Towards New Metrics for High-Performance Computing Resilience*

Saurabh Hukerikar, Rizwan A. Ashraf, Christian Engelmann
Computer Science and Mathematics Division
Oak Ridge National Laboratory
1 Bethel Valley Road
Oak Ridge, Tennessee 37831
{hukerikarsr,ashrafra,engelmannc}@ornl.gov

ABSTRACT

Ensuring the reliability of applications is becoming an increasingly important challenge as high-performance computing (HPC) systems experience an ever-growing number of faults, errors and failures. While the HPC community has made substantial progress in developing various resilience solutions, it continues to rely on platform-based metrics to quantify application resiliency improvements. The resilience of an HPC application is concerned with the reliability of the application outcome as well as the fault handling efficiency. To understand the scope of impact, effective coverage and performance efficiency of existing and emerging resilience solutions, there is a need for new metrics. In this paper, we develop new ways to quantify resilience that consider both the reliability and the performance characteristics of the solutions from the perspective of HPC applications. As HPC systems continue to evolve in terms of scale and complexity, it is expected that applications will experience various types of faults, errors and failures, which will require applications to apply multiple resilience solutions across the system stack. The proposed metrics are intended to be useful for understanding the combined impact of these solutions on an application's ability to produce correct results and to evaluate their overall impact on an application's performance in the presence of various modes of faults.

CCS CONCEPTS

• **General and reference** → **Reliability**; **Metrics**; • **Software and its engineering** → **Software reliability**; **Software fault tolerance**; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks*;

KEYWORDS

High Performance Computing; Resilience; Fault Tolerance; Metrics

ACM Reference format:
Saurabh Hukerikar, Rizwan A. Ashraf, Christian Engelmann. 2017. Towards New Metrics for High-Performance Computing Resilience. In *Proceedings of FTXS'17, Washington , DC, USA, June 26, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3086157.3086163

1 INTRODUCTION

The resilience of high-performance computing (HPC) systems to frequent fault events is a critical challenge as extreme-scale systems continue to increase component counts while the individual component reliability decreases (due to shrinking process technology and near-threshold voltage operation), and as software complexity continues to increase. Resilience is concerned with the correctness of an HPC application in lieu of, or even at the expense of, the reliability of the system [6]. Resilience solutions are designed to enable effective and cost efficient management of faults, errors and failures in systems. Therefore, the application correctness and execution efficiency are the essential aspects of a resilience solution. Yet the HPC community continues to rely on metrics that don't adequately provide a quantitative assessment of this perspective about HPC resilience.

In the design and engineering of fault tolerant systems, the term dependability refers to a property of a system that indicates whether the system is operating properly. The term is formally defined as *the quality of delivered service by a computing system such that reliance can justifiably be placed on the service* [8]. Since the notion of dependability is concerned with a system's ability to deliver service, the criteria used when deciding whether a system is dependable is in terms of the system's ability to avoid service failures. To quantify dependability, we use attributes such as availability, reliability, safety, integrity, maintainability, etc. For each of these attributes there are mathematical measures that provide a definitive perception about the system's dependability. The various dependability attributes are shown in Figure 1 [2]. It is important to stress the difference between an attribute and its measurement. A measurement quantifies an attribute of a system; the term metric is a standardized method to measure an attribute. For example, the reliability attribute is described in terms of continuous service accomplishment, or the time to failure from a reference point in time, and is measured in terms of the system's mean time to failure (MTTF) and mean time to repair (MTTR). The metrics for the availability attribute provide a measure of the service accomplishment with respect to the alternation between the two states of service accomplishment and

*This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

FTXS'17, June 26, 2017, *Washington , DC, USA*

978-1-4503-5001-3/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3086157.3086163

interruption (expressed as the ratio of MTTF to MTTF + MTTR). Both of these metrics are based on a service-based view of the computing platform. While the MTTF and MTTR are appropriate for measuring the reliability and availability of a computing platform, the use of these metrics to measure an HPC application's resilience incorrectly imposes a service-based view on the application's execution as well, and these metrics are therefore not suitable for quantifying the resilience of an HPC application. Even a metric such as the application's mean time to fatal error (AMTTFE) [5] is in reality a system-based metric, since it is based on the assumption that the MTTF of the hardware and system software components is equivalent to the time to failure of an application running on the system. Furthermore, these metrics do not take into account the fact that not every fault and error results in fatal failure, or the fact that an HPC application might be able to survive partial system failures and continue executing towards a correct outcome at a degraded performance level.

