Resilience Strategies for Future Systems

The Salishan Conference on High Speed Computing

April 24, 2012

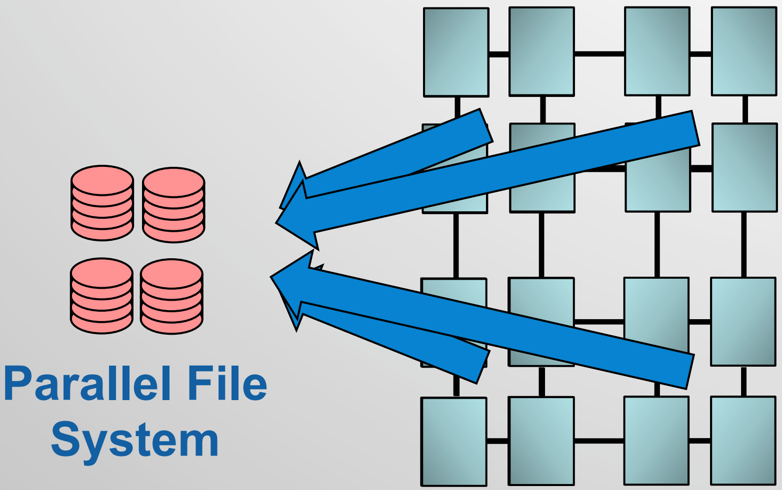
Bronis R. de Supinski

Lawrence Livermore National Laboratory

LLNL-PRES-549691

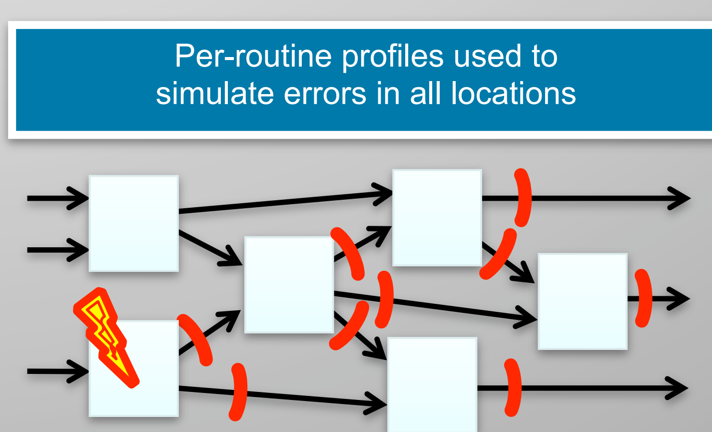
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

We are developing a comprehensive strategy for application resilience on large-scale systems

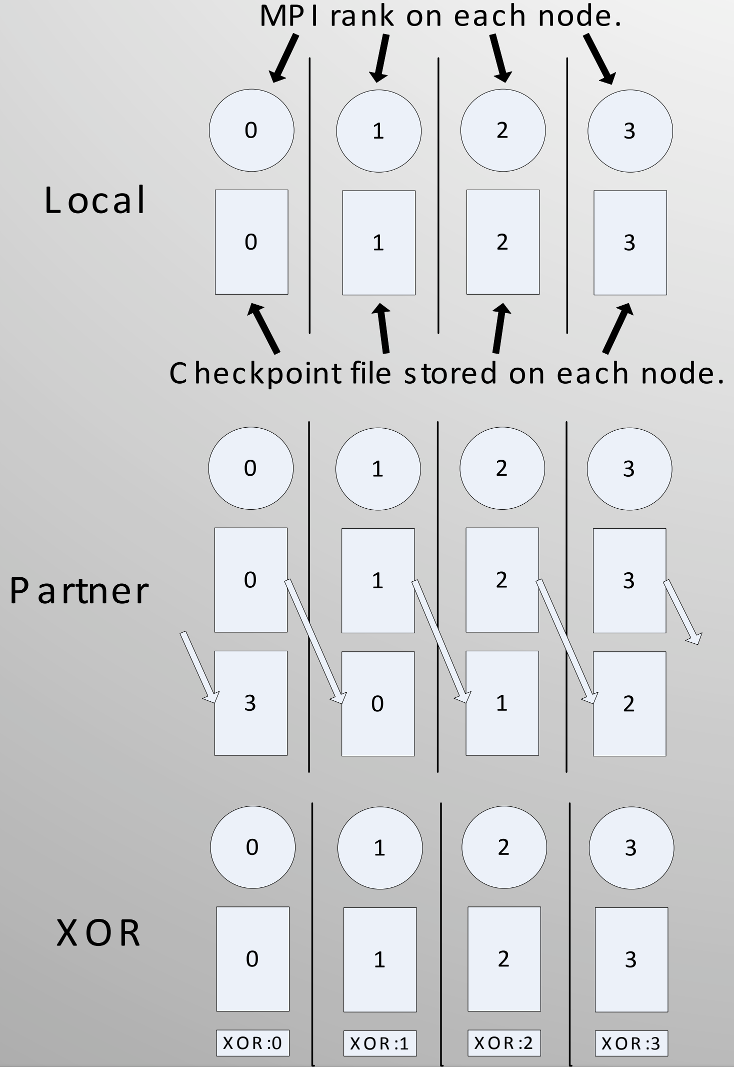


- Checkpoint/restart too slow even on current large-scale systems
- Reduce current checkpoint times
  - In-memory techniques
  - Compression of existing checkpoints
  - Compression across tasks
- Coordinate checkpoints across task subsets
  - Limit scope of restart

- Algorithmic-based fault tolerance (ABFT)
  - Application fault vulnerability models
    - Fault injection studies
    - Compiler assisted analysis
    - Target use of expensive solutions
  - Solvers that detect and correct errors
  - Targeted techniques for specific applications
  - Statistical error detection techniques
  - Automation of resilience transformations

