Detection and Correction of Silent Data Corruption for Large-Scale High-Performance Computing

David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, Ron Brightwell
Resilience in HPC

- **HPC**: 10k-100k nodes
  - Some component failure likely
  - System MTBF becomes shorter
  - Processor/memory/IO failures

- Currently FT exists but...
  - not scalable
  - mostly reactive: process checkpoint/restart
  - restart entire job → inefficient if only one/few node(s) fail
  - overhead: re-execute some of prior work

### System MTBF Examples

<table>
<thead>
<tr>
<th>System</th>
<th># CPUs</th>
<th>MTBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCI White</td>
<td>8,192</td>
<td>5/40 hrs</td>
</tr>
<tr>
<td>Google</td>
<td>1,5000</td>
<td>20 reboots/day</td>
</tr>
<tr>
<td>ASC BD/L</td>
<td>212,992</td>
<td>7 hrs</td>
</tr>
<tr>
<td>Jaguar</td>
<td>300,000</td>
<td>5/52 hrs</td>
</tr>
</tbody>
</table>
Checkpoint/Restart Overhead

- Apps req’d to support C/R paradigm
  - As we add cores:
    - C/R overhead grows exponentially
    - Inc. probability of failure

- Sandia study:

<table>
<thead>
<tr>
<th># Nodes</th>
<th>work</th>
<th>checkpoint</th>
<th>recomp.</th>
<th>restart</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>96%</td>
<td>1%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>1,000</td>
<td>92%</td>
<td>7%</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>10,000</td>
<td>75%</td>
<td>15%</td>
<td>6%</td>
<td>4%</td>
</tr>
<tr>
<td>100,000</td>
<td>35%</td>
<td>20%</td>
<td>10%</td>
<td>35%</td>
</tr>
</tbody>
</table>
Exascale Resilience

- 1 billion cores
- ~ 1 million components
- MTBF/node 50 yrs (52 hrs for Jaguar)
- Goal: MTBF ~ 1 day
- 10x-100x > components
- Reliability ~ # components

  - need 10x-100x reliability improvement
    - Hardware: 10x (or less → smaller fabs)
    - Software: 10x (or more → focus of this talk)

- How can this be achieved?

<table>
<thead>
<tr>
<th>System attributes</th>
<th>2010</th>
<th>“2015”</th>
<th>“2018”</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak FLOPS</td>
<td>2 Peta</td>
<td>200 Peta</td>
<td>1 Exa</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~15 MW</td>
<td>~20 MW</td>
</tr>
<tr>
<td>System memory</td>
<td>0.3PB</td>
<td>5 PB</td>
<td>32-64PB</td>
</tr>
<tr>
<td>Node performance</td>
<td>125 GF</td>
<td>0.5TF or 7 TF</td>
<td>1 TF or 10x</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25GB/s</td>
<td>0.1TB/s or 10x</td>
<td>0.4TB/s or 10x</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
<td>O(100)</td>
<td>O(1k) or 10x</td>
</tr>
<tr>
<td>TotalNode Interconn BW</td>
<td>1.5 GB/s</td>
<td>20 GB/s or 10x</td>
<td>200GB/s or 10x</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
<td>50,000 or 1/10x</td>
<td>O(100,000) or 1/10 x</td>
</tr>
<tr>
<td>MTTI</td>
<td>days</td>
<td>O(1day)</td>
<td>O(1 day)</td>
</tr>
</tbody>
</table>
Silent Data Corruption

- Silent Data Corruption (SDC) faults → bit flips in
  - storage or CPU cores
  - Some not detectable / correctable
  - Undetected → invalid results, app doesn’t stop

  - Severe problem for today’s large-scale simulations

- Memory bit flips correctable by ECC
  - Each ECC algorithm may have an upper limit of bit flips
  - Uncorrectable for an instant reboot

Undetectable errors are expected to occur once or twice per day on ORNL’s Jaguar Supercomputer [Geist, Monster in Closet]
Contributions

• Design & impl. of novel mechanisms for FT in HPC
  — Propose efficient protocols for SDC protection
  — Investigate cost of different levels of redundancy

• Demonstrate capabilities of SDC protection at comm. layer
  — Assess cost of redundancy
  — Fault injection → study failures on native cluster
  — SDC Propagation Study
Design