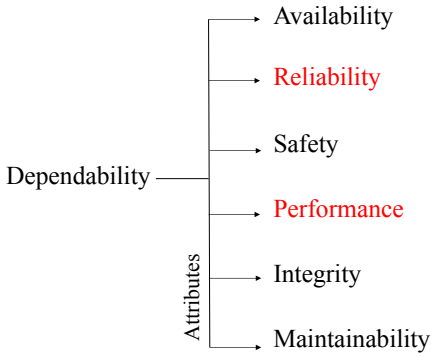

Figure 1: Dependability Attribute Tree

Resilience is also a dependability attribute that encompasses the properties of reliability and performance. However, the concept of resilience in HPC is based on an application-centric paradigm. Resilience solutions are designed to ensure the fidelity of the scientific output of simulation, modeling and analysis applications without the expectation of correct operation from the HPC hardware and software environment. The solutions seek to ensure that the presence of faults and failures minimally impact the computation and communication of the application. The performance efficiency of handling the fault events is also an important aspect of a resilience solution. Therefore, for evaluating and quantifying the impact of a resilience solution in these terms, the most commonly used platform reliability metrics, including the MTTF (and its variants such as mean time to interrupt (MTTI)) and MTTR, are not suitable.

As the scale and complexity of HPC systems continues to grow, it is vital to clearly articulate the resilience through metrics for the application's reliability and performance attributes. Our quest for resilience metrics is driven by the need to accurately measure the ability of an application running on an unreliable HPC system to complete with correct results in a performance efficient manner independent of whether the event is a fault, error or failure. The metrics are intended to complement platform-based metrics such as MTTF (and its variants). We develop outcome-based metrics called the *resilience factor* (RF) and *resilience factor yield* (RY) to measure the impact of fault events on the application's ability to produce correct results and the efficiency of detecting and recovering from errors and failures. The metrics enable HPC designers, programmers and users to contemplate about questions such as: *(i) how do the inclusion of specific hardware or software-based solutions improve an application's ability to deliver a correct outcome and its effect on the application's performance (independent of the improvement in platform dependability)? (ii) how does the combination of multiple resilience solutions implemented across multiple layers of the system stack impact application reliability and performance?* This paper examines both of these questions and explores how the proposed metrics quantify application resilience. We also demonstrate the utility of the designed metrics by applying them to a linear solver application, which is enhanced with a multilayered resilience solution to evaluate its reliability and performance characteristics in the presence of hard and soft errors.

2 MEASURING HPC RESILIENCE

While the term resilience generally refers to the ability of a computing system to withstand or recover from anomalous effects or behaviors, different domains of computing use it in subtly different ways. For example, the dependable computing community defines resilience as the persistence of service delivery that can justifiably be trusted, when facing *changes* [7]. The changes here may refer to unexpected threats, accidents, or failures. In computer networking, resilience is defined broadly as a combination of trustworthiness (dependability, security, performability) and tolerance (survivability, disruption tolerance, and traffic tolerance) [1]. In computer security, the concept of resilience is used in relation to the concepts of privacy, integrity and confidentiality [3].

In the context of HPC, resilience is concerned with the reliability and performance attributes of an HPC application. HPC resilience solutions are built on the assumption that the hardware and system software platform that the application runs on is unreliable, and are therefore focused on ensuring the reliability of an application's computation and the correctness of its outcome. Additionally, HPC resilience places as much emphasis on the efficiency of managing the impact of fault events as on the correctness of the HPC application's execution. Therefore, the key attributes that define the property of HPC resilience are application reliability and performance.

The development of metrics that capture these attributes is difficult for two reasons. First, the nature of the operating environments of HPC systems are such that faults often develop during the course of regular system operation resulting in bit flips in the system components. However, simply the occurrence of a fault event in a HPC system does not result in fail-stop behavior. Certain faults events never activate, and produce no error notifications. Such faults are called benign faults. Other faults may activate and result in an error condition. It may be possible to detect, and even correct errors. By including capabilities to only detect an error and not correct it, we avoid generating incorrect outputs. When an error is detected and corrected, a fatal application failure may be prevented. A failure in HPC systems is typically observed through an application abort. In some circumstances, an error remains undetected and its impact

is only observed in the erroneous output of an application. Such errors are called silent data corruptions (SDC). In some instances, a failure may be experienced after a long interval following the fault that caused it. Due to the complexity of hardware and software environment of HPC systms, diagnosing the root cause and identifying the activation of a fault and the propagation of the resulting error leading to an incorrect result or fatal failure is difficult. Due to these challenges in categorizing the severity and understanding the impact of various types of faults on an application's execution and outcome, developing standardized metrics that measure an application's reliability is hard.

