SANDIA REPORT

SAND2011-5909
Unlimited Release
Printed August, 2011

A Model-Based Case for Redundant Computation

Jon Stearley, David Robinson, Kurt Ferreira, Rolf Riesen

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.
Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2011-5909
Unlimited Release
Printed August, 2011

A Model-Based Case for Redundant Computation

Jon Stearley (Org. 01422) jrstear@sandia.gov
David Robinson (Org. 01464) drobin@sandia.gov
Kurt Ferreira (Org. 01423) kbferre@sandia.gov
Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM 87185-1319

Rolf Riesen rolf.reisen@ie.ibm.com
IBM Research, Ireland

Abstract

Despite its seemingly nonsensical cost, we show through modeling and simulation that redundant computation merits full consideration as a resilience strategy for next-generation systems. Without revolutionary breakthroughs in failure rates, part counts, or stable-storage bandwidths, it has been shown that the utility of Exascale systems will be crushed by the overheads of traditional checkpoint/restart mechanisms. Alternate resilience strategies must be considered, and redundancy is a proven unrivaled approach in many domains. We develop a distribution-independent model for job interrupts on systems of arbitrary redundancy, adapt Daly's model for total application runtime, and find that his estimate for optimal checkpoint interval remains valid for redundant systems. We then identify conditions where redundancy is more cost effective than non-redundancy. These are done in the context of the number one supercomputers of the last decade, showing that thorough consideration of redundant computation is timely - if not overdue.

1 Introduction

Supercomputer scale is growing in both component count and societal impact. Once tools for governments only, today they impact everything from the evening weather forecast to tire and drug design and fission and fusion research. Their unparalleled capabilities enable breathtaking science, and demand huge financial, power, and human resources. A dark cloud of concern is rising however, due to the unavoidable outcome on current systems that increasing component counts (up to 220K sockets by 2015 [12]) and stalled component failure rates [38] result in decreasing time to job interrupt. This stems directly from the fact that current systems are designed such that the failure of any non-redundant component interrupts the entire job - and most components in today's capability systems are not redundant. The most common failure mitigation strategy is to save application state at fixed intervals, such that when interrupts do occur, jobs can restart from their last checkpoint. However, checkpoint sizes are increasing faster than checkpoint bandwidths [38]. It has been shown that the collision of these trends will render Exascale systems as "useless" due to checkpoint/restart overheads [12], and thus it is time for new reliability strategies to be explored [38, 13, 48].

A variety of approaches is possible, including message logging, uncoordinated checkpointing, checkpoint compression, and checkpointing to rack-based flash with asynchronous push to disks [20], but this paper focusses on "N+1 Redundancy" since it is unequalled as a proven-effective resilience strategy in other domains. It has been studied widely, and deployed in technical and non-technical settings ranging from circuits and space ships to communications and governance. In return for its benefits to completion and correctness in the midst of failures and faults, it unarguably involves significant costs. However, its benefits have been shown to outweigh its costs for many purposes. Redundancy is already deployed at various scales within supercomputers, including power, disks, and memory, but it has not been deployed at the macro-scale of computation itself. At that point, it is roughly assumed to completely double the cost of systems, for which the return on investment is unclear to most and silly to some. We quantitatively explore these issues, and make the following contributions:

- We derive a distribution-independent model for job interrupts on systems of arbitrary redundancy, including nonuniform, enabling greater understanding of system properties and direct calculation of quantities of interest such as mean time to job interrupt.
- We combine this with Daly's model for total application runtime, and examine the impact of redundant computation on HPC's paramount metric - total time to solution.
- We show that Daly's estimate for optimal checkpoint interval remains valid for dual-redundant systems, assuming poisson process node failures.
- We do these in the context of the top ranked supercomputers of the last decade, and find that the point at which redundancy is more cost effective than non-redundant falls well within existing system sizes and failure rates, making thorough consideration of redundant computation sensible and timely, if not overdue.

1.1 Related Work

Redundancy is well studied and broadly deployed, such that a thorough review of its use and properties is out of scope for this work. We thus touch on its use in computing in general, and move to supercomputing in specific. At the micro-scale, extra gates are used in a wide variety of circuits in order to compensate for manufacturing defects. Moving higher, memory chips implement error checking and correction through extra bits and logic. Taking major steps upward, computational redundancy is performed in high-availability servers [28, 32], and has been proposed for security [30].

Redundancy is already used in major supercomputer subsystems, most notably for storage of checkpoints (and all other data for and from the massive computation subsystems). Redundancy is present at multiple layers of the I/O subsystem (input/output), such as those using the Lustre filesystem [42]. First, the service

nodes which receive the data from compute nodes can be configured in a redundant fashion. Both types of such nodes - metadata servers (MDS) and object storage servers (OSS) - can be configured in a failover mode such that the failure of either node type does not interrupt the storage process. Secondly, OSS nodes are typically connected to multiple object storage targets (OST), which are typically redundant arrays of independent disks (RAID [33]) - specifically designed to survive disk failures. It is ironic that even with such extensive redundancy, I/O subsystems are still a significant cause of service interrupts [37], and that avoiding them by writing fewer checkpoints has been observed to actually improve job mean time to failure [29].

The compute subsystem is however the largest, and a primary source of interrupt causes [38]. Computational redundancy for HPC however has been neither extensively studied nor deployed. Factors for this include the extreme demand for compute cycles (the core purpose of supercomputing) coupled with their significant procurement and power expenses, and insufficient research on the efficacy of computational redundancy. Recently, Schroeder [38] and others [52] have suggested redundant computation (also termed process pairs) as a possible path forward. Engelmann et al. [15] make a case for double and triple redundancy for HPC in terms of the availability of the system, particularly regarding the dramatic increase of the availability of a system with various levels of redundancy. One of his observations is that the availability gains of redundancy are so great that, in order to offset costs, less reliable and less expensive components can be used. His thesis describes active/active redundancy, includes surveys on systems and strategies, and includes models similar to ours, but he focusses on availability, and dual and triple redundancy in the service section of HPC systems [14]. We instead explore mean time to job interrupt and total application runtime [10] for arbitrary redundancy in the compute section. Daly has also developed a model for job interrupts (which he refers to as "application fatal errors") [9], but whereas he focusses on dependency hierarchies to model interrupt rates, we explore redundant computation as an interrupt reduction strategy.

We have implemented a library called rMPI which performs redundant computation as described in this paper, without requiring any changes to existing applications or systems, via the MPI profiling layer. The user specifies how many ranks should be replicated, and the library takes care of the details from there. It is a proof-of-concept implementation, and does not provide full functionality (such as, it does not currently handle I/O), but we have run micro-benchmarks and four full Sandia-significant applications at scales up to 2,048 nodes. Total slowdown of the four applications were 5%, 10%, 10% and 20%. Additional details are available elsewhere [16]. More extensive description of our simulator and lessons learned from it are also available [36], but the focus of the current paper is modeling.

We now survey existing systems in Section 2, develop our model in Section 3, and explore exponential failures in Section 4. In Section 5 we extend Daly's model to address non-exponential interrupts, and identify the conditions where redundancy is more cost-effective than non-redundancy in Section 6. We then conclude with future work and call for more effort to directly measure the impact of failures on real systems.

