# Resilience in High-Performance Computing

- Increasingly important area
  - Exascale systems will have extremely large numbers of cores
  - Failures will happen with high probability
  - Something has to be done, or exascale simply won’t happen

# State of the art for Reliability in HPC: Checkpointing

- Basic idea:
  1. Stop entire application at a specific global synchronization point
  2. All nodes send part of their system state (memory, generally) to dedicated I/O nodes in a *checkpoint* operation
     - Whatever is needed to re-start application
  3. I/O nodes store checkpoint info to disk
  4. If the application crashes, execute a *restore from checkpoint* operation

# State of the art for Reliability in HPC: Checkpointing

- Some questions with checkpointing
  - Is it application-level or system-level?
    - Former is generally used; requires user intervention (bad) but is much more lightweight and efficient (good)
  - Is it coordinated (e.g., at barrier) or uncoordinated?
    - Former is simpler to implement but has high overhead; latter is more efficient but the implementation requires more work (e.g., keeping message logs)
    - Uncoordinated may allow checkpointing to be overlapped with idle time, but also can have multiple rollbacks (bad)

# Why has Checkpointing been dominant?

- Application state can be saved/restored much more quickly than mean time to interrupt (MTTI)
- Checkpointing has generally been only 10-20% of total program execution time
- Non-crash system faults are rare
  - E.g., per the paper you read, few memory errors
  - Note: since publication of the paper you read, there is more concern about memory (“soft”) errors

# Why Checkpointing (may be) Doomed at Exascale

- Paper claim: exascale socket counts likely to be in the 100,000 range***
  - MTTI for the *system* has been projected in the 3-37 *minute* range
  - Checkpoints themselves at this range might take longer than this!
    - And if they don't, the checkpoint + restart might!

****This doesn't seem to be happening, but this is still interesting as newer systems with "fat nodes" will likely fail more often, since any component can fail*

# Why Checkpointing (may be) Doomed at Exascale

- Paper points out that even assuming optimistic checkpoint time (15 mins) and MTTI (1 hour):
  - System should checkpoint once every 27 minutes, which means utilization below 50% (assumes checkpoint time equals restart time)
    - Unacceptable!
  - Would require checkpoint time of 1 minute to achieve > 80% utilization
    - Here, utilization means % of the time computing
  - Checkpointing *preserves* soft (e.g., memory) errors
    - This is bad---it means that errors can persist on disk

# Other Approaches

- High-speed storage
  - E.g., Flash memory
  - Main problems: expense and durability
    - Expense has decreased a lot recently; exists on some systems
- Memory-based checkpointing
  - Checkpoint to remote memory rather than disk
  - Main problem: expense
- Create nodes with larger mean time between failures (MTBF)
  - Doesn't really address the problem (and nodes are now more significant, not less)

# Different idea: Replication

- Simple idea: have a replica node for each primary node
  - In general, every node does not have to be replicated, but we'll assume full replication
    - E.g., might know that certain nodes in the data center are much more likely to fail (e.g., not near the A/C)
- Still need checkpoints, but can greatly decrease frequency

