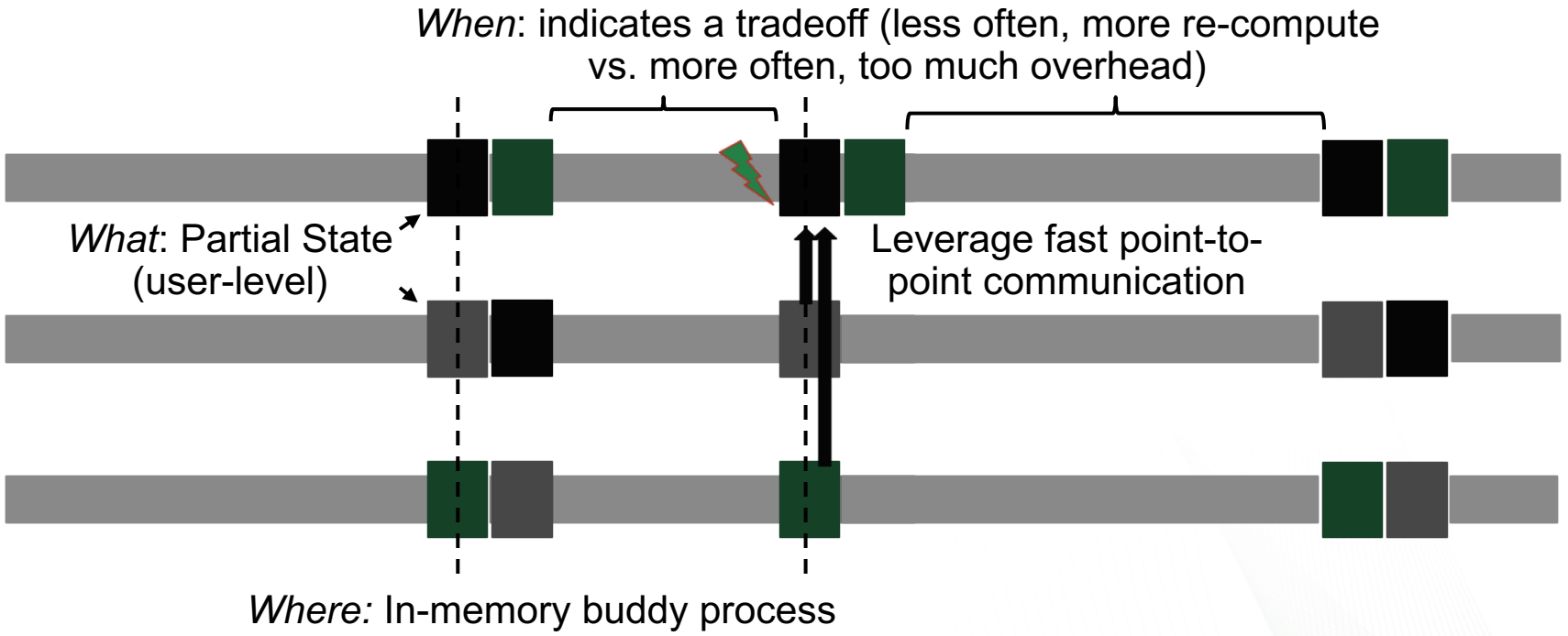
Synopsis: Checkpointing (1/2)

- Checkpoints: facilitate recovery of lost state.
  - Replicate static and dynamic state distributed across parallel processes.
  - *What, when and where?*