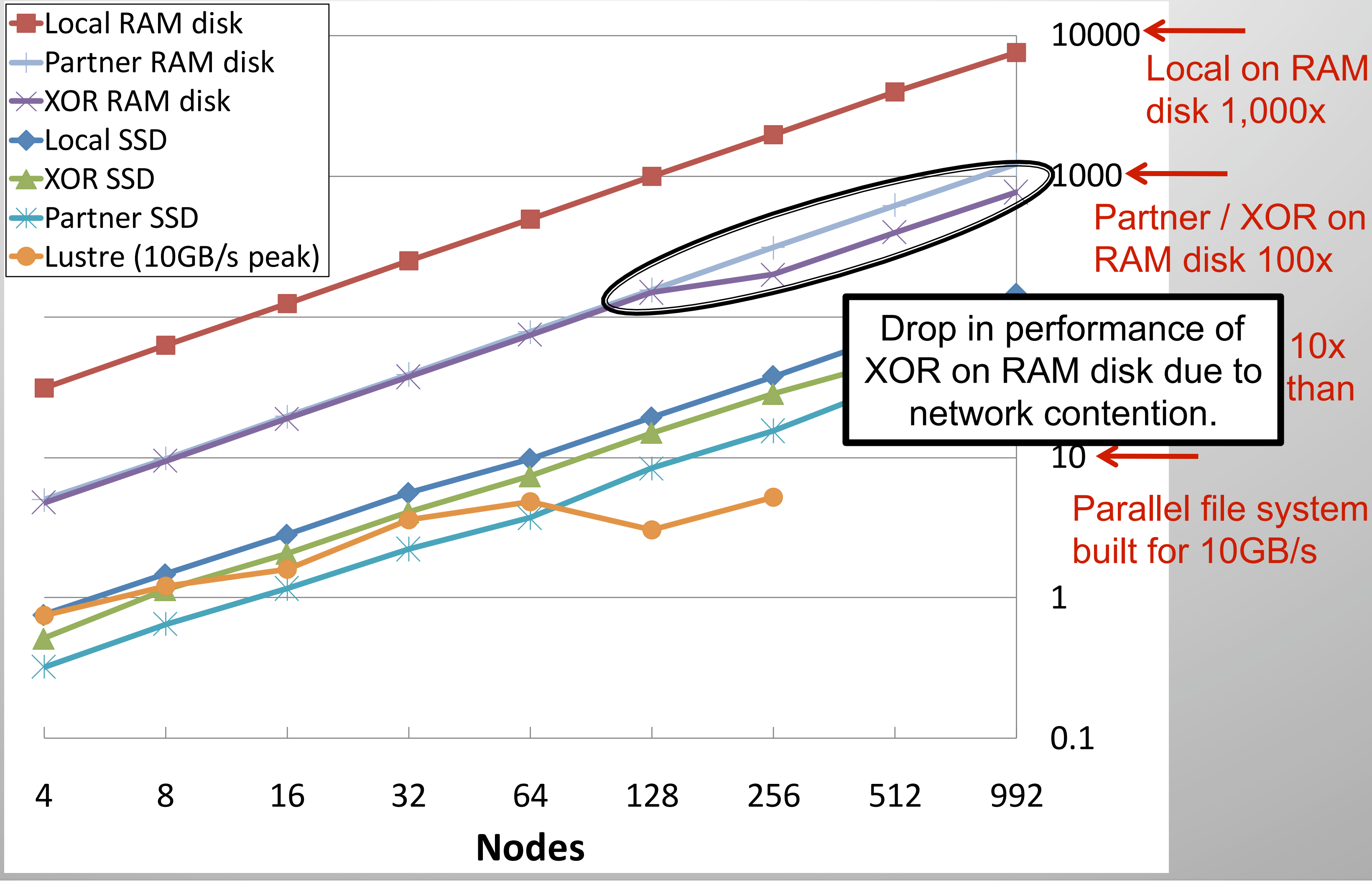


SCR provides an easy-to-use multi-level checkpoint mechanism

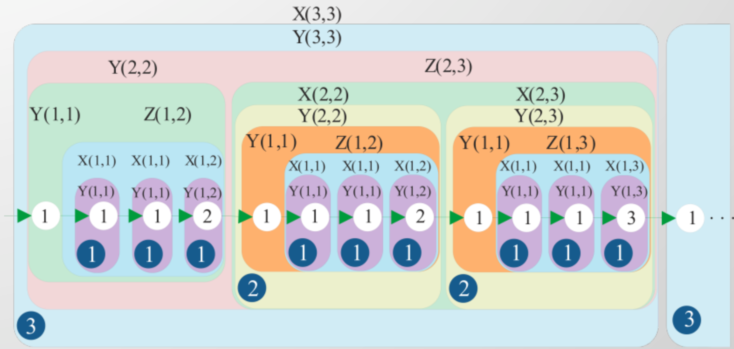


- Simple, portable API integrates around application's checkpoint code
- Instructs application to write checkpoint files to storage local to each compute node
  - RAM disk, SSD, hard drive, etc.
- Applies redundancy scheme to withstand common failures
  - Local, Partner, or XOR
- Also writes checkpoints to parallel file system
- Upon failure:
  - Kills current job
  - Finds most recent checkpoint
  - Rebuilds missing files and distributes files among compute nodes
  - Restarts job

Node-local aggregate checkpoint scales linearly on Coastal



We have formulated a novel Markov model to predict the optimal checkpoint interval at each level

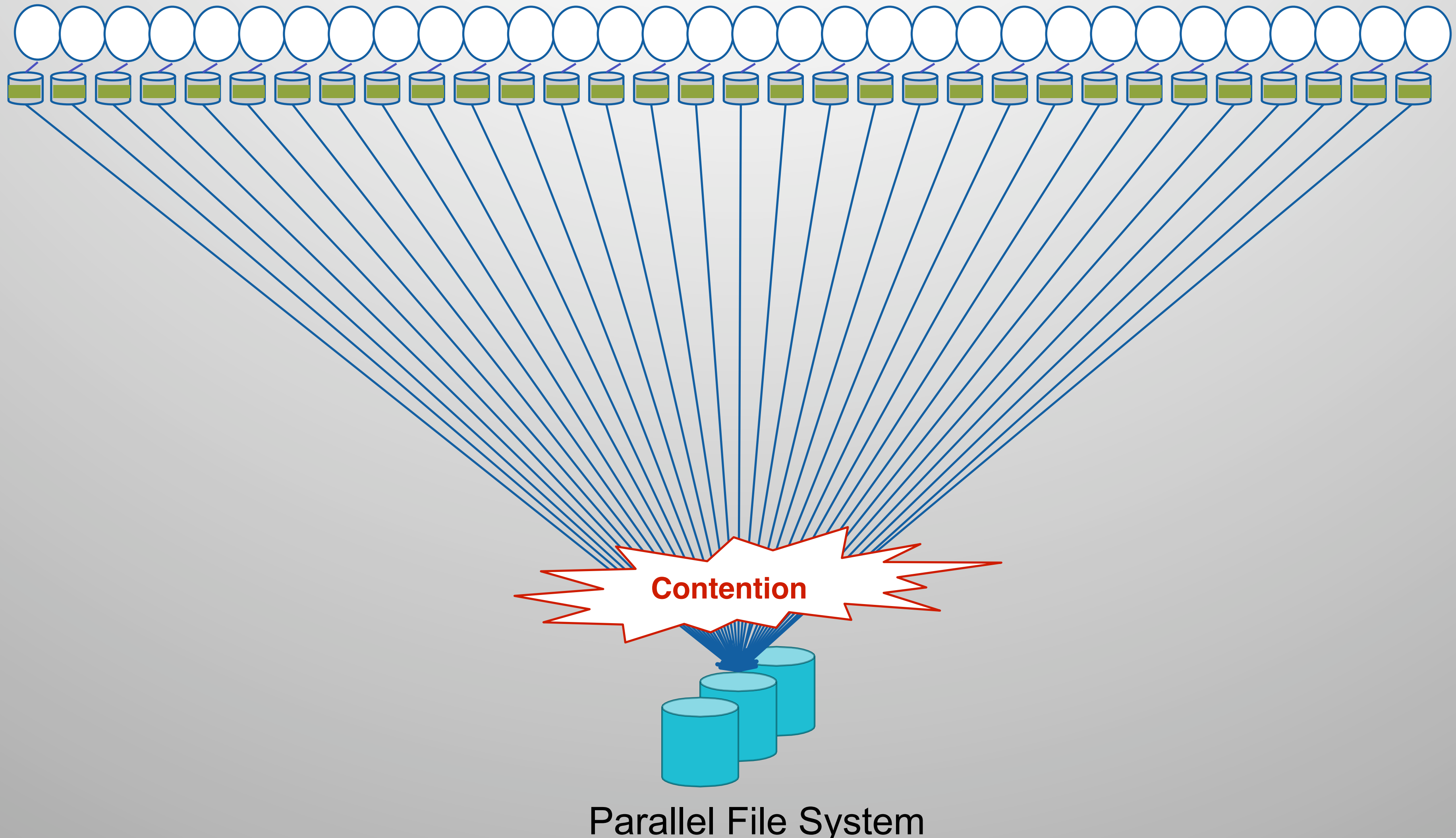


- SCR:
  - Dramatically increases utilization
  - In pF3D production use
- Used pF3D reliability data with novel Markov Model to explore future design space for local storage
  - Predictions highly accurate for current systems (Atlas and Coastal)
  - Multilevel checkpointing significantly alleviates burden on parallel file system
  - Model demonstrates that allocating extra (idle) nodes for restart of failed processes will usually improve overall utilization with SCR

