The Reliability Wall for Exascale Supercomputing

Xuejun Yang, *Member, IEEE*, Zhiyuan Wang,
Jingling Xue, *Senior Member, IEEE*, and Yun Zhou

Abstract—Reliability is a key challenge to be understood to turn the vision of exascale supercomputing into reality. Inevitably, large-scale supercomputing systems, especially those at the peta/exascale levels, must tolerate failures, by incorporating fault-tolerance mechanisms to improve their reliability and availability. As the benefits of fault-tolerance mechanisms rarely come without associated time and/or capital costs, reliability will limit the scalability of parallel applications.

This paper introduces for the first time the concept of "Reliability Wall" to highlight the significance of achieving scalable performance in peta/exascale supercomputing with fault tolerance. We quantify the effects of reliability on scalability, by proposing a reliability speedup, defining quantitatively the reliability wall, giving an existence theorem for the reliability wall, and categorizing a given system according to the time overhead incurred by fault tolerance. We also generalize these results into a general reliability speedup/wall framework by considering not only speedup but also costup. We analyze and extrapolate the existence of the reliability wall using two representative supercomputers, Intrepid and ASCI White, both employing checkpointing for fault tolerance, and have also studied the general reliability wall using Intrepid. These case studies provide insights on how to mitigate reliability-wall effects in system design and through hardware/software optimizations in peta/exascale supercomputing.

Index Terms—fault tolerance, exascale, performance metric, reliability speedup, reliability wall, checkpointing.

1 Introduction

In these past decades, the performance advances of supercomputers have been remarkable as their sizes are scaled up. Some fastest supercomputers crowned in the TOP500 supercomputing list in the past decade are ASCI White in 2000, an IBM SP (Scalable POWERparallel) cluster system consisting of 8,192 cores with an Rpeak of 12 TeraFlops, Blue Gene/L in 2005, an IBM MPP system consisting of 131,072 cores with an Rpeak of 367 TeraFlops, and Tianhe-1A in 2010, currently the world's fastest PetaFlop system, a NUDT YH MPP system consisting of 186,368 cores with an Rpeak of 4.701 PetaFlops. Therefore, scalable parallel computing has been the common approach to achieving high performance.

Despite these great strides made by supercomputing systems, much of scientific computation's potential remains untapped "because many scientific challenges are far too enormous and complex for the computational resources at hand" [1]. Planned exascale supercomputers (capable of an exaflop, 10^3 petaflops, or 10^{18} floating point operations per second) in this decade promise to overcome these challenges by a revolution in computing at a greatly accelerated pace [2]. However, exascale supercomputing itself faces a number of major challenges, including technology, architecture, power, reliability, programmability, and usability. This research addresses the reliability challenge when building scalable supercomputing systems, particularly those at the peta/exascale levels.

The reliability of a system is referred to as the probability that it will function without failure under stated conditions for a specified amount of time [3]. MTTF (*mean time to failure*) is a key reliability metric. It is generally accepted in the supercomputing community that exascale systems will have more faults than today's supercomputers do [4], [5] and thus must handle a continuous stream of faults/errors/failures [6]. Thus, fault tolerance will play a more important role in the future exascale supercomputing.

While many fault-tolerance mechanisms are available to improve the reliability of a system, their benefits rarely come without time and/or capital costs. For example, checkpointing [7], [8] improves system reliability but incurs the additional time on saving checkpoints and performing rollback-recovery (as well as the capital cost associated with the hardware used). According to some projections [9] made based on existing technologies, the time spent on saving a global checkpoint may reach or even exceed the MTTF of a system when the system performance sits between the petascale and exascale levels. Therefore, reliability will inhibit scalability in peta/exascale supercomputing.

In this research, we analyze and quantify the effects of reliability, which is realized at the extra time and

- *X. Yang, Z. Wang and Y. Zhou are with the School of Computer, National University of Defense Technology, ChangSha, Hunan, 410073, China.*
  E-mail: {xjyang, zy_vxd}@nudt.edu.cn and wang_zhiyuan17@yahoo.com.cn
- *J. Xue is with the School of Computer Science and Engineering, University of New South Wales, Australia.*
  E-mail: jingling@cse.unsw.edu.au

dollar cost induced by fault tolerance, on the scalability of peta/exascale supercomputing. Our case studies provide insights on how to mitigate reliability-wall effects when building reliable and scalable parallel applications and parallel systems at the peta/exascale levels.

The main contribution are summarized below:

- We introduce for the first time the concept of "Reliability Wall", thereby highlighting the significance of achieving scalable performance in peta/exascale supercomputing with fault tolerance.
- For a fault-tolerant system, we present a reliability speedup model, define quantitatively the reliability wall, give an existence theorem of the reliability wall, and categorize it according to the time overhead induced by fault tolerance (Section 3).
- We generalize our results to obtain a general reliability speedup model and to quantify the general reliability wall for a system when the dollar cost induced by the fault-tolerance mechanism used is also taken into account, by considering simultaneously both speedup and costup [10] (Section 4).
- We conduct a series of case studies using two real-world supercomputers, Intrepid and ASCI White, both employing checkpointing, the predominant technique for fault tolerance used in practice (Sections 5.1 – 5.3). Intrepid is an IBM Blue Gene/P MPP system while ASCI White is an IBM SP cluster system, with both being representatives of the majority of the TOP500 list. As exascale systems do not exist yet, we use these two supercomputing systems to verify, analyze and extrapolate the existence of the (general) reliability wall for a number of system configurations, depending on whether checkpointing is full-memory or incremental-memory and whether I/O is centralized or distributed, as the systems are scaled up in size. Furthermore, we show that the checkpoint interval optimization does not affect the existence of the (general) reliability wall.
- We describe briefly how to apply our reliability-wall results to mitigate reliability-wall effects in system design (e.g., on developing fault tolerance mechanisms and I/O architectures) and through hardware/software optimizations in supercomputing, particularly peta/exascale supercomputing (Section 5.4).

2 Related Work

Supercomputing systems have mainly relied on checkpointing for fault tolerance. As this paper represents the first work on quantifying the Reliability Wall for supercomputing systems based on a new reliability speedup model, we review below the prior work on speedup and costup modeling and on checkpointing techniques.

2.1 Speedup and Costup

Speedup has been almost exclusively used for measuring scalability in parallel computing, such as Amdahl's speedup, Gustafson's speedup or memory-bounded speedup [11], [12], [13], [14]. These models are concerned only with the computing capability of a system.

Amdahl proposed a speedup model for a *fixed-size problem* [11]:

$$S_{P,\mathit{Amdahl}} = \frac{P}{1+f(P-1)} \tag{1}$$

where P is the number of processors and f is the serial part of the program. Interestingly, Amdahl's speedup argues against the usefulness of large-scale parallel computing systems.

To overcome the shortcomings of Amdahl's speedup model, Gustafson [12] presented a scaled speedup for a *fixed-time problem*, which scales up the workload with the increasing number of processors in such a way as to preserve the execution time:

$$S_{P,\mathit{Gustafson}} = f + P(1-f) \tag{2}$$

Later, Wood and Hill considered a costup metric [10]:

$$C = \frac{C_P}{C_1} \tag{3}$$

where C_P is the dollar cost of a P-processor parallel system and C_1 is the dollar cost of a single-processor system. They argued that parallel computing is cost-effective whenever speedup exceeds costup.

In this work, we have developed our reliability wall framework based on these existing speedup and costup models.

2.2 Fault-Tolerant Techniques

A plethora of fault-tolerance techniques have been proposed to improve the reliability of parallel computing systems. We consider software solutions first and then hardware solutions.

Of many software techniques developed for fault tolerance, checkpointing has been the most popular in large-scale parallel computing. Checkpointing can occur at the operating system level or the application level. In the first approach, system-level checkpointing [15], [16] only requires the user to specify a checkpoint interval, with no additional programmer effort. On the other hand, application-level checkpointing [17], [18], [19] requires the user to decide the placement and contents of the checkpointing. This second approach requires more programmer effort but provides the user an opportunity to place checkpoints in such a way that checkpoint contents and checkpointing time are reduced. In order to combine the best of these two approaches, compiler-assisted checkpointing [20], [21], [22], once coupled with application-level checkpointing, can provide a certain degree of transparency to the user. Our case studies focus on system-level checkpointing techniques. By checkpointing, we mean system-level checkpointing from now on unless indicated otherwise.

Saving the states for thousands of processes at a checkpoint in a system can result in a heavy demand on the underlying I/O and network resources, rendering this part of the system a performance bottleneck. To alleviate this problem, many checkpointing optimizations have been considered. Instead of full-memory checkpointing, which involves saving the entire memory context of a process at a checkpoint, incremental-memory checkpointing opts to save only the dirty pages, i.e., the ones that have been modified since the last checkpoint [23]. Thus, incremental-memory checkpointing can reduce the amount of memory contexts that need to be saved, especially for large systems. In our case studies discussed in Section 5, both types of checkpointing techniques are considered.

In diskless checkpointing [24], [25], each processor makes local checkpoints in local memory (or local disk), eliminating the performance bottleneck of traditional checkpointing. Some form of redundancy is maintained in order to guarantee checkpoint availability in case of failures. However, diskless checkpointing doubles the memory occupation and may not be well suited to exascale supercomputing [4].