Year	Name	TFlops	Cores	Sockets	Nodes	Memory (TB)	Disk I/O (GB/s)	Checkpoint (Minutes)	Clock (GHz)	Power (MW)
2000	ASCI Red	2	9,632	9,632	4,562	1	5	4	.33	.85
2001	ASCI White	7	8,192	8,192	512	8	12	14	.37	2.0
2004	Earth Simulator	35	5,120	5,120	640	10	16	13	1.0	2.5
2007	BlueGene/L	478	212,992	106,486	53,248	69	35	41	.70	2.3
2008	Roadrunner	1,105	129,600	18,360	3,060	97	220	9	1.8	2.4
2009	Jaguar	1,759	224,162	37,376	18,688	299	240	26	2.6	6.9
2010	Tianhe-1A	2,566	202,752	21,504	7,168	229	n/a	n/a	2.9	4.0

Table 1. The number one supercomputers from the top500.org list over the last decade [4, 8, 49, 26, 3, 44], at earliest year of highest performance. "Disk" indicates maximum theoretical I/O rate. "Checkpoint" indicates time to save full memory to disk, assuming 80% of "Disk" rate [31].

2 Systems Survey

We now survey existing systems, in order to inform our terminology and scope parameter ranges to explore. Data on the number one supercomputers from the Top500 list over the last decade has been assembled in Table 1. Total processing capability (TFlops) has increased by three orders of magnitude, accomplished by increased core count, clock speed, and node count (these factors are listed in decreasing contribution order). Tianhe-1A's use of Graphics Processing Units (GPUs) is a key factor to its lower node count and power usage. Due to this, node count growth from 2000 to 2010 has only doubled. Prior to GPU's however, node count increased by a factor of 22 from ASCI Red to BlueGene/L.

Although node count is not the primary contributor to processing capability, it is a natural description of system size: it corresponds to physical footprint, and is a common unit of management and replacement. A Linux node is the most common administrative unit of HPC systems today, and providing multiple CPU and GPU cores via a single operating system instantiation. While redundant computation could take place among cores or CPU's within a node, many failure types within a node result in the entire node going offline (for example, non-correctable memory errors or failed CPU's cause the Linux kernel to panic). Since our consideration of redundancy is motivated by fault tolerance, it makes sense to speak of replicating computation across nodes, rather than cores or other unit. We thus select *node* to be a worthwhile term to describe redundancy. This selection is purely semantic however - we will describe redundancy at the node level, but this does not constrain the generality of our models or analysis. Redundancy could occur at the core level for example, but from a fault-tolerance perspective it would be most robust if those cores were physically independent. Our discussion of redundancy focuses on overall effect - not granularity of implementation.

We were unable to find sufficient data to include a failure rate column in Table 1, but now report what we could find. Based on a study of twenty-two (anonymous) HPC systems, Gibson and Schroeder observe that failure rate grows in proportion to the number of sockets, and give an "optimistic" estimate of 0.1 failures per socket per year [18]. Dual-socket nodes are common in that study, corresponding to a mean time to node failure of 5 years (also used in [31]). ASCI White's MTBI has been reported to be 5 hours in 2001 and 40 hours in 2003 (after the platform had stabilized). At 512 nodes, this corresponds to a mean time to node failure of 0.3 and 2.3 years respectively [24]. Consistently, ASCI Q's reported MTBI is 6.5 hours (this had the same architecture as ASCI White) [24]. Detailed metrics are reported for Red Storm (number 2 on Top500 in 2006), corresponding to a mean time to node failure of 22 years (based on an $MTBI_{job}$ of 25 hours on 7,660 nodes [40]). Per-node MTTF is also reported as 1-45 years in [46]. While the measurement and reporting is not as common as we would like [39], these do provide independent estimates to motivate the range of failure rates explored in this study.

As the last three systems in Table 1 are petascale, the HPC community has now set its sights on exascale. Projections based on transistor density and clock frequency trends indicate that such systems will likely require on the order of 100,000 sockets [7]. There is already one system with this many sockets in Table 1, but it had comparatively low memory leading to smaller checkpoints. It did encounter excessive faults with L2 cache [19], which was then overcome by greater cooperation among hardware and application layers in dealing with faults. Nevertheless, component failure rates have been observed to be stable over the last decade [38], and there is no indicator of significant improvements on the horizon. Add to this the fact that I/O bandwidth rates have not kept pace with memory size (to checkpoint) or flop rates [7], and the prudence of exploring alternate failure mitigation strategies for exascale is clear.

Other strategies include building systems with rack-local flash memory for checkpoints, which are automatically bled to traditional stable storage over time [20]. This has the advantage of being evolutionary in the sense that checkpoint-restart remains the core strategy, albeit to a different (and more expensive) hardware type. Redundancy however has a more fundamental goal - reduce the number of interrupts themselves. Message logging is another strategy. With any approach, it is likely that in order to realize sustained exascale computing rates, a higher cost for the enabling fault tolerance will have to be paid.

3 Distribution-Independent Formulation

Reliability reporting is not consistent across HPC sites, let alone distribution models. We therefore develop a model which is independent of failure distribution, using basic reliability theory. Consider a node which is put into service at time $t = 0$, and fails at time T. The probability that it fails by time T is denoted $F(t) = P(0 \leq T \leq t)$, and is known as its cumulative distribution function (CDF). The probability of failure after time T is known as its *reliability* $R(t)$:

$$R(t) = 1 - F(t) = P(T > t). \tag{1}$$

The time derivative of a CDF is a probability density function (PDF), and describes the likelihood of failure at any given time t, $f(t) = \frac{d}{dt}F(t)$. A PDF can have values greater than 1 so is not strictly a probability, but is often normalized for this purpose. The expected time of failure $E[T]$ is commonly referred to as mean time to failure (MTTF), and corresponds to the area under the reliability curve, and first moment of the PDF, namely:

$$MTTF = E[T] = \int_0^\infty R(t)dt = \int_0^\infty f(t)tdt. \tag{2}$$

Now consider systems composed of multiple devices. A job performing work occurring on n devices in series is interrupted upon the earliest device failure. Let T_i be the time of failure for the i'th device. In order to identify the earliest T_i, we test the case that all T_i are greater than t - this corresponds to the time of job interrupt, denoted T_j. So, the probability that interrupt will occur after time t is $R_j(t) = P(T_j > t) = P\{(T_1 > t) \cap (T_2 > t) \cap \cdots \cap (T_n > t)\}$. If the devices fail independently[1], this reduces to simply multiplying all of the device reliabilities, $R_j(t) = \prod_{i=1}^{n} R_i(t)$.

On the other hand, work occurring on m devices performing redundant computation in parallel continues until the failure of the last device, so we test for the case that all T_i are less than t. The probability that interrupt will occur by time T_j is then $F_j(t) = P(T \leq t) = P\{(T_1 \leq t) \cap (T_2 \leq t) \cap \cdots \cap (T_m \leq t)\}$. Again assuming independence, this becomes $F_j(t) = \prod_{i=1}^{m} F(t)$.

Equations 1 and 2 of course hold for all systems (series, parallel, or combination thereof), so the mean time to job interrupt M_j is

$$M_j = E[T_j] = \int_0^\infty R_j(t)dt = \int_0^\infty f_j(t)tdt. \tag{3}$$

Any book on standard reliability theory will provide greater detail on the above relations, we used [47].

From this point forward, we will generally omit the "(t)" notation for brevity, such that F refers to node CDF. Furthermore, we assume that nodes fail identically (all nodes have the same CDF), and independently (one node's failure does not affect another). These assumptions are common [10, 52], but not universal [38, 9] regarding HPC.

