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

NC STATE UNIVERSITY Department of Computer Science

Sandia National Laboratories

IBM®

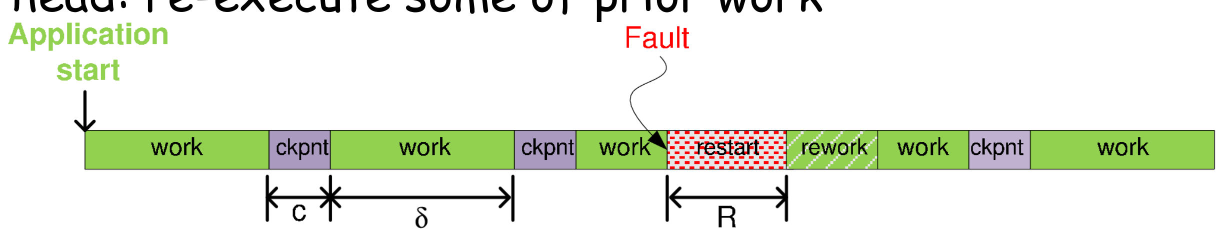
OAK RIDGE National Laboratory

Resilience in HPC

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

System	# CPUs	MTBF
ASCI White	8,192	5/40 hrs
Google	1,5000	20 reboots/day
ASC BD/L	212,992	7 hrs
Jaguar	300,000	5/52 hrs

- 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



Checkpoint/Restart Overhead

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

- Sandia study:

168-HOUR JOB, 5 YEAR MTBF

# Nodes	work	checkpt	recomp.	restart
100	96%	1%	3%	0%
1,000	92%	7%	1%	0%
10,000	75%	15%	6%	4%
100,000	35%	20%	10%	35%

Exascale Resilience

- 1 billion cores
- ~ 1 million components
- MTBF/node 50 yrs (52 hrs for Jaguar)

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

- 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?**

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

	No Redundancy	Dual Redundancy	Triple Redundancy
Live SDC Detection	✗ No	✓ **Yes**	✓ **Yes**
Live SDC Correction	✗ No	✗ No	✓ **Yes** (via voting)

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

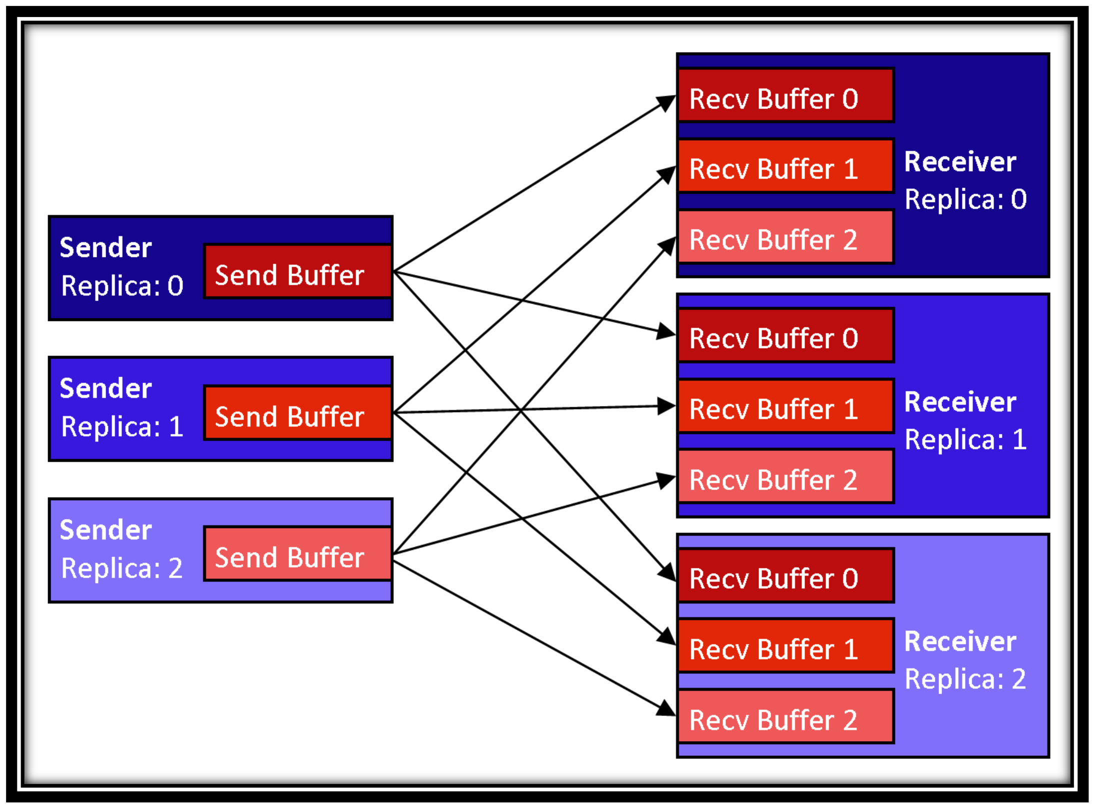
```
mpirun -np <nativesize>

virtualRank == MPI_Comm_rank()
```

