On Determining a Viable Path to Resilience at Exascale

Frank Mueller
Dept. of Computer Science
North Carolina State University

NC STATE UNIVERSITY
Department of Computer Science

Resilience in HPC

System	# CUPs	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

- HPC: 10k-100k nodes
  - Some component failure likely
  - System MTBF becomes shorter
  - processor/memory/IO failures
- MPI widely used for scientific apps
  - Problem w/ MPI: no recovery from faults in the standard
- Currently FT exist 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
  - issues: checkpoint at what frequency?
- 100 hr job → +150 hrs for chkpt / 55%-85% time wasted [Philp'05,Daly'08]

Exascale Resilience

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

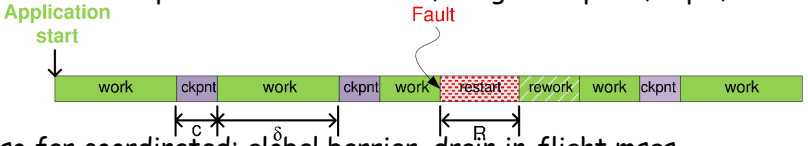
System attributes	2010	"2015"	"2018"
System peak FLOPS	2 Peta	200 Peta	1 Exa
Power	6 MW	~15 MW	~20 MW
System memory	0.3PB	5 PB	32-64PB
Node performance	125 GF	0.5TF or 7 TF	1 TF or 10x
Node memory BW	25GB/s	0.1TB/s or 10x	0.4TB/s or 10x
Node concurrency	12	O(100)	O(1k) or 10x
TotalNode Interconn BW	1.5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

- 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?**

Resilience Advances in HPC (1)

Checkpoint/restart (C/R)

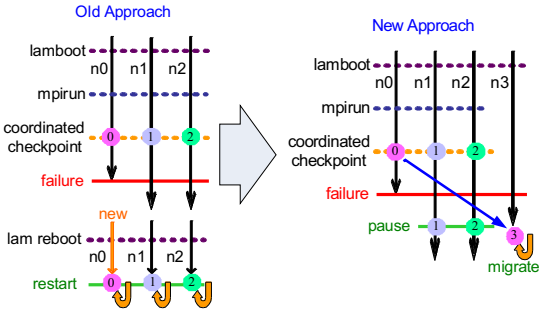
- @ app: intrinsic → -high dev cost, +small footprint, -unoptimized
- @ app: API → medium dev cost, +small footprint, +optimized
  - SCR (LLNL): local disk/SSD/RAM neighbor + RAID: 10X-100X
- @ process: transparent → +low dev cost, medium footprint, +opt.
  - BLCR (LBNL): disk (GFS/local+GFS)
- @ VM: transparent → low dev cost, -large footprint, +opt., -overhead



- so far coordinated: global barrier, drain in-flight msgs
- Uncoordinated: log msgs: +low dev cost, -large footprint, +optimized
  - Send-deterministic: +low dev cost, medium footprint, +opt.
    → control flow to msgs must be input agnostic

C/R: Our BLCR Job-pause Mechanism

- Integrate dynamic group communication (Open MP/LAM+BLCR)
  - Add/delete nodes
  - Detect node failures automatically
- Processes on live nodes remain active (roll back to last checkpoint)
- Only processes on failed nodes dynamically replaced by spares
  - resumed from the last checkpoint

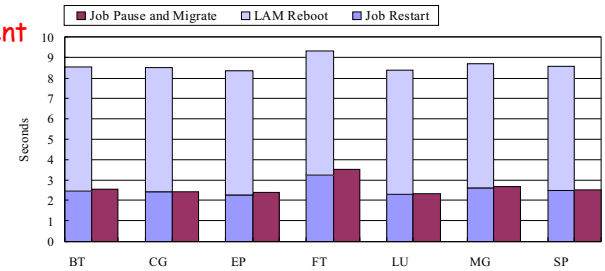


Hence:
- no restart of entire job
- no staging overhead
- no job requeue penalty
- no MPI runtime restart

Contributions (1) [ICS'06+IPDPS'07]

Job-Pause C/R

- Designed for any MPI impl
- Implemented: Open MPI/LAM+BLCR
- Decentralized P2P scalable membership
- Job-pause/rollback for operational nodes
- Restart from chkpt for failed nodes
- Completely transparent → fast ~ 10 sec.
- 69.6% time saved