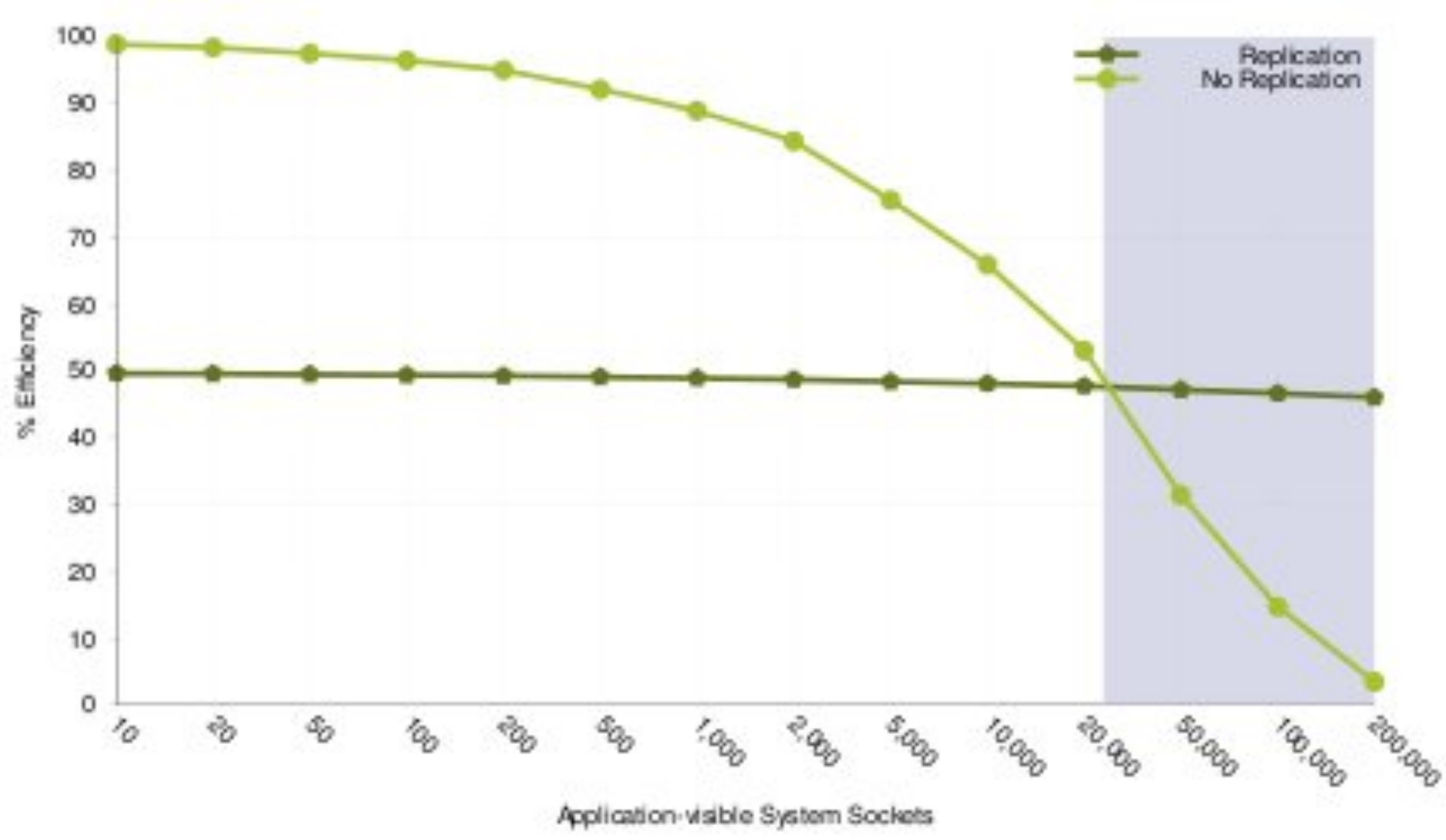
# Different idea: Replication

- Costs
  - Full replication doubles hardware requirement, or takes away half of the performance immediately
    - These are logically equivalent
  - Maintaining replicas has overhead
    - So, it’s actually slightly more than half of the performance that’s lost

# Different idea: Replication

- Benefits
  - Increased MTTI
    - All replicas must fail
  - Reduced I/O requirement
    - Spend less on I/O system
  - Can potentially detect soft errors
    - E.g., checksum memory regions
    - Could be quite expensive, though
  - Increased system flexibility
    - Could, depending on application and number of nodes, vary the actual number of nodes for replication