We define a *bundle* as a set of m nodes performing redundant computations in an active-standby manner, such that a job using that bundle will not be interrupted until all nodes in the bundle fail. Since a bundle is a parallel system of nodes, as we have seen earlier its interrupt CDF will be F^m, and bundle reliability will be $1 - F^m$. If we then run a job on a set of n bundles, it will be interrupted upon the earliest bundle failure. This is depicted in Figure 1. We now see that job reliability will be

$$R_j = (1 - F^m)^n. \tag{4}$$

[1] Although independence is "suspect" [38], no other model has been proposed and doing so is beyond the scope of this work. Daly proposes a dependency hierarchy [9], but assumes independent failures.

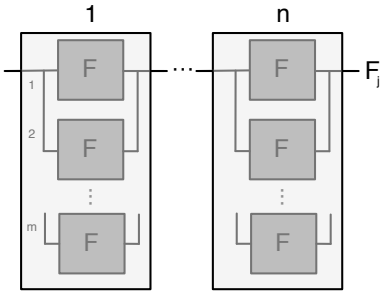


Figure 1. A series of n bundles, each bundle composed of m nodes in parallel, where each node has a failure CDF of F. The interrupt CDF of a job running on such a system is $F_j = 1 - (1 - F^m)^n$

Although not with HPC in mind, Turner explored this exact model and noted [43]:

> It is obvious that in order to maximize (R_j), n should be as small as possible, and m as large as possible. Normally n will be decided by the functional requirements of the system, but m will depend more on cost, space available, and similar considerations.

This of course remains true today. Capability HPC systems have required large job size n, and the cost of redundant computation has been avoided (keeping $m = 1$) because checkpoint-restart has proven to be a sufficient fault-tolerance strategy. Given the expected increases in n for Exascale systems in particular, it is prudent to evaluate the benefits and costs of increasing m for fault-tolerance.

Equation 4 describes a system composed of bundles all having the same number of nodes, which we refer to as *uniform* redundancy. We refer to *nonuniform* redundancy in the case that a job is running on bundles of different size. If we let m_i be the number of nodes in bundle i, job reliability will be

$$R_j = \prod_{i=1}^{n} (1 - F^{m_i}). \tag{5}$$

As before, this can be used in Equation 3 to determine the mean time to job interrupt on a system of arbitrarily complex computational redundancy.

3.1 Count of Failures

If a job running on a system with no redundancy is interrupted upon the earliest node failure, how many node failures can a job running on redundant bundles endure? Let k be the expected number of failed nodes in a bundle and p_k be the probability of that occurring, so the expected number of node failures in each bundle is

$$E[k] = \sum_{k=1}^{m} k p_k. \tag{6}$$

The discrete definition of expectation sums from zero to infinity, but no bundle has zero nodes, and no bundle has more than m. At time of job interrupt we know with certainty ($p_m = 1$) that there is only[2] one

[2]This assumes that nodes do not fail at the same instant, and that the jobs interrupt immediately.

bundle with m failed nodes, leaving $n-1$ bundles with less. Since a bundle either has k failed nodes or not (two possible outcomes), p_k is appropriately modeled by the binomial distribution,

$$p_k^{(r)} = \binom{m}{k} \mathcal{F}^k (1-\mathcal{F})^{m-k} \tag{7}$$

where $\mathcal{F}$ is the probability of a node failing by time the time of job interrupt, $\mathcal{F} = F(t = M_j)$. Combining equations 6 and 7, we have the expected cumulative number of node failures by the time of job interrupt H_j as being

$$H_j = m + (n-1) \sum_{k=1}^{m-1} k p_k^{(m-1)}. \tag{8}$$

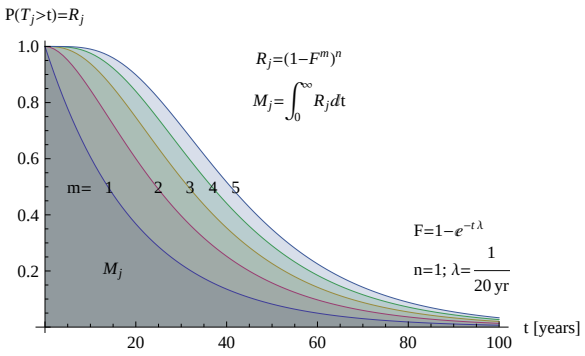


Figure 2. The mean time to interrupt M_j (the area under the R_j curve) of a job increases with redundancy level (m), but the increase diminishes with each successive level. Here, we examine a job on a single bundle ($n = 1$) for simplicity.

4 Exponentially-Distributed Node Failures

Up to this point we have made no assumptions regarding the actual failure distribution of nodes. In order to simulate and examine predictions however, a distribution must be chosen. In this Section we examine the case that nodes fail at a constant rate of λ, and equivalently, that failure times are exponentially distributed, such that node reliability is

$$R = e^{-\lambda t}. \tag{9}$$

This is a common assumption [10, 48], and an appropriate starting case. It is not a universal assumption however [38, 51], and we address alternate distributions in Sections 5 and 6.

4.1 Uniform Redundancy

We begin by examining how increasing m increases bundle reliability. Setting $n = 1$ and $m = 2$ for dual-redundancy on a single bundle, by equations 1 and 4 $R_j = 1 - (1 - R)^2 = 2R - R^2$, and by equations 3 and 9 we have mean time to interrupt being

$$M_j = \int_0^\infty R_j dt = \int_0^\infty 2e^{-\lambda t} - \int_0^\infty e^{-2\lambda t} = \frac{2}{\lambda} - \frac{1}{2\lambda} = \frac{3}{2\lambda}. \tag{10}$$

Similarly, on triple and higher redundancy ($m = 3, 4, 5$), M_j is $\frac{11}{6\lambda}$, $\frac{25}{12\lambda}$, and $\frac{137}{60\lambda}$ respectively. Successive levels of redundancy yield diminishing returns in mean time to bundle failure - namely 50% from no redundancy to dual, 22% from dual to triple, triple to quadruple 14%, and quadruple to quintuple of 9%. R_j and M_j of a single bundle are depicted in Figure 2.

The benefits of dual redundancy include detection of soft errors, reduction in mean time to job interrupt, possible minimization of rework via immediate checkpoint upon the first node failure (but before bundle failure), and the possibility of checkpoint-free operation if failed nodes can be replaced quickly enough. For example, halt a computation upon first node failure, repair or replace it, and resume computation (discussed again in Section 6). In addition to these, triple redundancy enables correction of soft errors via voting among bundled nodes [15]. The advantages of higher levels of redundancy are less pronounced. Given the

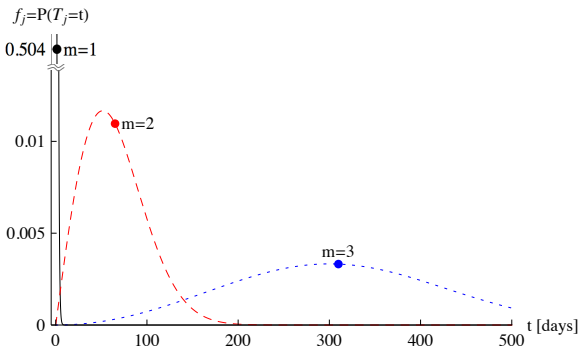


Figure 3. The time to interrupt distribution (PDF) for a 10,000 node job, using nodes with mean time to failure of 20 years. Black, red, and blue lines correspond to non-redundant, dual-redundant, and triple-redundant bundles, and dots indicate mean time to job interrupt (M_j = 17 hours, 63 days, and 309 days respectively).

significant costs of large systems, diminishing gains in successive redundancy levels, and distinct resilience opportunities, we focus our discussion on dual and triple redundancy.