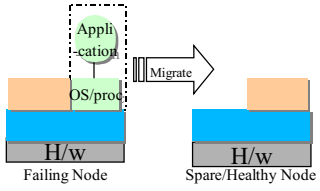
Resilience Advances in HPC (2)

Proactive Fault tolerance

- Migrate when health deteriorates → low dev cost, larger footprint, optimized [e.g., our BLCR extensions]
  - @ process: our work
  - @ VM: Xen, VMWare…

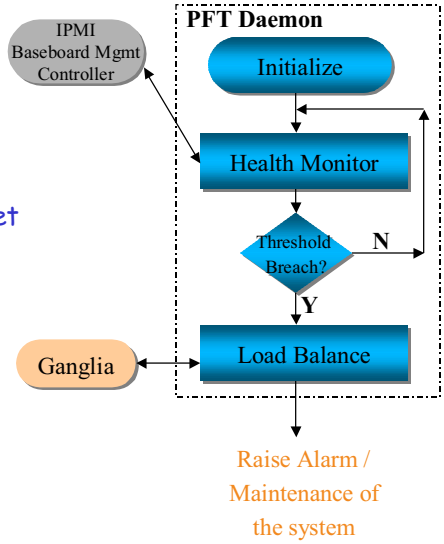
Proactive Resilience: Live Migration

- OpenIPMI health monitoring → predict node failure
- takes **preventive action** (instead of "reacting" to a failure)
  - Live migration of process/OS → healthy node
  - transparent to app/process/OS)
- OS vs. process level: Abstraction vs. overhead tradeoff
  - Copy pages while running
  - Then stop & copy rest
  - Kill src, continue dst
- Implemented over
  1. Xen
  2. Ours: Open MPI/LAM + BLCR + Linux kernel
     - BLCR extensions
     - Kernel enhancements (dirty bit tracking in PTEs)
     - Add'l MPI support



PFTd: Proactive Fault-Tolerance Daemon

- Runs on privileged VM (host)
- OpenIPMI to read sensors
- Periodic sampling of data
- threshold exceeded → control handed over to load balancing
- PFTd determines migration target
- contacting Ganglia → lowest load



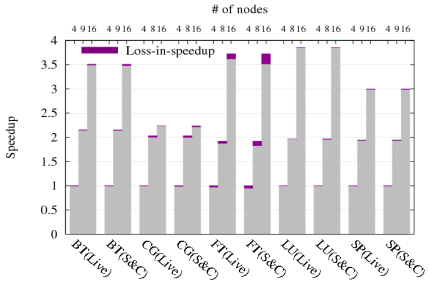
Process vs. OS Migration [ICS'07+SC'08]

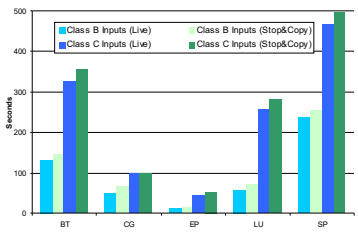
Process-level

- 2.6-6.5 sec live migration
- 1-1.9 sec frozen migration
  - xfer subset of OS image
- 1-6.5 secs prior warning

Xen virtualization

- 14-24 sec live migration
- 13-14 sec frozen migration
  - xfer entire VM image
- 13-24 sec prior warning





Proactive FT Complements Reactive FT

$T_c = \sqrt{2 \times T_s \times T_f}$ **[J.W.Young Commun. ACM '74]**

Tc: time interval between checkpoints

Ts: time to save checkpoint information (mean Ts for BT/CG/FT/LU/SP Class C on 4/8/16 nodes is 23 seconds)

Tf: MTBF, 1.25hrs **[I.Philp HPCRI'05]**

$$T_c = \sqrt{2 \times 23 \times (1.25 \times 60 \times 60)} = 455$$

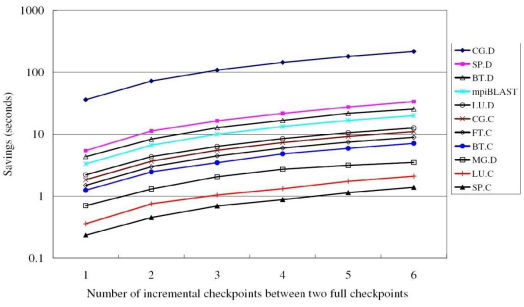
70% faults **[R.Sahoo et.al KDD '03]** can be predicted and handled proactively

$$T_c = \sqrt{2 \times 23 \times (1.25/(1-0.7) \times 60 \times 60)} = 831$$

Cut the number of chkpts in half: 455→831 seconds

Incremental Chkpting **[Linux Symp'11,ICPADS'11]**

- Diff since last chkpt
- BLCR enhancement
- Reuse dirty bit at PTE (h/w support)
- Hybrid: 1 full, k incr. Chkpts
- Model savings [Nakisanehaboon et al.]



