Combing Partial Redundancy and Checkpointing for HPC

James Elliott, Kishor Kharbas, David Fiala, Frank Mueller, Kurt Ferreira, and Christian Engelmann

North Carolina State University

Sandia National Laboratory

Oak Ridge National Laboratory

NC STATE UNIVERSITY
Department of Computer Science

Motivation

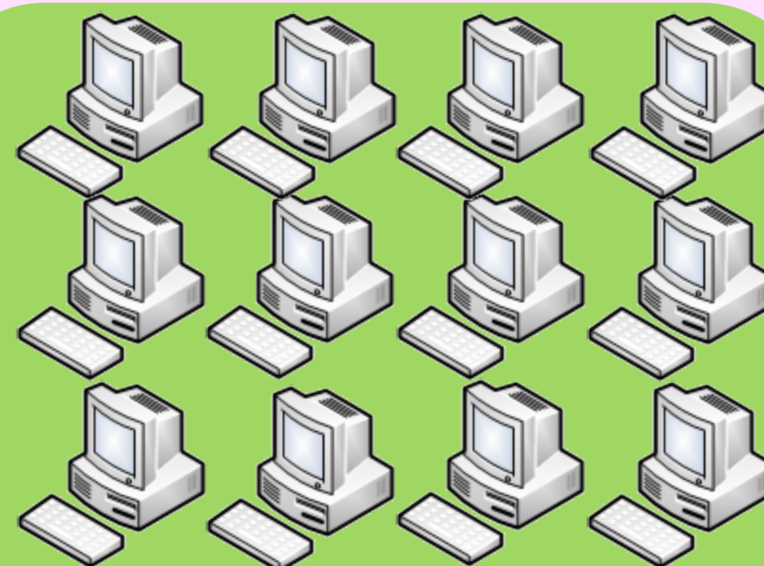
- Target:
  - High Performance Computing (HPC)
  - Assumes *capability computing* (uses entire system)
- Systems classified by floating point operations per second (FLOPS)
  - teraflop : 10^{12}; petaflop : 10^{15}; exaflop : 10^{18}
  - terascale 1990s, petascale 2008-, exascale ?
- Trends
  - Roadrunner: 1 petaflops 2008
  - K: 10 petaflops 2011
  - Sequoia: 16.32 petaflops 2012
  - Exascale by 2020
    - Top500 projection

Motivation

Assumes capability computing (using entire system)

Components have reliability

- Reliability follows a statistical distribution e.g., Exponential
- Mean Time Before Failure
  MTBF denoted as θ
- Nodes form a **system**, with **system MTBF** Θ

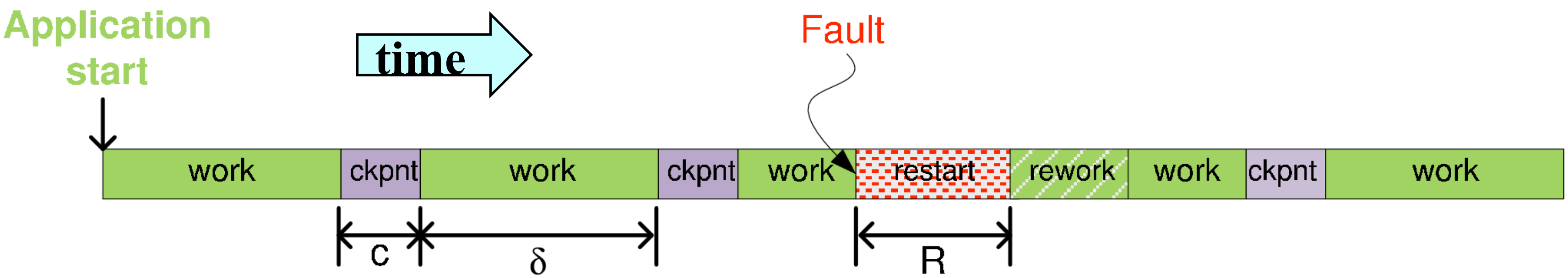


Θ = 43,800 hr.

Θ : <u>**System**</u> MTBF

Assume a node has 5yr MTBF (θ = 43,800 hours)

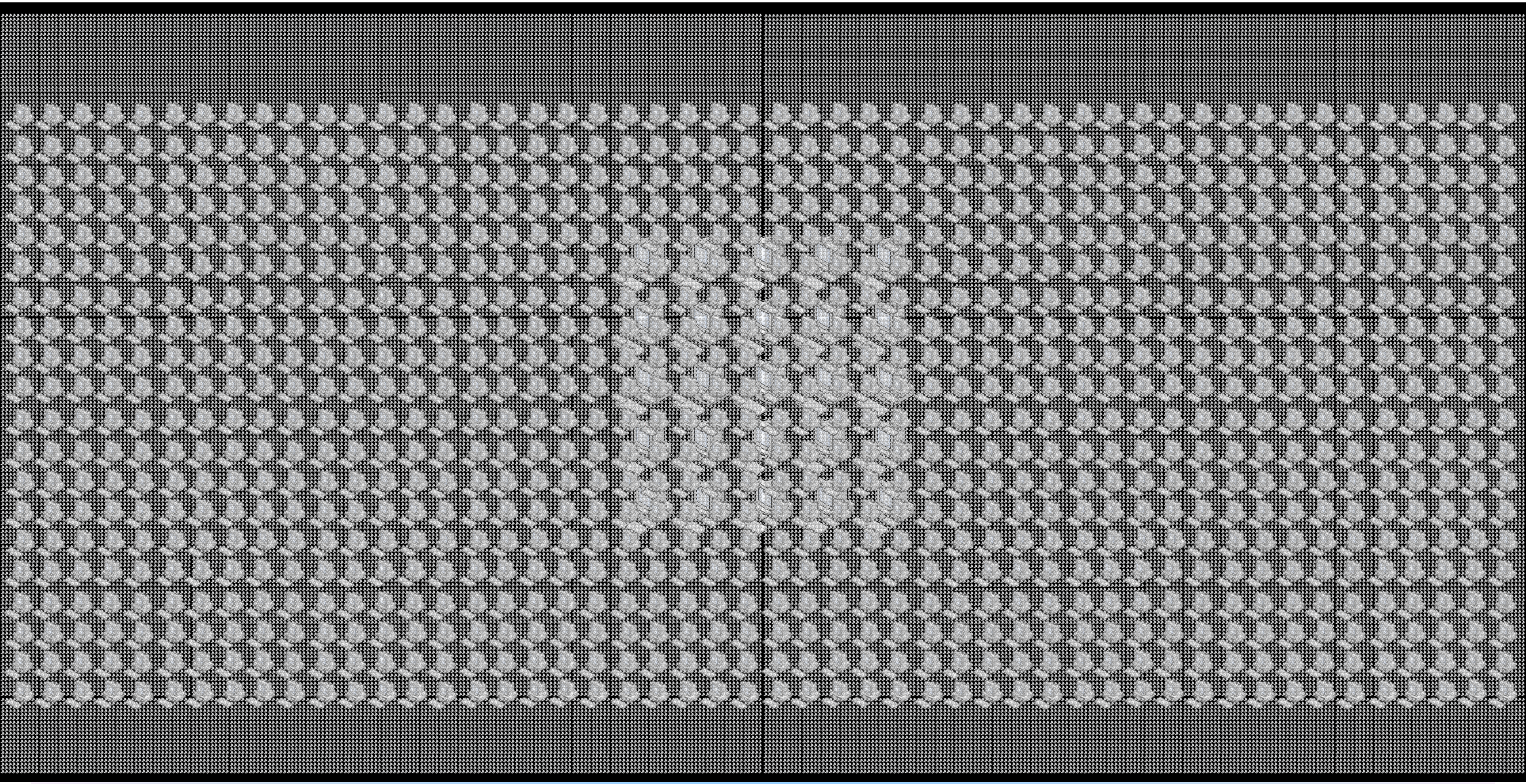
Motivation



Checkpoint Restart (C/R)

- Enable unreliable systems to complete jobs that exceed the system's reliability.
  e.g., job runtime > system MTBF
- C/R has no impact on system reliability
- ***Any component fails => application fails***
- Idea: periodically save state (**checkpoint**), if failure occurs: load prev chkpt and **restart**
- I/O from parallel file system (not local disk)

Motivation



N = 250,000 Θ = 1.75 hr. δ = 6 min.