Our next step is to examine how increasing job size n affects job mean time to interrupt. Figure 3 shows the time to interrupt distribution for non-redundant, dual, and triple redundancy for a system of $n = 10,000$ bundles and node mean time to failure of 20 years (a large system, with good MTTF nodes, based on Section 2). That mean time to job interrupt is increased from 17 hours to 63 days via (only) dual-redundancy suggests promise. As with bundle failures, increasing redundancy levels yield decreasing gains in mean time to job interrupt - in this case a factor of 89 for none to dual ($\simeq$ 63 days / 17 hrs) versus 5 for dual to triple ($\simeq$ 309 days / 63 days).

However, when multiple bundles are placed in series, the system reliability is the product of each of the bundle's reliability. Thus a plot of system PDF looks just like Figure 2, except that the compounding effect of the number of bundles n results in much smaller reliability areas, and more pronounced differences among redundancy levels. The product of 10,000 $m = 1$ bundles is much smaller than that of 10,000 $m = 2$ bundles, as shown by the dots in Figure 3. So as job size increases, the relative gain in mean time to job interrupt from non-redundant to redundant increases. This is intuitive - the more bundles, the less likely that any one will encounter two node failures. We also note that when the ratio of improvement in M_j from one redundancy level to another is calculated, the node failure rate λ cancels out, and as trivia, order of magnitude improvements in M_j for dual over no redundancy occur at $n = 115$ and $n = 12,606$.

Note that job interrupts on redundant systems are not exponentially distributed, as they are on non-redundant systems ($m = 1$). Thus, checkpoint models based on exponentially distributed job interrupts (such as [10]) may not be accurate for redundant systems. We will explore this in Section 6, where we evaluate redundancy's impact on the paramount HPC metric, total time to solution.

Birthday Problem

We now review the case of $m = 2$ as presented previously [35], and suggest generalization to higher order of m. The classic birthday problem states, "what is the average number of people required to find a pair with the same birthday?" It can be shown that if one samples uniformly, with replacement, from a population of size N, the number of trials required for the first repeated sampling of some individual has expected value $1 + Q$, where $Q = \sum_{k=1}^{N} \frac{N!}{(N-k)!N^k}$ [27, 17, 25]. Holst later showed that $Q \approx \sqrt{\frac{\pi}{2}N} - \frac{1}{3}$ for large N [23]. For

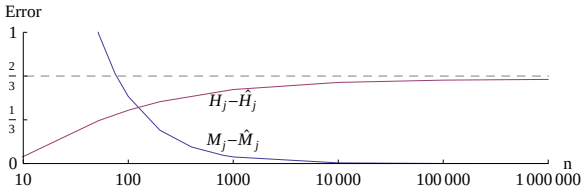


Figure 4. The birthday problem estimate for dual-redundancy agrees with our model, at large n.

dual-redundancy, the total population of nodes is twice the job size ($N = 2n$), and we are interested in large job size so $1 - \frac{1}{3}$ is negligible. By analogy, consider nodes as individuals and bundles as birthdate: "how many node failures can occur before a single bundle experiences two?" The answer is

$$\hat{H}_j = \sqrt{\pi n} \qquad \approx 1 + Q = 1 + \sqrt{\frac{\pi}{2} 2n} - \frac{1}{3}. \tag{11}$$

This remarkable result provides a simple estimate for the number of node failures by time of job interrupt on dual redundant systems. Note that H_j is completely independent of node failure rate.

To find the total number of times a device with constant failure rate will fail, simply multiply failure rate by total time. This is a direct result of *cumulative failure rate*, $H = \int_0^t \frac{F'}{1-F} dt = \int_0^t \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} dt = \lambda t$. To find the total number of failures in a population, simply multiply by the population size. Given $2n$ total nodes for dual-redundancy, operating until time of job interrupt $t = M_j$, the total number of node failures is $\hat{H}_j = 2n\lambda M_j$. Combining this with equation 11 and solving for mean time to job interrupt, we have

$$\hat{M}_j = \frac{\sqrt{\pi}}{2\lambda\sqrt{n}}. \tag{12}$$

Figure 4 depicts that this simple birthday-problem estimate for dual-redundancy agrees with our full model, at large n.

Current systems are non-redundant, corresponding to $m = 1$. In this case, equation 3 reduces wonderfully to the well-known $M_j = 1/\lambda n$. The practical meaning is not wonderful however - that M_j varies inversely with n is bane for capability users, and boon to resilience researchers. Above we see that M_j varies as $1/\sqrt{n}$ for $m = 2$ and large n. Figure 5 shows M_j as a function of n for triple and quad redundancy as well. The linear slopes on the log-log plot indicate a general power law, and were numerically fit to be $-1/m$ at large n, suggesting that

$$\lim_{n \to \infty} M_j \propto \frac{1}{\sqrt[m]{n}}. \tag{13}$$

We can not currently provide an analytical proof of this starting from equation 4, but note related results for interested readers [22, 23, 45]. This can be left for future work, since successive levels of redundancy yield diminishing returns, and even dual-redundant computation is not yet an accepted supercomputing resilience strategy.

4.2 Nonuniform Redundancy

Above, we explore uniform redundancy. However, the financial cost of dedicating and powering redundant nodes is significant. Does running redundant computations on only one-half of job bundles provide one-half

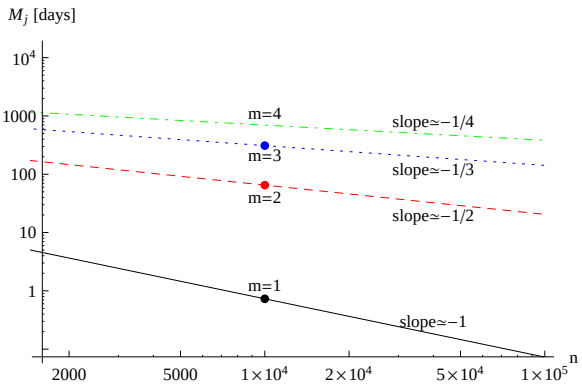


Figure 5. Mean time to job interrupt M_j as a function of job size n, exhibiting a power-law relationship (linear on a log-log plot). Slopes of $-1/m$ are independent of the underlying node failure rate, which in this case is the same as in Figure 3. The dots also match Figure 3.

the benefit? Or, if we are running fully redundant, but then one bundle experiences a node failure, we are now running on bundles of varying size - is the best response strategy to checkpoint immediately to minimize rework, or to wait a while longer before checkpointing? Or should we try to replace the failed node with a hot spare? These motivate us to study the case of nonuniform redundancy, where a job runs on bundles of different sizes.

For clarity and relative simplicity, we focus here on the case of a job running on only two distinct bundle sizes - for example, some single-node bundles (non-redundant) and some dual-redundant bundles. Given N total nodes on which to run a job of size n, all bundles can have a redundancy level of at least $m = \lfloor \frac{N}{n} \rfloor$. In addition, $x = N - mn$ nodes are left over, which can be distributed to other bundles to increase their redundancy level by one. We then have $n - x$ bundles of redundancy m, and x bundles of redundancy $m+1$, and equation 5 becomes

$$R_j = (1 - F^m)^{n-x}(1 - F^{m+1})^x. \tag{14}$$

This of course reduces to equation 4 for uniform redundancy, where $\frac{N}{n}$ is a positive integer and thus $x = 0$.