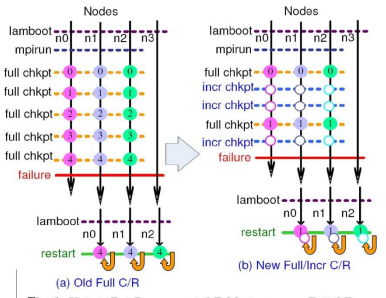
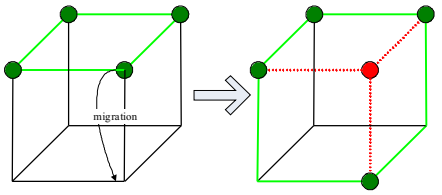


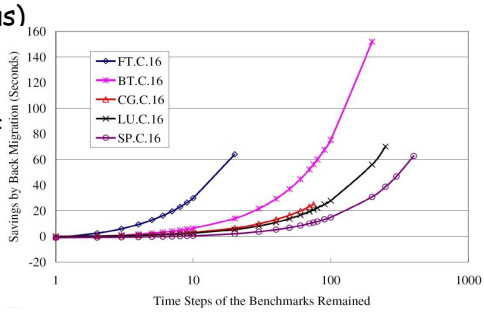
Fig. 1: Hybrid Full/Incremental C/R Mechanism vs. Full C/R

- Reduced I/O pressure
  - Fewer writes
- Faster restarts
  - Fast reads, 1 pass
- 1:9 full/incr. ratio optimal

Back Migration [JPDC'12]

- Node fails → migrate
- Node recovers -> migrate back
- Why?
  - Heterogeneous nodes
  - MPI task sharing on nodes
  - Increased hop counts (torus)
- Experiments
  - slower spare nodes: CPU freq. nearly cut in half
- Benefits
  - Lower MPI cost
  - Reduced I/O bandwidth
- Wins when >10% work left





Contributions (2)

- Reactive FT
  - Save restart cost: **70%** < job queuing, MPI startup
- Novel, proactive fault resilient scheme w/ process live migration
  - Provides transparent & automatic FT for arbitrary MPI apps
  - Less overhead than reactive
  - Also complements reactive → lower checkpoint frequency
  - Process-level: **½ overhead** of OS-level
  - **½ the chkpts** when 70% faults handled proactively
- Incr. Chkpt → less overhead & I/O pressure, **1:9 full/incr. is opt.**
- Back migration → original performance, **wins if >10% work left**

Resilience Advances in HPC (3)

Redundancy: double/triple each MPI task

- Either need 2x/3x more nodes (and 2x/3x # msgs) [our work]
- Or need 2x/3x more bandwidth [SNL]

- Why? (*) [Ferreira at al. SC'11]

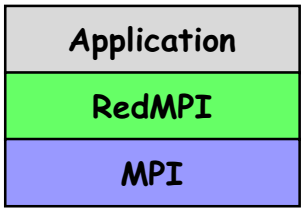
No. of Nodes	Work	Checkpoint	Re-computation	Restart
100	96%	1%	3%	0%
1,000	92%	7%	1%	0%
10,000	75%	15%	6%	4%
100,000	35%	20%	10%	35%

- C/R not scalable: > 50% of time spent in C/R
  - (maybe less due to C/R optimizations)

Sandia National Laboratories

Design of Redundancy: RedMPI [SC'12]

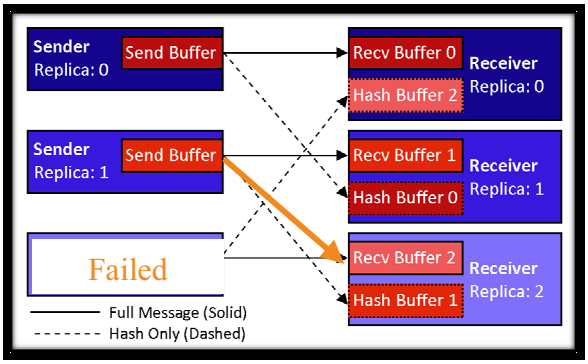
- RedMPI library, related to
  - MR-MPI [Engelmann&Boehm PDCN'11]
  - rMPI [Ferreira et al. SC'11]