# Model-Based Analysis (courtesy of Ferreira et al.)

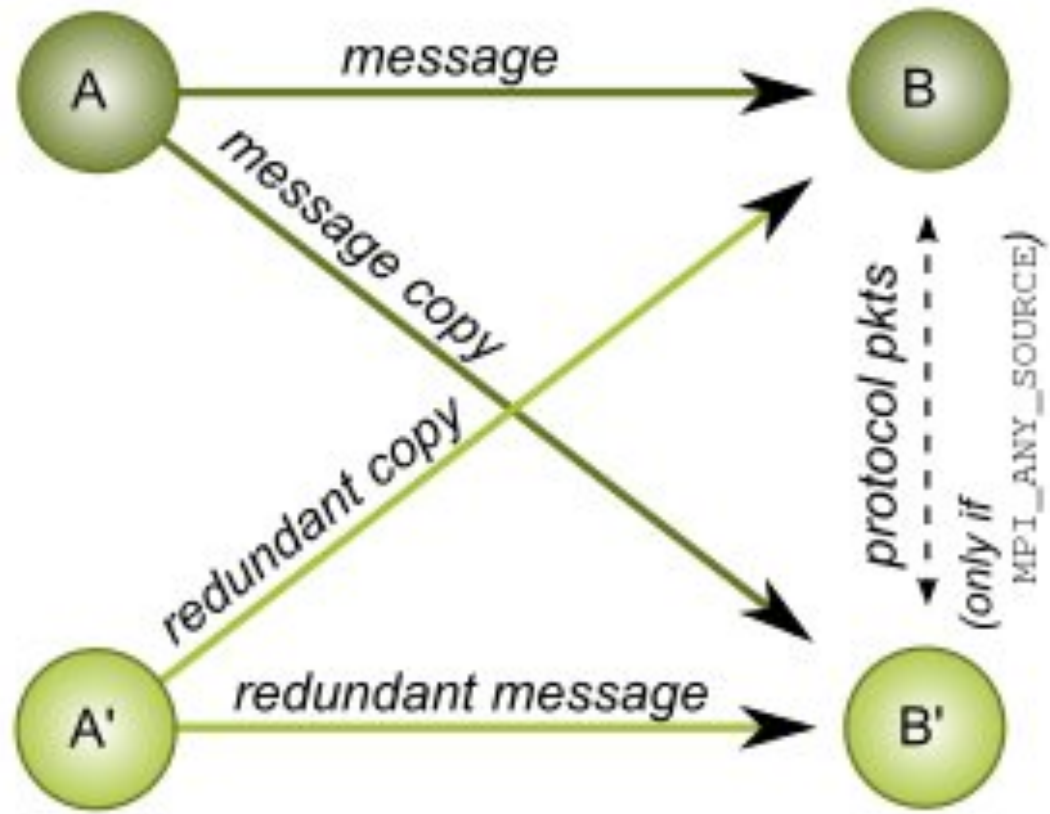



Shaded areas correspond to exascale socket counts

# rMPI

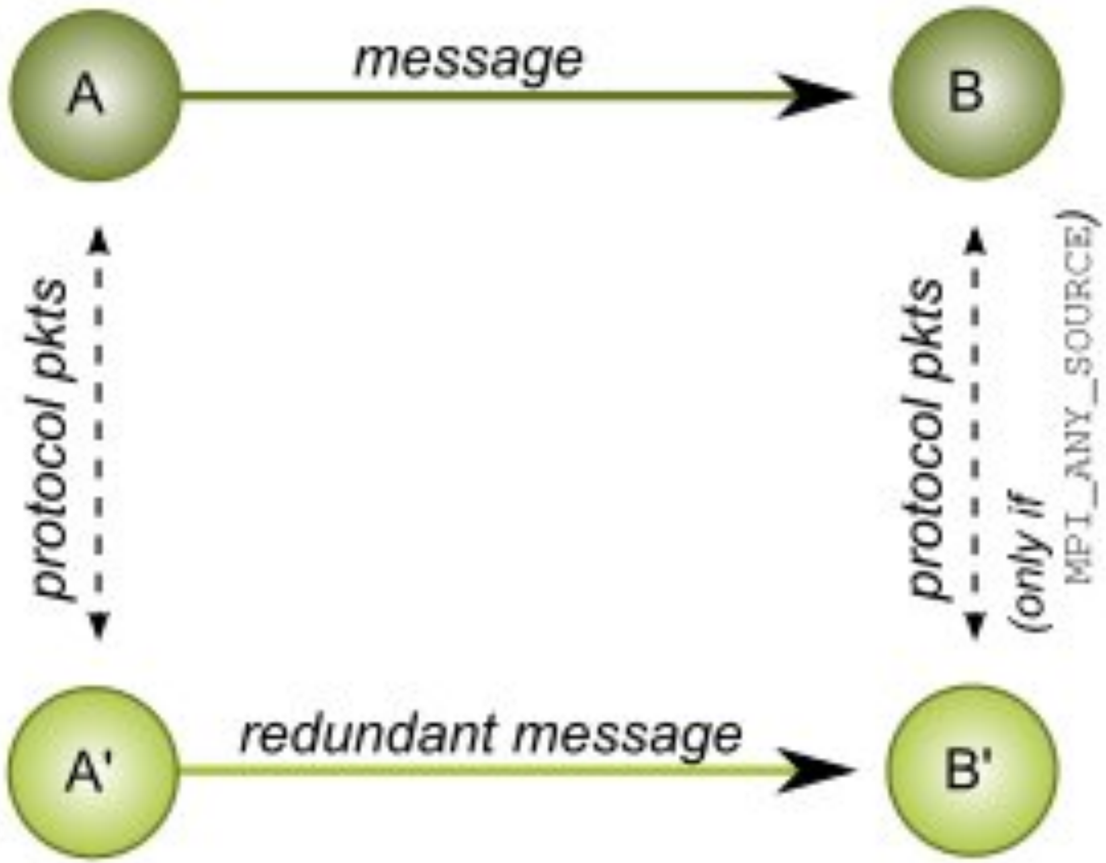
- Implements replication within MPI
- Two possible protocols: mirrored and parallel
  - Mirrored: every message is sent to primary and replica
  - Parallel: duplicate messages only sent when a rank is down, and the remaining shadow rank "covers"
- Interesting implementation issues with Mirrored protocol
  - E.g., multiple messages with same tag from a rank
    - Set high-order (unused) bit for redundant messages