At petascale 50yr node MTBF (438,000 hours)

Motivation

- Scalability limitations of Checkpoint/Restart

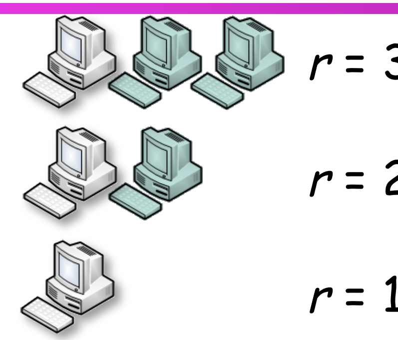
No. of Nodes	Work	Checkpoint	Re-computation	Restart
[illegible]	[illegible]	[illegible]	[illegible]	0%
1,000	92%	7%	1%	0%
10,000	75%	15%	6%	4%
100,000	35%	20%	10%	35%

Less than 50% time spent doing meaningful work

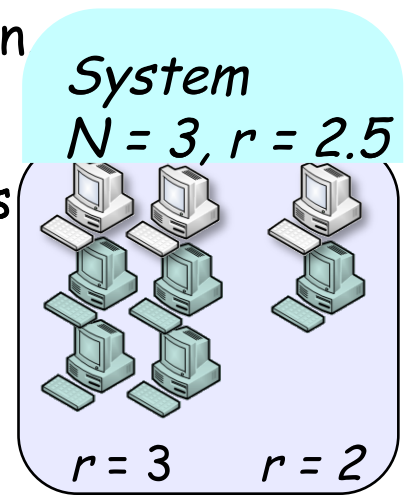
- Redundancy is expensive: **Is it advantageous to use various degrees of redundancy in conjunction with C/R to minimize job execution time?**
- **Can this relationship be modeled analytically?**
- **What are the *optimal parameters* for degree of redundancy and checkpoint interval to achieve the *lowest wallclock* time?**
- **Goal: maximize time spent in useful application work**
  - **not fault tolerance code.**

Redundancy and Partial Redundancy

- Virtual process: contains ***r*** physical processes
  - — in a parallel (redundant) configuration.



- ***r*** *:= degree of redundancy.*
  - — State machine replication.
  - — Active and redundant nodes perform same computation.
  - — Upon failure, replica process takes over execution
  - — Substantial **increase** in process MTBF.

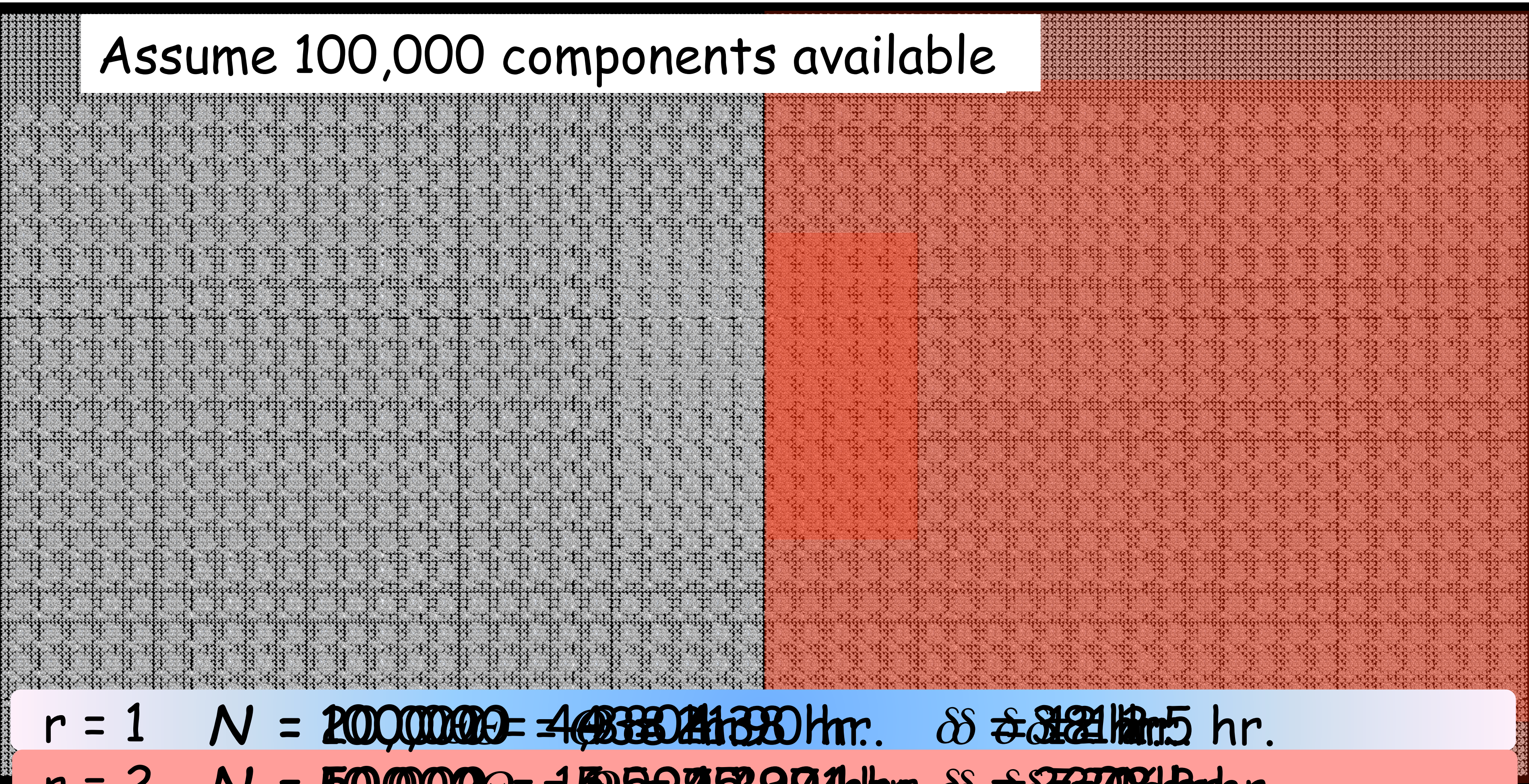


- A system of ***N*** virtual processes connected in a series configuration (**single failure = total system failure**)
- Traditional redundancy: all *N* virtual processes

  have same *r* and *r* must be a positive integer.
- **Partial redundancy**, *N* virtual processes have **ceiling(r)** or **floor(r)** level of redundancy, *r* may be a real number ≥ 1

Motivation Revisited

Assume 100,000 components available



r = 1 N = [illegible] θ = [illegible] hr. δ = [illegible] hr.

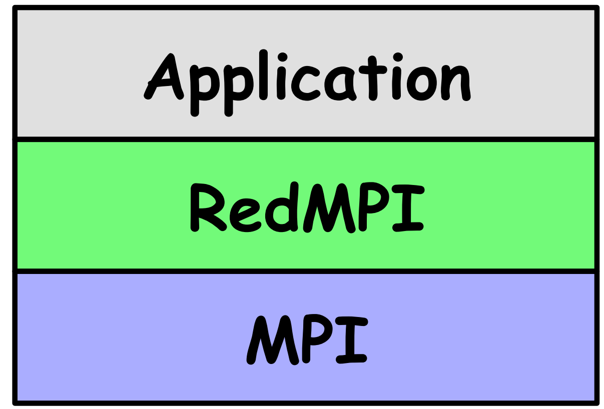
r = 2 N = [illegible] θ = [illegible] hr. δ = [illegible] hr.

(r = 1 N = [illegible] θ = [illegible] hr. δ = [illegible] hr.)