- Works at profiling layer
- Goal: guard faults that leak into msgs (IO)
  - file IO also handled [Engelmann PDP'12]
- Intercepts MPI function calls
- Each redundant copy needs to receive same messages in same order
- Each message is sent/received **r** number of times
  - opt. hashes to detect silent data corruption (SDC)
  - Why?Multi-bit flips,DRAM err in 2% of DIMMs/year [Schroeder'11]

RedMPI – MsgPlusHash Protocol [SC'12]

- optimization for critical path : msg not corrupt
- Send r msgs + r small hash messages: $(r_{data}+r_{hash})$
- faster than r^2 msgs
- Patches faulty nodes
- SDC: detect&correct



- **Main objective: catch memory errors (interconnects have CRC)**

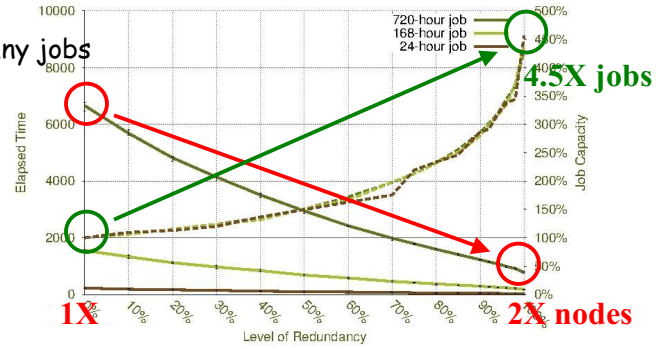
RedMPI Overhead & Benefit

- Overhead: 1-11% time

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

- Benefit:
  at 2X # nodes
  → run 4.5X as many jobs

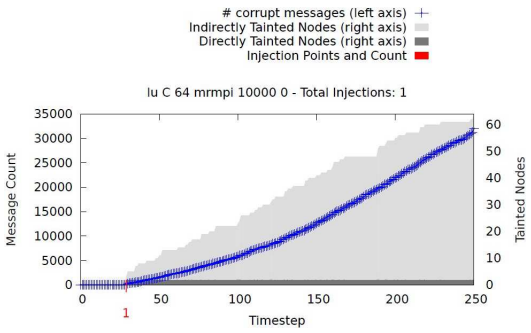
Caveat:
simplistic model
→ fixed next

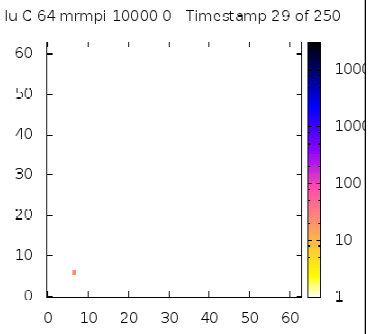


- Exascale: capacity computing ✓, ~~capability computing~~

Fault Injection: SDC Correction (TMR)

- Inject 1 bit flip / 5M msgs
  1. Keep on running: single, corrected msg → 90% of cases, others:
  2. > 1 sent corrupt msgs simultaneously → detected & job failed
  3. Tainted buffer reuse, propagates





Modeling Preliminaries [ICDCS'12]

- A physical process (node) follows an exponential failure distribution
  - θ - Mean Time Between Failures (MTBF)
- A system of virtual processes has an exponential failure distribution
  - Θ - system MTBF
  - **r** - Degree of Redundancy
  - α - Communication to Computation ratio
- Failures arrive following a Poisson process
- **Redundancy increases the system reliability.**

Modeling Preliminaries

- Effect of Redundancy on Execution Time
  - Application execution time ≥ base execution time
  - Dependent upon many factors
    - Placement of processes, communication to computation ratio, degree of redundancy, relative speed, etc.
  - Consider ideal execution environment:

$$\underbrace{t}_{Total\ time} = \underbrace{\alpha t}_{Communication} + \underbrace{(1-\alpha)t}_{Computation}$$

$$t_{Red} = (\alpha t)r + (1-\alpha)t$$

System Reliability Model

- Probability of failure of a physical node:

$$\Pr(Node\ Failure) = 1 - e^{\frac{-t}{\theta}} = 1 - (1 - t/\theta) = t/\theta$$