Also for simplicity, first consider a job on $n = 5$ bundles. All bundles can be non-redundant ($N = 5$), or four non-redundant and one dual-redundant ($N = 6$), and so on up to five dual-redundant bundles ($N = 10$). The resulting R_j and M_j are depicted in Figure 6. Adding nodes to a subset of a job's bundles does not result in a linear increase in mean time to job interrupt. Just above uniform redundancy, additional nodes make a very small difference compared to their effect just below a uniform redundancy level. The smaller reliability of non-redundant bundles has a greater effect in the overall product than the larger reliability of redundant bundles. Figures 2 and 6 are complimentary (and use the same node failure rate), but note that the time axis here is reduced by a factor of 5, corresponding to a job size of 5 - the reliability of a job on n bundles is the product of the reliability of all n bundles, then integrate for mean time to interrupt.

Figure 7 shows the trend for large job sizes. Again, it can be clearly seen that loosing a few nodes below a uniform redundancy level has a much greater impact than adding a few above a uniform redundancy level. We observe steps at uniform redundancy levels, and non-linear increases between them, caused by the exponents in equation 14. The dots in Figures 3, 5, and 7 all indicate the same points, in order to facilitate comparison from multiple perspectives. The practical lesson here is that nonuniform redundancy is of limited value. If redundancy is to be deployed, it is most beneficial to deploy it at uniform levels. Given uniform levels, extra nodes could be used as hot-spares to repair non-uniform bundles; we discuss this in section 6.

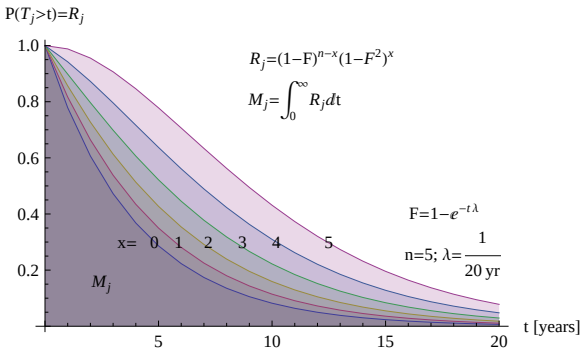


Figure 6. The increases in mean time to job interrupt M_j accelerate towards uniform redundancy levels. This plot shows a small job ($n = 5$), running at no redundancy ($x = 0$), increasing non-uniform redundancy ($x = 1$ to 4), and fully dual-redundant ($x = 5$).

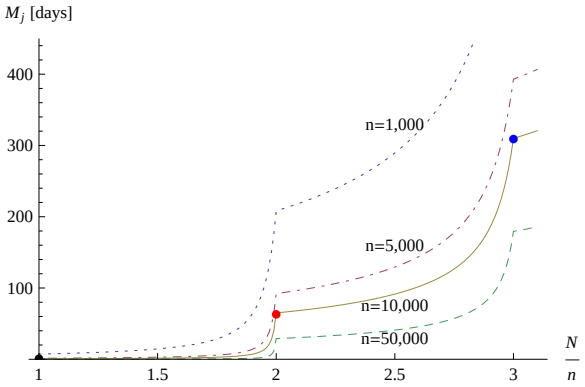


Figure 7. Mean time to job interrupt M_j does not increase linearly with increasing total nodes N given to jobs of size n running with non-uniform redundancy. M_j is affected most just below uniform redundancy boundaries, where $\frac{N}{n}$ is a positive integer. The dots match those in Figures 3 and 5.

5 Modeling Application Runtime, with Arbitrarily Distributed Job Interrupts

Daly has developed a model of total application runtime, taking into consideration the time spent writing checkpoints at regular intervals, and time spent restarting from such a checkpoint [10]. We refer interested readers to his full derivation, and here jump in with his distribution-independent equation for the total time an application runs in order to reach a solution, T_w (equation 18 in [10]). This is distinct from the time a job runs, because in order to reach a solution in the presence of interruptions, multiple jobs are needed.

$$T_w = \frac{M_j\{T_s - \delta + \delta T_s/\tau\}}{M_j - \{E(\tau+\delta)+R\}R_j(R+\tau+\delta) - E(R+\tau+\delta)F_j(R+\tau+\delta)} \tag{15}$$

This is composed of T_s, the time required to solve a computational problem (eg, without any interruption overhead), a fixed interval of time between checkpoints τ, time to write the checkpoint δ, and time to restart R after an interruption has occurred. We have used our notation of M_j, F_j, and R_j, rather than his M, $P(\tau)$ and $1 - P(\tau)$ respectively. $E(\Delta t)$ denotes the expected time of interruption within the interval Δt, so $E(\tau + \delta)$ is the expected time of interrupt during a compute-checkpoint interval, and $E(R + \tau + \delta)$ is the same but including a restart (from the preceding interruption).

Daly's next steps assume that job interrupts are exponentially distributed, which follows from exponentially distributed node failures on non-redundant systems. As seen in Figure 3 however, even exponentially distributed node failures do not result in exponentially distributed job interrupts on redundant systems, and furthermore we want to explore non-exponential node failures [38]. Only a minor revision to how $E(\Delta t)$ is calculated is necessary to enable these paths of exploration. As in Daly's case

$$E(\Delta t) = \int_0^{\Delta t} t g(t) dt, \tag{16}$$

but $g(t)$ (the probability of interrupt at time t into any Δt interval), becomes

$$g(t) = \sum_{i=0}^{\infty} f_j(t + i\Delta t), \tag{17}$$

where $f_j(t)$ is a job interrupt PDF for a system of arbitrary distribution or redundancy from Section 3. For $m = 1$ and exponentially distributed failures, this reduces exactly to Daly's model, as it should.

In this Section only, we use fixed parameters matching [10] Figure 5: $T_s = 500$ hours, $R = 10$ minutes, $\delta = 5$ minutes, and $M_j = 15$ minutes (for $m = 1$ and $n = 100$). While these are not practical per Section 2, they do provide a direct bridge of comparison with Daly's work, upon which we are building. Importantly, we note that this section's results are consistent with those in the next (where more practical values are used), so our conclusions here are general rather than limited to these parameter values. Figure 8 shows application runtime T_w, as a function of time between checkpoints τ. The non-redundant case ($m = 1$) agrees with [10], and the redundant cases show that redundant computation can yield a solution much faster than non-redundant. We have also written a simple discrete-event simulator, and show its results as dots in Figure 8, showing consistency with the model.

The minimums for each curve in Figure 8 correspond to the checkpoint interval which minimizes time to solution, τ_{opt}. We have already seen that redundant computation provides longer mean time to job interrupt than non-redundant, it follows that optimal checkpoint interval is also longer. Fewer interrupts are encountered, meaning fewer restarts and fewer rework segments, thus requiring fewer checkpoints. In this case, dual redundancy enables a minimum total runtime of 722 hours (with $\tau_{opt} = 32$ minutes, determined by numerical minimization) compared to $2,504$ hours (with $\tau_{opt} = 9$ minutes) without redundancy. For

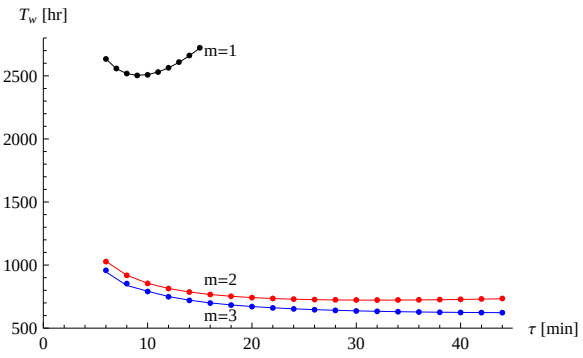


Figure 8. Model (lines) and simulation (dots) results for total application runtime T_w (hours) as a function of checkpoint interval τ (minutes). For a cost of twice the nodes ($m = 2$), time to solution is reduced by more than two thirds (T_w of 722 vs 2,504 hours) - justifying dual and even triple redundant computation for Daly's parameter values.

twice the nodes, time to solution is cut to less than one third - clearly demonstrating that redundancy's benefits can overtake its costs. In this case, the non-redundant system spends a ridiculous four fifths of its runtime dealing with interrupts rather than making meaningful progress solving the problem at hand ($\simeq (T_w - T_s)/T_w$). The next subsection identifies where the crossover occurs from being ridiculous to run redundant, versus being ridiculous not to (such as in this case). However, we must first touch on estimates for optimal checkpoint interval and total runtime.