Fault tolerance at the hardware level has been achieved mostly through redundancy or replication of resources, either physically (as in RAIDed disks [26] and backup servers) or information-wise (as in ECC memory [27], parity memory [28] and data replication [29]). In the case of N modular redundancy [30], for example, N systems perform a common operation and their results are processed by a voting system to produce a single output, at the cost of employing $N-1$ redundant duplicate copies. In data replication, data are copied into a local node from remote nodes to facilitate local rather than remote reads, improving the overall system performance. Due to data redundancy, data replication may also improve the reliability of communication, at the cost of additional storage space.

In summary, the overhead introduced by a fault-tolerance mechanism includes both a time component and a dollar cost component, limiting scalability as the system size is scaled up. Below we analyze and quantify the effects of reliability achieved through fault tolerance techniques on scalability in supercomputing.

3 The Reliability Wall

Table 1 provides a list of main symbols used, their meaning and where they are defined in the paper.

Section 3.1 characterises the time overhead incurred by fault-tolerance. Section 3.2 introduces a new reliability speedup model. Section 3.3 quantifies the reliability wall based on our reliability speedup model. Section 3.4 provides a classification of supercomputing systems based on their fault-tolerance time overhead.

3.1 Analyzing the Fault-Tolerance Time Overhead

Consider a parallel system consisting of P single-core processor nodes, denoted by $N_1,\ldots,N_P$. Let a parallel program G be executed on the system with one process per node, resulting in a total of P processes. (If a processor has more than one core, then P denotes the number of cores in the system with cores being treated as nodes. In this case, there will be one process running on each core.)

A system is called an R *system* if it uses some fault-tolerance mechanism, denoted R, to improve its reliability and R-*free* otherwise.

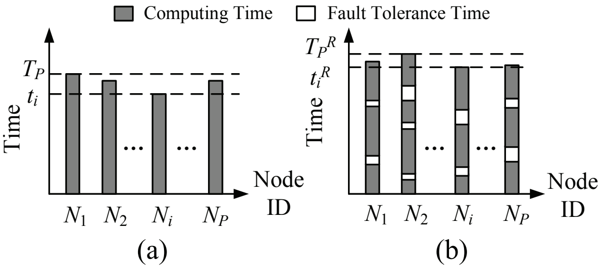


Fig. 1. Execution state of a program on a P-node system. (a) R-free system. (b) R system.

Figure 1 compares and contrasts the execution states of a program G on an R-free system and an R system. Let t_i (t_i^R) be the execution time of program G on node N_i in the R-free (R) system. Then the execution time of G on the R-free system is:

$$T_P = \max\{t_i, i = 1, \ldots, P\} \tag{4}$$

Similarly, the execution time of G on the R system is given by:

$$T_P^R = \max\{t_i^R, i = 1, \ldots, P\} \tag{5}$$

Thus, the time overhead induced by the fault-tolerance mechanism R when program G is run on an R system is found to be:

$$H = T_P^R - T_P \tag{6}$$

In general, most fault-tolerance mechanisms have positive time overhead values, i.e. $H > 0$. However, there are cases where $H \leqslant 0$, when, for example, a program runs on a data replication system with a

TABLE 1
List of main symbols, their meanings and definitions.

Symbol	Meaning	Definition
P	Number of single-core processor nodes	Sect. 3.1
R	Fault tolerance mechanism	Sect. 3.1
G	Parallel program with P processes	Sect. 3.1
f	Serial part of parallel program G	Sect. 2.1
T_P	Execution time of program G on an R-free system (with P nodes)	(4)
S_P	Traditional speedup on an R-free system	Sect. 3.2
$S_{P,\mathit{Amdahl}}$	Amdahl's speedup on an R-free system	(1)
$S_{P,\mathit{Gustafson}}$	Gustafson's speedup on an R-free system	(2)
E_P	Traditional efficiency on an R-free system	Sect. 3.2
T_P^R	Execution time of program G on an R system (with P nodes)	(5)
F	Average number of failures that may occur when program G runs on an R-free system	(7)
H	Time overhead induced by the fault-tolerance mechanism R when running program G on an R system	(6)
H_P	Average time incurred per failure for fault tolerance on an R system	(8)
M_P	MTTF of an R-free system	Sect. 3.1
$R(P)$	Fault tolerance time factor	(11)
E_P^R	Efficiency of the fault tolerance technique R	(12)
S_P^R	Reliability speedup	(9)
$S_{P,\mathit{Amdahl}}^R$	Amdahl's reliability speedup	(14)
$S_{P,\mathit{Gustafson}}^R$	Gustafson's reliability speedup	(15)
$C(P)$	Fault tolerance cost factor	(17)
C	Costup	(3)
C_P	Dollar cost of a P-processor parallel system	Sect. 2.1
C_1	Dollar cost of a single-processor system	Sect. 2.1
c_P	Dollar cost of the fault tolerance mechanism R	Sect. 4.1
C^R	Costup of an R system	(19)
C_P^R	Dollar cost of a P-node R system	Sect. 4.1
E_P^C	Efficiency of fault tolerance cost	(18)
S_P^{GR}	General reliability speedup	(16)
$\sup\limits_P S_P^R$	Reliability wall	Sect. 3.3
$\sup\limits_P S_P^{GR}$	General reliability wall	Sect. 4.2
m	Average number of checkpoints between failures	(21)
S_P^{R-full}	Reliability speedup for full-memory checkpointing	(25)
S_P^{R-inc}	Reliability speedup for incremental-memory checkpointing	(26)
$\sup\limits_P S_P^{R-full}$	Reliability wall for full-memory checkpointing	(28)
$\sup\limits_P S_P^{R-inc}$	Reliability wall for incremental-memory checkpointing	(29)
$S_P^{GR-full}$	General reliability speedup for full-memory checkpointing	(31)
$\sup\limits_P S_P^{GR-full}$	General reliability wall for full-memory checkpointing	Sect. 5.2

CC-NUMA storage architecture. By adopting the first-touch page allocation strategy, such a system only copies read-only pages of the program. If the program reads frequently and seldom modifies the page copies, then its execution time can be equal to or even smaller than that on the system without data replication. Obviously, such "overhead-free" fault-tolerance mechanisms do not limit scalability but are rarely achievable in peta/exascale supercomputing. Thus, this work focuses only on the fault tolerance mechanisms with positive time overhead values (when $H > 0$).

Let us use M_P to denote the MTTF of an R-free system. Suppose that $T_P \gg M_P$ for a large-scale parallel program G. When program G runs on an R-free system, the average number of failures that may occur can be approximated by:

$$F = \left\lfloor \frac{T_P}{M_P} \right\rfloor \tag{7}$$

Statistically, it is assumed that one failure occurs during every MTTF-sized time interval and no failure occurs during the last residual interval of length $T_P \bmod M_P$.

The average time incurred per failure for fault tolerance is defined as:

$$H_P = \frac{H}{F} = \frac{H}{\left\lfloor \frac{T_P}{M_P} \right\rfloor} \tag{8}$$

3.2 Defining the Reliability Speedup

In this section, we generalize the traditional concept of speedup to reliability speedup. In Section 3.3, we apply this generalization to analyze and quantify the effects of fault tolerance time overhead on scalability.

Recall that T_P given in (4) represents the (parallel) execution time of program G on an R-free system. Let T_{seq} be the (sequential) execution time of the sequential version of G. Thus, the traditional speedup is $S_P = \frac{T_{\text{seq}}}{T_P}$ and the efficiency achieved is $E_P = \frac{S_P}{P}$.

Definition 1 (Reliability Speedup)**.** When program G runs on an R system, the *reliability speedup* achieved is defined as follows:

$$S_P^R = \frac{T_{\text{seq}}}{T_P^R} \tag{9}$$

Our reliability speedup formula S_P^R can be refined below. To this end, let us refine T_P^R first. According to (6) – (8), we have:

$$\begin{aligned} T_P^R &= T_P + H = T_P + \left\lfloor \frac{T_P}{M_P} \right\rfloor H_P \\ &= T_P + T_P \left(\frac{1 - (T_P \bmod M_P)/T_P}{M_P} \right) H_P \\ &\approx T_P + T_P \frac{H_P}{M_P} = T_P \left(1 + \frac{H_P}{M_P} \right) \end{aligned} \tag{10}$$

where $\frac{T_P \mod M_P}{T_P}$ can be dropped since $\frac{T_P \mod M_P}{T_P} < \frac{M_P}{T_P} \ll 1$ when $(T_P \bmod M_P) < M_P$ and $T_P \gg M_P$.

Let us define:

$$R(P) = \frac{H_P}{M_P} \tag{11}$$

which represents the ratio of the average fault tolerance time per failure over MTTF and is called the *fault tolerance time factor*. In addition, we define:

$$E_P^R = \frac{1}{1 + R(P)} \tag{12}$$

which is called the *efficiency of the fault tolerance technique R*.

As a result, our reliability speedup formula S_P^R is refined to:

$$S_P^R = \frac{T_{\text{seq}}}{T_P(1+\frac{H_P}{M_P})} = \frac{S_P}{1+R(P)} = S_P \times E_P^R = P \times E_P \times E_P^R \tag{13}$$

In (13), as $E_P \times E_P^R$ means the efficiency of program G running on an R system, E_P^R represents the efficiency-preserving rate by R with respect to the efficiency of G running on an R-free system.