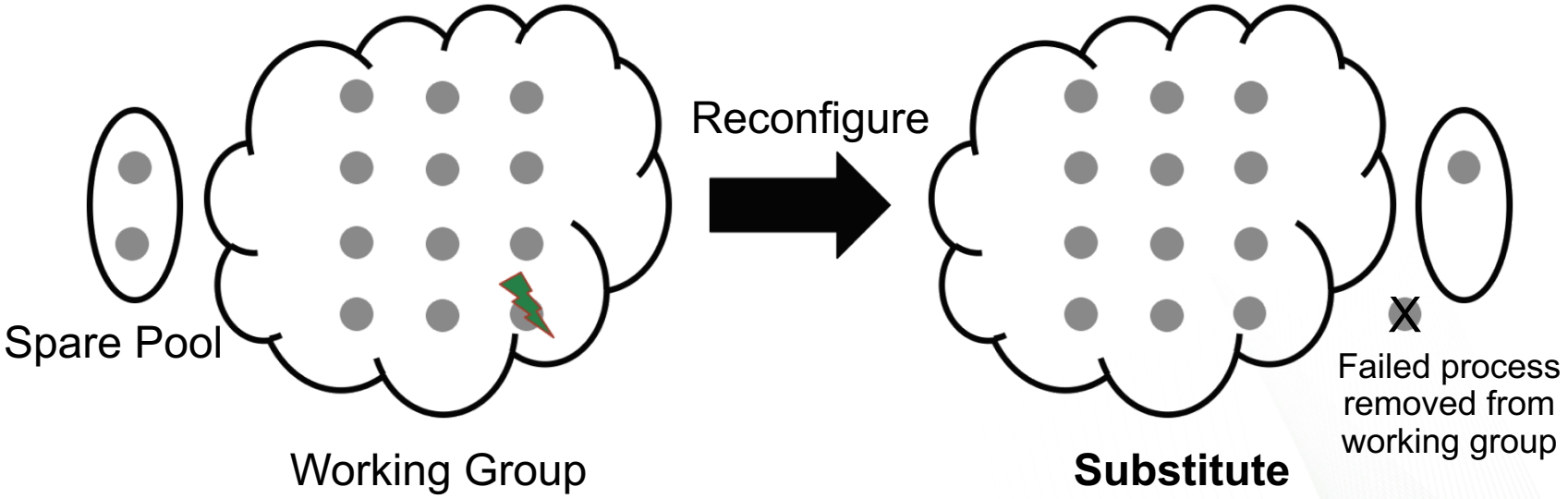
Synopsis: Checkpointing (2/2)

- Checkpoints: facilitate recovery of lost state.
  - Replicate static and dynamic state distributed across parallel processes.
  - *What, when and where?*



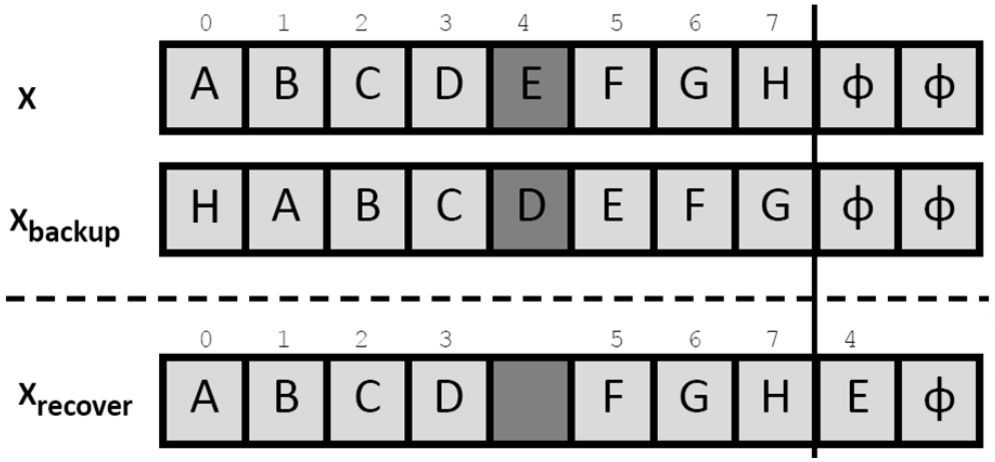
Goal: *reduce* waste = re-compute + time to perform checkpoint

Substitute



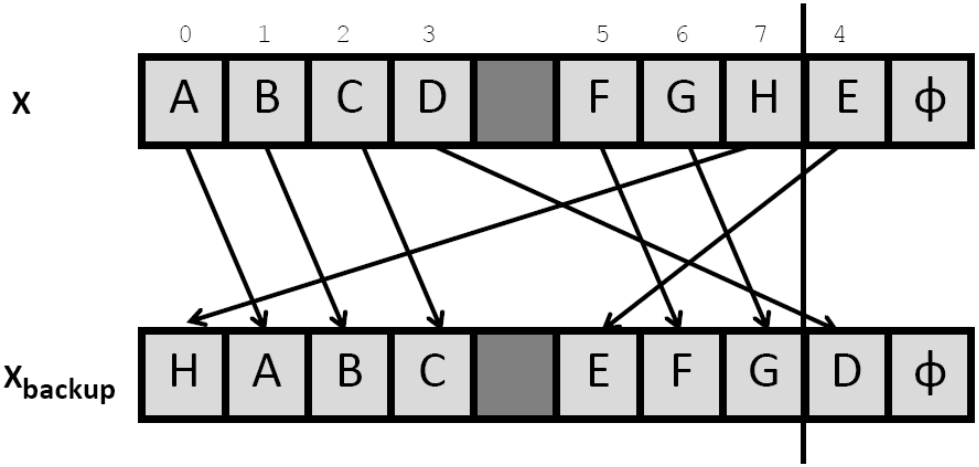
Substitute: Supplemental Computation with Spares (1/2)