While closed-form solution of τ_{opt} on redundant systems is a topic of future work, we can provide observations from numerical evaluation of the model and simulation. Young proposed [50]

$$\tau_{opt} = \sqrt{2\delta M_j}, \tag{18}$$

which Daly revised to

$$\tau_{opt} = A\sqrt{2\delta M_j} - \delta, \tag{19}$$

where τ_{opt} is within 5% of true optimal for $A = 1$ and $\delta < \frac{1}{2}M_j$. He also provided a "higher-order" estimate within 0.2% of true optimal, where $A = 1+\frac{1}{3}\left(\frac{\delta}{2M_j}\right)^{1/2}+\frac{1}{9}\left(\frac{\delta}{2M_j}\right)$ and $\delta < 2M_j$ [10]. Table 2 shows the results of using each of these estimators for the experiment in Figure 8. Although these all assume exponentially distributed job interrupts (which we have shown not to be the case), they all result in a T_w within 1% of true optimal (per numeric optimization of equation 15). This is because the redundant-computation systems are operating in a very stable region, where mean time to interrupt is significantly less than time to checkpoint. Given its simplicity and prevalence in the literature, we will use Daly's estimate with $A = 1$ for the rest of this paper, which focusses on dual-redundancy.

Since Daly's estimate for τ_{opt} is still valid for dual-redundant systems, perhaps his exponential-interrupt solution for T_w is as well? Using our birthday-problem estimates dual-redundancy, this would be

$$\hat{T}_w = \hat{M}_j e^{R/\hat{M}_j}\left(e^{(\tau+\delta)/\hat{M}_j} - 1\right)\frac{T_s}{\tau}. \tag{20}$$

We find that this significantly underestimates T_w for dual-redundancy, as shown in Figure 9, so the full formulation in equation 15 for total runtime should be used instead for accuracy. We reiterate that τ_{opt} and

τ_{opt} [min] : T_w [hr]	$m = 1$	$m = 2$	$m = 3$
Young	12.2 : 2,573	37.5 : 724.8	57.0 : 622.0
Daly ($A = 1$)	7.2 : 2,546	32.5 : 722.1	52.0 : 621.5
Daly (higher-order)	9.1 : 2,504	34.1 : 722.5	53.8 : 621.5
Numeric solution	9.1 : 2,504	31.8 : 722.1	61.2 : 621.4

Table 2. Total application runtime T_w resulting from different estimates for optimal checkpoint interval τ_{opt}. The total difference in T_w among estimates is minisule.

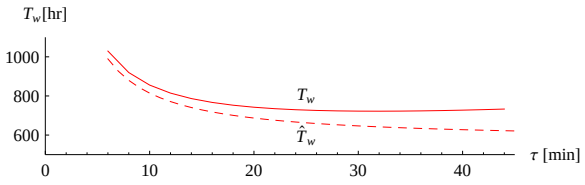


Figure 9. Simply estimating $\hat{M}_j$ and plugging it into Daly's runtime equation for non-redundant systems (Equation 20) can significantly underestimate true runtime T_w on dual redundant computation systems, 12% at τ_{opt} in this case. Parameter values here match those in Figure 8.

$\hat{T}_w$ were evaluated for the experiments in the next section, and were consistent with the above conclusions; Daly's parameter values were only used here as a bridge of comparison to his work.

6 Conditions Where Dual-Redundant Computation Is Cost Effective

It has been intuitively suggested that process-pairs may be cost-effective once an application spends roughly half of its runtime on fault-related activities such as checkpoints and restarts [18]. We have studied various aspects of redundant computation, but none is more important than this bottom line: at what point does redundant computation become cost effective? Regarding fault-tolerance, application efficiency is the ratio of solve time to total runtime, T_s/T_w [11]. When this falls below 50%, the application is spending more than half its time coping with interrupts rather than solving the problem the application was written for. Here we examine the straightforward case that a given system of N nodes is used to run an application. It can run the system with N application ranks in non-redundant mode, or can run the application with $N/2$ ranks in dual-redundant computation mode via rMPI or similar [36]. In this case, total runtime is a complete cost comparison, as the equipment and power costs are the same per unit time.

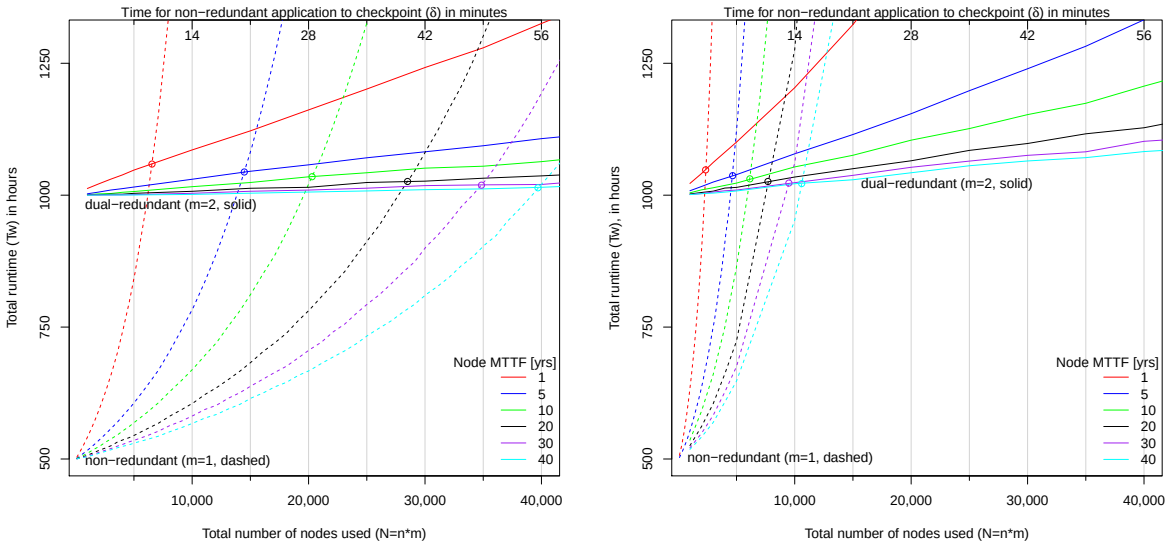
Given a total of N nodes that never fail, let us assume that the total runtime of the $N/2$ rank application would be twice that of the N rank application (weak scaling). This is pessimistic because real applications are rarely embarrassingly parallel - in actuality, running on half the nodes will require less than twice the time due to decreased parallelism overhead. However, let us also assume that the overhead incurred by the redundant computation mechanism is zero. Thus, we use balanced first-order assumptions - the runtime cost of redundancy is paid for by gains in application scaling.

The aforementioned simulator was used to generate the results in this section, as it was faster than our tools for numerical evaluation of the analytical model for redundant systems at large N. The simulator was carefully verified as matching all the analytical model results we have presented thus far.