Redundancy improves system reliability

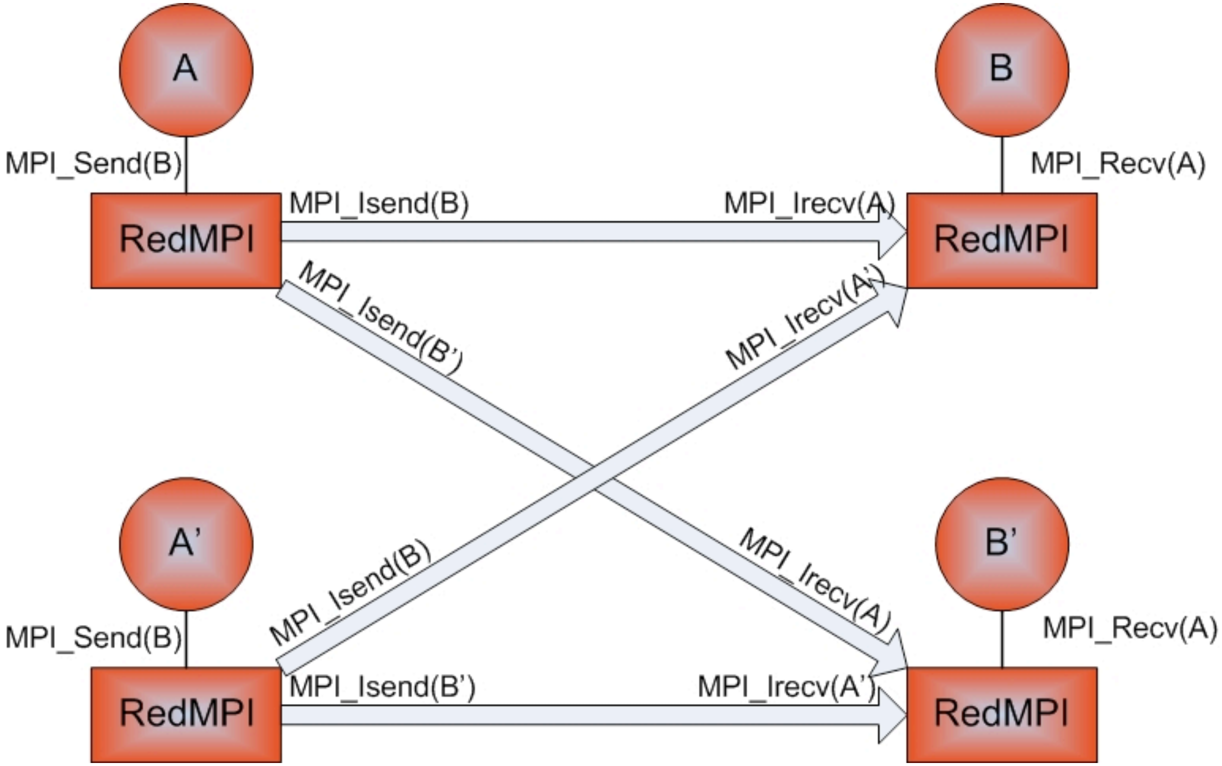
Design of Redundancy

- RedMPI library



- Works at profiling layer
- **Goal**: ensure output is correct
  - Related work already handles file IO (Böhm and Englemann '12)
  - We focus solely on MPI messages
- Intercepts MPI function calls
- MPI_Comm_rank() returns same value for replica processes
- Each redundant copy needs to receive same messages in same order
- Each message is sent/received *r* number of times.

Design of Redundancy: Blocking MPI P2P calls



- MPI_Send() -> MPI_Isend()s
  MPI_Waitall()
- Allocation of additional buffers

Design of Redundancy: Other MPI functions

- Non-blocking MPI calls
  - maintain list of MPI_Requests

- Collectives : e.g. MPI_Bcast(), MPI_Alltoall()
  - use redundant point-to-point calls

- Same info return by MPI_Probe() , MPI_Test() and MPI_Wtime() functions

Modeling Preliminaries

- 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:

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

$$\Pr(Node\ Failure) = 1 - (1 - t/\theta) = t/\theta$$

- 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

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

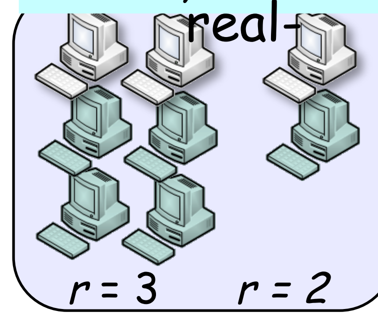
- Reliability of the system may be expressed as

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

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

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

System
N = 3, *r* = 2.5



System Reliability Model

- Assuming an Exponential distribution,

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

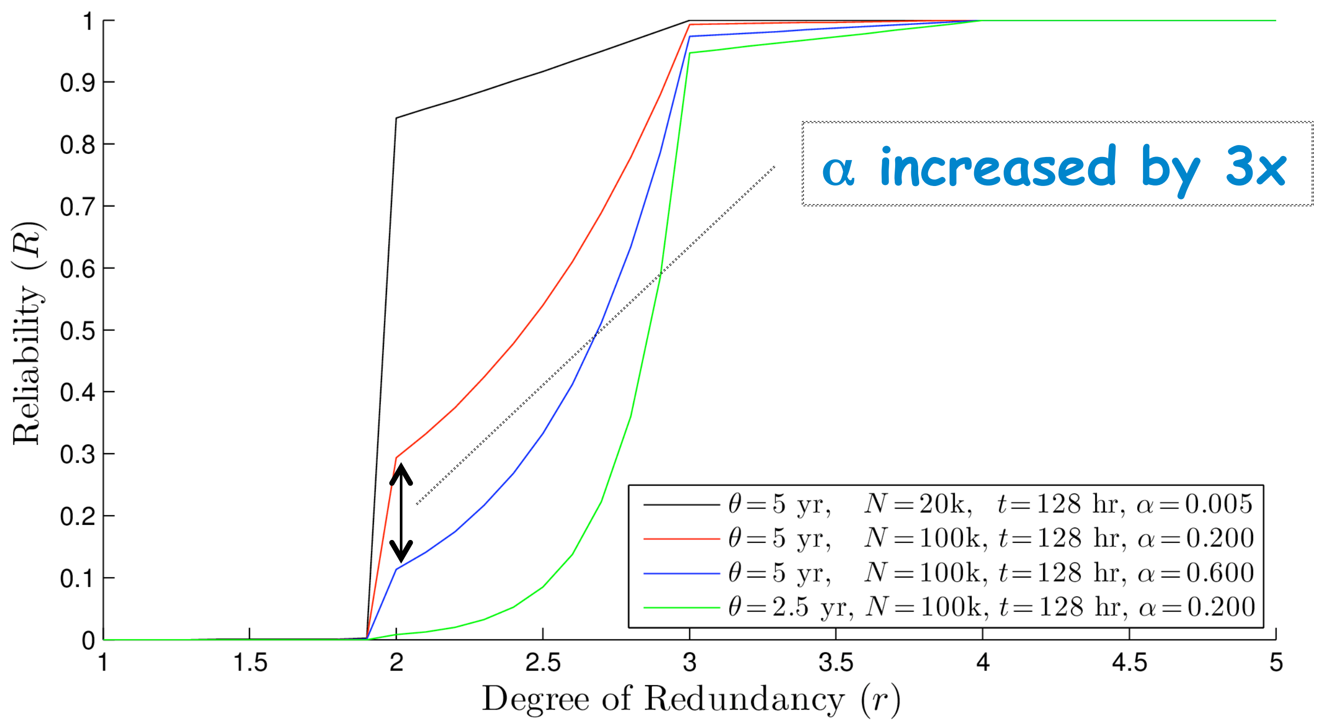
- The system failure rate is

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

- System MTBF is

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

Effect of Redundancy on Reliability



- Reliability spikes at whole number redundancy levels
  - (stepping function as component count increases)
- Reliability now depends on Communication to Computation ratio
  - **Time is a function of alpha**