- Use pool of spare processes to replenish lost processes.
  - **Warm/Cold Spares:** Design-time vs. runtime spawned spare processes. *Employ as need arises.* Cold is intractable since some environments prohibit runtime allocation.
  - **Hot Spares:** Redundant computation (every process has an active spare), resource heavy. Sudden degradation in reliability once failures start to occur.
- **Advantages**: No workload re-distribution. Fulfills problem decomposition restrictions, e.g., `LULESH` needs cubic number of processes.
- **Disadvantages**: Non-utilization of resources; Determination of spares; Programming effort; Disruption of communication pattern.

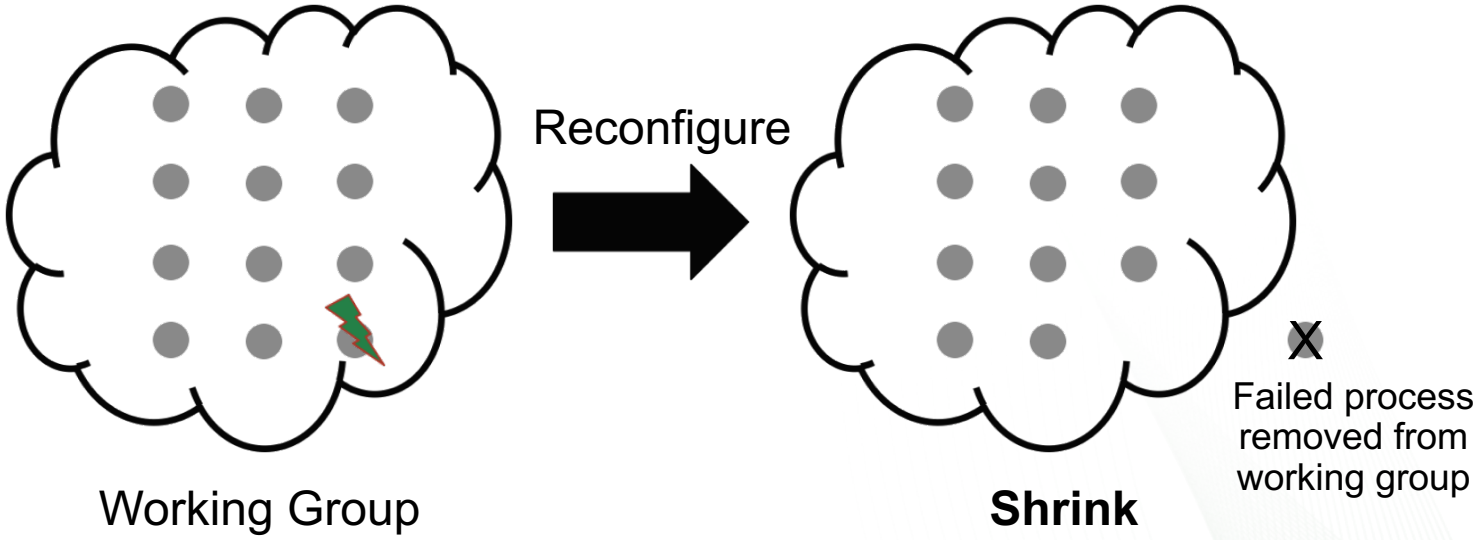


Substitute: Supplemental Computation with Spares (2/2)

- Use pool of spare processes to replenish lost processes.
  - **Warm/Cold Spares:** Design-time vs. runtime spawned spare processes. *Employ as need arises.* Cold is intractable since some environments prohibit runtime allocation.
  - **Hot Spares:** Redundant computation (every process has an active spare), resource heavy. Sudden degradation in reliability once failures start to occur.
- **Advantages**: No workload re-distribution. Fulfills problem decomposition restrictions, e.g., `LULESH` needs cubic number of processes.
- **Disadvantages**: Non-utilization of resources; Determination of spares; Programming effort; Disruption of communication pattern.