- Probability of survival of a virtual node with some integer **k** degree of redundancy

$$\Pr(Virtual\ Node\ Survival) = 1 - \prod_{i=1}^{k} t/\theta = 1 - (t/\theta)^k$$

- Partition **N** virtual processes into sets of real-world redundancy levels

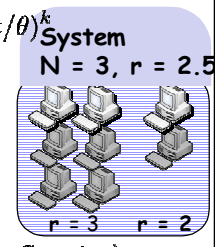
$$N = N_{\lfloor r \rfloor} + N_{\lceil r \rceil}$$

- Reliability of the system may be expressed as

$$\Pr(All\ Virtual\ Processes\ Survive)$$

$$\Pr(All\ N_{\lfloor r \rfloor}\ Processes\ Survive \text{ and } All\ N_{\lceil r \rceil}\ Processes\ Survive)$$

$$R_{sys} = \left[1 - (t_{Red}/\theta)^{\lfloor r \rfloor}\right]^{N_{\lfloor r \rfloor}} \times \left[1 - (t_{Red}/\theta)^{\lceil r \rceil}\right]^{N_{\lceil r \rceil}}$$



System Reliability Model

- Assuming an Exponential distribution,

$$R_{sys} = e^{-\lambda_{sys} t_{Red}}$$

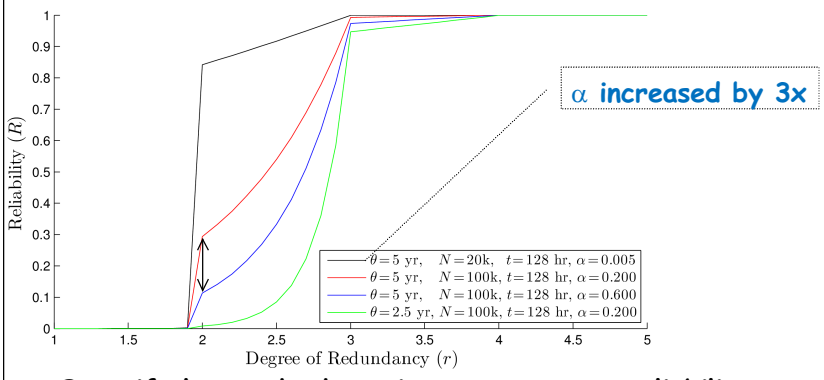
- The system failure rate is

$$\lambda_{sys} = -\ln R_{sys}/t_{Red}$$

- System MTBF is

$$\Theta_{sys} = \frac{1}{\lambda_{sys}}$$

Effect of Redundancy on Reliability [ICDCS'12]



Quantify how redundancy increases system reliability

- Reliability spikes at whole number redundancy levels
- Reliability now depends on α = communicate/compute ratio
  - **Time is a function of alpha**

$$R_{sys} = \left[1 - (t_{Red}/\theta)^{\lfloor r \rfloor}\right]^{N_{\lfloor r \rfloor}} \times \left[1 - (t_{Red}/\theta)^{\lceil r \rceil}\right]^{N_{\lceil r \rceil}}$$

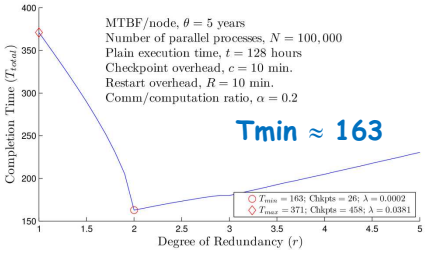
Mathematical Analysis

- Using system MTBF, optimal checkpoint interval may be calculated from Daly (Daly 2003)
- Cost function to compute total wallclock time derived by
  - Computing expected lost work
  - Computing amount of rework using lost work.
  - Total time = t + num_chkpts*chkpt_overhead + rework
- Formally,
  - **c** – time to write a checkpoint to storage
  - **R** – time to load a checkpoint from storage
  - δ **- optimal checkpoint interval**

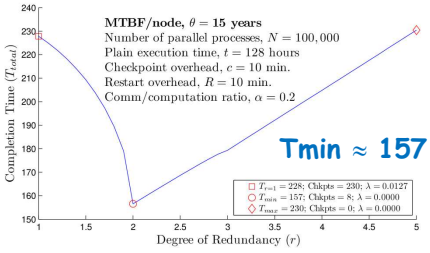
$$T_{total} = \frac{t_{Red} + \frac{t_{Red} \times c}{\delta}}{1 - \lambda_{sys} \times t_{ReWork}}$$

Model Evaluation

Base Configuration



Increased node MTBF



Minimum runtime similar, even though components are 3x less reliable.