- Create clones of MPI processes
  - Clones run same app, deterministically
  - Clones always send same msgs when no corruption
- Double modular redundancy (2x processes - one “shadow”)
  - Clones perform online (live) message verification
- Triple modular redundancy (3x processes - two “shadows”)
  - Clones perform verification and correction

<table>
<thead>
<tr>
<th></th>
<th>No Redundancy</th>
<th>Dual Redundancy</th>
<th>Triple Redundancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live SDC Detection</td>
<td>X No</td>
<td>✓ Yes</td>
<td>✓ Yes</td>
</tr>
<tr>
<td>Live SDC Correction</td>
<td>X No</td>
<td>X No</td>
<td>✓ Yes (via voting)</td>
</tr>
</tbody>
</table>
Message comparison: Point-to-point

- Instrument send op (MPI_Isend)
  - Each message now becomes one message per replica

- Instrument replicas’ receiver op (MPI_Irecv)
  - receive 1 message from each sender replica
    (instead of just 1 message total)

- Receiver responsible for verification
  - general case ➔ msg is correct: msgs from replicas match

receivers may verify/correct message
sender continues immediately after transmission
Design Assumptions

- Transport layer reliable (TCP Ethernet / Infiniband)
  - Already covered by checksums / correction codes on fabrics

- Not protected: app instructions / control-flow

- 56 MPI functions supported, incl.
  - pt-2-pt, collectives, wildcards...
Implementation of RedMPI

- Implemented MPI instrumentation lib: RedMPI
  - provides transparent protection to MPI processes
  
  - interposes MPI functionality via PMPI (MPI profiling layer)
  
  - extra processes created when MPI applications are launched
    - extra processes become replicas

- MPI job w/ 128 tasks now becomes
  - 256 tasks for 2x redundancy
  - 384 tasks for 3x redundancy
Redundant MPI Ranks

- Each MPI task/process is a rank
- RedMPI transparently creates \( r \) replicas per normal MPI rank
- Virtual rank: as seen by app.
- Native rank: as seen by MPI
- Replica rank: 0…\( r-1 \) identifies the replica

```bash
mpirun -np <nativesize>
virtualRank == MPI_Comm_rank()
```
SDC Method 1: All-to-all

- \( r \) replicas \( \Rightarrow \) each sender xmits full copy of msg to each receiver

- Requires:
  - \( r \) receive buffers
  - \( r^2 \) messages

- Simple, naïve approach
  - \( r \)-way comparison
  - for \( >2 \) buffers, compare & replace mismatch
SDC Method 2: MsgPlusHash

- An optimization for the general case
  - Most messages are not corrupt

- $r$ messages + $r$ small hash messages (instead of $r^2$)
  ($r_{\text{data}} + r_{\text{hash}}$)

- More efficient, but requires corruption discovery protocol
Dealing with Non-Determinism

- Some MPI ops are non-deterministic
  - RedMPI’s control-flow between replicas must be identical

- MPI_Wtime returns current time
  - Almost guaranteed to be divergent between replicas

- MPI_Iprobe checks if a message has already arrived
  (without making an actual request)

- MPI_Probe - blocking equivalent of MPI_Iprobe

- Wildcard operations: MPI_ANY_TAG, MPI_ANY_SOURCE
Extending coverage: Collectives

- MPI implementations employ collective operations
  - broadcast, reduce, etc.

- MPI library determines underlying communication pattern
  - Pt-to-pt ops not visible to the app /profiling layer

- Collectives are blocking - impossible to overlap

- RedMPI implements own linear collectives
  - not necessarily performant for large jobs
    - Aforementioned pt-to-pt protection is used
Interposing collectives

- RedMPI exploits topology-aware algorithms
  - Redirects MPI’s low-level pt-to-pt comm back through RedMPI’s communication layer
  - Uses optimal comm. from MPI implementation
Experimental Framework

- RedMPI run on ARC cluster at NCSU
  - 108 compute nodes, 1700+ cores
    - 32GB DRAM/node
    - 2-way SMPs with AMD Opteron 6128 processors with 8 cores per socket
    - 16 cores per node
  - Open MPI 1.5
    - Evaluated with RedMPI’s collectives module
  - 40Gbit/sec Infiniband interconnect
  - Evaluated with up to 1536 processes per job
Results: Benchmarks (Weak Scaling)