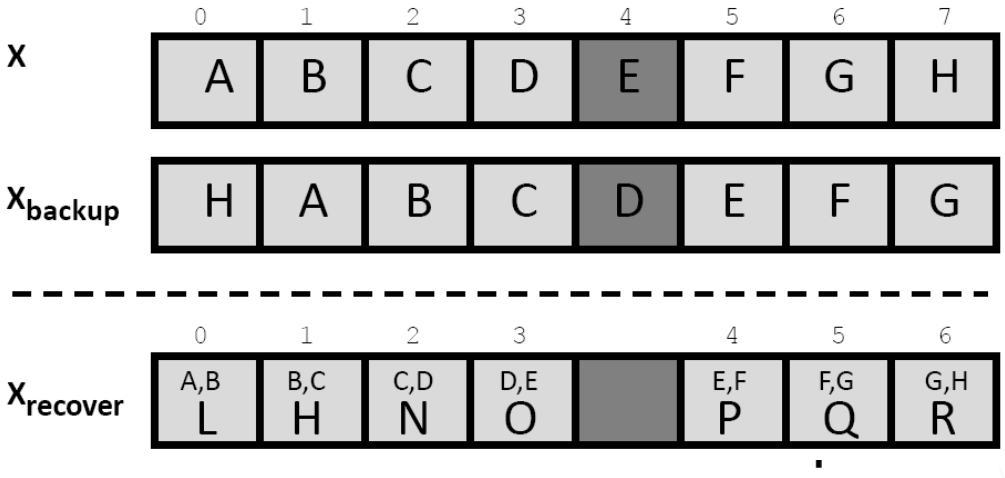


Shrink


Reconfigure
X
Failed process removed from working group
Working Group
Shrink

Shrink: Graceful Degradation with Survivors

- Continue execution with surviving processes.
  - Need to dynamically adjust workload after a failure.
  - All processes do useful work throughout.
- **Advantages**: Utilizes all resources; Maintain communication pattern.
- **Disadvantages**: Mandatory workload re-distribution. Application dependent effort; Recovery involves inter-process communication; Workload increases per process with increase in failures mitigated.