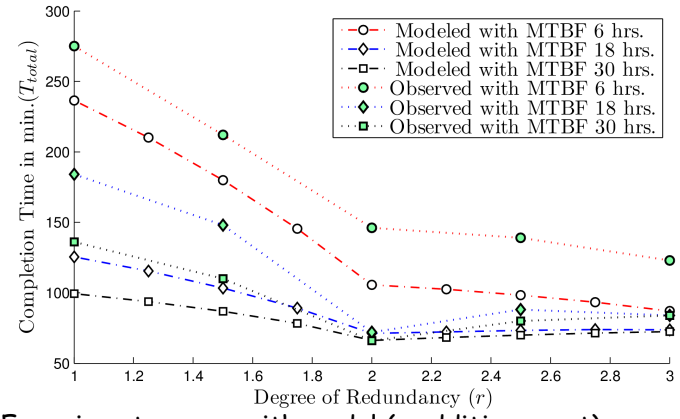
Experiments Redundancy + C/R

Optimal: Depends on MTBF

- Lower MTBF - 3x optimal redundancy
- Higher MTBF - 2x optimal redundancy

Redundancy degree MTBF per node	1x	1.25x	1.5x	1.75x	2x	2.25x	2.5x	2.75x	3x
6 hrs	275	279	212	189	146	158	139	132	***123***
12 hrs	201	207	167	143	103	113	***98***	111	125
18 hrs	184	179	148	120	***72***	126	88	80	84
24 hrs	159	143	133	100	***67***	92	78	84	83
30 hrs	136	128	110	101	***66***	73	80	82	84

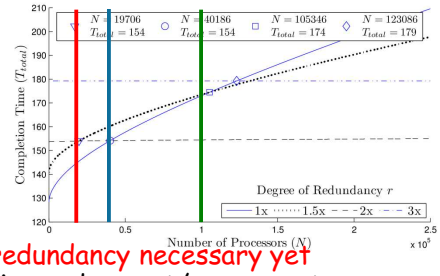
Results – Model vs. Experiment



- Experiments agree with model (+ additive const)
  - **minimum runtime always achieved at 2x redundancy**

Results – Extrapolation based on Jaguar

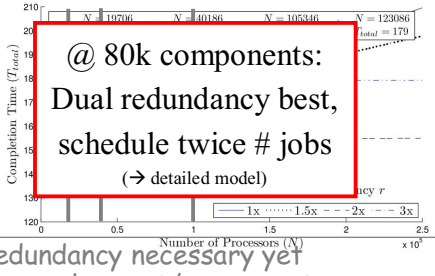
- Jaguar: node MTBF ~ 50 years (on 18,688 nodes)
- K-Computer: has 2.3X more components (equiv. 44,064)
- Exascale lane 1: ~100k nodes



- Jaguar: No redundancy necessary yet
- Titan maintains node count/component
  - **increases core count by 33%, adds GPUs→effect?**
- K-Computer: Dual redundancy break-even
- Exascale: 12% faster under dual redundancy than single,
  - close to triple redundancy for free (free SDC correction)

Results – Extrapolation based on Jaguar

- Jaguar: node MTBF ~ 50 years (on 18,688 nodes)
- K-Computer: has 2.3X more components (equiv. 44,064)
- Exascale lane 1: ~100k nodes

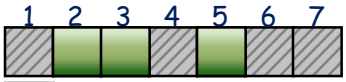


- Jaguar: No redundancy necessary yet
- Titan maintains node count/component
  - **increases core count by 33%, adds GPUs→effect?**
- K-Computer: Dual redundancy break-even
- Exascale: 12% faster under dual redundancy than single,
  - close to triple redundancy for free (free SDC correction)

LIBSDC –Protect Memory from SDC [subm.]