Our analogues of Amdahl's speedup and Gustafson's speedup when S_P in (13) is replaced with $S_{P,\mathit{Amdahl}}$ in (1) and $S_{P,\mathit{Gustafson}}$ in (2) are:

$$S_{P,\mathit{Amdahl}}^R = S_{P,\mathit{Amdahl}} \times E_P^R = \frac{P}{1+f(P-1)} E_P^R \tag{14}$$

$$S_{P,\mathit{Gustafson}}^R = S_{\mathit{Gustafson}} \times E_P^R = (f + P(1-f))E_P^R \tag{15}$$

When $E_P^R = 1$, our two reliability speedup models are simplified into the traditional Amdahl's and Gustafson's speedup models.

3.3 Quantifying the Reliability Wall

Definition 2 (Reliability Wall)**.** When a program G runs on an R system, the reliability wall is defined as the supremum[1] of the corresponding reliability speedup S_P^R, denoted $\sup\limits_P S_P^R$.

Theorem 1. *When a program G runs on an R system, the reliability wall exists if and only if $\lim\limits_{P\to\infty} S_P^R \neq \infty$.*

Proof: ⇒ By Definition 2, when the reliability wall exists, the supremum of the corresponding reliability speedup S_P^R exists. Suppose $\sup\limits_P S_P^R = Z < \infty$, where Z is a positive constant. Then $\lim\limits_{P\to\infty} S_P^R \leqslant \sup\limits_P S_P^R = Z < \infty$. Thus, $\lim\limits_{P\to\infty} S_P^R \neq \infty$ holds.
⇐ Since $\lim\limits_{P\to\infty} S_P^R \neq \infty$, $\lim\limits_{P\to\infty} S_P^R = Z < \infty$ holds. Thus, $\forall\ \varepsilon > 0$, $\exists\ N \in Z^+$ s.t. $|S_P^R - Z| < \varepsilon$ when $P > N$. Then $Z - \varepsilon < S_P^R < Z + \varepsilon$. So S_P^R has an upper bound. According to supremum principle[2] [31], the supremum of S_P^R exists. By Definition 2, the reliability wall exists for the program G. □

This theorem has the following important implication. If $\lim\limits_{P\to\infty} S_P \neq \infty$, then $\lim\limits_{P\to\infty} S_P^R \neq \infty$ since $S_P^R < S_P$. Thus, the reliability wall always exists.

For example, Amdahl's speedup metric falls into this special case. According to (14), we always have:

$$\lim_{P\to\infty} S_{P,\mathit{Amdahl}} = \lim_{P\to\infty} \frac{P}{1+f(P-1)} = \frac{1}{f}$$

Since $S_{P,\mathit{Amdahl}}^R < S_{P,\mathit{Amdahl}}$, $\sup\limits_P S_{\mathit{Amdahl}}^R \leqslant \frac{1}{f}$ holds. So for Amdahl's speedup, the reliability wall always exists, and is bounded at $\frac{1}{f}$ no matter on what R system the program runs. In the rest of the paper, we therefore focus on the cases when $\lim\limits_{P\to\infty} S_P = \infty$ holds.

3.4 Categorizing Fault-Tolerant Systems

Observing from our reliability speedup given in (13), we can see that $R(P)$ (introduced in (11)) is the key factor in determining the existence of the reliability wall, allowing R systems to be classified this way.

By convention, we write $f(x) \succcurlyeq g(x)$ if $\lim\limits_{x\to\infty} \frac{f(x)}{g(x)}$ is a positive constant or ∞ and $f(x) \succ g(x)$ if $\lim\limits_{x\to\infty} \frac{f(x)}{g(x)}$ is ∞ always. In addition, the operators $\preccurlyeq$ and $\prec$ are used in the standard manner.

Definition 3 (Constant and Incremental R Systems)**.** Suppose that a program G satisfying $\lim\limits_{P\to\infty} S_P = \infty$ runs on an R system. The R system (with G being executed on it) is said to be a *constant system* if $R(P) \preccurlyeq \Theta(1)$ and an *incremental system* if $R(P) \succ \Theta(1)$.

As is customary, the Θ notation[3] describes asymptotically both an upper bound and a lower bound.

For example, if $R(P) = kQ \preccurlyeq \Theta(1)$, where k and Q are positive constants (and remain so for the rest of this paper), then the R system is a constant system. By (15), we have:

$$\lim_{P\to\infty} S_{P,\mathit{Gustafson}}^R = \lim_{P\to\infty} \frac{f + P(1-f)}{1+kQ} = \infty$$

Then the reliability wall does not exist according to Theorem 1.

If $R(P) = kP \lg P \succ \Theta(1)$, then the R system is an incremental system. Similarly, by (15), we have:

$$\lim_{P\to\infty} S_{P,\mathit{Gustafson}}^R = \lim_{P\to\infty} \frac{f + P(1-f)}{1+kP\lg P} = 0$$

This time, however, the reliability wall exists by Theorem 1.

The following theorem presents a necessary and sufficient condition for the existence of the reliability wall on an R system.

Theorem 2. *Suppose a program G that satisfies $\lim\limits_{P\to\infty} S_P = \infty$ runs on an R system. The reliability wall exists if and only if $R(P) \succcurlyeq S_P$.*

Proof: ⇒ If the reliability wall exists, then $\lim\limits_{P\to\infty} \frac{S_P}{1+R(P)} = \lim\limits_{P\to\infty} S_P^R \neq \infty$ by Theorem 1. Thus, $R(P) \succcurlyeq S_P$ holds.
⇐ If $R(P) \succcurlyeq S_P$, then $\lim\limits_{P\to\infty} S_P^R = \lim\limits_{P\to\infty} \frac{S_P}{1+R(P)} < \infty$.

[1] For a subset Φ of an arbitrary number set Ψ, a supremum or least upper bound of Φ is an element η in Ψ such that 1) $x \leqslant \eta$ for all x in Φ, and 2) for any α in Ψ such that $x \leqslant \alpha$ for all x in Φ it holds that $\eta \leqslant \alpha$.

[2] Supremum principle. Suppose S_P is a non-empty number set. If S_P has an upper bound, then S_P has a supremum.

[3] Suppose $\mathfrak{R}(P)$ is the set consisting of all functions of P, $f(P) \in \mathfrak{R}(P)$ and $g(P) \in \mathfrak{R}(P)$. $f(P) = \Theta(g(P))$ if both $\lim\limits_{P\to\infty} \frac{g(P)}{f(P)}$ and $\lim\limits_{P\to\infty} \frac{f(P)}{g(P)}$ are positive constants.

Thus, $\lim_{P\to\infty} S_P^R \neq \infty$. By Theorem 1, the reliability wall exists for the program G. □

For an incremental R system, $R(P) \succ \Theta(1)$, by definition. Therefore, the reliability wall may exist according to Theorem 2. On the other hand, a program that satisfies $\lim_{P\to\infty} S_P = \infty$ remains scalable on a constant R system since $R(P) = \Theta(1) \prec S_P$, as summarized below.

Corollary 1. *When a program G that satisfies $\lim_{P\to\infty} S_P = \infty$ runs on a constant R system, the reliability wall does not exist.*

TABLE 2
Existence of reliability wall for R systems.

System	Categorization		Reliability wall
Incremental	$R(P) \succ \Theta(1)$	$R(P) \succcurlyeq S_P$	Yes
		$\Theta(1) \prec R(P) \prec S_P$	No
Constant	$R(P) \preccurlyeq \Theta(1)$		No

The conditions that govern the existence of the reliability wall for a program that satisfies $\lim_{P\to\infty} S_P = \infty$ running on constant and incremental systems are summarized in Table 2.

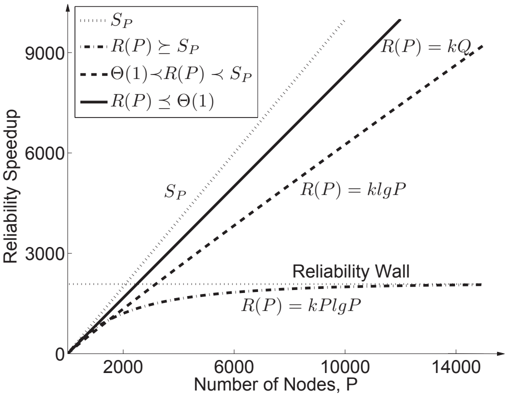


Fig. 2. Reliability speedup and wall for a constant system (in the solid line) and two incremental systems (in dashed lines).

By instantiating S_P with Gustafson's speedup in (13), Figure 2 illustrates three different systems when $R(P)$ is set to be as $kP\lg P$, $k\lg P$ and kQ, respectively, to represent the three conditions $R(P) \succcurlyeq S_P$, $\Theta(1) \prec R(P) \prec S_P$ and $R(P) \preccurlyeq \Theta(1)$ in Table 2.

4 Generalization

We generalize our reliability wall theory introduced in Section 3 by considering not only speedup but also costup [10]. This generalization is practically important for peta/exascale supercomputing systems as the fault-tolerant mechanisms employed can be costly. The generalized theory allows us to investigate the effects of both the time and cost induced by fault tolerance on scalability.

4.1 General Reliability Speedup

Definition 4 (General Reliability Speedup)**.** When program G runs on an R system, the *general reliability speedup* is defined as:

$$S_P^{GR} = \frac{S_P^R}{C^R} \tag{16}$$

where C^R is the costup of the R system.