SDC Method 1: All-to-all

- *r* replicas ➔ each sender xmits full copy of msg to each receiver

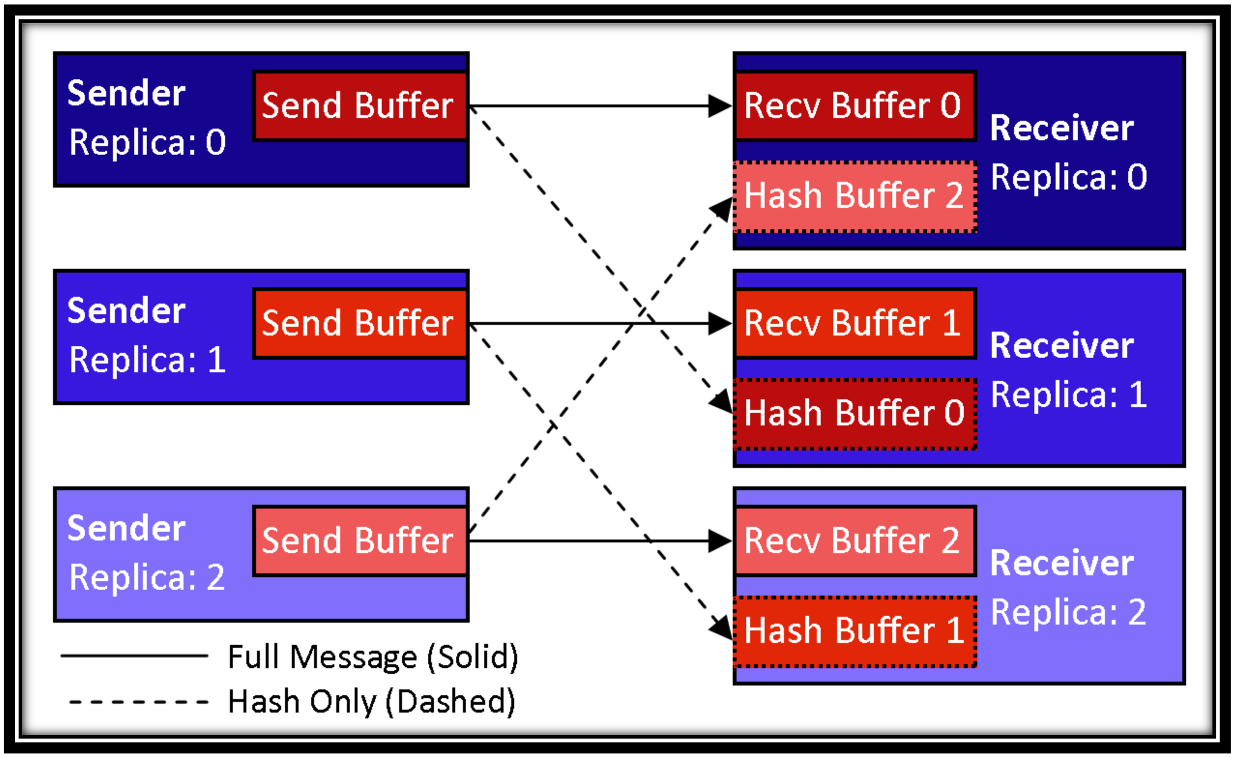
- Requires:
  - *r* receive buffers
  - r^2 messages

- 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)
  ($r_{data}+r_{hash}$)
- More efficient, but requires corruption discovery protocol



Dealing with Non-Determinism

- SomeMPI 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

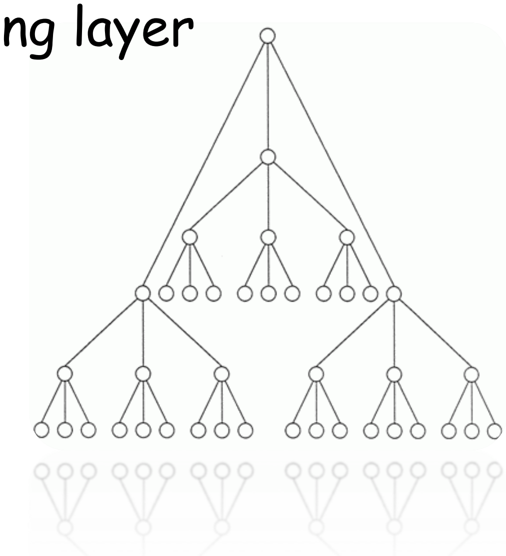
- MPI implementations employee collectives
  — broadcast, reduce, etc.