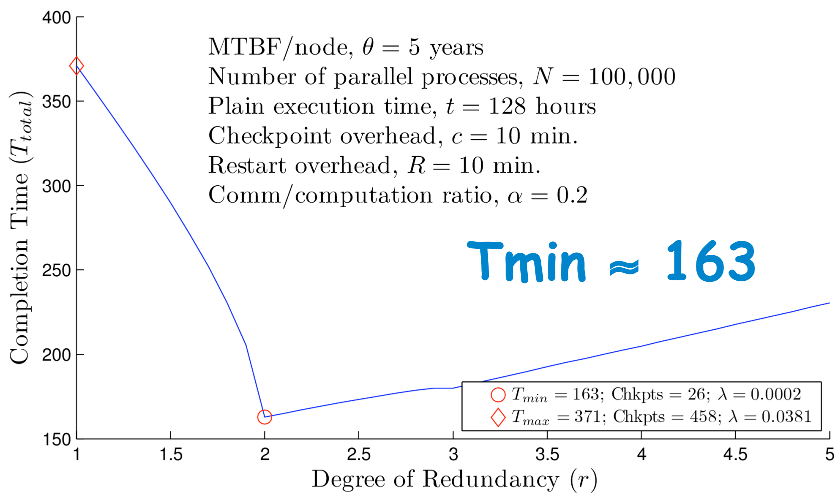
Mathematical Analysis

- 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*

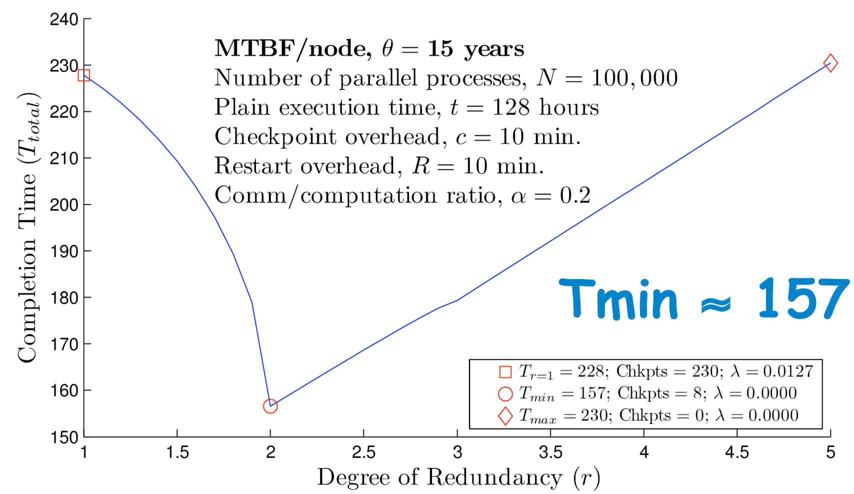
$$T_{total} = \frac{t_{Red} + \frac{t_{Red} \times c}{\delta}}{1 - \lambda_{sys} \times t_{ReWork}}$$

Model Evaluation

Base Configuration



Increased node MTBF

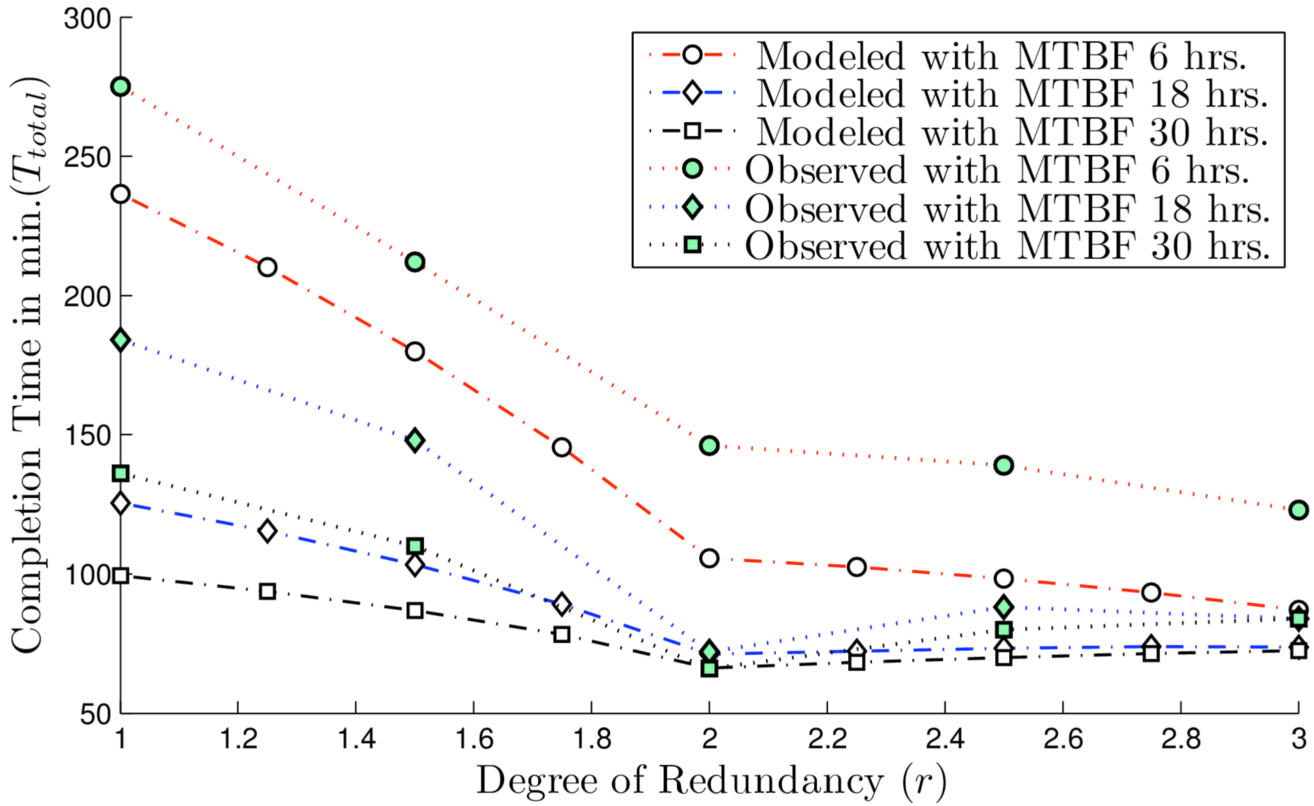


Minimum runtime similar, even though components are 3x less reliable.

Simulation Environment

- Architecture:
  - 108 node cluster (w/ 16 cores each)
  - QDR Infiniband
  - 2-socket shared-memory nodes
  - octo-core AMD Opterons per socket
- OpenMPI, BLCR, RedMPI
- NPB-CG, class D for 128 processes
- Base execution time: 46 min.
- MTBF: 6 hrs, 12 hrs, … 30 hrs
- Redundancy degree: 1x, 1.25x, 1.5x, … 3x