# Mirrored Protocol
# (courtesy of Ferreira et al.)




(a) Mirror Protocol

# Parallel Protocol (courtesy of Ferreira et al.)

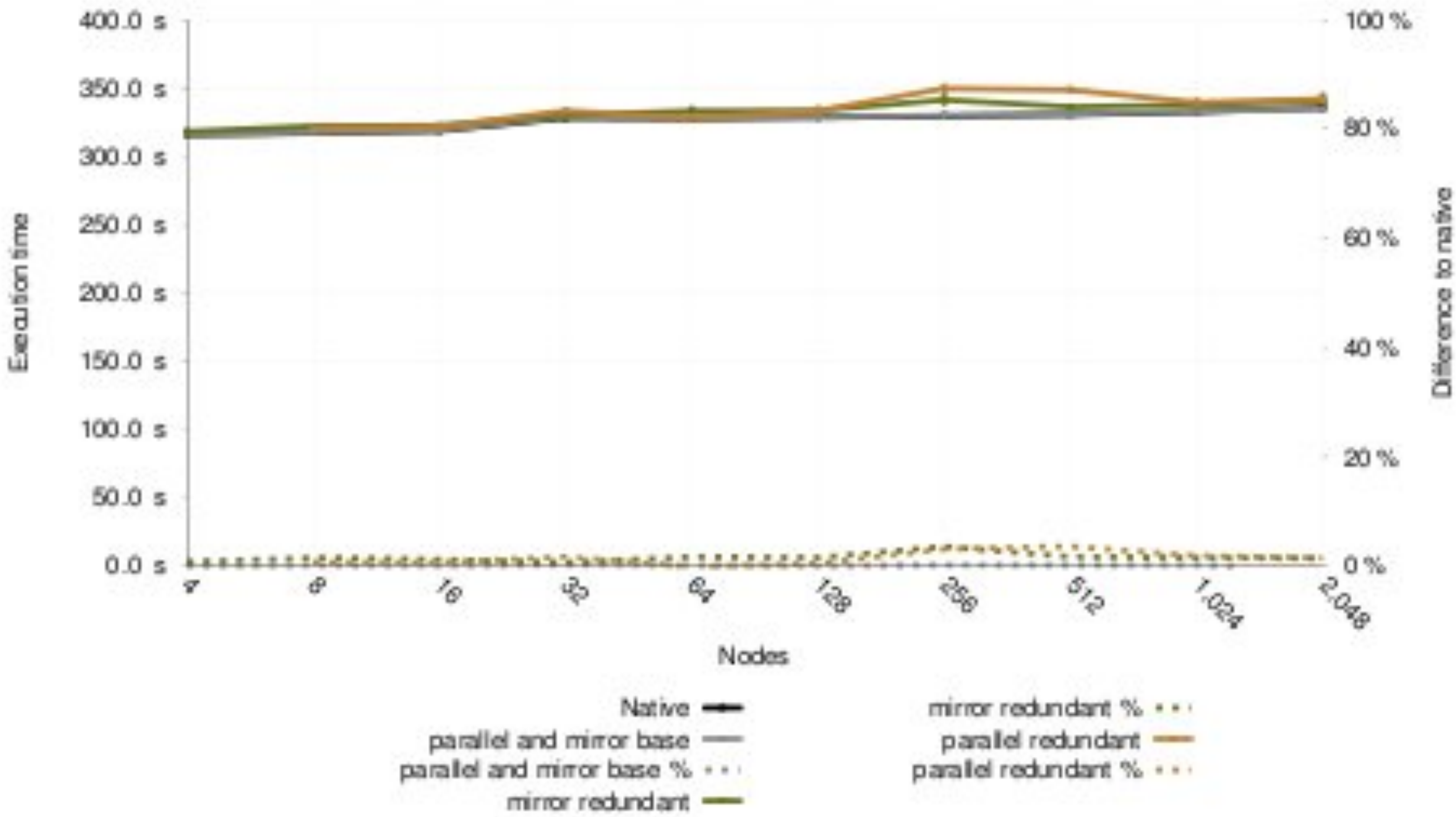


(b) Parallel Protocol

# rMPI

- Problematic: need all replicas to receive from the same rank on wildcard receives (ANY_SOURCE)
  - E.g., can’t have rank A receive ANY_SOURCE and have that match rank B, while rank A’ receives ANY_SOURCE and matches rank C
- Solution: Designate one node per pair the leader
  - Replicas replace their wildcard receive with a receive of metadata from the leader
  - Leader sends who the matching sender was
  - Replica then posts a non-wildcard receive from the correct sender