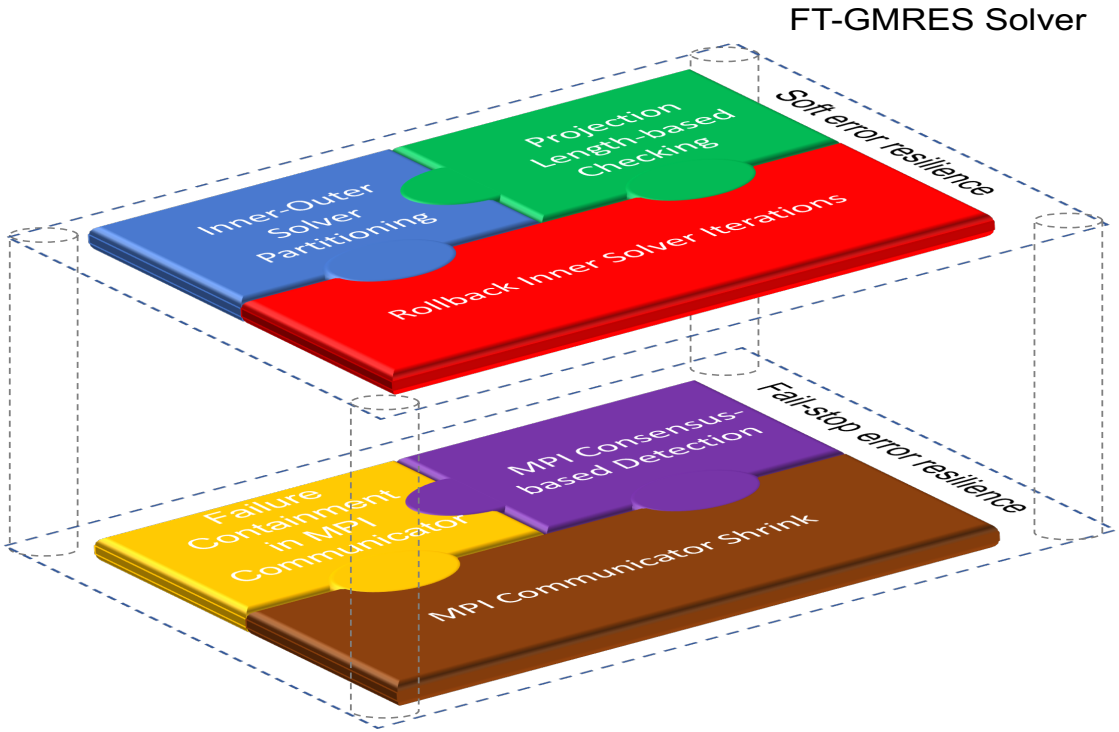
Synopsis: *Shrink* or *Substitute*

Substitute	Shrink
Allocate spares at design-time (depending on MTTF, expected execution time, etc.)	N/A
Spares do not do **useful work** until employed (resources wasted if failure-free execution)	All processes do useful work from the beginning
Maintains original **workload** on all survivors	Balanced workload re-distribution required to fully utilize resources and optimal performance
Employing spares disrupts regular **communication pattern** of application	Communication pattern disruption is less severe or may not occur (depends on network topology)
Checkpoint size as per design, does not increase with failures	Checkpoint size increases with number of failures (may run out of resources, memory checkpoints)
Only couple of communication channels required during **state recovery,** survivors use local state	All processes may need to talk to each other during state recovery
Size of spare pool determines **# of process failures** which can be mitigated	Arbitrary # of failures can be mitigated until computation is too expensive (or out-resourced)

Use Case: Linear Solver

- GMRES minimal residual method for solving non-symmetric linear systems.
  - Solve: `Ax = b`
  - Iterative algorithm

- Static State: Matrix `A`, vector `b`.
- Dynamic State: solution vector `x` (checkpoint periodically)



Implementation and Experimental Setup

- FT-GMRES implemented using Trilinos 12.6.4 framework, https://trilinos.org/.
  - Tpetra package for parallel linear algebra using MPI.
- Parallel Environment: ULFM release 1.1, based on Open MPI 1.7.1 http://fault-tolerance.org/.
- Our contributions to FT-GMRES include:
  - **Utilization of MPI-ULFM** instead of standard MPI.
  - **Support for in-memory checkpoints** of Tpetra objects (matrices, vectors).
  - **Dynamic workload re-distribution** of Tpetra objects such as matrices and vectors in case of shrink strategy.
  - Use of **warm spares** for recovery in case of substitute strategy.
- Test problem: Discretization of 3D mesh. Sparse Matrix with about 7 million rows and 186 Million non-zeros.
- 40-node Linux cluster with AMD Opteron processors.
  - Cores/node: 24 (Total: 960 cores); Memory/node: 64 GB; Point-to-point bandwidth: 215 MB/s.

Goals of Experiments

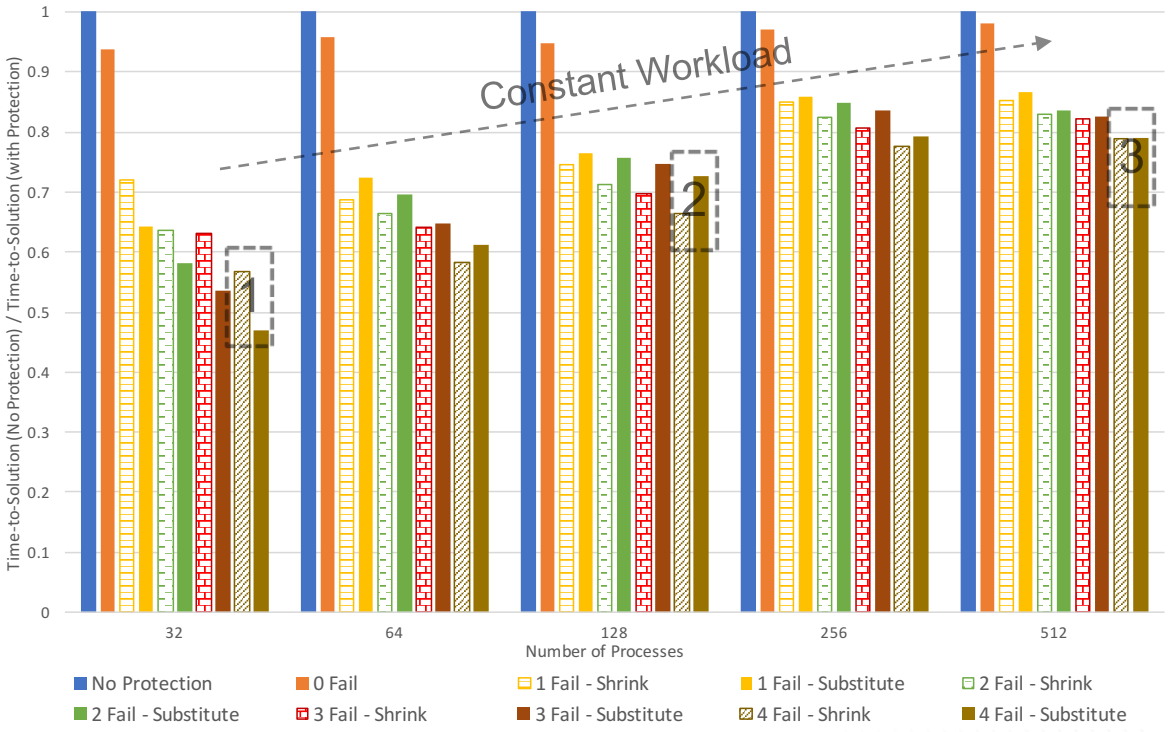
- **Goal # 1:** Evaluate overall performance impact of employing substitute vs. shrink approach for in-situ recovery.
- **Goal # 2:** Compare checkpoint and state recovery overheads of substitute and shrink approaches.
- **Goal # 3:** Evaluate the effect on performance, checkpoint and state recovery overhead of increasing number of process failures.