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

- Collectives are blocking - impossible to overlap

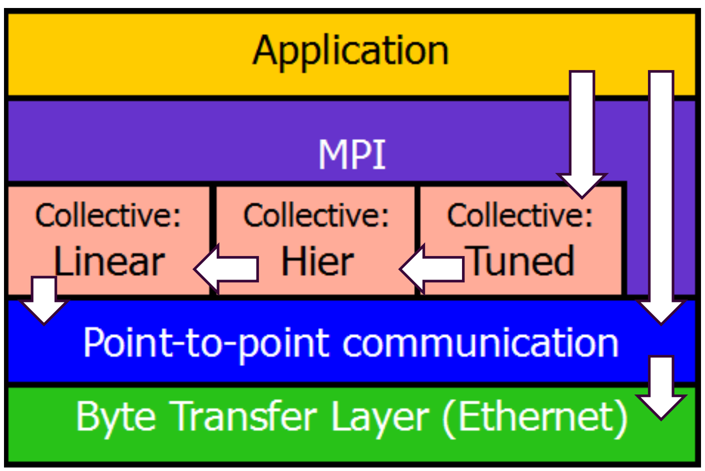
- RedMPI implements own ***linear*** collectives
  — not necessarily performant for large jobs

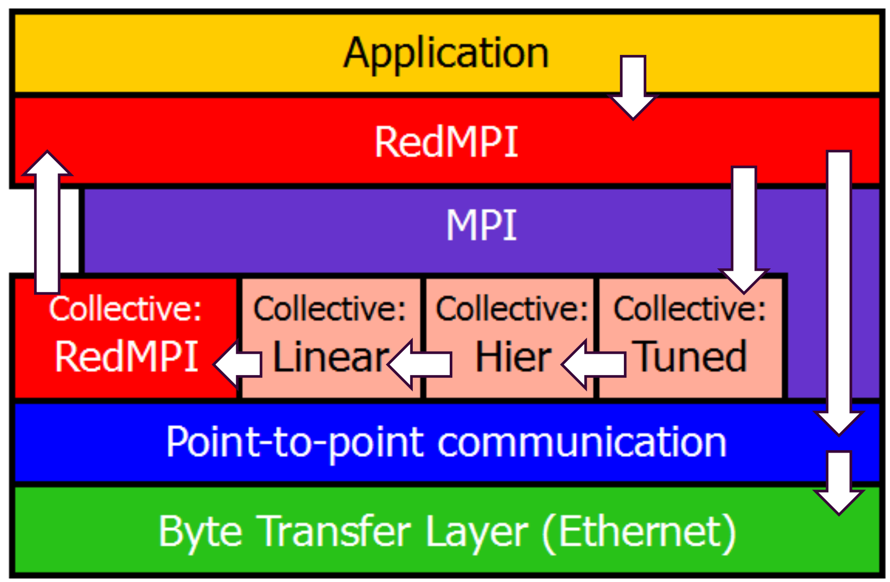
  — Aforementioned pt-to-pt protection is used