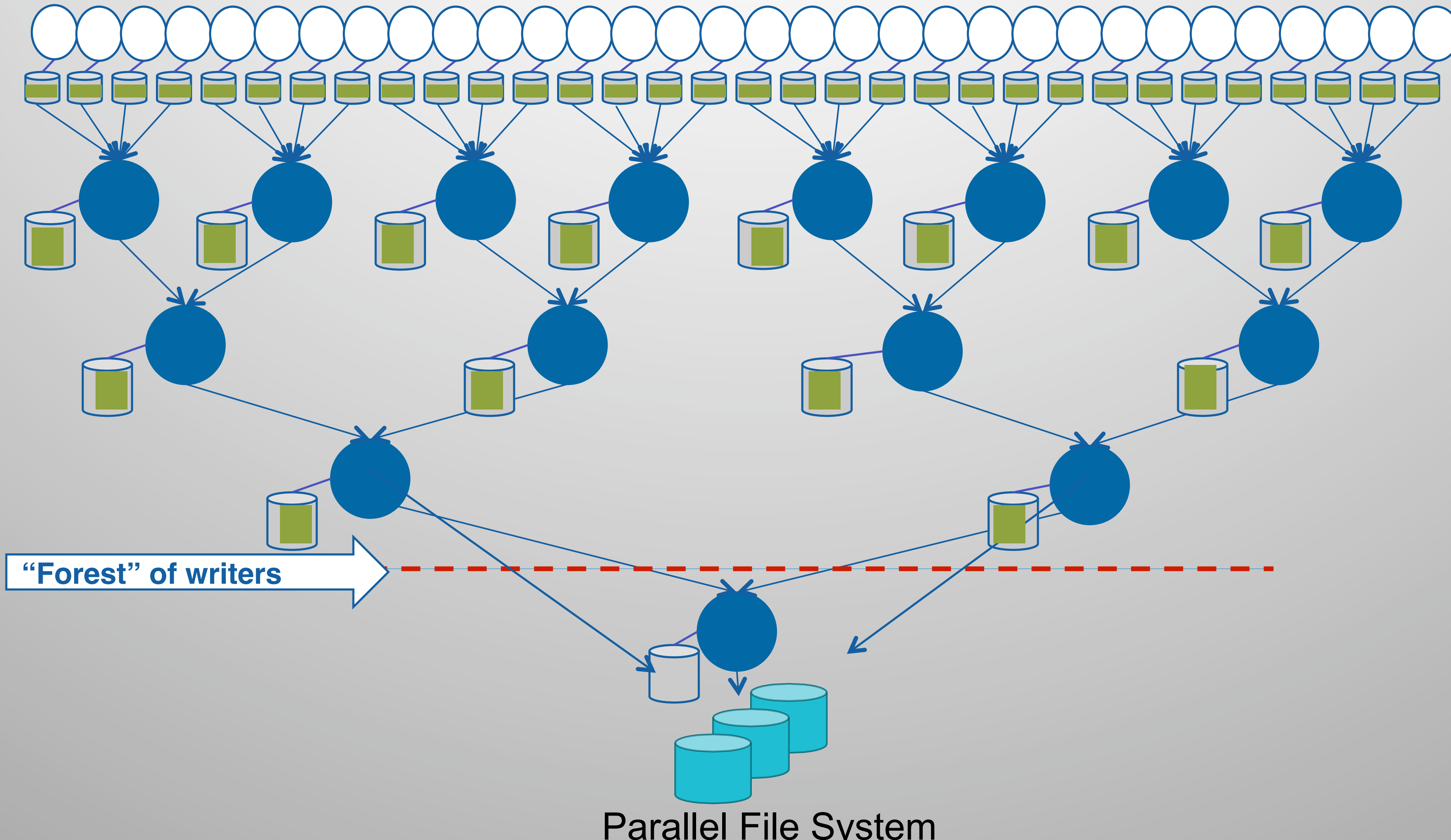
pF3D FAILURES ON THREE DIFFERENT CLUSTERS

Clusters	Coastal	Hera	Atlas	Total
Time span	Oct 09 - Mar 10	Nov 08 - Nov 09	May 08 - Oct 09	
Number of jobs	135	455	281	871
Node hours	2,830,803	1,428,547	1,370,583	5,629,933
Total failures	24	87	80	191
LOCAL required	2 (08%)	36 (41%)	21 (26%)	59 (31%)
PARTNER/XOR required	18 (75%)	32 (37%)	54 (68%)	104 (54%)
Lustre required	4 (17%)	19 (22%)	5 (06%)	28 (15%)

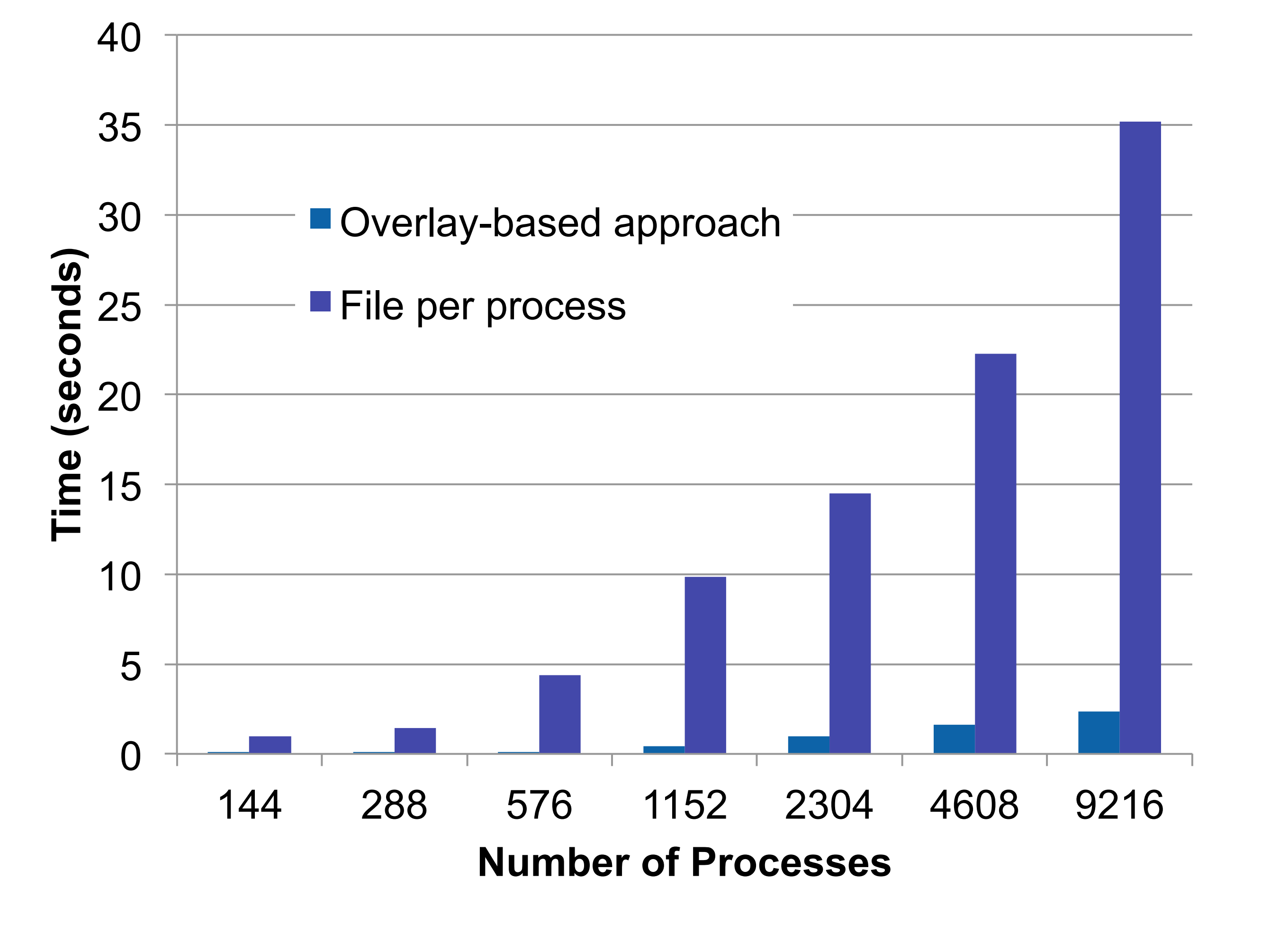
Current SCR implementation can still suffer from parallel file system contention and meta-data bottleneck