- Page-level on-demand memory verification
- Memory model: Set of locked/unlocked pages
- Locked = memory protected
  - intercepts reads/writes
    1. SDC Detection: Hashes provide verification only
    2. SDC Correction: ECC/Hamming codes correct SDCs
  - Then unlock page

Access pattern: Pages 2,3,2,2,5



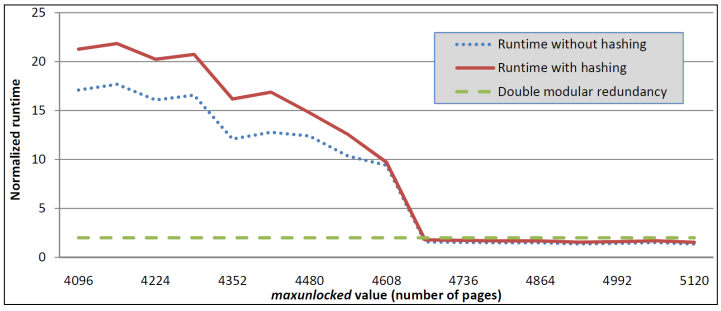
Method	SHA1 Hashing (4KB pages)	72/64 Hamming Codes (ECC)
Storage Overhead	0.49%	12.5%

Locked/Protected Page – LIBSDC must validate before next use

Unlocked/Unprotected Page – Recently validated – may be read/written

Experimental Results

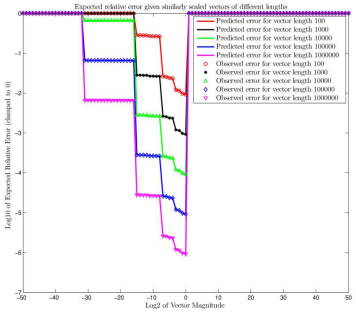
- HPCCG – A Sandia Natl. Labs kernel conjugate gradient solver from the Mantevo Miniapps



- Ideal max-unlocked around 4096-5120 to match working-set size
- On average, about 15% of overhead spent on page hashing

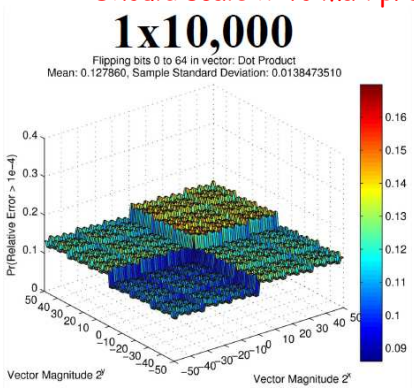
Quantify Impact of SDC on FP Ops [subm.]

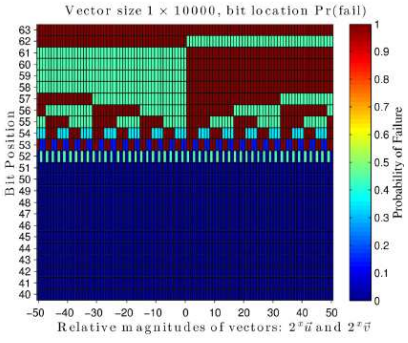
- Model likelihood of bit flip to affect results
  - Mantissa vs. exponent
- For vector dot product (DP): $\vec{u} \cdot \vec{v} = \sum_{i=1}^{N} c_i; \quad \text{where} \quad c_i = u_i v_i.$
- Plot for same |vector|:
  Expected relative error [y axis]
  over vector magnitude [x axis]:
  - (DP – flipped DP) / DP
- Flip lower 10 bits of exponent
  - Spikes due to patterns:
    1023 vs. 1024
    many 1s, few 0s vs. …
- ➢ Model fits experiments



Quantify Impact of SDC on FP Ops (2)

- Monte-Carlo sampling
  - via random # gen.
  - Expected # flips
    → Pr(Error>10^{-4}) [y axis]
- Slice across
  - Similar magnitudes
    (front to back)
- Shows bit position of error
- ➢ Should scale # to max precision→few flips affect you

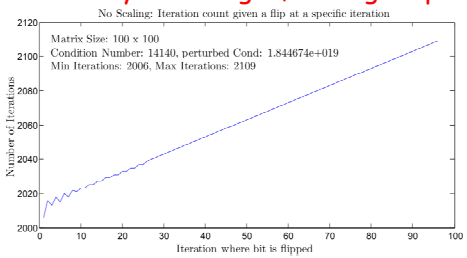


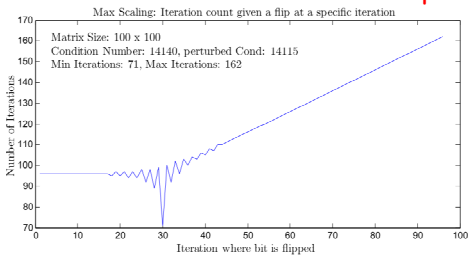


Quantify Impact of SDC on FP Ops (3)