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

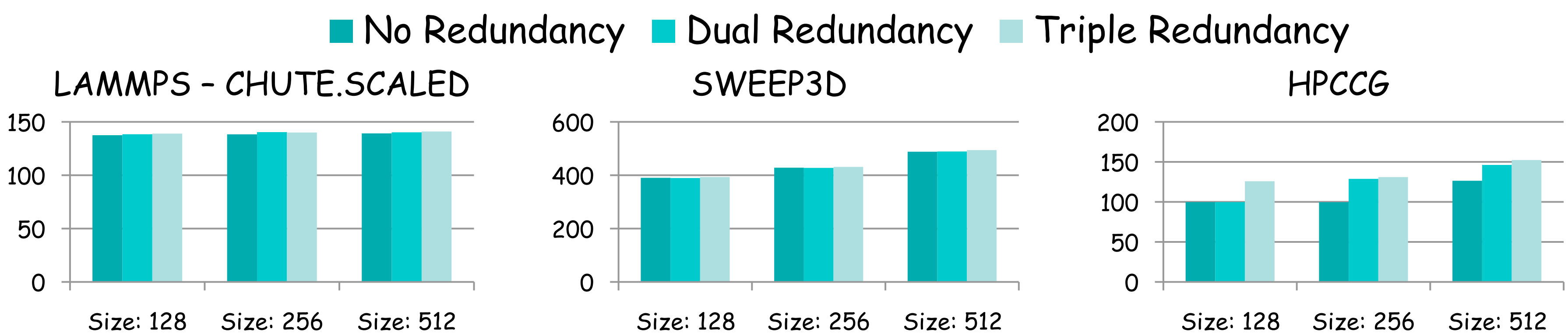




Experimental Framework

- 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

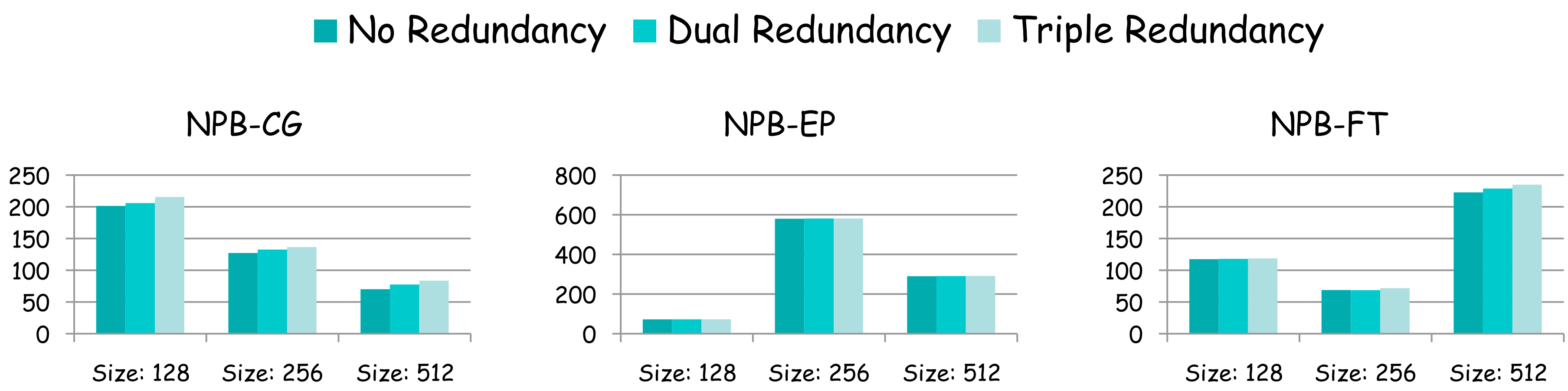
Results: Benchmarks (Weak Scaling)



HPCCG – wildcard receives present					
Size	1x [sec]	2x [sec]	3x [sec]	2x OV	3x OV
128	99.8	99.8	125.8	0.0%	26.0%
256	99.6	128.8	131.0	29.3%	31.5%
512	126.4	146.2	152.3	15.7%	20.5%

- Increased job size ➔ Comm/Comp ratio same
- Negligible overhead for weak scaled apps

Results: Benchmarks (Strong Scaling)