Our prototype overlay network solution reduces the bottleneck and supports our compression strategy

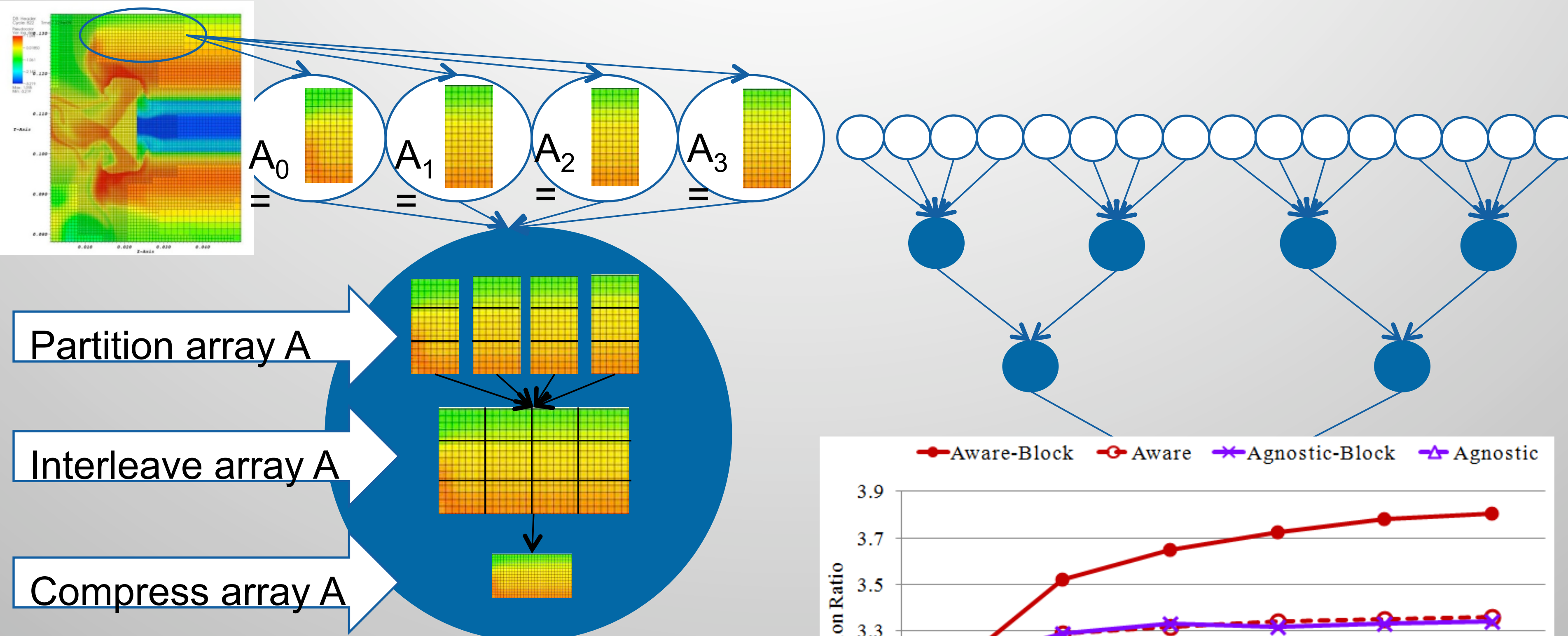


Initial results demonstrate this strategy will improve average total I/O time per checkpoint

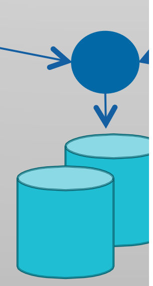


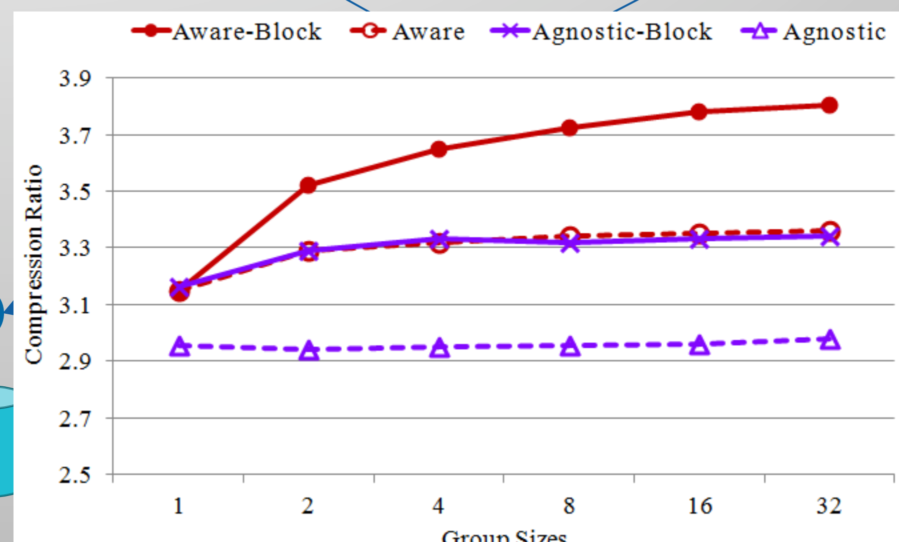
- Overlay network uses a single writer
- Both strategies write every checkpoint to the parallel file system

Data-aware cross-checkpoint compression can reduce parallel file system bandwidth demand