Let us write c_P as the cost of the fault tolerance mechanism R. We define:

$$C(P) = \frac{c_P}{C_P} \tag{17}$$

which is simply the ratio of the cost of fault tolerance mechanism R over the cost of a P-node R-free system and is referred to as the *fault tolerance cost factor*. In additin, we define:

$$E_P^C = \frac{1}{1 + C(P)} \tag{18}$$

which is called the *efficiency of the fault tolerance cost*.

Suppose that C_P^R is the cost of a P-node R system. According to (3), we have:

$$\begin{aligned} C^R &= \frac{C_P^R}{C_1} = \frac{C_P + c_P}{C_1} = C + \frac{c_P}{C_1} = C\left(1 + \frac{c_P}{CC_1}\right) \\ &= C\left(1 + \frac{c_P}{C_P}\right) = C(1 + C(P)) = \frac{C}{E_P^C} \end{aligned} \tag{19}$$

Then, the general reliability speedup becomes:

$$S_P^{GR} = \frac{S_P}{C(1 + R(P))(1 + C(P))} = \frac{P \times E_P \times E_P^R \times E_P^C}{C} \tag{20}$$

If the time overhead incurred by fault tolerance can be omitted, i.e., if $H \approx 0$, then we can make the following simplification, by using (8):

$$E_P^R = \frac{1}{1 + R(P)} = \frac{1}{1 + \frac{H_P}{M_P}} = \frac{1}{1 + \frac{H}{FM_P}} \approx 1$$

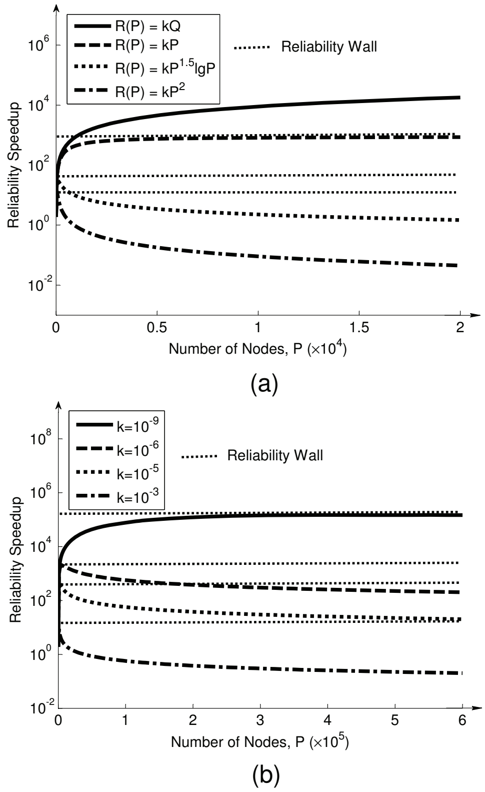
As a result, the general reliability speedup can be simplified to:

$$S_P^{GR} = \frac{P \times E_P \times E_P^R \times E_P^C}{C} \approx \frac{P \times E_P \times E_P^C}{C}$$

which considers essentially the effects of only the dollar costs incurred by fault tolerance on scalability.

For example, parity encoding techniques [28] fall into this category; they incur some capital cost due to extra encoding bits added in hardware but negligible time overhead.

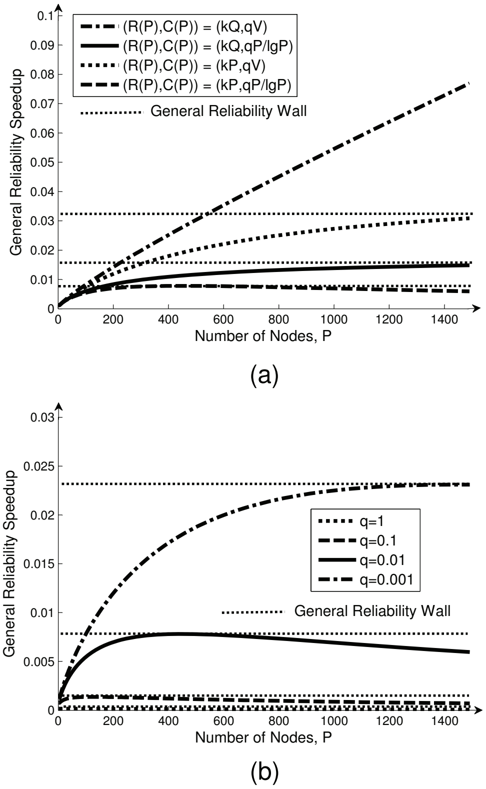
Therefore, our generalization allows us to study the effects of either the time overhead or dollar cost or both introduced by a fault-tolerance mechanism on scalability.



(a)

(b)

Fig. 3. Trends of the reliability speedup S_P^R and the existence of the reliability wall, where S_P is taken to be Gustafson's speedup. The Y axis is drawn in the logarithmic scale with base 10. (a) Four forms of $R(P)$ shown with fixed $k = 10^{-3}$ and $Q = 10^{-3}$. (b) Four forms of $R(P) = kP^{1.5} \lg P$ shown with varying k as shown.



(a)

(b)

Fig. 4. Trends of the general reliability speedup S_P^{GR} and the existence of the general reliability wall, where S_P is Gustafson's speedup and $C = 5000 \lg P$ is the costup used in (20). (a) Four form combinations of $(R(P), C(P))$ shown with fixed $k = 10^{-3}$, $V = 10$, $q = 0.01$ and $Q = 10^{-3}$. (b) Four forms of $(R(P), C(P)) = (kP, \frac{qP}{\lg P})$ with fixed $k = 10^{-3}$ and varying q as shown.

4.2 General Reliability Wall

Definition 5 (General Reliability Wall). *When a program G runs on an R system, the general reliability wall is defined as the supremum of the corresponding general reliability speedup S_P^{GR}, denoted $\sup_P S_P^{GR}$.*

Theorem 3. *When a program G runs on an R system, the general reliability wall exists if and only if $\lim_{P\to\infty} S_P^{GR} \neq \infty$.*

Proof: Proceeds similarly as in the proof of Theorem 1. □

As before, this theorem has the following immediate implication. If $\lim_{P\to\infty} S_P \neq \infty$, then $\lim_{P\to\infty} S_P^{GR} \neq \infty$ since $S_P^{GR} < S_P$. Thus, the general reliability wall always exists. For example, Amdahl's speedup metric falls into this special case.

The general reliability wall is affected by not only the time overhead but also the dollar cost induced by fault tolerance. Therefore, the general reliability wall exists if the reliability wall does.

4.3 Understanding the General Reliability Speedup and Wall

According to (13) and (20), $R(P)$ and $C(P)$, which characterize the time overhead and dollar cost incurred by the fault tolerance technique R, respectively, are the key scalability-limiting factors of a program that satisfies $\lim_{P\to\infty} S_P = \infty$ running on an R system. Their effects on the (general) reliability speedup and the existence of (general) reliability wall are plotted in Figures 3 and 4 for a number of representative function forms.

In order to determine realistically the two parameters, k and Q), used in $R(P)$, we have assumed an abstract P-node system, where the MTTF of a single node in the R-free system is 512×3600 secs or 21 days [32], calculated according to the failure data collected from 22 high-performance computing systems (mostly NUMA or SMP clusters) between 1996 and 2005 at the Los Alamos National Lab [33]. The two parameters, q and V, used in $C(P)$, are set approximately according to the dollar cost of a single node in the current market.

5 Case Studies

In our case studies, we use existing representative supercomputing systems with representative fault-tolerance techniques deployed to verify, analyze, and extrapolate the existence of the (general) reliability wall for forthcoming exascale systems. (A simulator for an exascale system, which does not exist yet, may not be helpful unless it can keep track of the execution state of the entire system.) Our case analysis also provides insights on how to mitigate reliability-wall effects in system design and through applying optimizations for large-scale parallel computing.

MPPs and clusters are the two most popular architectures in the TOP500 supercomputing list and checkpointing is a fault-tolerance mechanism widely adopted in large-scale parallel systems. In our case studies, we have selected Intrepid [34], an MPP system, and ASCI White [35], a cluster system, with their architectural details discussed later. We have selected these two systems also because their performance data, including MTTF, are publicly available. We consider two checkpointing techniques, full-memory and incremental-memory [23], and two I/O architectures, centralized and distributed to evaluate their impact on scalability.

We write $Interval$ to represent the checkpointing interval used in a system and m to represent the average number of checkpoints between failures. The two parameters are related to each other as follows:

$$m = \left\lfloor \frac{M_P}{Interval} \right\rfloor \tag{21}$$

In Section 5.1, we focus on reliability speedup and wall by considering four system configurations, depending on whether checkpointing is full-memory or incremental-memory and whether I/O is centralized or distributed. Both of these affect the existence of the reliability wall. For each configuration, we first develop its reliability speedup model and then analyze the existence of the reliability wall. In Section 5.2, we move to the general reliability speedup and wall by considering one configuration with full-memory checkpointing and a centralized I/O architecture. In Section 5.3, we further analyze the the existence of the (general) reliability wall for the above configuration when checkpoint interval optimization is adopted. In Section 5.4, we summarize how to mitigate the reliability wall effects through reducing scalability-limiting overhead induced by fault tolerance.

In this section, the traditional speedup S_P is always instantiated with Gustafson's speedup, $S_{P,\mathit{Gustafson}}$.

In our case analysis conducted in Sections 5.1 and 5.2, we use only a fixed value for m to analyze the reliability walls for Intrepid and ASCI White. In Section 5.3, we show that our results are valid when different values of m are used.