0-20% overhead

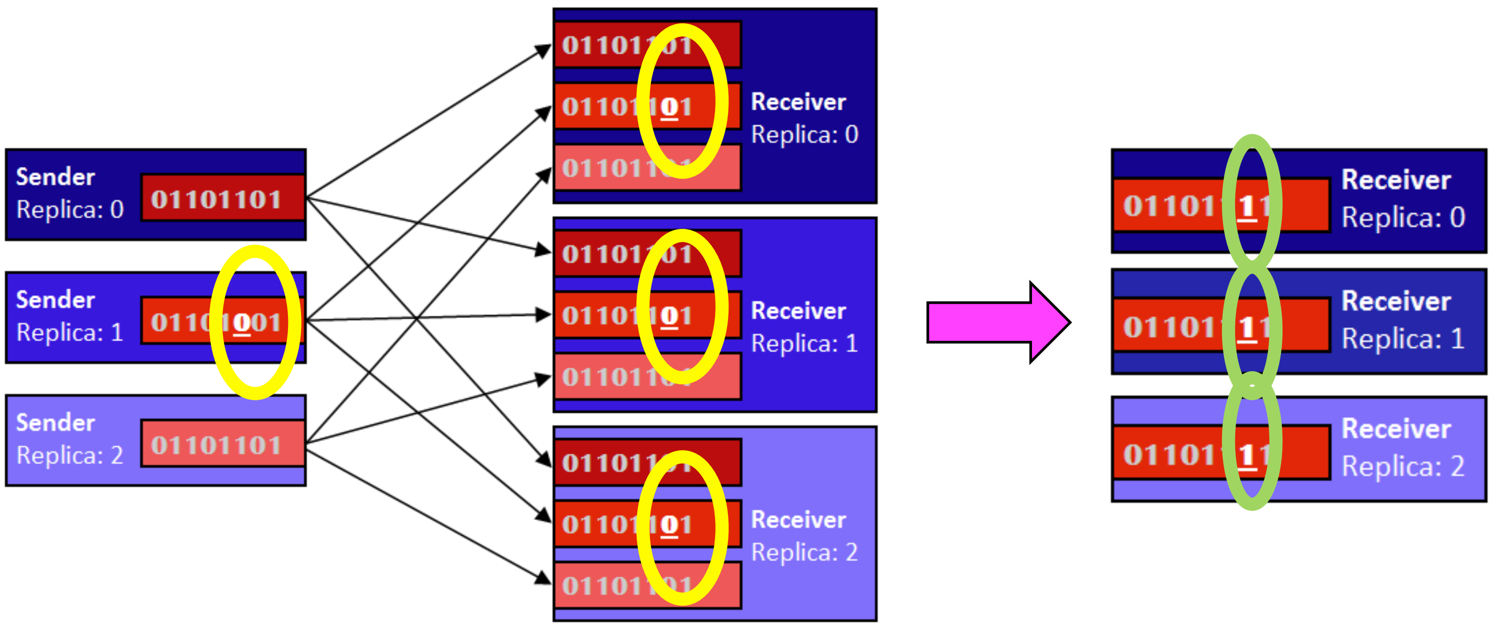
<1% overhead

0-5% overhead

- Increased job size ➔ 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

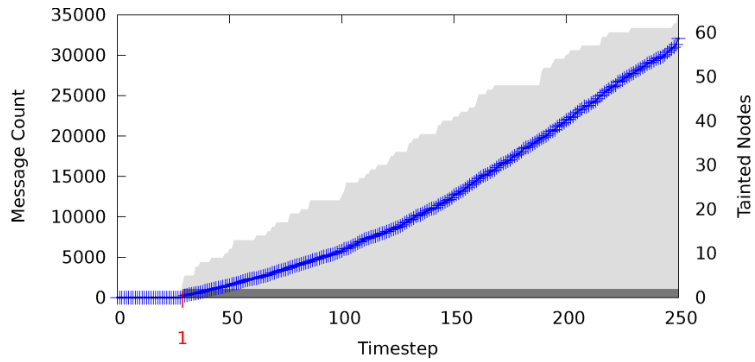
Retains only correct msg

Fault Injection Experiments (TMR)

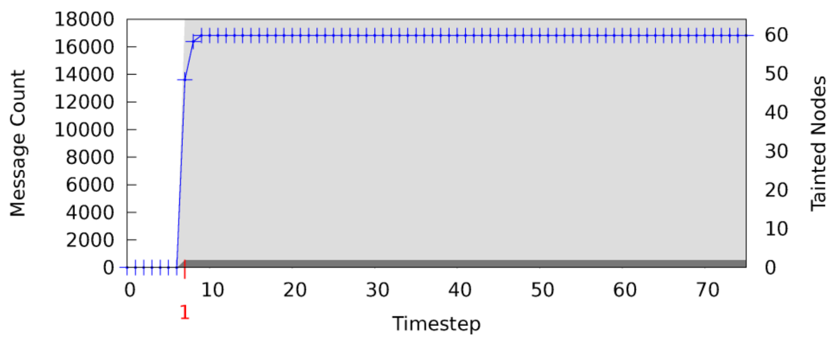
- High injection rates → good stress test
- Experiment #1:
  - Injection rate: **1 bit flip / 5 million messages**
  - 9/10 runs ➔ 1 corrected message
  - 1 run ➔ 6,242 bad messages
    - Likely due to data reuse in corrupted send buffer
  - All runs pass benchmark's built-in verification
- Experiment #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