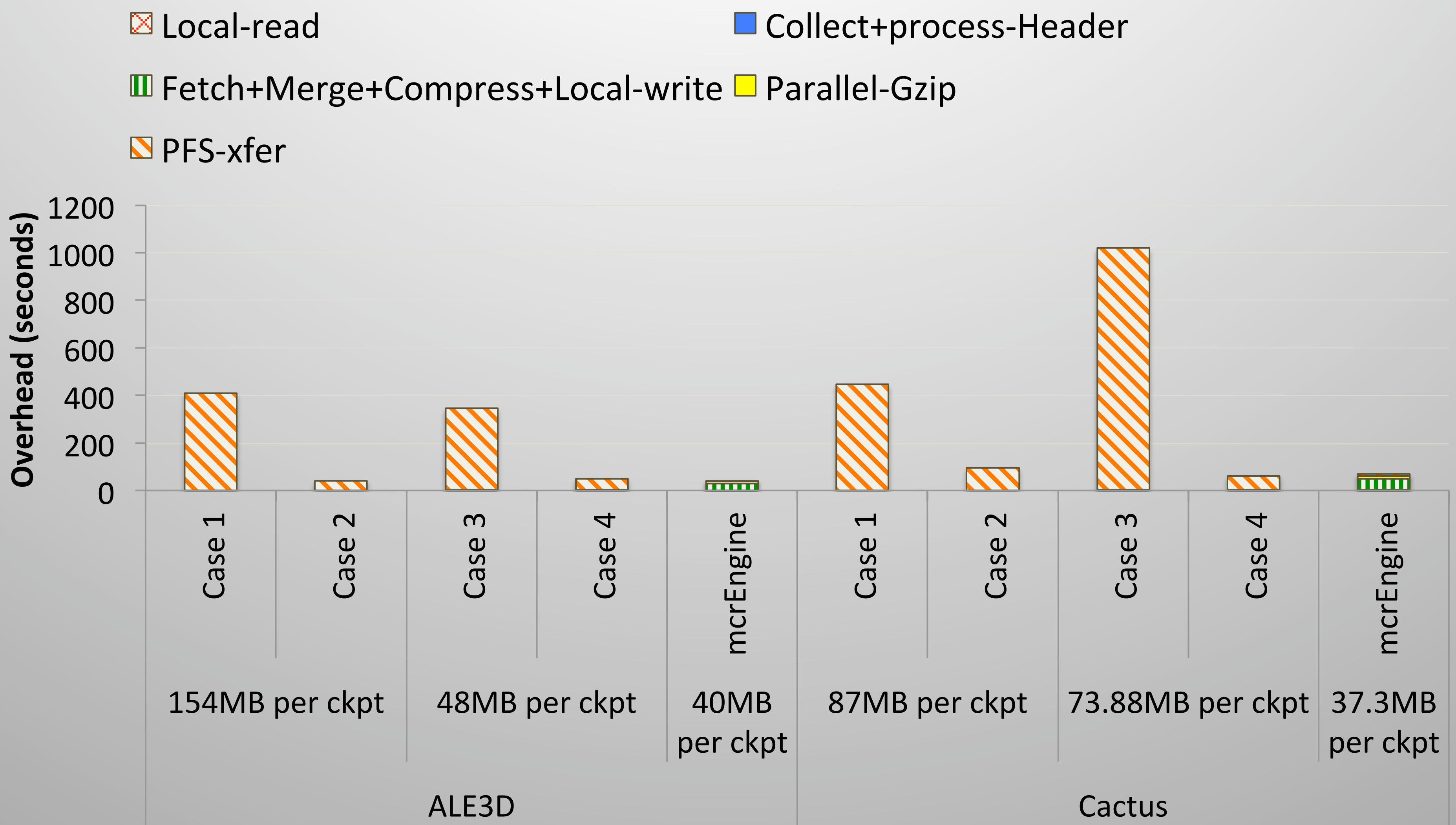


- Interleave like with like data from checkpoints across tasks
- Apply compression algorithm targeted to data type