Take the case where T_s=500 hours for $m = 1$ and T_s=1000 hours for $m = 2$: Figure 10(a) depicts total runtime T_w as a function of total nodes $N = nm$, for a reasonable range of node failure rates, and assuming exponentially-distributed failures. If node mean time to failure ($1/\lambda$) is 5-10 years [18, 31], it will be more cost-effective to run redundant rather than non-redundant on a system size of 15-20 thousand nodes. If $1/\lambda$ is 20 years, crossover occurs at a system size of approximately 28k nodes. All of these values are well within the practical bounds of section 2.

Whereas some studies assume a fixed time to write a checkpoint [10, 41, 34], the simulations underlying Figure 10(a) assume that checkpoint time δ increases linearly with job size n (as does [31]). This is practical - the more nodes, the more memory to checkpoint. Although network capacity also increases with n, the I/O subsystem quickly becomes the limiting factor [31]. We also assume a fast filesystem per Section 2, and apply the same 80% of theoretical bandwidth limitation, such that $\delta = n * 16GB/(.8 * 240GB/s)$. So, checkpoint time δ increases linearly in Figure 10(a): from 9 minutes at the crossover for node failure rate λ of 1 per year, to 56 minutes for $\lambda = 1/40yr$, for the non-redundant case of N application ranks. For the dual-redundant case of $N/2$ application ranks, the checkpoint time is half those values, as only one node in each bundle needs to write a checkpoint. This opens another opportunity - perhaps the non-checkpointing nodes could continue computing, and when the checkpointing nodes finish they could fast-forward to the state of the non-checkpointing nodes, further decreasing the runtime impact of checkpoints. It is common for HPC filesystems to be tuned for writes rather than reads, and restarts require some computational setup, so we use a restart time of $R = 2\delta$. Notice also that crossover occurs at an efficiency T_s/T_w of slightly less than 50%, because the dual-redundant system is also spending time on checkpoints, restarts, and rework. However, it is operating very efficiently - better than 95% for all cases in Figure 10(a) at crossover.

So far, we have only considered the case where a job is interrupted upon the first bundle failure. However, this is not the only option available. Lets say that we can repair or replace the failed node of a bundle before its partner fails, 50% of the time. For instance, modify the Linux kernel so that it does not panic upon double-bit memory errors, but instead remaps memory to avoid the associated memory cell. It could then replace the contents of the failed memory cell using the good copy of the parter, and resume computation. Or, we quickly replace failed nodes in bundles with hot-spares, half of the time. These correspond to a 50%



(a) Exponential node failures [10].

(b) Weibull node failures with 0.8 hazard rate [38].

Figure 10. Here we show total time to solution T_w as a function of the total number of nodes used $N = nm$, for non-redundant (dashed) and dual-redundant (solid) systems, assuming solve times T_s of 500 hours and 1000 hours respectively. Each simulation was run 100 times, and optimal checkpoint interval τ_{opt} was calculated appropriately throughout. The crossover points (circles) where redundancy becomes more cost effective than non-redundant fall well within the size and failure rates of existing systems - especially in the Weibull case.

reduction in node failure rate λ, and a crossover point even closer to 50% application efficiency. However, again, the redundant system is operating in a range that a 50% decrease in node failure rate yields only a very small net reduction in runtime. Thus, while repair/replace mechanisms which redundancy would enable would be a worthwhile feature, they are not the killer feature. The killer features are the increase in mean time to interrupt, such that fewer interrupts are encountered and fewer checkpoints and restarts are required, and secondly that checkpoint and restart times are cut in half due to running on half the nodes. However, using repair/replace to avoid the non-uniform redundancy cliff described in Section 4, would yield dramatic benefits - jobs would only interrupt when bundle repair does not succeed, and checkpoint intervals could be adjusted accordingly.

We have focussed so far on exponentially distributed node failures, but Schroeder has shown this assumption to be "suspect" [38], and found that Gamma or Weibull distributions with a decreasing hazard rate fits the data more accurately. Figure 10(b) is based on her suggestion of Weibull node failures with a 0.8 hazard rate. The result is that cost-crossover points for each failure rate occur at significantly lower node counts than with exponential failures. Thus, if the literature on failure rates (Section 2) and models (Weibull) are correct, thorough consideration of redundant computation for extreme scale systems is overdue.

7 Conclusion and Recommendation

Via standard reliability modeling methods, we have examined the properties of failures and interrupts on redundant computation systems, including non-intuitive properties of non-uniform redundancy. We have adapted Daly's application runtime model to support arbitrarily distributed interrupts, and found that optimal checkpoint intervals are easily estimated for redundant systems as they are on non-redundant. We have shown that the crossover point where redundant computation is more cost effective than non-redundant is well within existing system sizes and node failure rates and distributions - justifying thorough and immediate consideration of redundant computation for extreme scale systems.

We strongly recommend that next steps include the direct measurement of application efficiency - including interrupt overheads - on real systems. This is certainly possible [11], but is rarely done (or reported) judging by its sparsity in the literature and our discussions with users. Such measurements will not only facilitate wise investment decisions regarding resilience, but should be of immediate value to the users, sites, and sponsors of large systems. We applaud the HPC community for increasing releases of failure data [2], and efforts to provide tools for measuring application efficiency [6, 1, 5], and recommend that simple tools be developed and made available for wide use. The required data resides within distinct communities (namely, administrators and users) - another example that solutions to the resilience issues facing exascale computing will require unified efforts to overcome [7, 21]. With teamwork and open minds, the prospects of yet-more-extreme computing remains bright, despite the anticipated resilience challenges.

References

[1] Advanced simulation and computing fy10-fy11 implementation plan. Technical report, WBS 1.5.4.7-TRI-004 Application Monitoring, https://e-reports-ext.llnl.gov/pdf/378111.pdf.

[2] Computer failure data repository.

[3] http://www.nccs.gov/jaguar/.

[4] http://www.sandia.gov/asci/red/redfacts.htm.

[5] Robert A. Ballance and Jonathan Cook. Monitoring mpi programs for performance characterization and management control. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2305–2310, New York, NY, USA, 2010. ACM.

[6] Robert A. Ballance and Nathan A. DeBardelen. Non-invasive job progress monitoring: The MoJo application monitoring tool suite. In *LCI Conference on High-Performance Clustered Computing*, Pittsburgh, PA, March 2010.

[7] Franck Capello, Al Geist, Bill Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. Toward exascale resilience. Technical report, Technical Report of the INRIA-Illinois Joint Laboratory on PetaScale Computing (TR-JLPC-09-01), 2009.

[8] Kim Cupps and M. Gary et al. Fy02 i/o integration blueprint (https://e-reports-ext.llnl.gov/pdf/246963.pdf). Technical report, Lawrence Livermore National Laboratory, 2001.

[9] J. T. Daly, L. A. Pritchett-Sheats, and S. E. Michalak. Application mttfe vs. platform mtbf: A fresh perspective on system reliability and application throughput for computations at scale. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 795–800, Washington, DC, USA, 2008. IEEE Computer Society.

[10] John Daly. A higher-order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22:303–312, 2006.

[11] John Daly. Performance challenges for extreme scale computing. Technical report, Los Alamos National Laboratories, 2007.

[12] Peter M. Kogge (editor). Exascale computing study: Technology challenges in achieving exascale systems. Tech. Report TR-2008-13, Univ. of Notre Dame, CSE Dept., Sept. 28 2008.

[13] E N Elnozahy, R Bianchini, T El-Ghazawi, and Armando Fox. System resilience at extreme scale. White Paper.