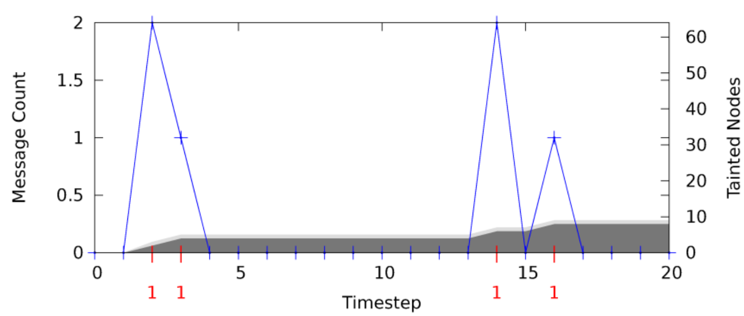
- Progressive



- Explosion

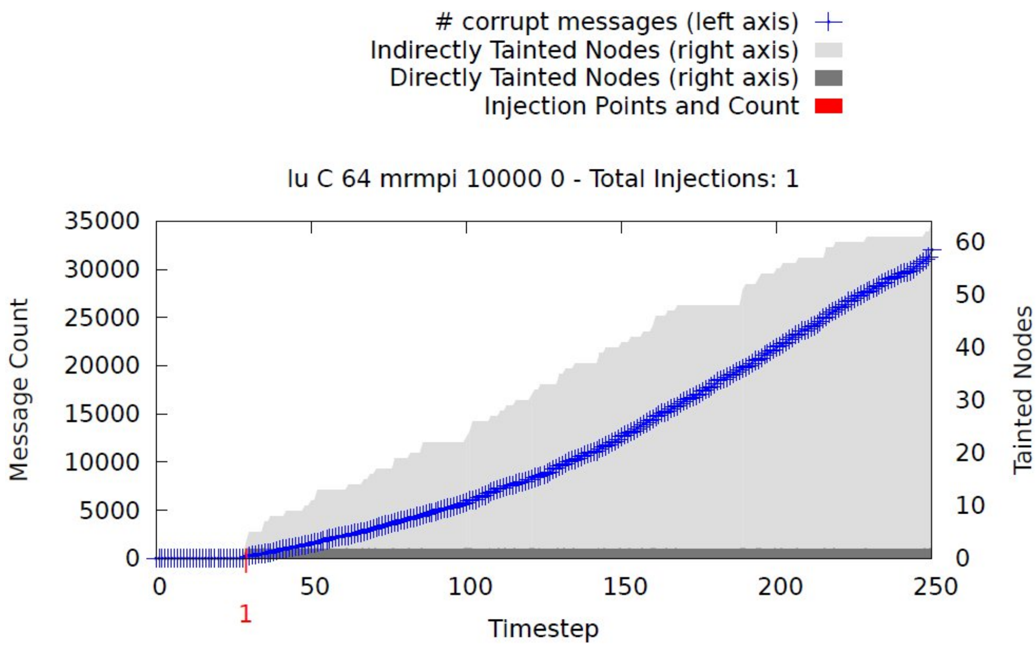


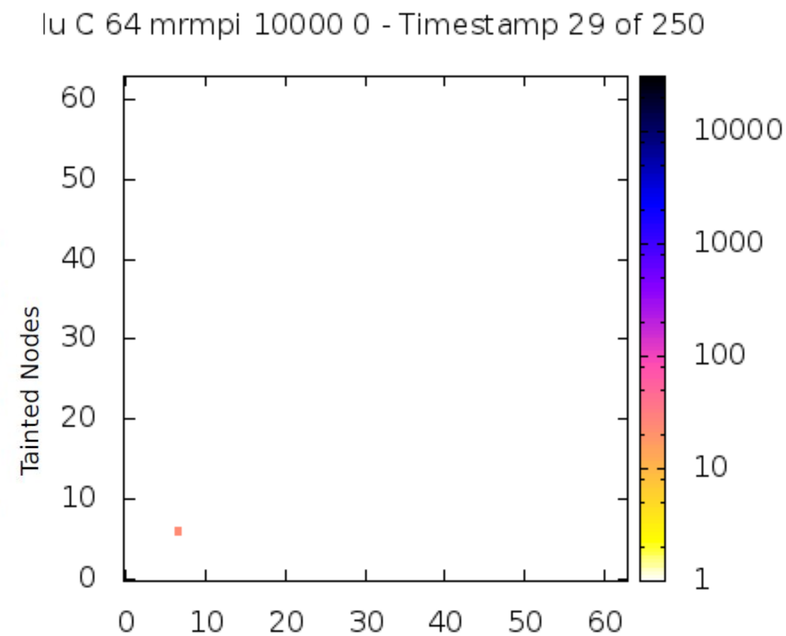
- Localized



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 in to Open MPI provided key performance enhancements over the fallback linear collectives

- Average overheads of select benchmarks using linear fallback:

	Dual Redundancy	Triple Redundancy
NPB CG	44%	53%
NPB LU	10%	19%
SWEEP3D	18%	23%

- Average overheads using RedMPI's enhanced collectives:

	Dual Redundancy	Triple Redundancy
NPB CG	6%	11%
NPB LU	8%	10%
SWEEP3D	0%	1%

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

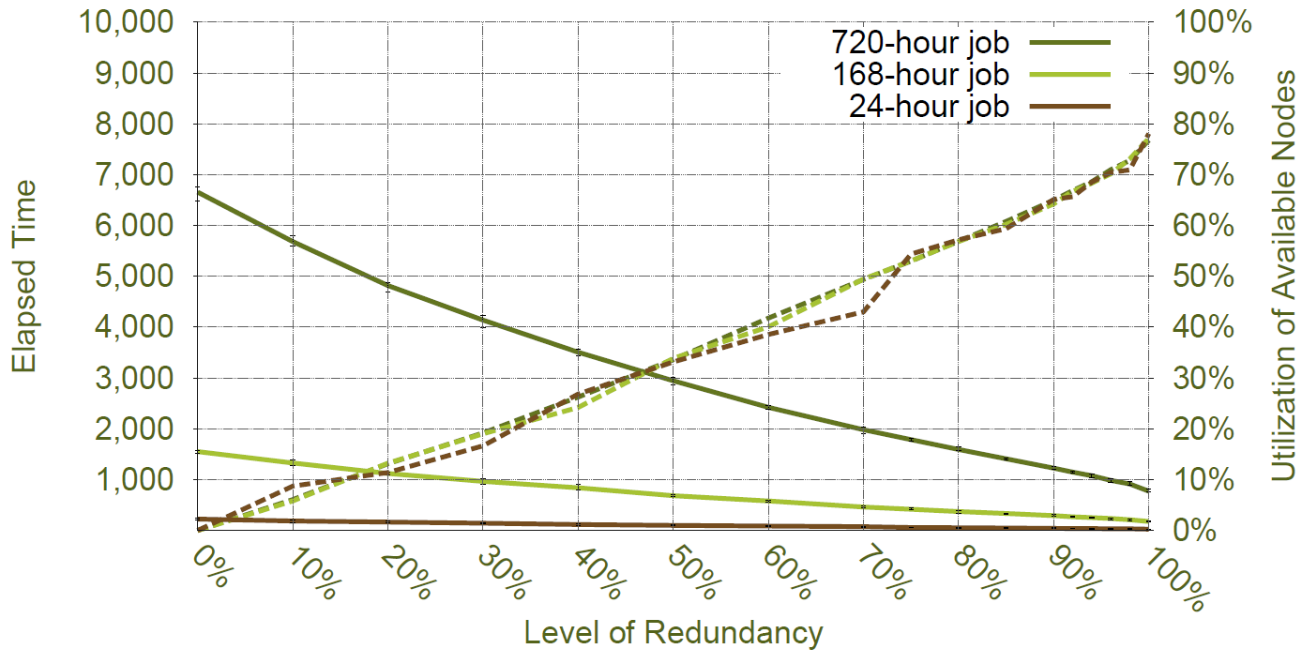
- Faults are now the norm for High Performance Computing (HPC)
  - Past reports attribute causes to hardware and software
    - I/O, memory, processor, power supply, switches
    - OS, runtime, unscheduled maintenance
  - Recent work finds that
    - Servers have a 2-5% failure rate
    - DRAM errors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

RELIABILITY OF HPC CLUSTERS

System	# CPUs	MTBF/I
ASCI Q	8,192	6.5 hrs
ASCI White	8,192	5/40 hrs ('01/'03)
PSC Lemieux	3,016	9.7 hrs
Google	15,000	20 reboots/day
ASC BG/L	212,992	6.9 hrs (LLNL est.)

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

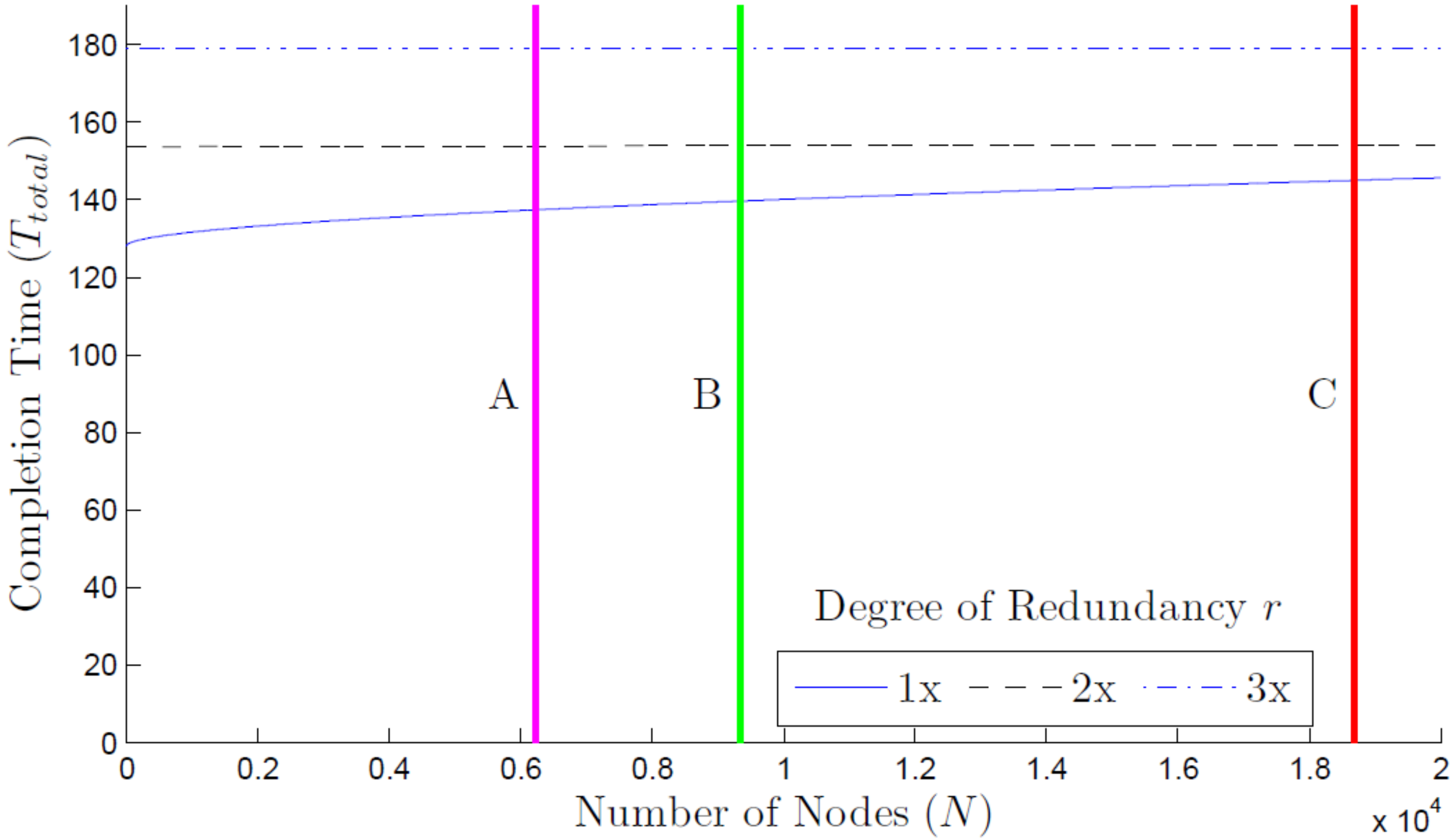
  - Similar to MPI_Wtime, have replica rank 0 decide