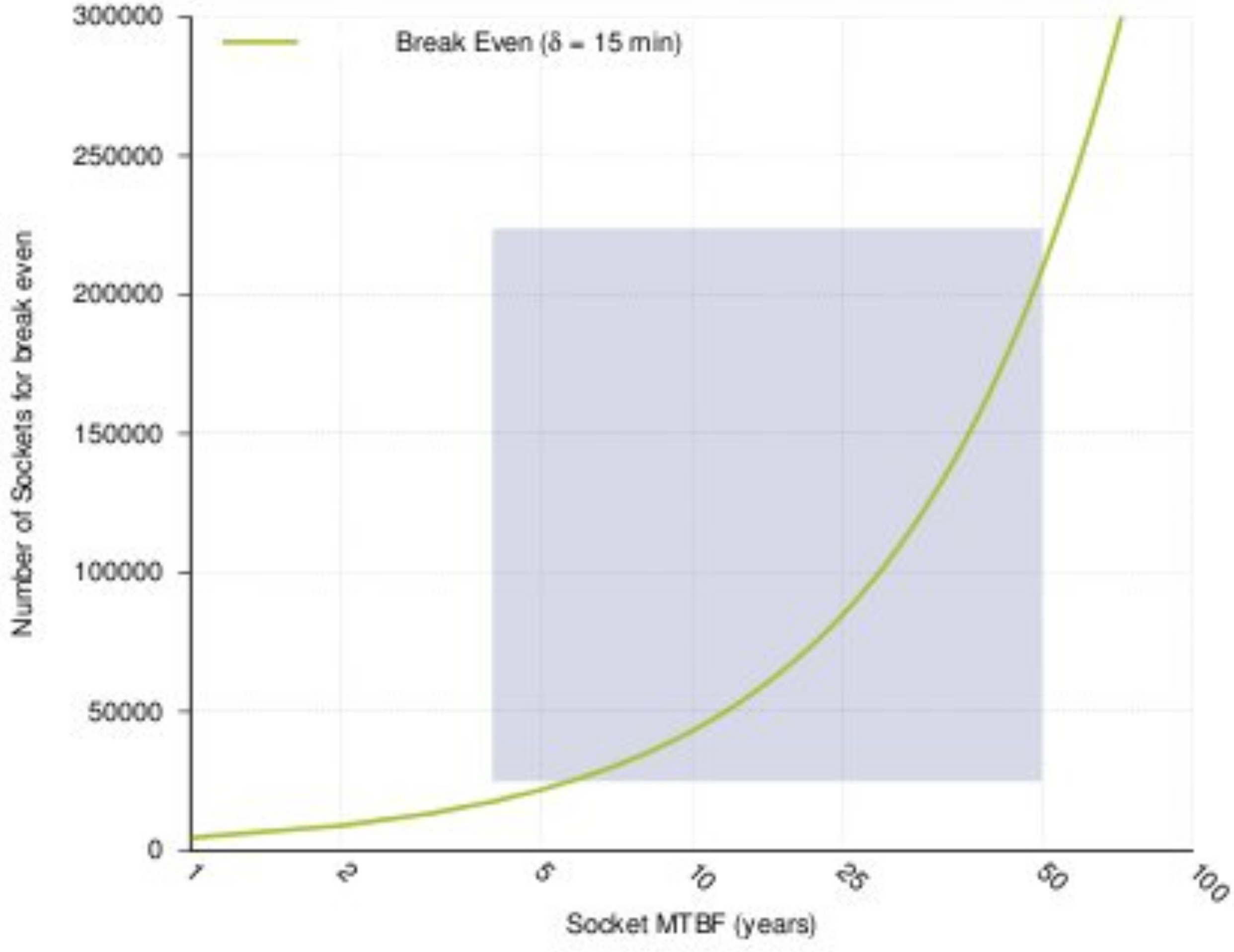
# Overhead of rMPI for LAMMPS (courtesy of Ferreira et al.)