5.1 Reliability Speedup and Wall

In Section 5.1.1, we build reliability speedup models for both full-memory and incremental-memory checkpointing techniques. We show that the existence of the reliability wall depends critically on two key parameters, one is related to checkpoint size and the other to the bandwidth of the underlying I/O architecture. In Section 5.1.2, we conduct our analysis using Intrepid with a centralized I/O architecture by considering full-memory and incremental-memory checkpointing. In Section 5.1.3, we repeat this analysis for ASCI White with a distributed I/O architecture.

5.1.1 Formulating Reliability Speedup for Checkpointing

If the fault-tolerance technique R used is checkpointing, then the average time incurred per failure due to fault tolerance, H_P given in (13), consists of mainly two parts. One part represents the average time spent on saving checkpoints, enforcing coordination among processors and orchestrating the communication among processes between failures. The other part represents the average time spent on roll-back and recovery when a failure occurs.

The bandwidth of the underlying I/O architecture is the bottleneck for checkpointing, which mainly affects the time on saving checkpoints between failures and the time on recovering a checkpoint per failure. To simplify our analysis, we assume below that H_P only includes these two components. Thus, we obtain:

$$H_P = \left\lfloor \frac{M_P}{Interval} \right\rfloor \frac{D^s_{size}}{W} + \frac{D^r_{size}}{W} = m\frac{D^s_{size}}{W} + \frac{D^r_{size}}{W} \tag{22}$$

where D^s_{size} is the average size of a saved checkpoint each time and D^r_{size} is the average size of a checkpoint per failure to be recovered. For some checkpointing techniques, such as incremental-memory checkpointing, D^s_{size} is not necessarily equal to D^r_{size}, as will be discussed below. W is the bandwidth of the underlying I/O architecture in Gbits/sec. The other two parameters, $Interval$ and m, are introduced in (21).

In 2006, Schroeder [36] tested 22 high performance computers in LANL and pointed out that failure rates are roughly proportional to the number of processors P in a system [36], implying that the time to failure is inversely proportional to P. Let $M > 0$ be the MTTF of a single node in the R-free system. Thus, we assume that M and M_P are related by:

$$M = \frac{M_P}{P} \tag{23}$$

According to (13), we have:

$$S^R_P = \frac{S_P}{1 + \frac{H_P}{M_P}} = \frac{S_P}{1 + \frac{(mD^s_{size} + D^r_{size})P}{WM}} \tag{24}$$

where $R(P) = \frac{(mD^s_{size} + D^r_{size})P}{WM}$.

In full-memory checkpointing, the entire memory context for a process at a checkpoint is saved. Thus, the checkpoint saved is of the same size every time, i.e. the size of the total memory. So is the size of a checkpoint to be recovered. Thus, $D^s_{size} = D^r_{size}$. According to (24), the reliability speedup for full-memory checkpointing is given by:

$$S_P^{R\text{–}full} = \frac{S_P}{1+\frac{(mD^s_{size}+D^r_{size})P}{WM}} = \frac{S_P}{1+\frac{(m+1)D^s_{size}P}{WM}} \qquad (25)$$

Unlike full-memory checkpointing, incremental-memory checkpointing reduces the amount of checkpoint contexts saved. Typically, the size of the first saved checkpoint file and the size of a checkpoint to be recovered per failure are equal to that of the entire memory, but the size of a subsequently saved checkpoint file may be smaller. Thus, the average size of a saved checkpoint between failures, D^s_{size}, is not larger than D^r_{size}, i.e., $D^s_{size} \leqslant D^r_{size}$. Thus, the reliability speedup for incremental-memory checkpointing is:

$$S_P^{R\text{–}inc} = \frac{S_P}{1+\frac{(mD^s_{size}+D^r_{size})P}{WM}} \leqslant \frac{S_P}{1+\frac{(m+1)D^s_{size}P}{WM}} \qquad (26)$$

In this section, we explain the ”if” condition governing the existence of the reliability wall in Theorem 1. Thus, the instance of the ”if” part of Theorem 1 for full-memory checkpointing or incremental-memory checkpointing is given as follows.

Theorem 4. *Suppose that a program G that satisfies $\lim_{P\to\infty} S_P = \infty$ runs on an R system using full-memory checkpointing with its reliability speedup given in (25) or incremental-memory checkpointing with its reliability speedup given in (26), where S_P is instantiated with $S_{P,\text{Gustafson}}$. If $\frac{D^s_{size}}{W} \succcurlyeq \Theta(1)$, then the reliability wall exists.*

Proof: In (25), m and M are positive constants independent of P and $S_{P,\text{Gustafson}} = \Theta(P)$. If $\frac{D^s_{size}}{W} \succcurlyeq \Theta(1)$ then $P \preccurlyeq \frac{(m+1)D^s_{size}P}{MW}$, which implies that $\frac{P}{1+\frac{(m+1)D^s_{size}P}{MW}}$ has a supremum. Note that

$$\frac{S_{P,\text{Gustafson}}}{1+\frac{(m+1)D^s_{size}P}{MW}} = \Theta\left(\frac{P}{1+\frac{(m+1)D^s_{size}P}{MW}}\right)$$

So S_P^{R-full} with S_P being replaced by $S_{P,\text{Gustafson}}$ has a supremum. Given (25) and (26), the reliability wall exists in both cases. □

Let us examine the two key parameters [4] in Theorems 4 that determine the existence of the reliability wall, and consequently, the scalability of a program G that satisfies $\lim_{P\to\infty} S_P = \infty$ on an R system.

D^s_{size}. For full-memory checkpointing, D^s_{size} is the size of the total memory, which increases linearly with the number of system nodes. For incremental-memory checkpointing, D^s_{size} is related to the characteristics and scalability of program G. In general, the amount of data required for computing the execution state of program G increases with P. So does D^s_{size}. However, the dirty pages of a non-first checkpoint does not necessarily increase with P. Thus, $\Theta(1) \preccurlyeq D^s_{size} \preccurlyeq P$ holds.

W. The I/O bandwidth W is determined by the underlying I/O architecture. The storage systems used in some modern large-scale MPP supercomputing systems are decoupled from the compute nodes [37], as in the IBM Blue Gene series, giving rise to a centralized I/O architecture. However, this design puts the storage system of a supercomputer on a separate network from the compute nodes, limiting the available I/O bandwidth. As a result, today’s limited I/O bandwidth is choking the capabilities of modern supercomputers, particularly making fault-tolerance techniques, such as checkpointing, prohibitively expensive. Thus, a scalable I/O system design, referred to as the distributed I/O architecture, may overcome this limitation. For example, the clusters such as ASCI White with local disks have certain advantages when adopting checkpointing techniques.

Below we take two systems, Intrepid and ASCI White, as examples to investigate the existence of reliability wall for two different checkpointing techniques and two different I/O architectures.

5.1.2 Intrepid

The BlueGene/P Intrepid system at Argonne National Laboratory is one of IBM’s massively parallel supercomputers consisting of 40 racks of $40,960$ quad-core computer nodes (totalling $163,840$ processors) and 80 TB of memory. Each compute node consists of four PowerPC 450 processors, and has 2 GB memory. The peak performance is 557 TeraFlops. A 10-Gigabit Ehternet network, consisting of Myricom switching gear connected in a non-blocking configuration, provides connectivity between the I/O nodes, file servers, and several other storage resources. The centralized I/O architecture of Intrepid is shown in Figure 5, where the compute node / I/O node ratio can be 16, 32, 64 or 128, resulting in the number of I/O controllers as n = 2560, 1280, 640 or 320. (We use Intrepid rather than BlueGene/L since Blue Gene/P is newer.)

The peak unidirectional bandwidth of each 10-Gigabit Ethernet port of an I/O node is 6.8 Gbits/sec by the internal collective network that feeds it. P stands for the number of cores as one process is assumed to run on one core. The MTTF of Intrepid (40 racks) is $M_P = 12.6$ days [38]. By (23), the MTTF of a single core is $M = M_P \times P = 1.8 \times 10^{11}$ secs. Generally, large-scale parallel computing systems take a checkpoint interval according to the system’s MTTF M_P, and current systems have a checkpoint interval in the order of several hours. Thus, our analysis has

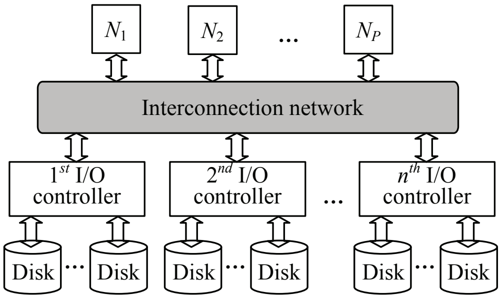


Fig. 5. The centralized I/O architecture with n I/O controllers.

assumed $m = 100$, implying that $Interval = 3$ hours for Intrepid (with $P = 163,840$ processors).

Since $\Theta(1) \preccurlyeq D^s_{size} \preccurlyeq P$, W is positive constants. Then $\Theta(1) \preccurlyeq \frac{D^s_{size}}{W} \preccurlyeq P$. According to Theorem 4, the reliability wall exists for an program G that satisfies $\lim\limits_{P\to\infty} S_P = \infty$ running on Intrepid when either full-memory or incremental-memory checkpointing technique is used. Below we calculate the smallest size of the system when the reliability wall is hit.