  - **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



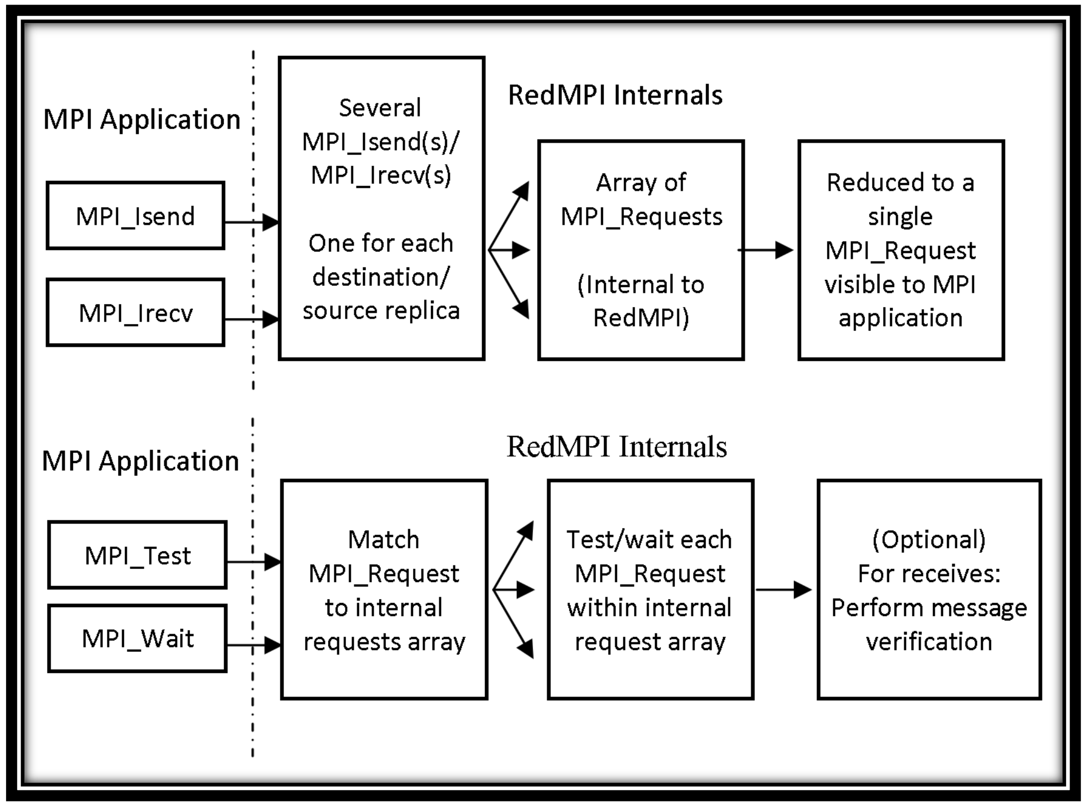
- Modeled Time to Completion with Redundancy
- B: ½ node count Jaguar   A: 1/3 node count Jaguar

All-to-all function interposition

- Each MPI request is converted in to *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