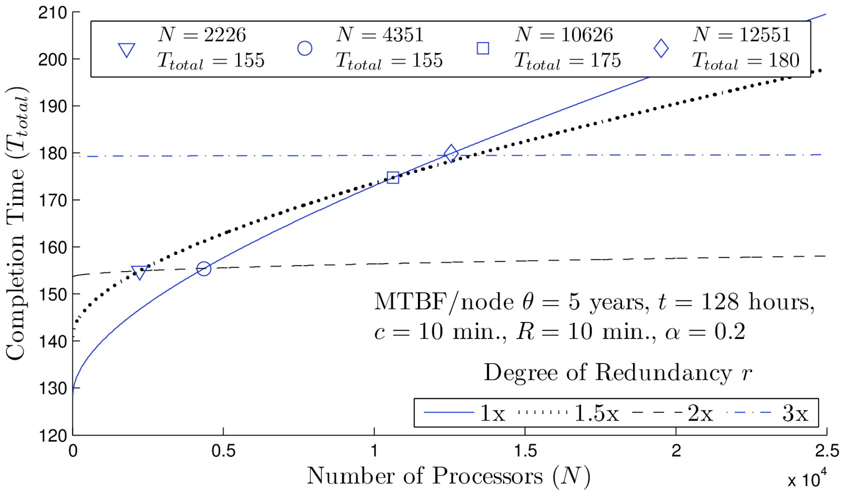
Results – Model vs. Experiment

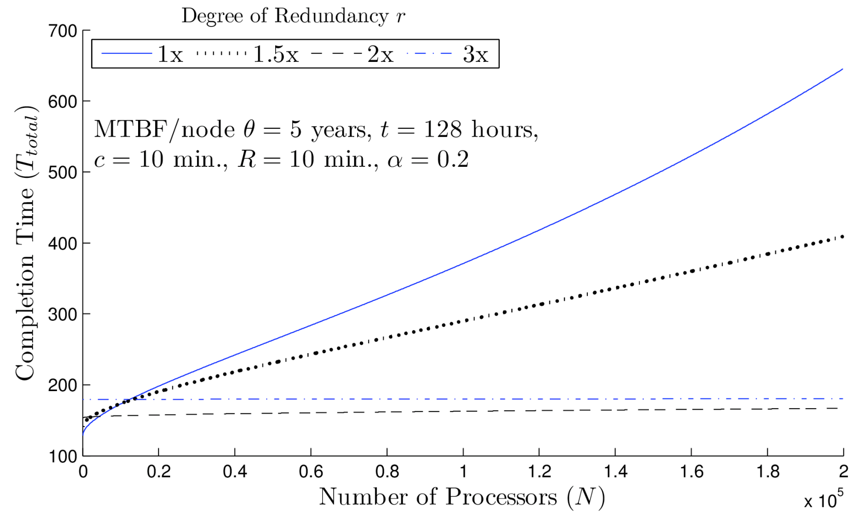


- Experiments agree with model (+ additive const)
  - **minimum runtime always achieved at 2x redundancy**

Results – Optimal Redundancy Level

- Determine when a redundancy level becomes beneficial
  - Assumes weak scaling





- Dual redundancy may be beneficial now
  - At 78,536 processes, **two dual redundant jobs** of 128 hours **can be run in the time of just one job without redundancy**.

Results – Extrapolation based on Jaguar

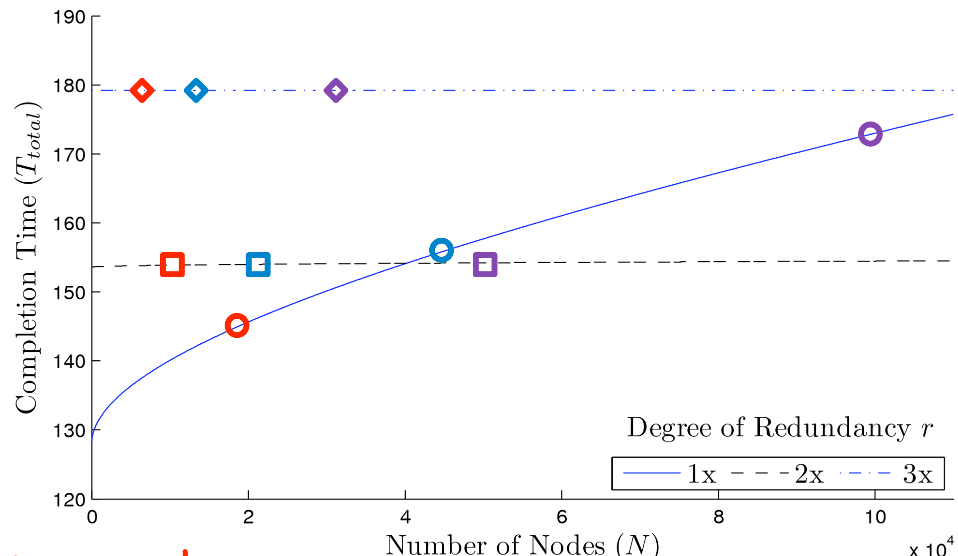
SDC – Silent Data Corruption – bit flips due to radiation, etc,…

ECC (error correcting code), correct single bit flip, detect double, triple

no protection.

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

	○ $r = 1$	□ $r = 2$	◇ $r = 3$
Jaguar	**145** (18,688)	154 (9,344)	179 (6,229)
K	156 (44,064)	154 (22,032)	179 (14,688)
Exascale	173 (100,000)	**154** (50,000)	179 (14,688)



- Jaguar: No redundancy necessary yet
- Titan maintains node count/component
  - **increases core count by 33%, adds GPUs→effect?**
- K-Computer: Dual redundancy possibly break-even
- Exascale: dual redundancy offers improves runtime over single,
  - triple redundancy still in the distance, unless SDC considered

Conclusions and Future Work

- Runtime of apps employing redundancy+C/R may be modeled.
  - For a large system or unreliable system
    → redundancy+C/R can achieve significantly shorter runtimes
  - @ 80,000 nodes:
    2x redundancy → 2x # resources but 2x # jobs
    @ exascale: 2x redundancy best!

- Future Work
  - Propose optimal checkpoint model that is redundancy aware
  - Work towards eliminating assumptions
    - exponential failure model of system…

Questions?

- Now is the time to ask.

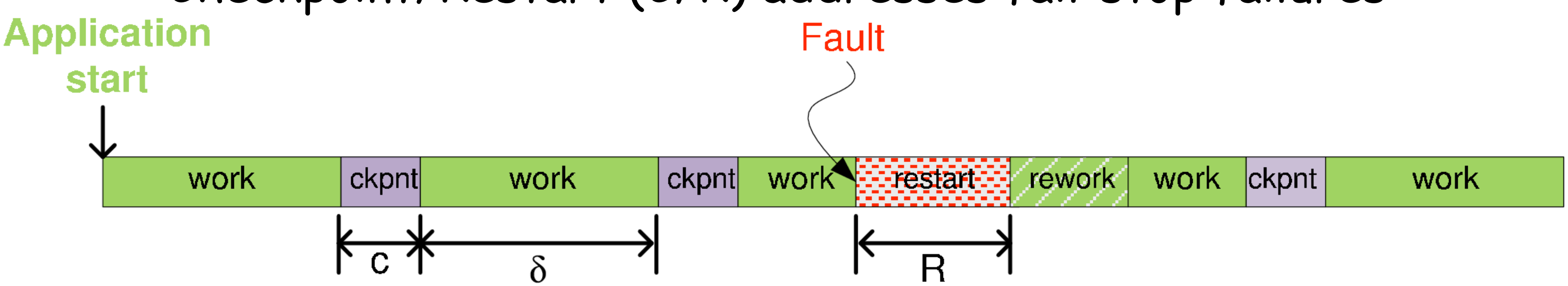
- Acknowledgements: This work was supported in part by
  - NSF grants 1058779, 0958311, 0937908,
  - DOE DE-AC05-00OR22725 as well as by subcontracts from
  - Sandia and
  - Lawrence Berkeley (LBL-6871849) National Laboratories.
  - The research at SNL was supported by DOE DE-AC04-94AL85000 and that at
  - ORNL by Office of Advanced Scientific Computing Research and DOE DE-AC05-00OR22725 with UT-Battelle, LLC.

Outline

- Motivation
- Overview of Redundancy and Partial Redundancy
  - Design of Redundancy
  - Preliminaries for Redundancy model and implementation
  - System Reliability Model
  - Effect of Redundancy on Execution Time and Reliability
- Mathematical Analysis
  - Wallclock Model
  - Model Evaluation
- Simulations and Model Comparison
  - Simulations performed on ARC
  - Extrapolated model of Jaguar
- Conclusions and Future Work