Results – Straight-up Comparison

- Up to four independent process failures are simulated using `SIGKILL`.
- Pre-selected processes for termination and fixed injection time window gives results with low standard deviation, e.g., coefficient of variation b/w 0.01 & 0.15.
- Mapping of spare processes is fixed and failed process is always on a different physical node.

Higher is better (less overheads)

1. Overheads of substitute approach are higher at small-scale

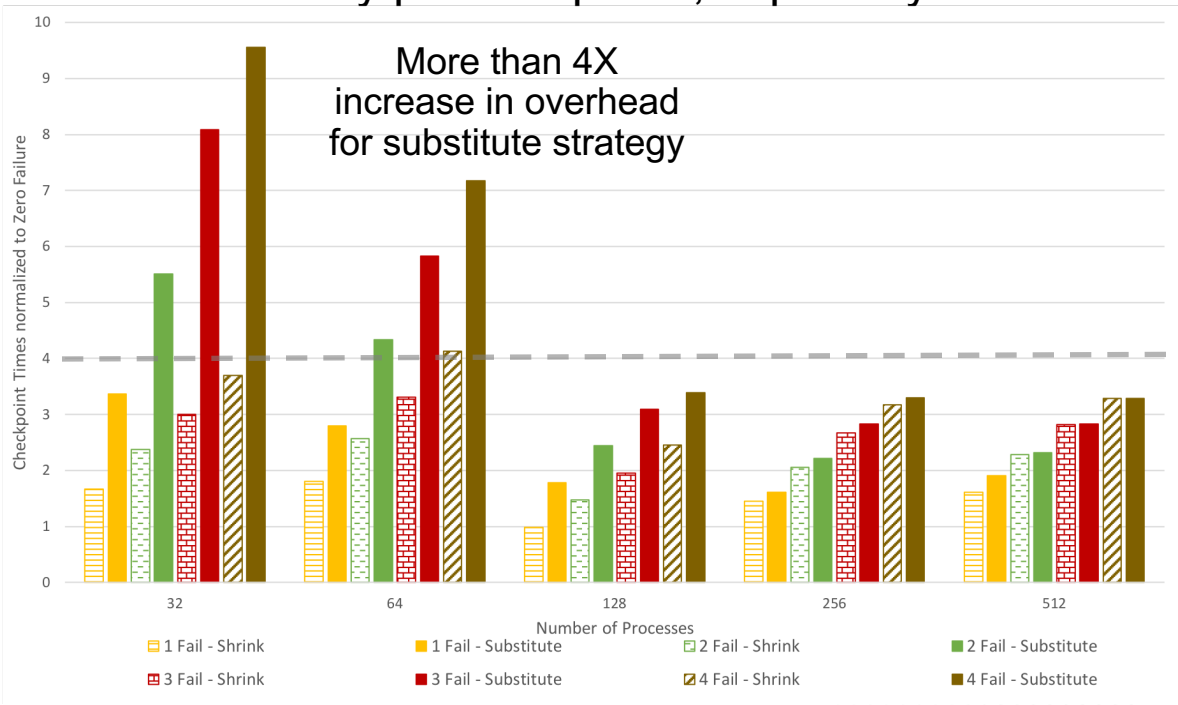


3. **Graceful Degradation** for shrink approach, when abundant workers present