5.1.2.1 Centralized I/O + Full-Memory Checkpointing: $D^s_{size} = 2 \times \frac{P}{4} \times 8 = 4P$ Gbits since each quad-core node has 2GB memory an P is the total number of cores. Observing from (25), we have $R(P) = kP^2 = \Theta(P^2)$, where

$$k = \frac{4(m+1)}{WM}$$

Substituting $S_{P,Gustafson}$ for S_P into (25), we obtain:

$$S_P^{R-full} = \frac{f + (1-f)P}{1 + R(P)} = \frac{f + (1-f)P}{1 + kP^2} \qquad (27)$$

Suppose S_P^{R-full} reaches the reliability wall at P_0, called the *optimal system size*, then we can find P_0 as follows:

$$\left.\frac{dS_P^{R-full}}{dP}\right|_{P=P_0} = \frac{(1-f)(1+kP_0^2) - 2kP_0(f+P_0(1-f))}{(1+kP_0^2)^2} = 0$$

Because f is small for a large-scale parallel program and can thus be omitted, we obtain $P_0 \approx \sqrt{\frac{1}{k}}$ and $\left.S_P^{R-full}\right|_{P=P_0} \approx \frac{P_0}{2}$. By Definition 2, the reliability wall is found to be:

$$\sup_P S_P^{R-full} = \sqrt{\frac{MW}{16(m+1)}} \equiv \sqrt{\frac{MWP}{4(m+1)D^s_{size}}} \qquad (28)$$

5.1.2.2 Centralized I/O + Incremental-Memory Checkpointing: As discussed earlier, the total size of all saved checkpoints is at least equal to that of the entire memory, since that is the size saved at the first checkpoint. Thus, D^s_{size} is at least the ratio of the total memory size to the number of checkpoints when program G runs on Intrepid. Suppose that program G runs on Intrepid for a month (of 30 days). Then $D^s_{size} = \frac{4P}{\frac{30\times 24}{Interval}}$ Gbits. Each checkpoint, except the first one only, saves only the modified pages, while the size to be recovered at a checkpoint, D^r_{size}, is the same as in the case of full-memory checkpointing, i.e., $D^r_{size} = 4P$ Gbits.

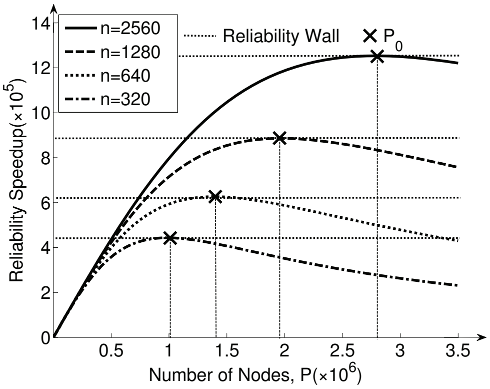


Fig. 6. Trend of $S^R_{P,Gustafson}$ for Intrepid with full-memory checkpointing ($m = 100$).

Based on (26), we also have $R(P) = kP^2 = \Theta(P^2)$, where

$$k = \frac{4(m\frac{Interval}{30\times 24} + 1)}{WM}$$

Substituting $S_{P,Gustafson}$ for S_P into (26), we obtain S_P^{R-inc} of the same form as in (27) except that k represents a different coefficient.

Proceeding exactly as in full-memory checkpointing, we obtain $P_0 \approx \sqrt{\frac{1}{k}}$ and $\left.S_P^{R-inc}\right|_{P=P_0} \approx \frac{P_0}{2}$. By Definition 2, the reliability wall for incremental-memory checkpointing is:

$$\sup_P S_P^{R-inc} = \sqrt{\frac{MW}{16(\frac{m\times Interval}{30\times 24}+1)}} \equiv \sqrt{\frac{MWP}{4(mD^s_{size}+D^r_{size})}} \qquad (29)$$

5.1.2.3 Reliability Wall: For a few combinations of different values taken by n (number of I/O controllers), Figure 6 and 7 display the trend of $S^R_{P,Gustafson}$ for Intrepid with full-memory and incremental-memory checkpointing techniques, respectively.

The optimal system size P_0 lies between 10^6 and 10^7 before the system hits the reliability wall, indicating that the reliability wall will exist at the Peta/exascale levels. By comparing full-memory and incremental-memory checkpointing techniques, we find that we can push the reliability wall forward by using less expensive fault-tolerance techniques, as further illustrated by an example given in Figure 8. In each case, the reliability speedup increases initially and starts to drop as soon as it hits the reliability wall, i.e., the maximum of the speedup achievable. Based on our earlier analysis results given in (28) and (29), the system designer can also increase M or W, i.e., adopt more reliable system components or improve the bandwidth, to push the reliability wall forward.

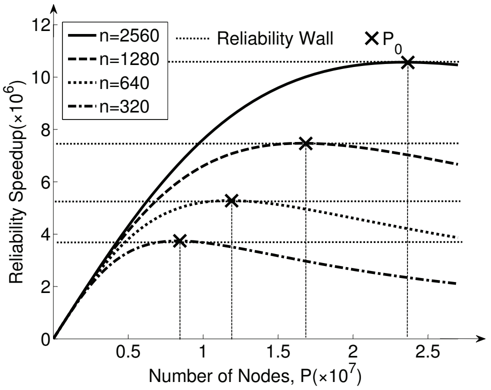


Fig. 7. Trend of $S^R_{P,Gustafson}$ for Intrepid with incremental-memory checkpointing ($m = 100$).

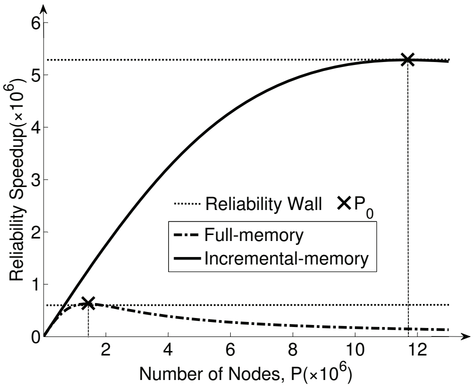


Fig. 8. Comparison of $S^R_{P,Gustafson}$ for Intrepid between full-memory checkpointing and incremental-memory checkpointing when $n = 640$ ($m = 100$).

5.1.3 ASCI White

We now turn to ASCI White with local disks to study the existence of the reliability wall for systems with scalable I/O architectures.

ASCI White is the third step in the DOE's five stage ASCI plan, with a peak performance of 12.3 TeraFlops. It is a computer cluster built based on IBM's commercial RS/6000 SP Power3 375MHZ computer. Each compute node contains 16 Power3-II CPUs built with IBM's latest semi-conductor technology (silicon-on-insulator and copper interconnects), with 16 GB memory. ASCI White uses a faster switching network (IBM's SP switch) to exchange information across the computer nodes. The number of compute nodes is 512 (totalling 8192 CPUs). The local disk per compute node is 70.9 GB, with the I/O bandwidth to local disk being 40 MBs/sec. As discussed below, ASCI White with scalable I/O bandwidth is better suited to checkpointing techniques than Intrepid with a centralized I/O architecture.

The MTTF of ASCI White is $M_P = 40$ hours (in 2003 with 8192 CPUs) [39]. P denotes the number of (single-core) CPUs with one process running per CPU. By (23), the MTTF of a single CPU is $M = M_P \times P = 1.2 \times 10^9$ secs. As in the case of Intrepid, we assume $m = 100$. This implies that $Interval = 0.4$ hours for ASCI White (with $P = 8192$ processors).

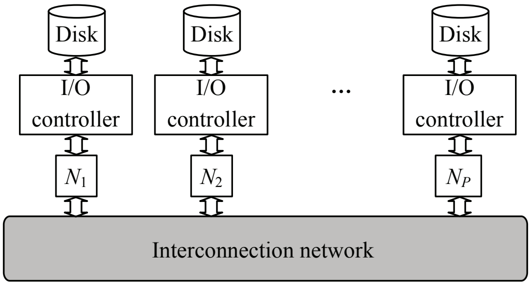


Fig. 9. The distributed I/O architecture.

The distributed I/O controller in ASCI White is shown in Figure 9. When the system size increases, the I/O bandwidth of the system increases too, i.e., $W = 0.04 \times P \times 8 = \Theta(P)$ Gbits/sec. According to Theorem 4, the reliability wall exists when $\frac{D^s_{size}}{W} \not\succcurlyeq \Theta(1)$. Thus, for ASCI White, the criterion for the existence of the reliability wall is $D^s_{size} \not\succcurlyeq \Theta(P)$. Note that $\Theta(1) \preccurlyeq D^s_{size} \preccurlyeq \Theta(P)$ as discussed in Section 5.1.1. According to Theorem 4, when a program G that satisfies $\lim_{P\to\infty} S_P = \infty$ runs on the ASCI White system with either full-memory or incremental-memory checkpointing, the reliability wall always exists because $D^s_{size}/W = \Theta(1)$ holds.