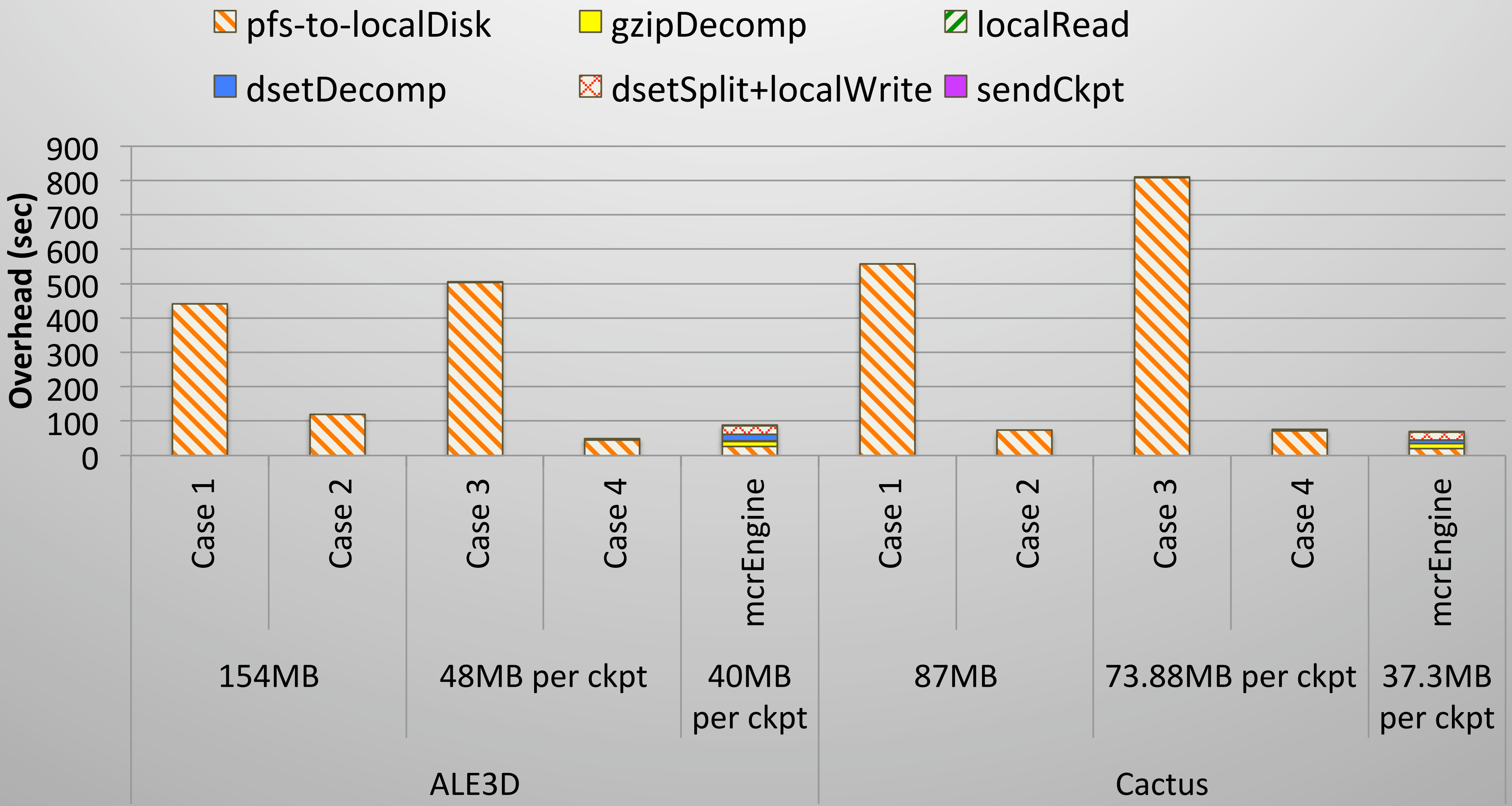




Compression often provides lower end-to-end checkpointing overhead



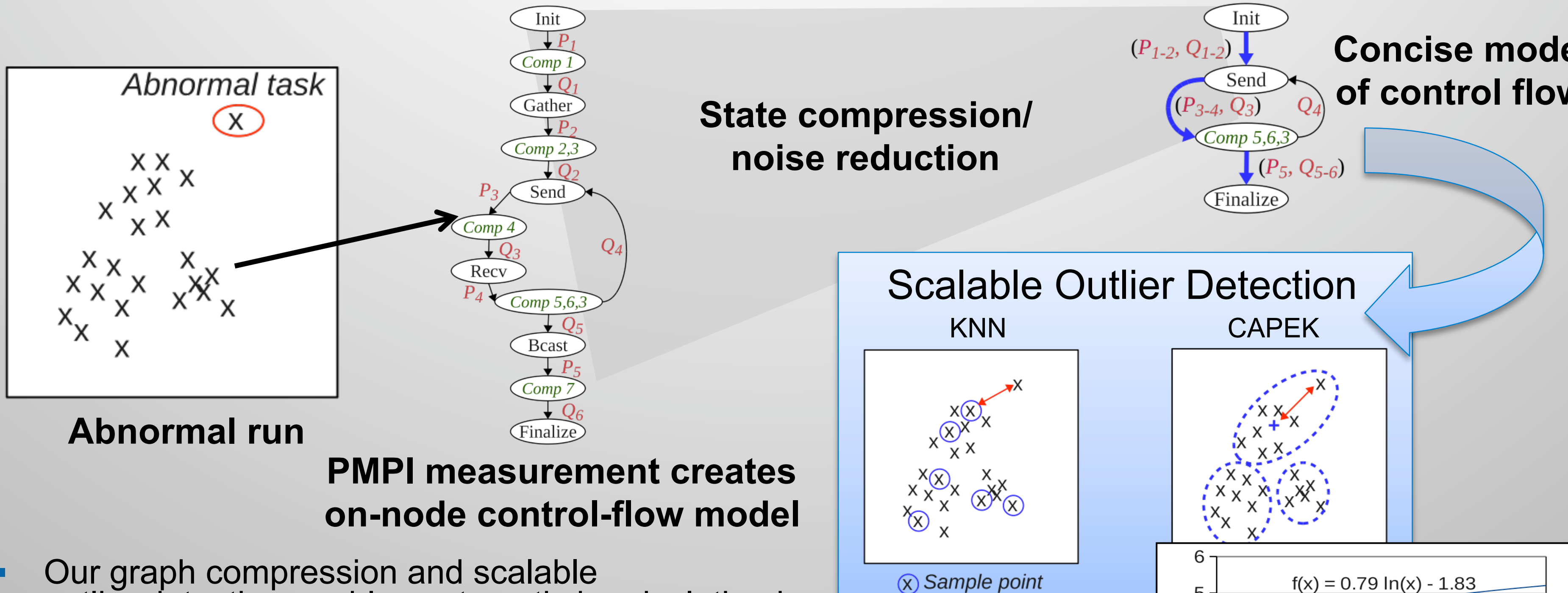
Recovery overhead, which is on the critical path, is low with compression

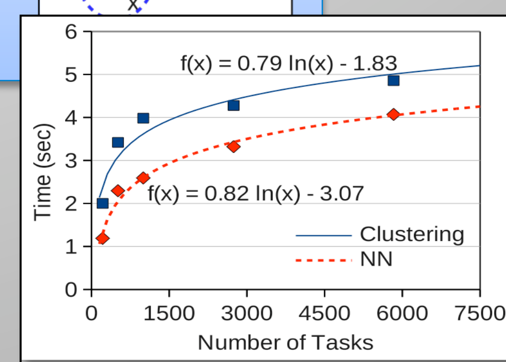


Our ABFT strategy will provide an integrated overall solution

- Tools to detect failures and to identify their causes
- Mechanisms to assess application vulnerability
  - Fault injection experiments
  - Evaluate improvements from hand transformations
- ROSE translators to automate transformations
  - Modular redundancy
  - Other techniques under consideration
- User annotations to guide translators
  - Identify regions, operations likely needing protection
  - Define possible protection mechanisms
- Autotune application of transformations

We are developing AutomaDeD into a framework for many types of distributed analysis





- Our graph compression and scalable outlier detection enables automatic bug isolation in:
  - ~ 6 seconds with 6,000 tasks on Intel hardware
  - ~ 18 seconds at 103,000 cores on BG/P
- Logarithmic scaling implies billions of tasks will still take less than 10 seconds
- We are developing new on-node performance models to target resilience problems as well as debugging

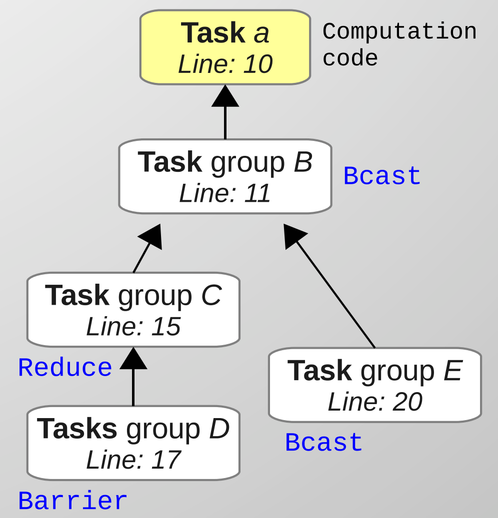
Probabilistic application progress supports finding root causes

- AutomaDeD finds:
  - Anomalous processes
  - Anomalous SMM transitions
- Programmers need insight: what code led to failure?
- Need distributed dependence information to understand distributed hangs
- We use progress dependence to provide that insight
  - Dynamic detection of MPI dependences
  - Similar to postdominance
    - Progress dependence does not require an exit node
    - May not have exit nodes in dynamic call tree, especially with a failure

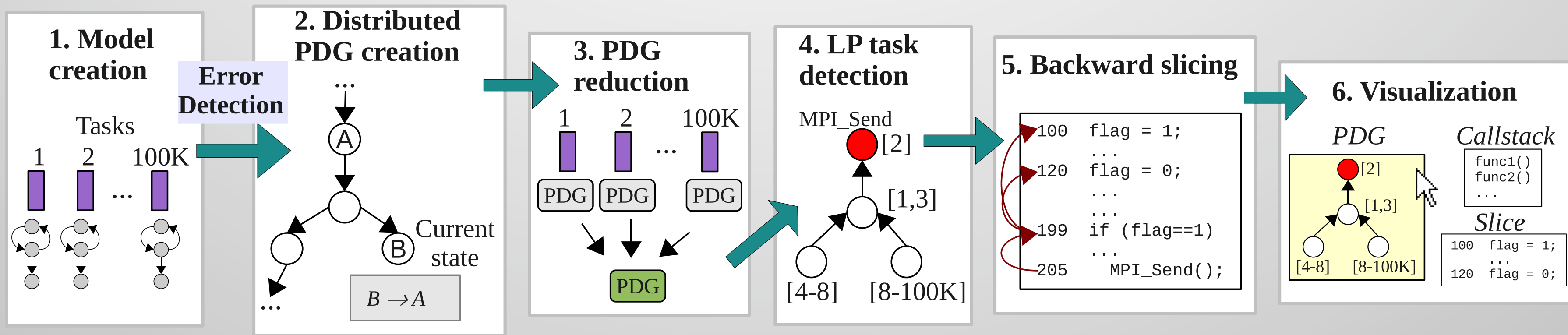
Sample code

```
10  // Computation code ...
11  MPI_Bcast(.., MPI_COMM_WORLD);
12  // ...
13  if (...) {
14    // ...
15    MPI_Reduce(..., comm_1);
16    // ...
17    MPI_Barrier(comm_1);
18  } else {
19    //...
20    MPI_Bcast(..., comm_2);
21  }
22  // ...
```