[14] Christian Engelmann. *Symmetric Active/Active High Availability for High-Performance Computing System Services*. PhD thesis, Department of Computer Science, University of Reading, UK, 2008.

[15] Christian Engelmann, Hong H. Ong, and Stephen L. Scott. The case for modular redundancy in large-scale high performance computing systems. In *Proceedings of the* 8^{th} *IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2009*, pages 189–194, Innsbruck, Austria, February 16-18, 2009. ACTA Press, Calgary, AB, Canada.

[16] Kurt Ferreira, Rolf Riesen, Ron Oldfield, Jon Stearley, James Laros, Kevin Pedretti, Ron Brightwell, and Todd Kordenbrock. Increasing fault resiliency in a message-passing environment. Technical report SAND2009-6753, Sandia National Laboratories, October 2009.

[17] Philippe Flajolet, Peter J. Grabner, Peter Kirschenhofer, and Helmut Prodinger. On Ramanujan's Q-function. *Journal of Computational and Applied Mathematics*, 58:103–116, 1992.

[18] G Gibson, B Schroeder, and J Digney. Failure tolerance in petascale computers. *CTWatch Quarterly*, 3, 2007.

[19] J. N. Glosli, D. F. Richards, K. J. Caspersen, R. E. Rudd, J. A. Gunnels, and F. H. Streitz. Extending stability beyond cpu millennium: a micron-scale atomistic simulation of kelvin-helmholtz instability. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, pages 58:1–58:11, New York, NY, USA, 2007. ACM.

[20] Gary Grider. Speed matching and what economics will allow. Technical report, HEC FSIO Research and Development Workshop, 2010.

[21] R. Gupta, P. Beckman, B. H. Park, E. Lusk, P. Hargrove, A. Geist, A. Lumsdaine, and J. Dongarra. Cifts: A coordinated infrastructure for fault-tolerant systems. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, 2009.

[22] Norbert Henze. A poisson limit law for a generalized birthday problem. *Statistics and Probability Letters*, 39(4):333–336, 1998.

[23] Lars Holst. The general birthday problem. *Random Structures and Algorithms*, 6(2-3):201–208, July 2007.

[24] Chung-hsing Hsu and Wu-chun Feng. A power-aware run-time system for high-performance computing. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 1, Washington, DC, USA, 2005. IEEE Computer Society.

[25] D. E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 1973.

[26] Ken Koch. Roadrunner platform overview. Technical report, http://www.lanl.gov/orgs/hpc/roadrunner/pdfs/Kochw/RR

[27] Frank H. Mathis. A generalized birthday problem. *SIAM Review*, 33(2):265–270, June 1991.

[28] Dennis McEvoy. The architecture of tandem's nonstop system. In *ACM '81: Proceedings of the ACM '81 conference*, page 245, New York, NY, USA, 1981. ACM.

[29] Adam Moody. The scalable checkpoint / restart (scr) library: Approaching file i/o bandwidth of 1 tb/s. Technical report, Lawrence Livermore National Laboratory, March 2009.

[30] Anh Nguyen-Tuong, David Evans, John C. Knight, Benjamin Cox, and Jack W. Davidson. Security through redundant data diversity. In *38th IEEE/IFPF International Conference on Dependable Systems and Networks*, 2008.

[31] Ron A. Oldfield, Patricia J. Teller, Maria Ruiz Varela, Philip C. Roth, Sarala Arunagiri, Seetharami Seelam, and Rolf Riesen. Modeling the impact of checkpoints on next-generation systems. In *In Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*, pages 30–43. ACM Press, 2007.

[32] Hewlett Packard. HP NonStop computing. `http://h20338.www2.hp.com/NonStopComputing/cache/76385-0-0-0-121.html`.

[33] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 109–116, New York, NY, USA, 1988. ACM.

[34] Rold Riesen, Kurt Ferreira, Jon Stearley, James Laros, Kevin Pedretti, and Ron Brightwell. Redundant computing for exascale systems. In *Proceedings of the 2011 EuroSys Conference - IN SUBMISSION*, 2011.

[35] Rolf Riesen, Kurt Ferreira, and Jon Stearley. See applications run and throughput jump: The case for redundant computing in hpc. In *1st Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2010)*, 2010.

[36] Rolf Riesen, Kurt Ferreira, Jon Stearley, Ron Oldfield, James H. Laros III, Kevin Pedretti, and Ron Brightwell. Redundant computing for exascale systems. Technical report SAND2010-8709, Sandia National Laboratories, December 2010.

[37] Bianca Schroeder and Garth A. Gibson. Disk failures in the real world: What does an mttf of 1,000,000 hours mean too you? In *5th Usenix Conference on File and Storage Technologies (FAST 2007)*, 2007.

[38] Bianca Schroeder and Garth A Gibson. Understanding failures in petascale computers. *Journal of Physics: Conference Series*, 78(1):012022, 2007.

[39] Jon Stearley. Defining and measuring supercomputer reliability, availability, and serviceability (ras). In *In Proceedings of the Linux Clusters Institute Conference*, 2005.

[40] Jon Stearley and Robert Ballance. A preliminary report on red storm ras performance. In *Proceedings of the 2006 Cray Users Group Meeting*, 2006.

[41] Rajagopal Subramaniyan, Eric Grobelny, Scott Studham, and Alan D. George. Optimization of checkpointing-related i/o for high-performance parallel and distributed computing. *J. Supercomput.*, 46:150–180, November 2008.

[42] Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. *Lustre 1.8 Operations Manual*, 1.3 edition, March 2010.

[43] J. C. Turner. Reliability and operability of systems of components in series and in parallel. *Electronics Reliability and Microminiaturization*, 1:21–26, 1962.

[44] Andrew Uselton and Brian Behlendorf. Visualizing i/o performance during the bgl deployment. In *Proceedings of the 2007 Linux Clusters Institute Conference*, 2007.

[45] David Wagner. A generalized birthday problem. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '02, pages 288–303, London, UK, 2002. Springer-Verlag.

[46] L. Wang, Karthik Pattabiraman, Z. Kalbarczyk, R.K. Iyer, L. Votta, C. Vick, and A. Wood. Modeling coordinated checkpointing for large-scale supercomputers. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 812 – 821, 2005.

[47] Richard Williams. *Electrical Engineering Probability*. West, 1991.

[48] Ming Wu, Xian-He Sun, and Hui Jin. Performance under failures of high-end computing. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–11, New York, NY, USA, 2007. ACM.

[49] Mitsuo Yokokawa. Present status of development of the earth simulator. *Innovative Architecture for Future Generation High-Performance Processors and Systems, International Workshop on*, 0:0093, 2001.

[50] John W. Young. A first order approximation to the optimum checkpoint interval. *Commun. ACM*, 17:530–531, September 1974.

[51] Liu Yudan, R Nassar, C Leangsuksun, N Naksinehaboon, M Paun, and S.L. Scott. An optimal checkpoint/restart model for a large scale high performance computing system. In *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008.*, pages 1 – 9, 2008.

[52] Ziming Zheng and Zhiling Lan. Reliability-aware scalability models for high performance computing,. In *Cluster'09: Proceedings of the IEEE conference on Cluster Computing*, 2009.

DISTRIBUTION:

1	MS 1319	Jon Stearley , 1422
1	MS 1319	David Robison , 1464
1	MS 1319	Kurt Ferreira , 1423
1	MS 0899	Technical Library, 9536 (electronic)
1	MS 0359	D. Chavez, LDRD Office, 1911

Sandia National Laboratories