Second, the HPC workload consists of scientific simulation, modeling and analysis computations, many of which are floating point intensive. These computations tend to be naturally tolerant to data errors since their algorithmic behavior might simply filter the occasional incorrect value, as is the case with many numerical iterative algorithms, or they might rely on pseudorandom processes, as is the case with Monte Carlo techniques. Many scientific and engineering codes that use numerical analysis methods tend to tolerate errors that result in limited loss in floating point precision. While the occurrence of errors may lead to variations in the outputs, often these perturbations are within an allowable range of correctness. Therefore, precisely defining notion of application reliability of HPC applications is highly contextual: for certain scenarios, the reliability of an application is defined by its ability to reproduce state precisely, whereas in other contexts the reliability may be defined in terms of application outcomes that are within a rounding error of the correct result. This makes it difficult to characterize and measure the application reliability.

With the growing scale and complexity of HPC systems, the concern is that the faults in the system and the resulting errors and fatal failures will increase exponentially, which could prevent future generations of extreme-scale HPC systems from running long enough for users to run their applications. The current generation of supercomputers already experience hundreds of faults per day. Therefore, resilience in HPC systems is becoming an important primitive along with performance and power efficiency. To evaluate the usefulness of existing and new resilience solutions for HPC applications and to optimize their implementations, it is important to quantify resilience in terms of its key attributes of application reliability and performance. There are several specific and commonly recognized needs and drivers for developing application-centric resilience metrics:

- In a typical multicomponent HPC environment, there are various types of fault, error and failure events, which impact an application with different levels of severity. The **need to holistically analyze the impact on the application's performance and reliability independent of the nature of the fault, its root cause and its location** is important to quantify application resilience.
- The **need to quantify application resilience independently from the platform's dependability measures**, since the platform MTTF and application MTTF may be different. Certain types of fault events in the system don't affect an application's correctness or performance significantly, while other types are fatal. For example, based on its location, an uncorrectable memory error may cause an application abort, but the system may still continue running. Likewise, during partial failures in a HPC system an application may be able to run to correct completion without interruption.
- The practical need to analyze the merits of new reliability solutions, whether hardware or software on an application's resilience. In order to perform a cost-benefit analysis, resilience metrics **must quantify the improvement in an application's resilience and the performance efficiency of the solution.**
- HPC applications are affected by different types of faults, and must often be supported by multiple resilience solutions to ensure comprehensive protection. The **need to understand the combined effects of employing multiple solutions on the application reliability and performance is important**. For example, a good resilience metric should be able to quantify the net improvement in application resilience achieved by the combination of a checkpoint/restart solution in concert with algorithm-based fault tolerance routine.
- The **need to quantify the performance and reliability impact** of self-correcting algorithmic solutions. Various applications tolerate the presence of faults by accepting a loss in precision, or provide algorithmic solutions that take additional cycles to converge to a solution without raising an interrupt. Resilience metrics must be able to articulate the cost of delivering inexact but acceptable outcomes to the application.
- Cross-layer resilience solutions, which use capabilities at multiple layers of the system stack, are increasingly being employed. Thus, the **need to quantify the improvement in application resilience** as result of combining partial detection, containment and mitigation solutions from different layers of the system stack into complete resilience solutions.
- The **need to quantify the impact of running on degraded platforms or software environments** on an application's performance and reliability. For example, the user-level fault mitigation (ULFM) library provides the ability to transparently reconstruct the MPI communicator upon occurrence of a MPI process failure. Although a parallel application is recovered by shrinking the communicator, the failure of a process causes loss of computing capacity as well as loss of application state. A resilience metric must quantify the reliability and performance impact of the failure of one or more system resources.

3 RESILIENCE METRICS

An application's performance analysis normally ignores the behavior of the application in the presence of faults in the system, and therefore tends to be optimistic. On the other hand, the reliability analysis focuses on understanding the impact of faults, and the protection coverage, and therefore does not adequately capture the impact of faults on an application's performance. For a complete assessment of an application's resilience, it is important to evaluate

both the reliability and performance attributes. These attributes can be measured in many different ways.

Due to the complexity of modern HPC environments, understanding the chain of events from the activation of a fault, the propagation of the resulting error, and the ultimate impact on an application's execution is hard. The models that describe individual component reliability and performance behavior are often imprecise when applied to the entire system. Furthermore, HPC systems tend to experience various types of faults and errors, which affect an application running on the system in different ways (ranging from having no effect at all to causing the application to abort). Developing formal methods for the measurement of the reliability and performance impact for every possible fault type rapidly becomes a complex problem.