# Simulations

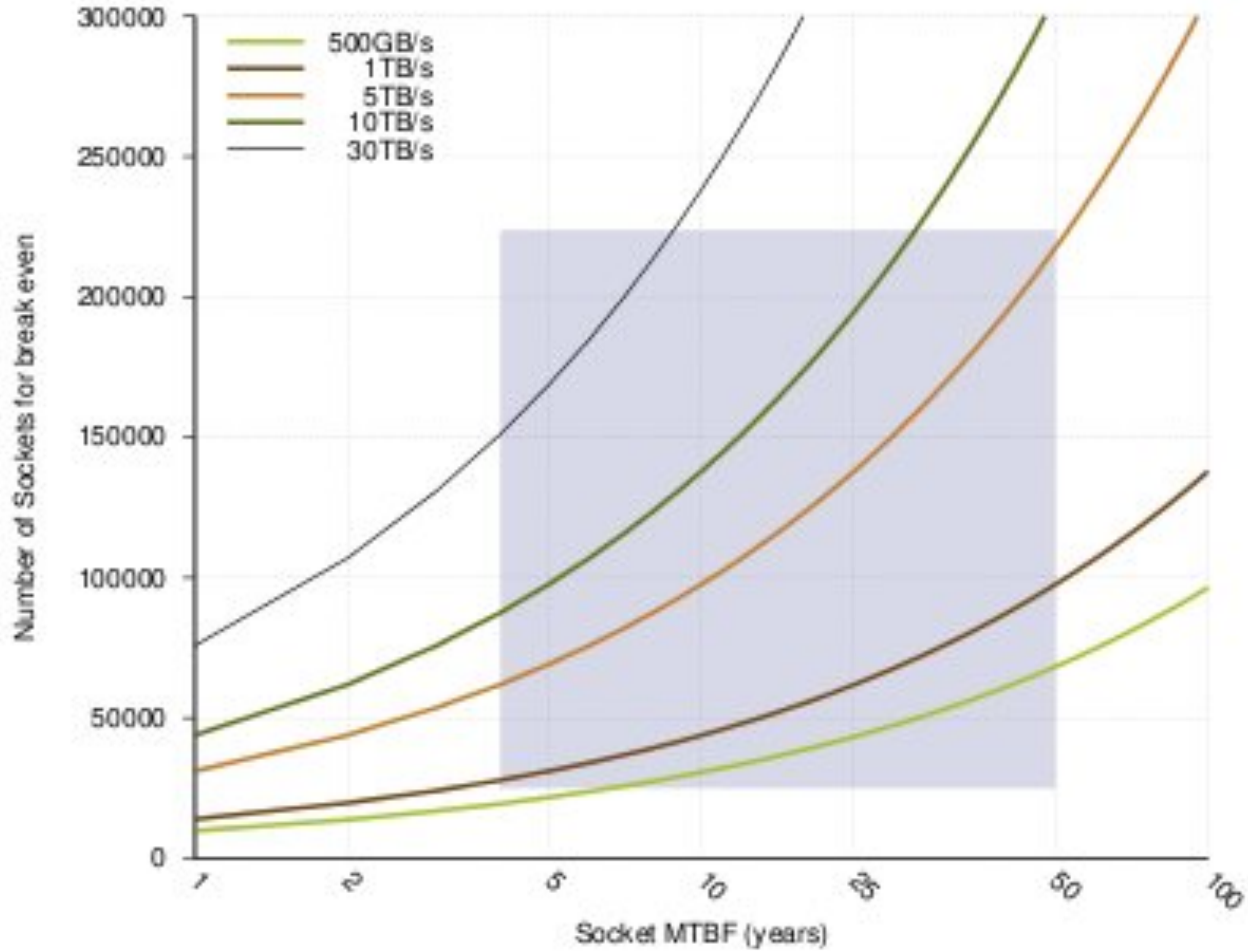
- Needed to examine potential exascale systems with large core counts
- Studied:
  - Different socket MTBFs
  - Different checkpoint rates
  - Different failure distributions