Motivation

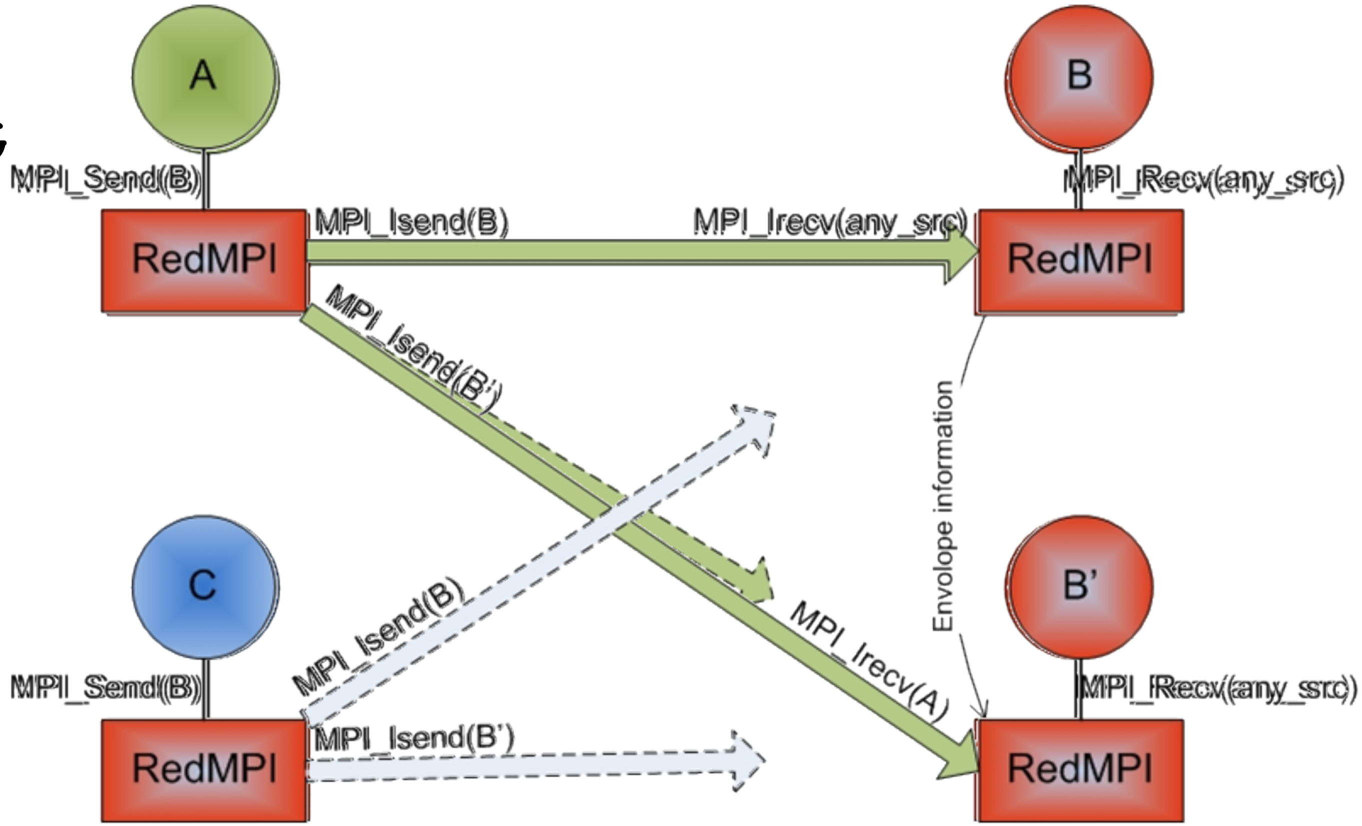
- Fault Tolerance and HPC
  - As # of components in a system increases → likelihood of failure increases
  - Fail-Stop failures
    - Node dies, switch fails,___ => running application fails
  - Checkpoint/Restart (C/R) addresses fail-stop failures



  - Periodically save application state
    - process level checkpoint on each node to shared storage, …
  - In event of failure, reload from last checkpoint

Design of Redundancy: MPI_ANY_SOURCE

- Message ordering requirement
- Primary replica posts MPI_Recv(any_src)
- Other replicas wait for primary
- Similarly for MPI_ANY_TAG