- Order-one iterative methods:
  - Always converges after bit flip
  - How about stationary methods?
- Case study: Jacobi unscaled
  - No fault → 98 iterations
  - Bit fliped @ iteration X
    → converges but 2000+ iters (20x)
- Jacobi max. scaled
  - Anomaly → flips can help !
  - Converges
    → 0-65 more iterations (1x-1.6x)

➢ always converges, scaling helps a lot → reduces overhead after flip





Contributions

1. Scalable network overlay (ICS'06)
   - track live nodes, group communication
2. Reactive fault tolerance (IPDPS'07, Linux'11, ICPADS'11)
   - job pause → 70% reduced resubmit overhead
   - Incr. Chkpts → 1:9 full/incr. Ratio best, reduce I/O
3. Proactive fault tolerance (ICS'07, SC'08, JPDC'12)
   - process virt. → ½ overhead of OS, health monitor
   - live migration → ½ # chkpts
   - back migration → wins if >10% work left
4. Redundancy + SDC Handling (ICDCS'12, SC'12)
   - 2x # nodes → 2x # jobs: capacity not capability comp.
   - dual for SDC check / triple SDC correction (msgs, RAM, I/O)
5. Algorithm-based Fault tolerance (Chen & others, subm.)
   - Complements above, sign. less overhead, only dense linear algebra
   - Modeling of SDC for numerical algorithms

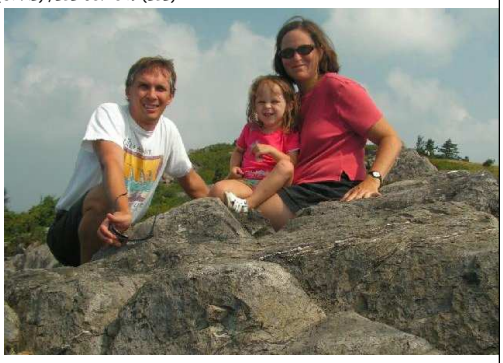
➢ Code contributed to BLCR, available for Open MPI, later RedMPI

Acknowledgement

Supp. in part by DOE/NFS grants, Humboldt fellowship

DOE DE-FG02-05ER25664, DE-FG02-08ER25837, DE-AC05-00OR22725, NFS 0237570, 0410203, 0429653, 1058779, 0958311, 0937908
DOE DE-AC04-94AL85000 (SNL) , DOE DE-AC05-00OR22725 (ORNL) , LBL-6871849 (LBL)

- NCSU: J. Varma, A. Nagarajan, Chao Wang (ORNL), M. Vasavada, K. Kharbas, D. Fiala, J. Elliott
- ORNL collaborators: Christian Engelmann, Stephen L. Scott
- LBL: Paul Hargrove
- SNL: Kurt Ferreira, Ron Brightwell
- IBM: Rolf Riesen



(4) Availability: Fault Tolerance for YOU

- BLCR: Collaboration w/ Paul Hargrove, Eric Roman… (LBL)
- BLCR (in repository / partly released):
  - Job Pause/Rollback → merged into V0.8.0, next BLCR release
  - Incremental checkpoints → in progress
    - Dirty vs. write-protect bits
    - Challenges: copy-on-write (fork), mprotect, map/unmap
    - Assess overheads of alternatives
  - Differential checkpoints → next release
  - Region-delimited checkpoints (semi-transparent) → next rel.
  - OpenMPI, MVAPICH2, Cray MPI, SLURM support, LSF plugin
  - Accepted by Debian + Fedora
- Open MPI support (not in repository yet):
  - Job pause, incremental chkpt, fault detector, live migration

Discussion

- Need CS cluster resources w/ root access for scaling
  - Later production deployment
- Job scheduler support for FT
  - Spare node pooling
- Chkpt PFS and I/O requirements
- Need studies on potential to detect health deterioration
  - Anomaly detection / threshold derivation
    - In progress: transparent fault detector for MPI

Discussion (cont.)

- MPI wish list:
  - Coordinated checkpointing:
  - MPI_QUIESCE_START/END(comm, info) → drain comm. Qs
    - Low-level control, MPI integrated
    - https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/Quiescence
  - MPI_CHKPT(bool force) → optional trigger
    - Higher-level abstraction
    - User/runtime defined, opt. MPI integrated
      - Just like MVAPICH2_Sync_Checkpoint();
- Checkpt in no more than **m** / every **m** minutes
  - MPI_NEXT_CHKPT(struct timespec abstime)
  - MPI_CHKPT_FREQ(struct timespec reltime)