Progress dependence graph



Our distributed pipeline enables fast root cause analysis

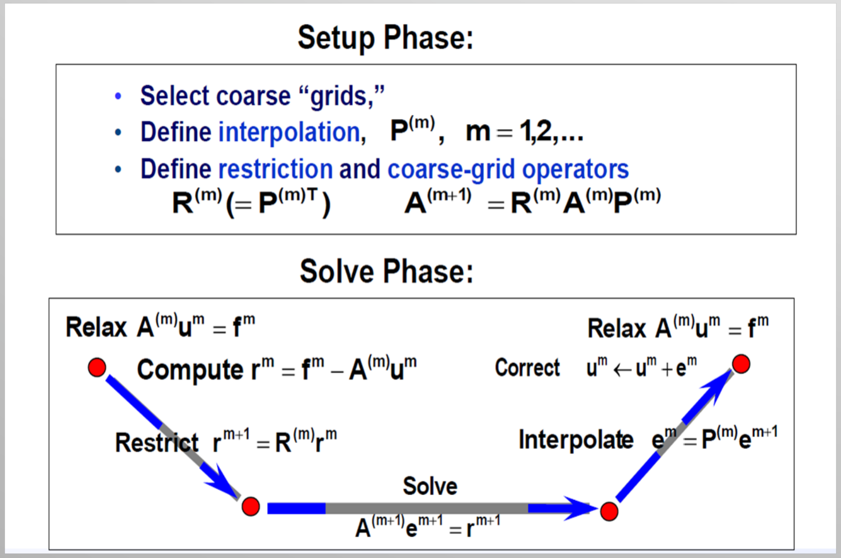


- Full process has O(log(P)) complexity
- Distributed analysis requires < 0.5 sec on 32,768 processes
- Gives programmers insight into the exact lines that could have caused a hang.
- We use DynInst's backward slicing at the root to find likely causes

We have analyzed the vulnerability of algebraic multigrid (AMG)

- AMG solves linear systems of equations derived from the discretization of PDEs.
- AMG is an iterative method that operates on nested grids of varying refinement.
- Two operators (restriction and interpolation) propagate linear system through the grids.

- Identify vulnerable data and code regions.
- Design and implement simple and effective resilience strategies to improve vulnerability of sensitive pieces of code.
- Long term: develop a general methodology to automatically improve the reliability of generic HPC codes.



AMG algorithmic structure

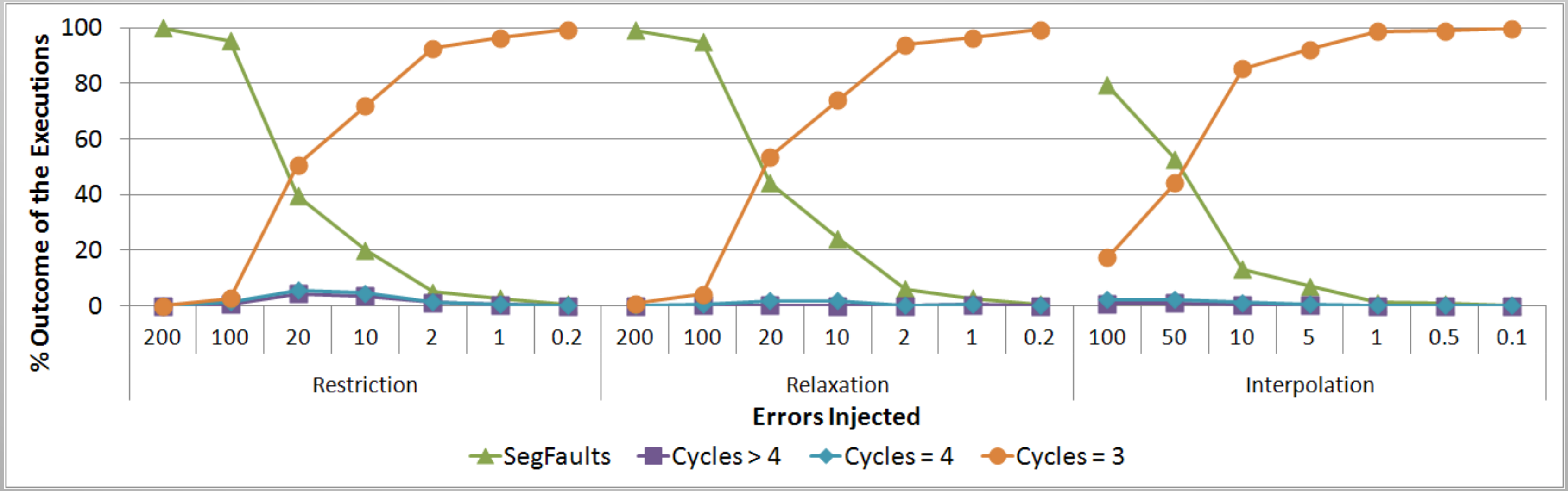
- Developed of a methodology to automatically inject faults to assess the vulnerability of codes to soft errors.
- Performed a vulnerability study of AMG.
- Determined that AMG is most vulnerable to soft errors in pointer arithmetic, which lead to fatal segmentation faults.
- Demonstrated that triple modular redundancy in pointer calculations reduces the vulnerability of AMG to soft errors
- Subject of paper to appear at ICS 2012

We evaluate the resilience of the solver phases

- Extensive fault injection campaign
  - 3 main AMG phases
  - 7 fault injection rates for each phase
  - 10,000 executions for each phase and fault injection rate

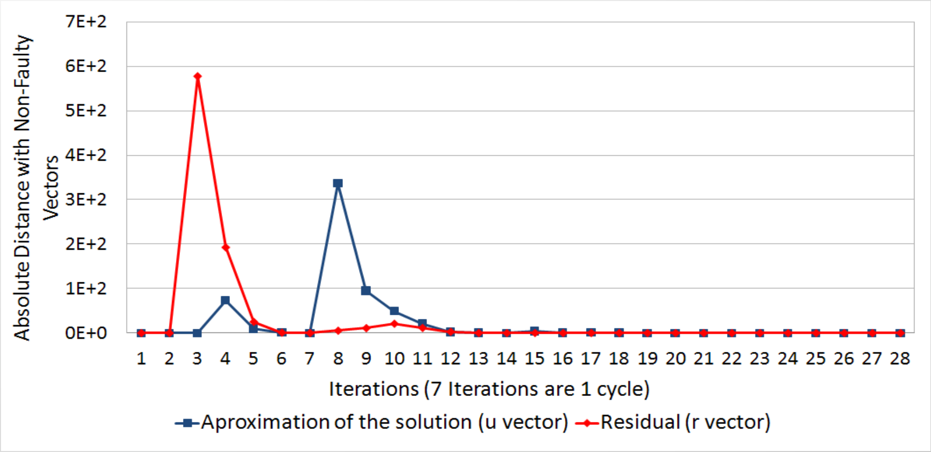
	Restriction	Interpolation	Relaxation
Rate 1	200 f, 4273 e/s	100 f, 5128 e/s	200 f, 1666 e/s
Rate 2	100 f, 2136 e/s	50 f, 2564 e/s	100 f, 833 e/s
Rate 3	20 f, 427 e/s	10 f, 513 e/s	20 f, 167 e/s
Rate 4	10 f, 214 e/s	5 f, 256 e/s	10 f, 83 e/s
Rate 5	2 f, 43 e/s	1 f, 51 e/s	2 f, 17 e/s
Rate 6	1 f, 21 e/s	0.5 f, 25 e/s	1 f, 9 e/s
Rate 7	0.2 f, 4 e/s	0.1 f, 5 e/s	0.2 f, 2 e/s