2. Use of spares pays off at certain scales, workload dependent

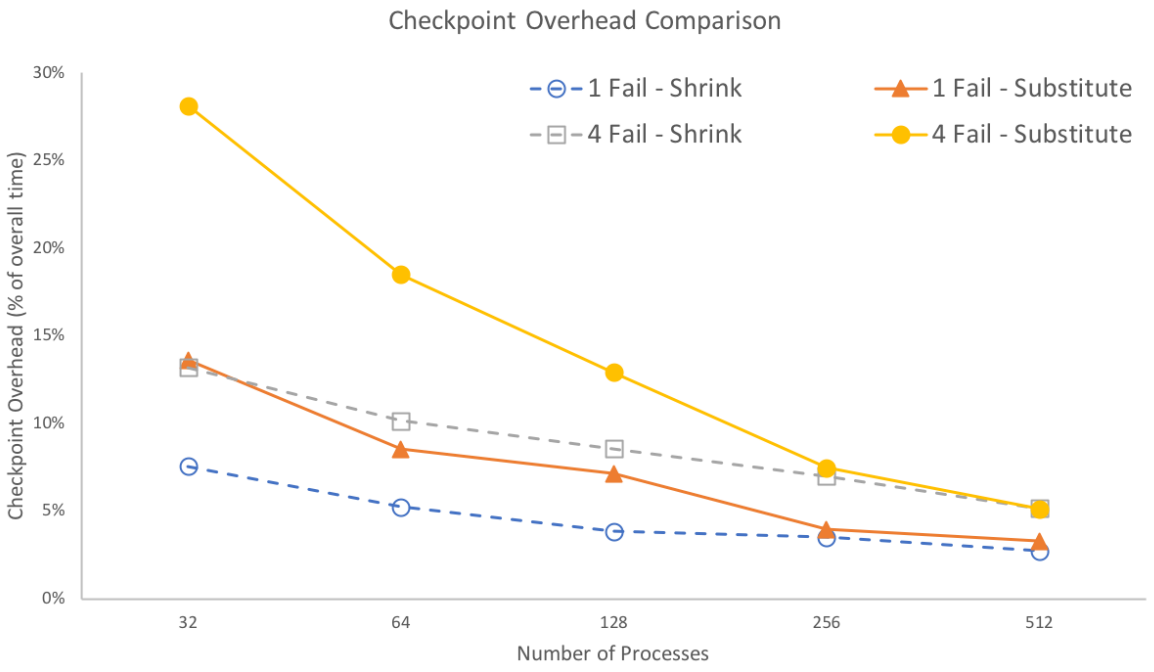
Results – Checkpoint overhead (1/2)

- Shrink strategy: A multiplicative linear increase in checkpoint overheads with increase in number of failures.
- Checkpoint overheads for substitute strategy do not increase multiplicatively with number of failures.
- Higher overheads are observed when using substitute strategy due to communication with statically placed spares, especially at small scales.



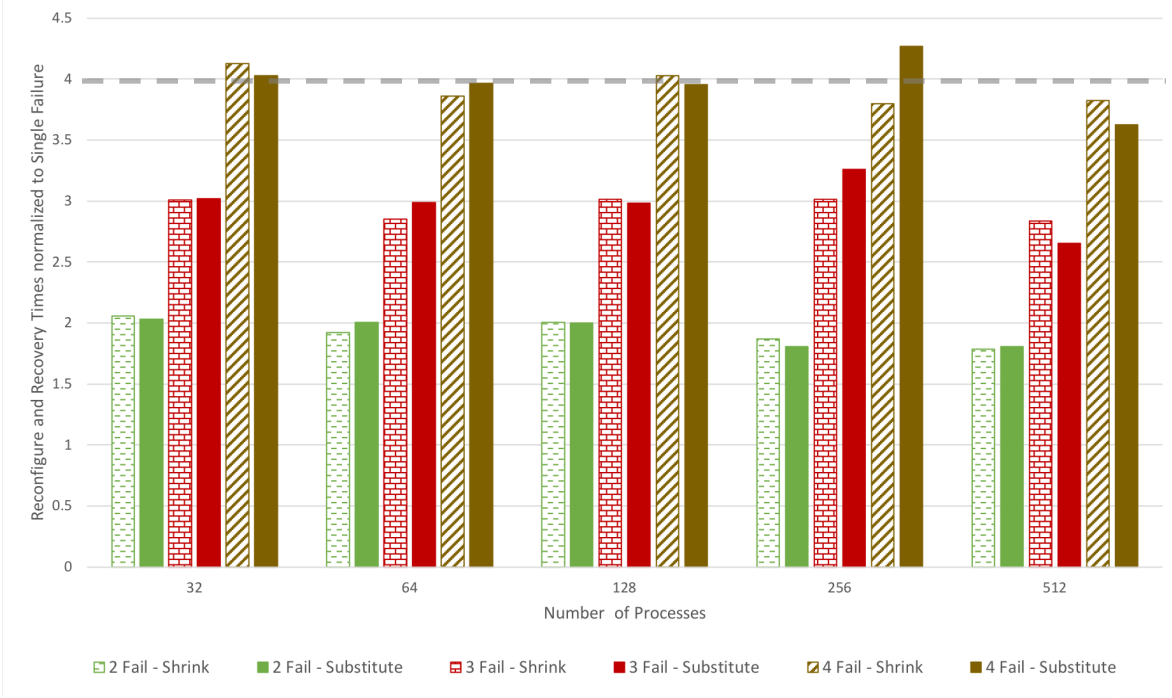
Results – Checkpoint overhead (2/2)