5.1.3.1 Distributed I/O + Full-Memory Checkpointing: $D^s_{size} = 16 \times \frac{P}{16} \times 8$ Gbits $= 8P$ Gbits as each compute node has 16 CPUs sharing 16 GB memory and P is the number of CPUs in ASCI White. In (25), we find that $R(P) = kP = \Theta(P)$, where

$$k = \frac{m+1}{0.04M}$$

Substituting $S_{P,Gustafson}$ for S_P into (25), we obtain:

$$S_P^{R-full} = \frac{f + (1-f)P}{1 + R(P)} = \frac{f + (1-f)P}{1 + kP} \qquad (30)$$

Since $\frac{dS_P^{R-full}}{dP} > 0$ for all P, the reliability wall is calculated as:

$$\sup_P S_P^{R-full} = \lim_{P\to\infty} S_P^{R-full} = \lim_{P\to\infty} \frac{f+(1-f)P}{1+kP} = \frac{1-f}{k}$$

Again, f is small and thus omitted. and we have:

$$\sup_P S_P^{R-full} \approx \frac{1}{k} = \frac{0.04M}{m+1} \equiv \frac{MW}{(m+1)D^s_{size}}$$

5.1.3.2 Distributed I/O + Incremental-Memory Checkpointing: As in the case for Intrepid, we consider running program G on the ASCI White for a month (of 30 days). By conducting a similar analysis, we have $D^s_{size} = \frac{8P}{\frac{30\times 24}{Interval}}$ Gbits and $D^r_{size} = 8P$ Gbits. In this case, we continue to have $R(P) = kP = \Theta(P)$, where

$$k = \frac{m\frac{Interval}{30\times 24} + 1}{0.04M}$$

Substituting $S_{P,Gustafson}$ for S_P into (26), we obtain S_P^{R-inc} of the same form as (30) except that k represents a different coefficient.

Proceeding exactly as in full-memory checkpointing, we obtain the reliability wall as follows:

$$\sup_P S_P^{R-inc} \approx \frac{1}{k} = \frac{0.04M}{m\frac{Interval}{30\times 24}+1} \equiv \frac{MW}{mD^s_{size}+D^r_{size}}$$

5.1.3.3 Reliability Wall: The same observations made earlier for Intrepid also apply here and are thus omitted.

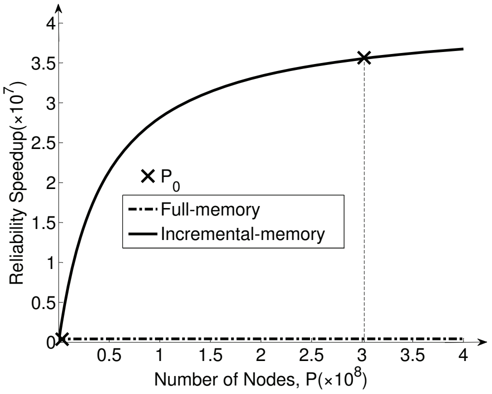


Fig. 10. Comparison of $S^R_{P,Gustafson}$ for ASCI White between full-memory checkpointing and incremental-memory checkpointing ($m = 100$).

Consider an example illustrated in Figure 10 for the two checkpointing techniques. In each case, the speedup function increases monotonically and approaches its supremum, i.e. reliability wall eventually.

In each case, the optimal system size can be determined by some simple heuristics. For example, let $threshold$ represent the growth rate (i.e. the first order derivative) of a given reliability speedup function, S_P^R. If the growth rate of S_P^R is slower than $threshold$, then any attempt for boosting performance by increasing the system size will be futile. Thus, the optimal system size P_0 can be calculated by simply solving $\frac{dS_P^R}{dP}\Big|_{P=P_0} = threshold$.

If $threshold = 0.01$, then P_0 is found to be 4.28×10^6 for full-memory checkpointing and 3.04×10^8 for incremental-memory checkpointing as shown in Figure 10. This implies that the reliability wall exists at the peta/exascale levels when the size of ASCI White increases and moves over the range between 10^6 and 10^8.

5.2 General Reliability Speedup and Wall

We consider Intrepid with a centralized I/O architecture using full-memory checkpointing for fault tolerance, due to the price information publicly available. We derive its general reliability speedup and discuss the existence of the general reliability wall.

For full-memory checkpointing, the cost of disks that are used to save checkpoint files is the cost incurred by checkpointing. c_P is the cost of RAIDs with a size being equal to the size of the total memory available in Intrepid.

TABLE 3
List prices for Intrepid in July, 2010 (US$)

Rack	100 TB RAIDs	Interconnect per Node	C_1
1.5 million	$500,000$	290	1170

According to Table 3, the cost of 100 TB RAIDs is $500,000$ dollars. So the cost of 1 GB is 5 dollars. Given that $P = 40,960 \times 4 = 163,840$, the (hardware) cost of full-memory checkpointing is $c_P = 40,960 \times 2 \times 5 = 2.5P$ dollars. Suppose program G consumes 6 GB of the RAIDs when it runs on Intrepid without full-memory checkpointing, the (hardware) cost is $6\times 5 = 30$ dollars. Thus, the cost of a P-node Intrepid system with full-memory checkpointing is $C_P = 1.5 \times 10^6 \times 40 + 40,960 \times 290 + 30 = 71.9$ million dollars.

Suppose the costup (without checkpointing) is $C = \frac{C_P}{C_1} = \Theta(\lg P)$, i.e., $C = \ell \lg P$, where ℓ is a positive constant. Then, we find that $\ell = \frac{C_P}{C_1 \lg P} \approx 1.2 \times 10^4$ when $P = 163,840$.

According to (20) and (25), when $n = 640$, the general reliability speedup for full-memory checkpointing is:

$$\begin{aligned} S_P^{GR-full} &= \frac{S_P^{R-full}}{C^R} = \frac{\frac{S_P}{1+\frac{(m+1)D^s_{size}P}{WM}}}{C(1+\frac{c_P}{C_P})} \\ &= \frac{S_P}{C}\frac{1}{1+\frac{4(100+1)P^2}{WM}}\frac{1}{1+\frac{2.5P}{C_1 llgP}} \\ &= \frac{f+(1-f)P}{1.2\times 10^4 lgP(1+5.2\times 10^{-13}P^2)(1+\frac{1.8\times 10^{-7}P}{lgP})}. \end{aligned} \tag{31}$$

It is easy to see that $\lim_{P\to\infty} S_P^{GR-full} = 0$. Thus, the general reliability wall exists in this situation according to Theorem 3.

Proceeding similarly as we did when analyzing the reliability wall for Intrepid with full-memory checkpointing in Section 5.1.2, the optimal system size P_0 for the general reliability wall can be calculated by solving $\frac{dS_P^{GR-full}}{dP}\Big|_{P=P_0} = 0$. We find that $P_0 = 1.26 \times 10^6$ and the general reliability wall is $\sup_P S_P^{GR-full} = 8.21$ as further illustrated in Figure 11. Due to the high costup being taken into account, the general reliability wall and its optimal system size are both much smaller than if the costup is ignored.

We can observe easily that the general reliability wall exists if the reliability wall does, but the converse is not always true.

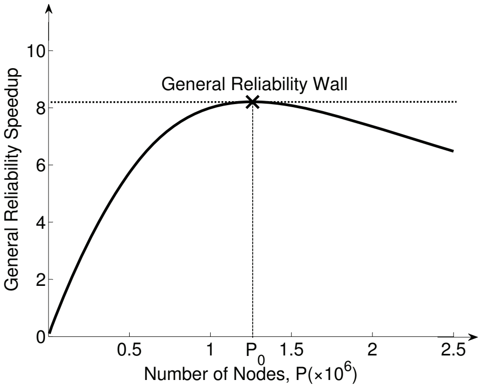


Fig. 11. Trend of $S^{GR-full}_{P,Gustafson}$ for Intrepid when $n = 640$ ($m = 100$).

5.3 On the Optimum Checkpoint Interval

In Sections 5.1 and 5.2, we consider $m = \lfloor \frac{M_P}{Interval} \rfloor$, i.e., the average number of checkpoints between failures as a constant. However, m can be optimized for a particular configuration by optimizing $Interval$ to obtain the *optimum checkpoint interval* [40] as follows:

$$Interval = \begin{cases} \sqrt{\frac{2D^s_{size}M_P}{W}} - \frac{D^s_{size}}{W}, & \frac{D^s_{size}}{W} < \frac{M_P}{2} \\ M_P, & \frac{D^s_{size}}{W} \geqslant \frac{M_P}{2} \end{cases} \quad (32)$$

There is a broad consensus in the fault-tolerance community that when the system size is scaled up continuously, the time spending on saving a global checkpoint will eventually reach and even exceed the MTTF of a system [9]. This implies that $\frac{D^s_{size}}{W} \geqslant \frac{M_P}{2}$ will hold eventually when P is in $[P', \infty)$ and $\frac{D^s_{size}}{W} < \frac{M_P}{2}$ will hold when P is in $[1, P'-1]$, where $\frac{D^s_{size}}{W}$ is roughly the time spent on saving a global checkpoint, and P' is a certain system size when $\frac{D^s_{size}}{W} = \frac{M_P}{2}$.

When P is in $[1, P'-1]$, i.e., $\frac{D^s_{size}}{W} < \frac{M_P}{2}$ holds, both S^{R-full}_P and S^{R-inc}_P are bounded and have the supremums. When P is in $[P', \infty)$, i.e., $\frac{D^s_{size}}{W} \geqslant \frac{M_P}{2}$ holds, we have $m = \lfloor \frac{M_P}{Interval} \rfloor = 1$, which falls into the cases discussed earlier where m is a constant. In both cases, Theorem 4 is still applicable when $Interval$ is instantiated with the optimum checkpoint interval.