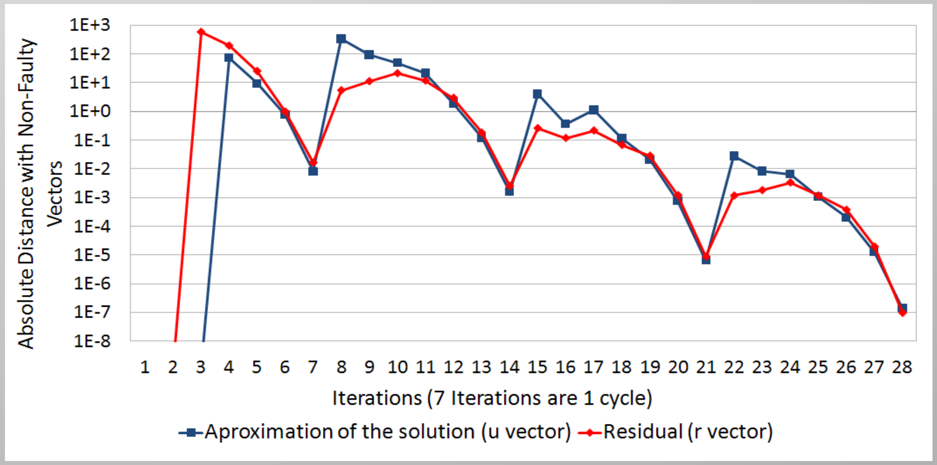
- Our analysis provides a vulnerability profile of each phase:



Our results show that AMG is naturally resilient to numeric faults

- Numeric errors smoothed out
  - The error spreads among the variables when execution returns to the finest levels.
  - The correction on the coarsest grids is more significant than the subsequent contamination
  - AMG's coarsening smoothes out the effects of large local errors
  - Other errors induce segmentation faults





We implement a simple algorithm that protects key pointers through replication

- Pointer replication
  - mantains multiple copies of each replicated pointer
  - compares them before every memory access
- Three copies are kept in our experiments

	Restriction	Interpolation	Relaxation
Rate 1	200 f, 4273 e/s	100 f, 5128 e/s	200 f, 1666 e/s
Rate 2	100 f, 2136 e/s	50 f, 2564 e/s	100 f, 833 e/s
Rate 3	20 f, 427 e/s	10 f, 513 e/s	20 f, 167 e/s
Rate 4	10 f, 214 e/s	5 f, 256 e/s	10 f, 83 e/s
Rate 5	2 f, 43 e/s	1 f, 51 e/s	2 f, 17 e/s
Rate 6	1 f, 21 e/s	0.5 f, 25 e/s	1 f, 9 e/s
Rate 7	0.2 f, 4 e/s	0.1 f, 5 e/s	0.2 f, 2 e/s

Original Code of Matrix-Vector Multiplication

```
for(jj = A_i[i]; jj < A_i[i+1]; jj++) {
   y_data[i] += A_data[jj] * x_data[A_j[jj]];
}
```

Transformed Code With Pointer Triplication

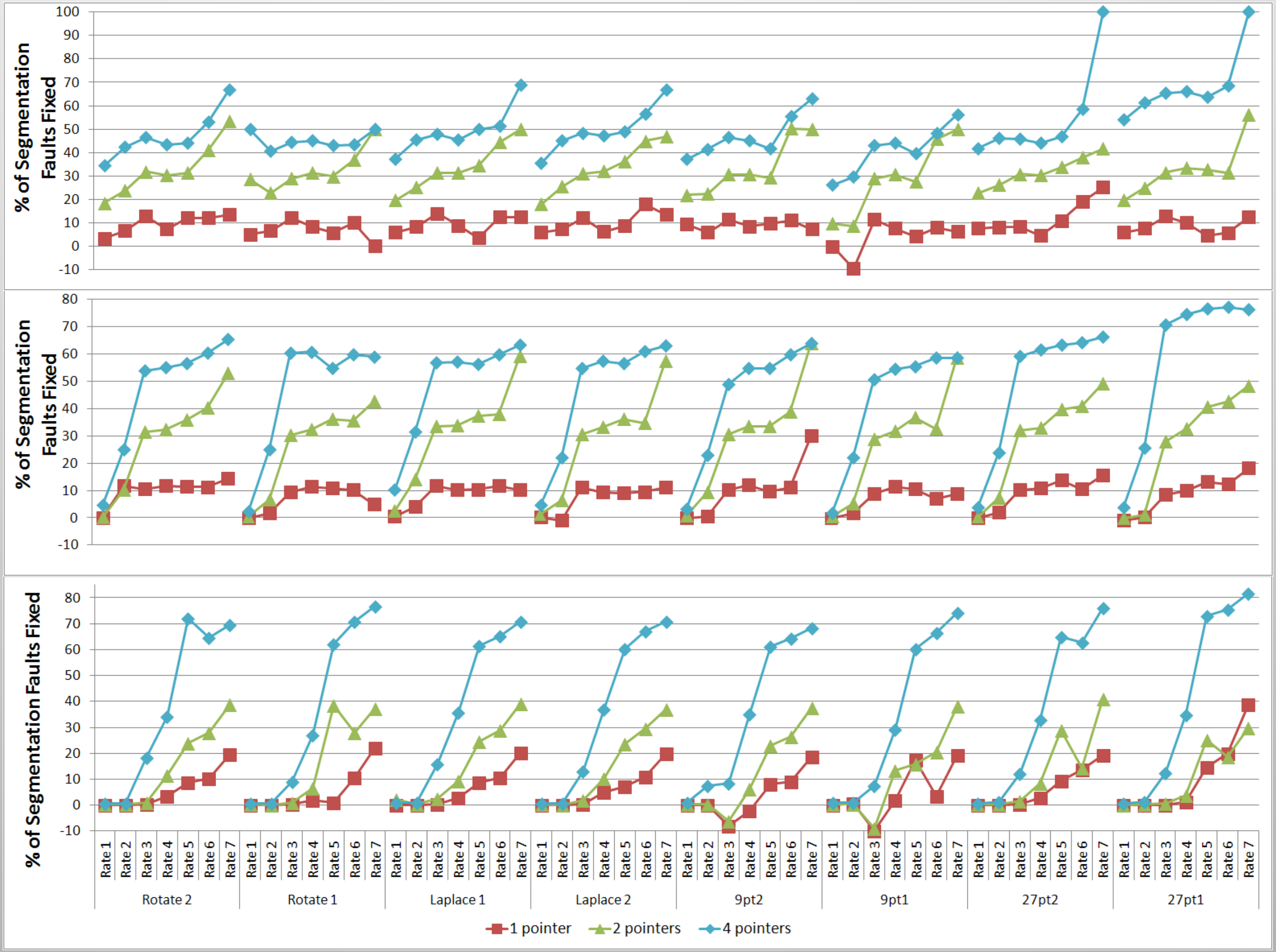
```
for(jj = A_i[i]; jj < A_i[i+1]; jj++) {
   y_data[i] += A_data[jj] * x_data[triplication(A_j, A_j_p1, A_j_p2)[jj]];
}
```

Pointer triplication is effective on a variety of inputs and different error injection rates

Interpolation

Restriction

Relaxation



We are addressing the looming resilience challenge

- Some question whether the sky is falling
  - Reduction in file system bandwidth relative to memory size
  - MTTI likely to decrease
- New resilience strategies will be essential
  - Provide critical checkpoint/restart efficiency improvements
  - Must understand fault vulnerability to address it appropriately
  - Must detect failures in order to respond to them
  - Must automate resilience transformations and trade-off assessment

Lawrence Livermore
National Laboratory