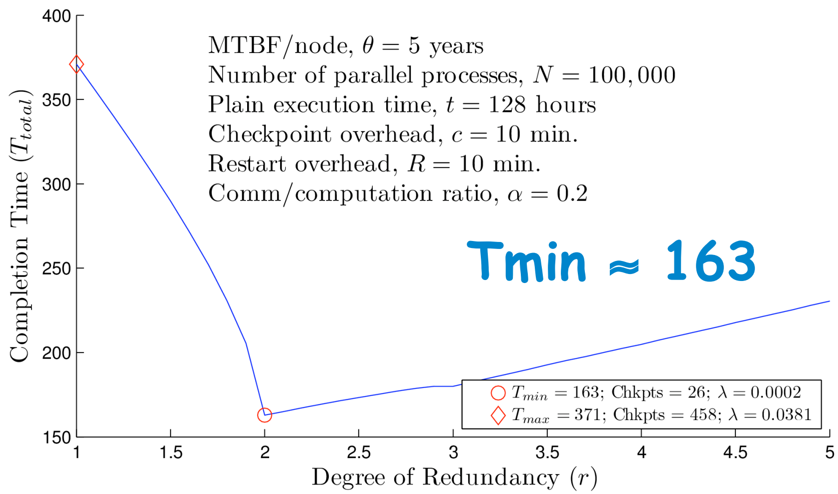


Model Evaluation

Base Configuration

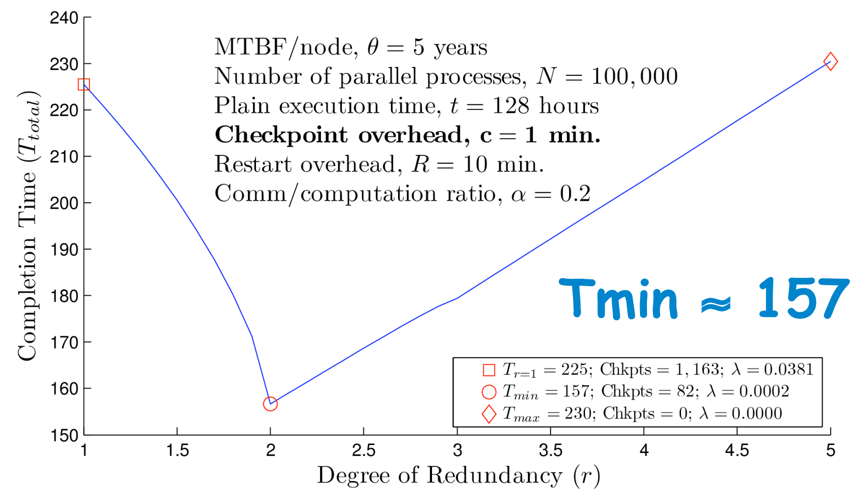
$$\delta_{opt} = 7.2$$

$$= 22.9/\sqrt{(c)} = 22.9/\sqrt{(10)}$$



Decreased Dump Time

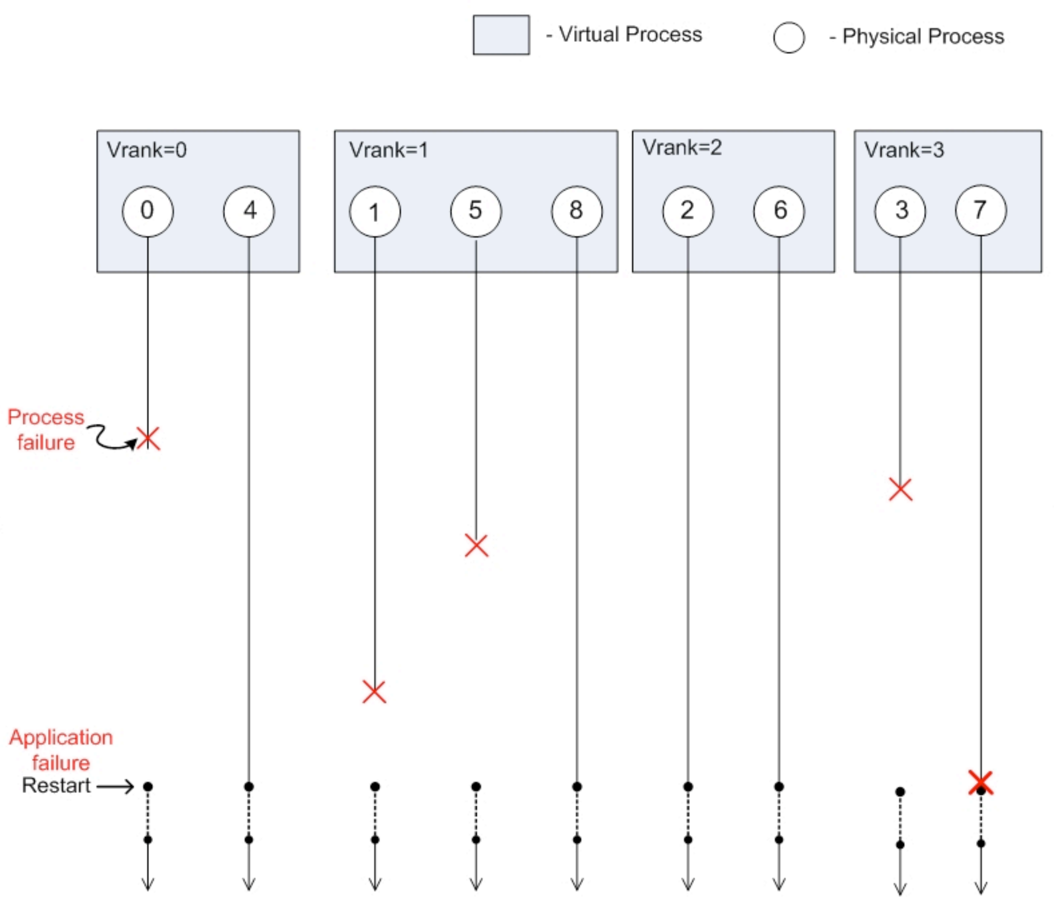
$$\delta_{opt} = 22.9$$



- Similar minimal runtime, even w/ 10X higher dump time
- Lower system MTBF = significantly fewer checkpoints
  - 458 vs 26 and 1,163 vs 82
  - minimizes impact of C/R overhead

Simulation Framework

- Background Processes
  - failure simulator
  - checkpointer
- Scaled down HPC Environment
- Goal : Validate analytical model



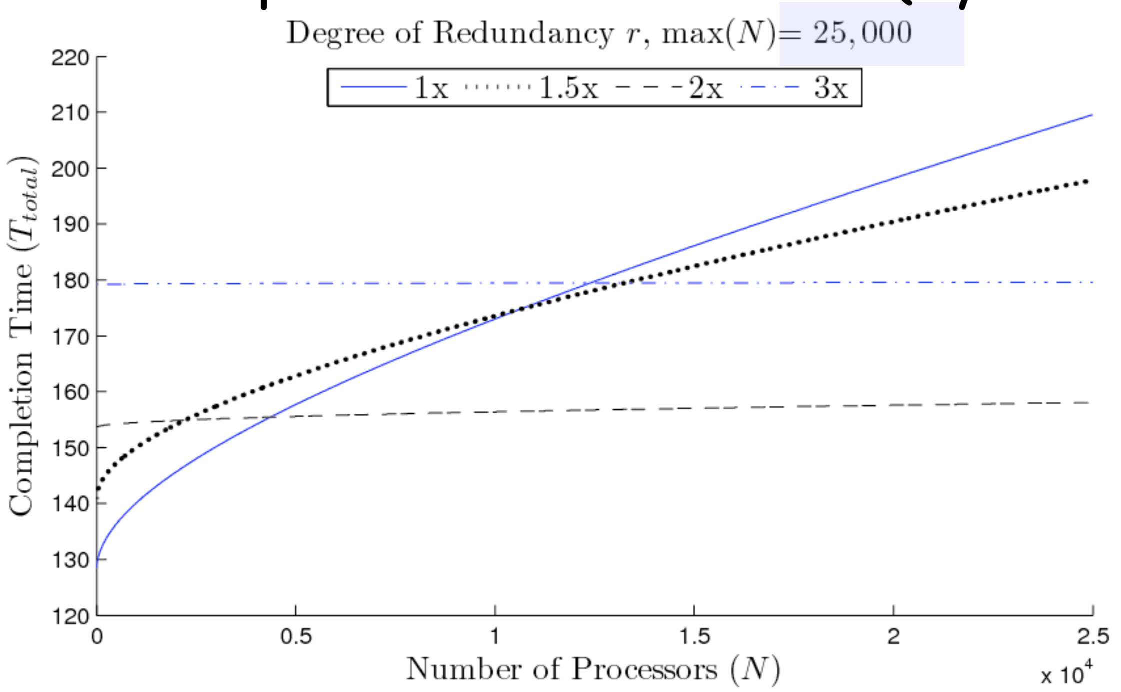
Results

- 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 – Animated Crossover

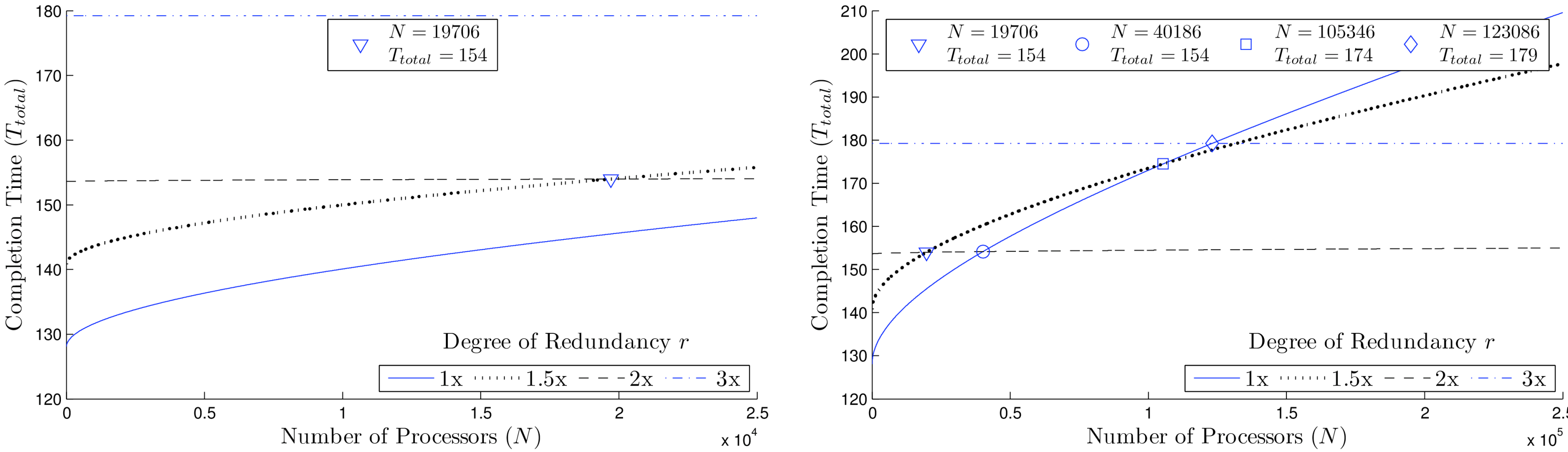
- Same params as published crossovers (5yr MTBF, etc..)



- 2x behaves like 1x, given large enough N
  - 3x should behave similarly given sufficiently large N.
- 1x fails at ~ 250k, reliability reaches floating limit for zero.