Thus, the existence of the (general) reliability wall does not change for the four system configurations discussed earlier in Sections 5.1 and 5.2.

5.4 Mitigating Reliability-Wall Effects

As demonstrated in our case studies, the reliability wall may exist in the future exascale supercomputing. Our reliability-wall theory provides us insights on reducing fault-tolerance overhead for improved scalability in a number of principal directions.

- Based on our (general) reliability-wall theory, the relationship between reliability and scalability can be revealed. A (general) reliability speedup function can be developed and the existence of the (general) reliability wall can be analytically predicted.
- By performing a reliability-wall analysis, the key scalability-limiting parameters, such as D^s_{size}, D^r_{size}, $Interval$, M and W, can be identified. The system designer can push the reliability-wall forward by deploying more reliable components and increasing the I/O bandwidth to optimize system-wide parameters M and W. For example, scalable diskless checkpointing [24] is designed for optimizing W. Simultaneously, application programmers or compiler researchers can develop optimizations to reduce fault-tolerance overhead by choosing appropriate values for the other parameters, such as D^s_{size}, D^r_{size} and $Interval$, which are the main design decisions to be made in application-level checkpointing [17], [18].
- As illustrated in Figures 8 and 10, the development of new fault-tolerance techniques and I/O architectures can be guided by our theory to focus on optimizing the key scalability-limiting factors.

6 Conclusion

This paper introduces for the first time the concept of "Reliability Wall" and proposes a (general) reliability-wall theory that allows the effects of reliability on scalability of parallel applications to be understood and predicted analytically for large-scale parallel computing systems, particularly those at the peta/exascale levels. The significance of this work is demonstrated with case studies using representative real-world supercomputing systems with commonly used checkpointing techniques for fault tolerance. Our work enables us to push the reliability-wall forward by mitigating reliability-wall effects in system design (e.g., by developing cost-effective fault-tolerance techniques and I/O architectures) and through applying compiler-assisted optimizations at the application-level.

Acknowledgments

The authors thank the reviewers for their helpful comments and suggestions, which greatly improved the final version of the paper. This work is supported by the National Natural Science Foundation of China (NSFC) No.60921062 and 61003087.

References

[1] D. B. Kothe, "Science prospects and benefits with exascale computing," Oak Ridge National Laboratory, Tech. Rep. ORNL/TM-2007/232, 2007.

[2] H. Simon, T. Zacharia, and R. Stevens, "Modeling and Simulation at the Exascale for Energy and the Environment," Website, http://www.sc.doe.gov/ascr/ProgramDocuments/ProgDocs.html.

[3] J. Stearley, "Defining and measuring supercomputer Reliability, Availability, and Serviceability (RAS)," in *Proceedings of the Linux Clusters Institute Conference*, 2005.

[4] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward exascale resilience," *Int. J. High Perform. Comput. Appl.*, vol. 23, pp. 374–388, November 2009.

[5] N. DeBardeleben, J. Laros, J. Daly, S. Scott, C. Engelmann, and B. Harrod, "High-End Computing Resilience: Analysis of Issues Facing the HEC Community and Path-Forward for Research and Development," White paper, 2009, http://www.csm.ornl.gov/~engelman/publications/debardeleben09high-end.pdf.

[6] E. N. Elnozahy, R. Bianchini, T. El-Ghazawi, A. Fox, F. Godfrey, A. Hoisie, K. McKinley, R. Melhem, J. Plank, P. Ranganathan, and J. Simons, "System resilience at extreme scale," 2008.

[7] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A Survey of Rollback-recovery Protocols in Message-passing Systems," *ACM Comput. Surv.*, vol. 34, no. 3, pp. 375–408, 2002.

[8] S. Chakravorty, "A Fault Tolerance Protocol for Fast Recovery," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2008, adviser-Kale, Laxmikant V.

[9] D. Scott, "HW & SW Challenges and Trends to Reach Exascale," in *HPCChina '09: High Performance Computing of China*, 2009.

[10] D. A. Wood and M. D. Hill, "Cost-Effective Parallel Computing," *Computer*, vol. 28, no. 2, pp. 69–72, 1995.

[11] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS '67 (Spring): Proceedings of the 1967 spring joint computer conference*, 1967, pp. 483–485.

[12] J. L. Gustafson, "Reevaluating Amdahl's law," in *Multiprocessor performance measurement and evaluation*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1995, pp. 92–93.

[13] X. H. Sun and L. M. Ni, "Scalable Problems and Memory-bounded Speedup," *J. Parallel Distrib. Comput.*, vol. 19, no. 1, pp. 27–37, 1993.

[14] X. H. Sun and D. T. Rover, "Scalability of Parallel Algorithm-Machine Combinations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 6, pp. 599–613, 1994.

[15] D. B. Johnson, "Distributed system fault tolerance using message logging and checkpointing," Ph.D. dissertation, Rice University, 1990, chairman-Zwaenepoel, Willy.

[16] A. Bouteiller, T. Herault, G. Krawezik, P. Lemarinier, and F. Cappello, "MPICH-V: A Multiprotocol Fault Tolerant MPI," *International Journal of High Performance Computing and Applications*, vol. 20, no. 3, pp. 319–333, 2006.

[17] G. Bronevetsky, D. Marques, K. Pingali, and P. Stodghill, "Automated application-level checkpointing of MPI programs," in *PPoPP '03: Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2003, pp. 84–94.

[18] A. Beguelin, E. Seligman, and P. Stephan, "Application level fault tolerance in heterogeneous networks of workstations," *J. Parallel Distrib. Comput.*, vol. 43, no. 2, pp. 147–155, 1997.

[19] Z. Chen, G. E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra, "Fault tolerant high performance computing by a coding approach," in *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2005, pp. 213–223.

[20] M. Beck, J. S. Plank, and G. Kingsley, "Compiler-assisted checkpointing," Knoxville, TN, USA, Tech. Rep., 1994.

[21] J. S. Plank, M. Beck, and G. Kingsley, "Compiler-assisted memory exclusion for fast checkpointing," *IEEE TECHNICAL COMMITTEE ON OPERATING SYSTEMS AND APPLICATION ENVIRONMENTS*, vol. 7, pp. 62–67, 1995.

[22] J. Li and W. K. Fuchs, "CATCH - Compiler Assisted Techniques for Checkpointing ," in *FTCS-20: 20th International Symposium on Fault-Tolerant Computing*, 1990, pp. 74–81.

[23] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent checkpointing under unix," University of Tennessee, Knoxville, TN, USA, Tech. Rep., 1994.

[24] C. D. Lu, "Scalable diskless checkpointing for large parallel systems," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2005, adviser Reed, Daniel A.

[25] J. S. Plank, K. Li, and M. A. Puening, "Diskless Checkpointing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 10, pp. 972–986, 1998.

[26] D. A. Patterson, G. Gibson, and R. 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*. ACM, 1988, pp. 109–116.

[27] S. Lin and D. J. Costello, *Error Control Coding, Second Edition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004.

[28] T. V. Ramabadran and S. S. Gaitonde, "A tutorial on crc computations," *IEEE Micro*, vol. 8, no. 4, pp. 62–75, 1988.

[29] O. Wolfson, S. Jajodia, and Y. Huang, "An adaptive data replication algorithm," *ACM Trans. Database Syst.*, vol. 22, no. 2, pp. 255–314, 1997.

[30] L. Mancini and M. Koutny, "Formal specification of N-modular redundancy," in *CSC '86: Proceedings of the 1986 ACM fourteenth annual conference on Computer science*. ACM, 1986, pp. 199–204.

[31] W. Rudin, *Principles of Mathematical Analysis, Third Edition*. R. R. Donnelley & Sons, 1976.

[32] M. Wu, X.-H. Sun, and H. Jin, "Performance under failures of high-end computing," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, ser. SC '07. New York, NY, USA: ACM, 2007, pp. 48:1–48:11.

[33] Los Alamos National Laboratory, "Operational Data to Support and Enable Computer Science Research," Website, http://institute.lanl.gov/data/lanldata.shtml.

[34] R. Gupta, H. Naik, and P. Beckman, "Understanding Checkpointing Overheads on Massive-Scale Systems: Analysis on the IBM Blue Gene/P System," *The International Journal of High Performance Computing Applications*, June 2010.

[35] UK High-End Computing, "Overview of the Advanced Simulation and Computing Program," Website, http://www.ukhec.ac.uk/publications/reports/asci.pdf.

[36] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proceedings of the International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 249–258.

[37] A. Moody and G. Bronevetsky, "Scalable I/O Systems via Node-Local Storage: Approaching 1 TB/sec File I/O," Lawrence Livermore National Laboratory (LLNL), Tech. Rep. LLNL-TR-415791, 2008.

[38] T. Budnik, A. Peters, and G. Thain, "Blue Heron Project," Website, http://www.cs.wisc.edu/condor/PCW2008/condor_presentations/peters_blue_heron.ppt.

[39] J. Dongarra, G. Bosilca, Z. Chen, V. Eijkhout, G. E. Fagg, E. Fuentes, J. Langou, P. Luszczek, J. Pjesivac-Grbovic, K. Seymour, H. You, and S. S. Vadhiyar, "Self-adapting numerical software (SANS) effort," *IBM J. Res. Dev.*, vol. 50, no. 2/3, pp. 223–238, 2006.

[40] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Gener. Comput. Syst.*, vol. 22, pp. 303–312, February 2006.