- **HPCCG – wildcard receives present**

<table>
<thead>
<tr>
<th>Size</th>
<th>1x [sec]</th>
<th>2x [sec]</th>
<th>3x [sec]</th>
<th>2x OV</th>
<th>3x OV</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>99.8</td>
<td>99.8</td>
<td>125.8</td>
<td>0.0%</td>
<td>26.0%</td>
</tr>
<tr>
<td>256</td>
<td>99.6</td>
<td>128.8</td>
<td>131.0</td>
<td>29.3%</td>
<td>31.5%</td>
</tr>
<tr>
<td>512</td>
<td>126.4</td>
<td>146.2</td>
<td>152.3</td>
<td>15.7%</td>
<td>20.5%</td>
</tr>
</tbody>
</table>

• Increased job size ➔ Comm/Comp ratio same

• Negligible overhead for weak scaled apps
Results: Benchmarks (Strong Scaling)

- No Redundancy
- Dual Redundancy
- Triple Redundancy

NPB-CG

- 0-20% overhead

NPB-EP

- <1% overhead

NPB-FT

- 0-5% overhead

- Increased job size $\rightarrow$ Comm/Comp ratio increases

- High communication gives greater impact
Fault injector

- **Sender side:** 1/x messages randomly receive 1 random bit flip
  - Internal, per-process seeded RNG

bit is permanently flipped in sender’s buffer ➔ passed to receivers

 Receivers detect corruption
 Retains only correct msg
Fault Injection Experiments (TMR)

- High injection rates → good stress test

- Experiment #1:
  - Injection rate: 1 bit flip / 5 million messages
  - 9/10 runs → 1 corrected message
  - 1 run → 6,242 bad messages
    - Likely due to data reuse in corrupted send buffer
    - All runs pass benchmark’s built-in verification

- Experiment #2: 1 bit flip / 2.5 million messages
  - avg. ~2.5 injections / run & 1000s bad msgs
  - 8/10 runs passed verification
  - 2/10 runs failed → 2+ clones sent corrupt msgs simultaneously
    - RedMPI forced corrupted job to fail
Propogation Study Classification

- Progressive

- Explosion

- Localized
Fault Injection: SDC Propagation

- Experiment #3: Inject 1 bit flip
- Error correction intentionally turned off
- Tainted buffer reuse, propagates
Conclusions

- Devised 2 SDC consistency methods
  - Efficient method: MsgPlusHash
    overheads: 0%-30% for dual / triple redundancy
  - Weak scaling apps ➔ particularly good candidates

- Error propagation study:
  - w/o detection mechanisms, SDC spreads across boundaries

- SDC coverage effective: All injected faults detected
  - If uncorrectable ➔ RedMPI forces a stop

- Cost of double & triple redundancy high
  - implementing redundancy is not ➔ avoids reruns of C/R

For applications experiencing high SDC rates, redundancy may be worth the cost to protect and ensure correct output
Acknowledgements

- This work was supported in part by
  - NSF grants CNS-1058779, CNS-0958311, DOE grant DE-FG02-08ER25837
  - a subcontract from Sandia National Laboratory
  - by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. De-AC05-00OR22725.
Extra Slides
Related Works

- **Software redundancy:**
  - PLR, DDMR *(Multicore redundancy)*

- **MPI:**
  - rMPI - K. Ferreira *(Sandia Labs)*
    - Built using MPICH source, handles node failures
  - MRMPI - C. Engelmann *(Oak Ridge Labs)*
    - MPI Interpositioning layer only
    - Provides redundancy
    - RedMPI borrows its linear collectives
    - Redundant IO
  - VolpexMPI
    - Provides polled-communication to handle node failures
    - Performant for smaller jobs, FT written in the comm. layer
Collectives Module Performance

- Switching to RedMPI’s enhanced collectives module integrated into Open MPI provided key performance enhancements over the fallback linear collectives.