# Different Socket MTBFs (courtesy of Ferreira et al.)



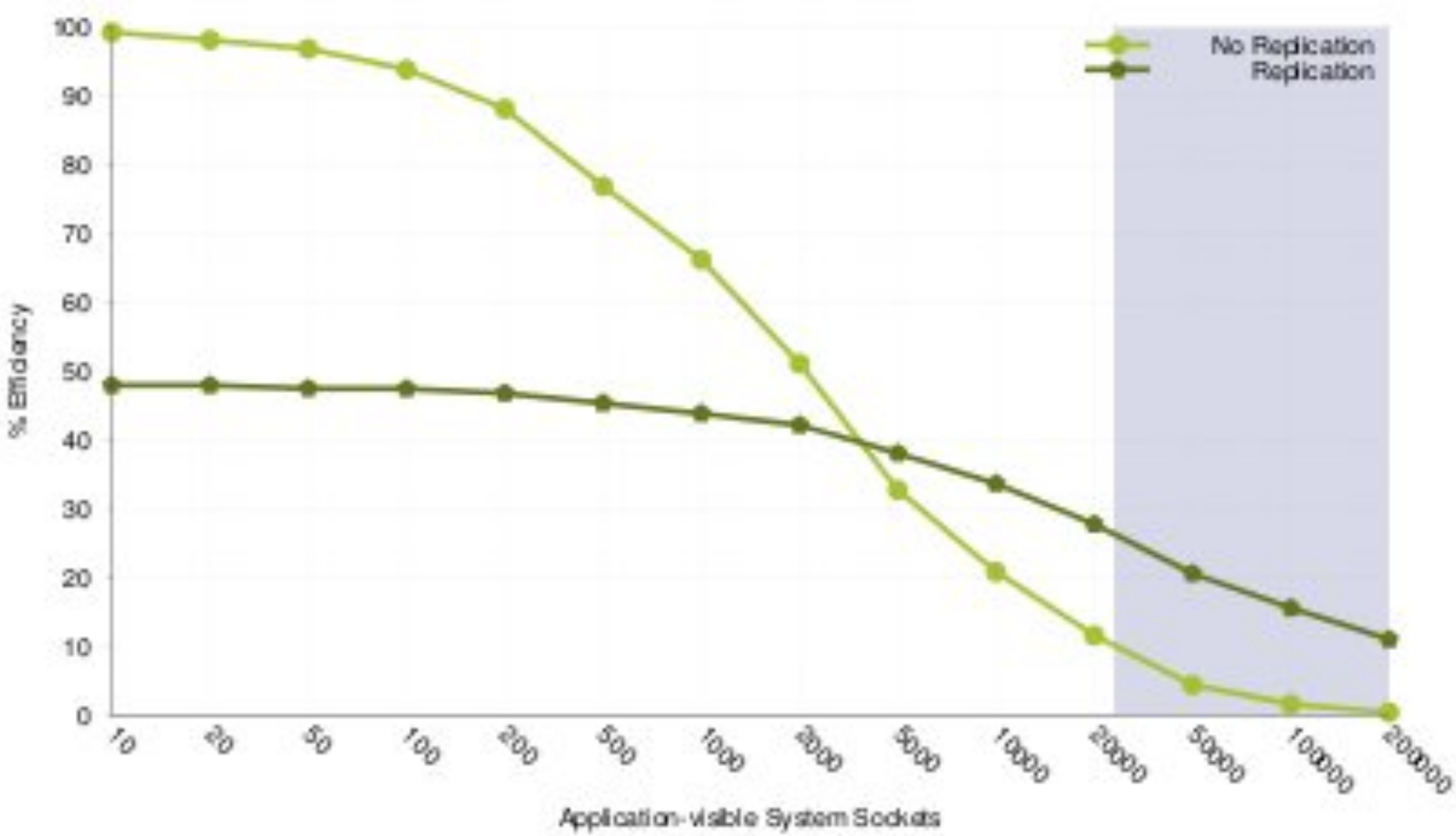

Shaded areas correspond to exascale socket MTBF

# Different Checkpoint Rates (courtesy of Ferreira et al.)




Shaded areas correspond to exascale socket MTBF

# Different Failure Distributions




(a) Weibull $\beta = 0.156$, socket MTBF = 12 years

Shaded areas correspond to exascale socket counts