- Shrink strategy: A multiplicative linear increase in checkpoint overheads with increase in number of failures.
- Checkpoint overheads for substitute strategy do not increase multiplicatively with number of failures.
- Higher overheads are observed when using substitute strategy due to communication with statically placed spares, especially at small scales.



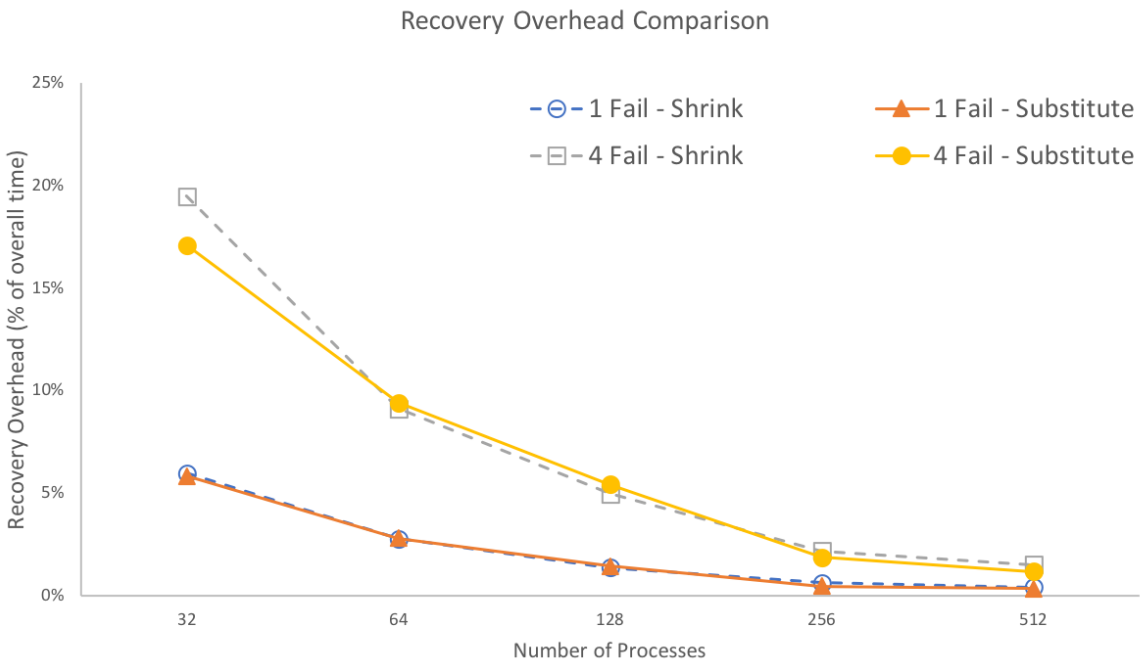
Results – Recovery overhead (1/2)

- A linear increase in recovery overheads with increase in number of failures is observed for both shrink and substitute approaches.
- Recovery overheads are comparable when using substitute and shrink strategies despite the complexity associated with workload re-distribution (shrink strategy).
- Reconfiguration overheads are observed to be negligible, e.g., they range b/w 0.01% and 0.05%.



Results – Recovery overhead (2/2)

- A linear increase in recovery overheads with increase in number of failures is observed for both shrink and substitute approaches.
- Recovery overheads are comparable when using substitute and shrink strategies despite the complexity associated with workload re-distribution (shrink strategy).
- Reconfiguration overheads are observed to be negligible, e.g., they range b/w 0.01% and 0.05%.



Conclusions

- Two alternative in-situ process failure recovery strategies are evaluated for a linear solver application.
- The shrink strategy demonstrated *graceful performance degradation* given abundant survivors exist to share the workload of lost processes.
- The performance advantage of having spares to substitute lost processes is diminished due to *static mapping of warm spares*, especially at small-scales. We expect this effect to hinder performance of other parallel operations as well.
- *Performance estimations* for shrink strategy to mitigate multiple process failures can be obtained from single failure experiments. Not possible for substitute strategy due to un-expected behavior of spare mapping (depends on failure location and network topo).
- *No winner*, depends on what you are doing!