- Average overheads of select benchmarks using linear fallback:

<table>
<thead>
<tr>
<th></th>
<th>Dual Redundancy</th>
<th>Triple Redundancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPB CG</td>
<td>44%</td>
<td>53%</td>
</tr>
<tr>
<td>NPB LU</td>
<td>10%</td>
<td>19%</td>
</tr>
<tr>
<td>SWEEP3D</td>
<td>18%</td>
<td>23%</td>
</tr>
</tbody>
</table>

- Average overheads using RedMPI’s enhanced collectives:

<table>
<thead>
<tr>
<th></th>
<th>Dual Redundancy</th>
<th>Triple Redundancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPB CG</td>
<td>6%</td>
<td>11%</td>
</tr>
<tr>
<td>NPB LU</td>
<td>8%</td>
<td>10%</td>
</tr>
<tr>
<td>SWEEP3D</td>
<td>0%</td>
<td>1%</td>
</tr>
</tbody>
</table>
Adding determinism: Wildcards

- MPI supports receiving messages from a previously unknown sender—and/or—a message with any "tag"
  - MPI_ANY_SOURCE MPI_ANY_TAG

- Only lowest ranked replica posts the wildcard receive
  - Others await an “envelope” message from the leader
  - Problematic: All subsequent receive operations on the followers must be buffered until all wildcards are resolved

  - Slows performance: MPI’s Unexpected buffer

- RedMPI handles both types of wildcards together or independently
Adding determinism: Lowest replica rank decides

- Idea: The replica with replica rank 0 is responsible for deciding the result of `MPI_Wtime`

- All replica ranks >0 await a control message from 0

- Result: all replicas return the same time for each call to `MPI_Wtime`

- Very useful for applications that use random number generators
  - Simply seed the RNG with `MPI_Wtime`
End Slide
SDC protection with redundancy

- Potential ideas:
  - Compare in-process memory during execution
    - Global synchronization, high memory usage for verification
    - Not feasible to correct errors while running
  - Frequent checkpoints & compare dumps
    - Checkpoints are huge, slow. Still needs rollback
  - Compare MPI messages
    - Minimized search space
    - Correct communication is a necessary condition for output correctness (but not sufficient)
Introduction

- Faults are now the norm for High Performance Computing (HPC)
  - Past reports attribute causes to hardware and software
    - I/O, memory, processor, power supply, switches
    - OS, runtime, unscheduled maintenance
  - Recent work finds that
    - Servers have a 2-5% failure rate
    - DRAM errors are occurring in 2% of all DIMMs per year

- Even small installations have a low MTBF (mean time between failure)
Redundancy in HPC

- Sandia’s study made an important finding:
  - Redundancy in computing can significantly reduce this trend

- Redundancy scales: Adding processes reduces the probability of simultaneous failure
Adding determinism: Probes

- Other MPI functions such as MPI_Iprobe may introduce non-deterministic behaviors
  - The arrival of a message depends on the network
  - While some replicas may have received a message, other may not have

  - Similar to MPI_Wtime, have replica rank 0 decide

  - **This is safe**: The arrival delayed arrival of any message that replica rank 0 has received will eventually arrive at other replicas
Benchmarks

**Weak scaling**
- Input size per process stays constant as we scale the number of total processes per job
  - LAMMPS - Molecular Dynamics code “chute” & “chain” inputs
  - ASCI Sweep3D - Neutron transport code
  - HPCCG - Finite elements app from Sandia Mantevo miniapps

**Strong scaling**
- Input size is invariant as we scale the number of processes
  - NAS Parallel Benchmarks: CG, EP, FT, LU, MG

**Total 9 benchmarks selected** - at 128, 256, & 512 ranks per benchmark for 27 experiments
Elliott et al – Partial Redundancy w Ckpt

- Modeled Time to Completion with Redundancy
- B: $\frac{1}{2}$ node count Jaguar   A: 1/3 node count Jaguar
All-to-all function interposition

- Each MPI request is converted into \( r \) requests internal to RedMPI
- App sees 1 request
- `MPI_Test`/`MPI_Wait` both wait for all messages to arrive before verification
MsgPlusHash Discovery Protocol

- If one sender becomes corrupt, two receivers will be affected
  - Receiver with the same replica rank has an invalid message
  - Receiver with [(replica rank + 1) % SIZE] has invalid hash