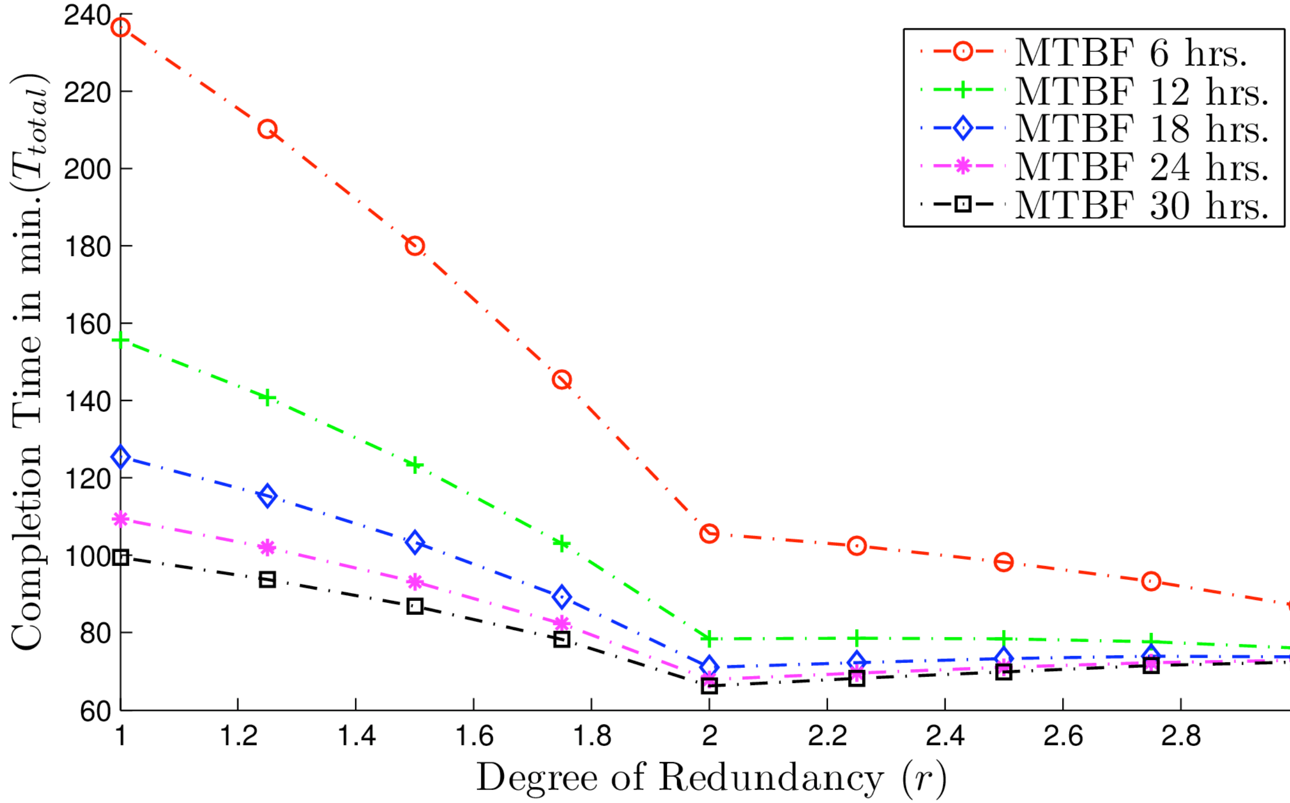
Results – Jaguar Extrapolation

- Jaguar node MTBF is estimated to be roughly 50 years
  - 18,688 nodes



- No redundancy necessary yet
- Dual redundancy in the very near future
  - Titan maintains node count
  - **increases core count by 33%, adds GPUs.**

Results – Model



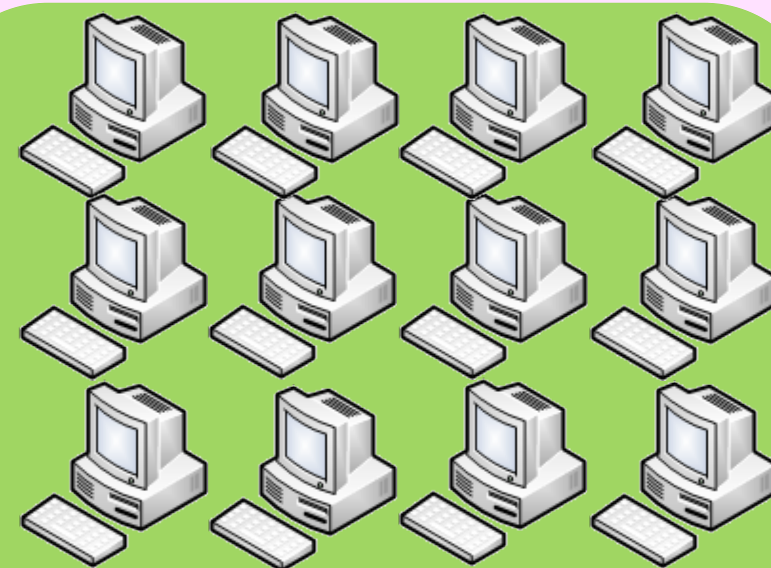
- minimum runtime always achieved at 2x redundancy

Motivation

Components have reliability

- Reliability follows a statistical distribution e.g., Exponential
- Mean Time Before Failure

MTBF denoted as θ

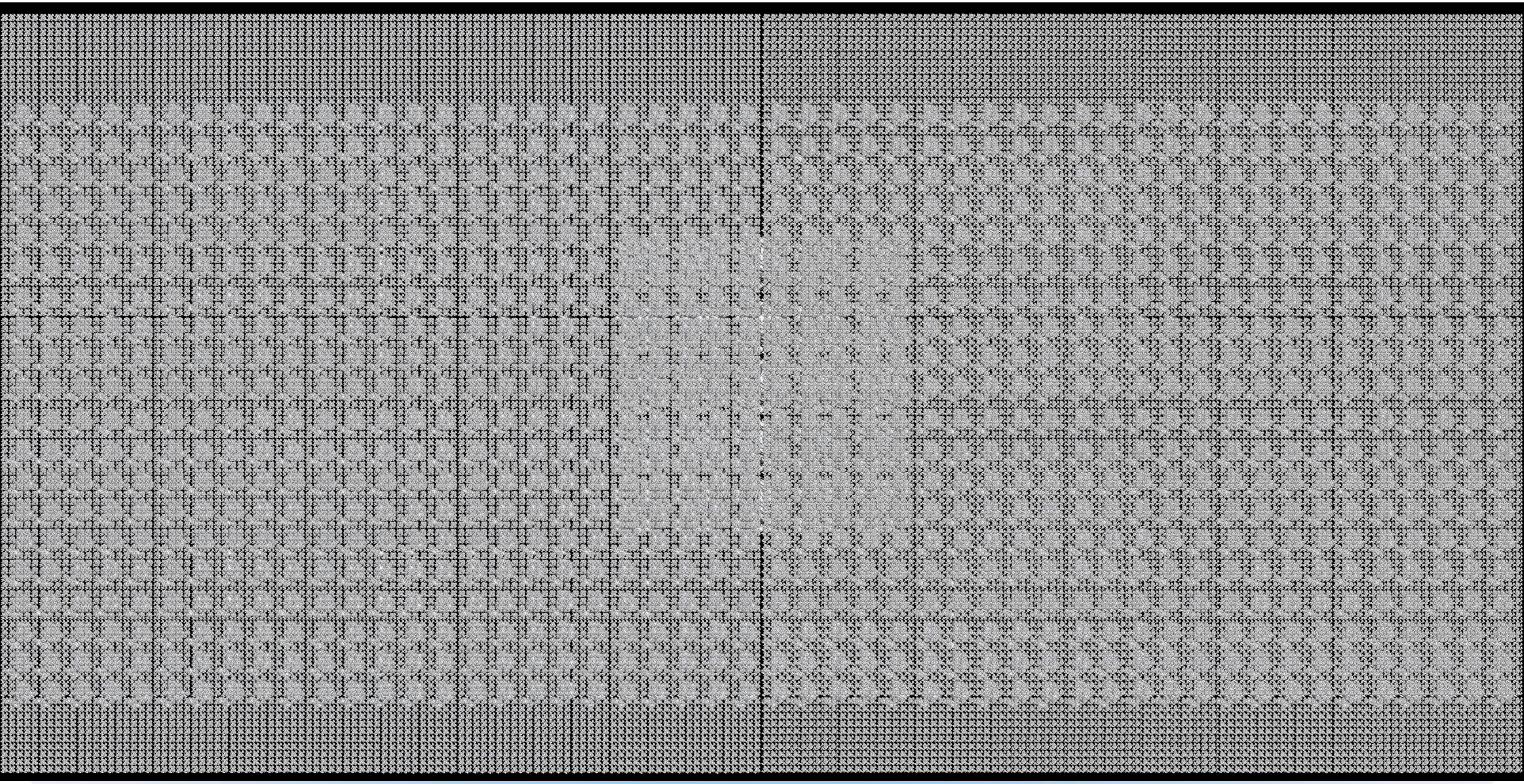


Θ = [illegible] hr.

Θ : **<u>System</u>** MTBF

Assume a node has <u>5yr MTBF (43,800 hours)</u>

Motivation



N = [illegible] Θ = [illegible] hr. δ = [illegible] min.

At petascale *50yr node MTBF (438,000 hours)*