Our goal is to develop metrics that provide a measure of the resilience of an application in a manner independent of the type and severity of fault events. We also seek to use the metric to evaluate the effectiveness of new solutions that claim to improve resilience, independent of the layer of abstraction at which the solution is deployed. Therefore, we have developed outcome-based metrics for quantifying the resilience of an application. In general, outcome metrics are based on the end results of a process that follow from a set of preceding events, and they are used to assess the ultimate impact of the events on the outcome of a system. These metrics are often used to evaluate competing objectives that may require tradeoffs. For our purposes, outcome-based metrics enable us to quantify the tangible impact of any fault event on an application's overall performance and reliability attributes.

3.1 Resilience Factor

We define an outcome metric called Resilience Factor (RF) to measure application resilience. Since the resilience of an application is concerned with the integrity of the output state and the performance efficiency of achieving the outcome in the presence of fault events, the RF has two distinct forms: the $\mathrm{RF}_{PerformanceEfficiency}$ (RF_{PE}) and the $\mathrm{RF}_{ValueEfficiency}$ (RF_{VE}).

The RF_{PE} is calculated as the ratio of the time to solution in the absence of fault events to the time to solution in the presence of fault events. This is a relative efficiency measure, which measures the extent to which the performance of an application is impacted by the occurrence of fault events:

$$RF_{PE} = \frac{time - to - solution_{event-free}}{time - to - solution_{events}} \tag{1}$$

The term time-to-solution$_{events}$ captures the overhead to the time to solution associated with dealing with faults events. It accounts for the performance overhead for the detection, diagnosis, and mitigation actions. However, since the RF_{PE} is an outcome metric, the term time-to-solution$_{events}$ is agnostic to the type of event, i.e., whether the event is a fault, error or failure, its root cause, or its location. The term also accounts for scenarios in which an application affected by an event continues running on a system operating in a partially failed or degraded mode. The time-to-solution$_{event-free}$ term can be obtained from an application run that is provably fault free, or by averaging the execution time for statistical confidence of several runs that produce a known *correct* outcome.

The RF_{PE} may also be used to measure the performance efficiency of introducing a new resilience technique for a given application. Here the RF_{PE} ratio captures the performance overhead incurred by the detection and mitigation mechanisms used by the solution. Therefore, the RF_{PE} is calculated as the ratio of the time to solutions of the original application to a version enhanced using a resilience solution X, with the assumption that both terms are measured for identical fault rates.

$$RF_{PE} = \frac{time - to - solution_{Original}}{time - to - solution_{SolutionX}} \tag{2}$$

The $RF_{PE} \leq 1$ and a value of RF_{PE} closer to 1 indicates higher intrinsic efficiency. Because the RF_{PE} expresses the ratio of the time to solution in the absence of fault events to the time to solution with fault events, describes the measurement of the intrinsic reliability of an application and the performance overhead of managing the faults.

The definition of resilience emphasizes the critical transformation of application data into scientific results. The impact of fault events on an application's data is measured by the RF_{VE} metric. It is used to measure the reliability of an application by computing the relative value efficiency of a program variable. The RF_{VE} is based on the basic idea of measuring application data efficiency as a distance from a known correct or approximated value function. At a conceptual level, the RF_{VE} provides a comparison between the value of a program variable when the application is affected by a fault event to a notional perfect value derived from an event-free application execution.

$$\begin{aligned} RF_{VE} &= \frac{ProgramValue_{event-free}}{ProgramValue_{events}} \\ &= \frac{V_x}{V_x + \mid \sigma \mid} \end{aligned} \tag{3}$$

The V_x is the expected correct value of a program variable. Similar to the time-to-solution$_{event-free}$, the V_x can be obtained from an application run that is known to be correct, or by averaging the variable value for statistical confidence from runs that produce an acceptable outcome. The σ represents the variance in a program's value due to the occurrence of fault events during its execution. Since the value efficiency metric is designed to measure the impact of faults on scientific outcome of an application, it is suitable only for application data values, and is not applicable to any control flow variables, pointer and address values.

Using only the RF_{PE} to measure performance efficiency does not provide a measure of the impact of faults on the reliability of the application. On the other hand, the RF_{VE} only measures reliability of application data values, and by itself does not account for the fault handling efficiency. However, for evaluating application resilience the two metrics when viewed in context help the HPC designers and users to compare and contrast the suitability of alternative resilience solutions, and select the most appropriate one for their application.

3.2 Resilience Factor Yield

To summarize the resilience properties of an application using a single number, the resilience factors must be combined in a way that measures the combined effect of all RFs. Because the RF is a ratio that calculates performance and value efficiency rather than an absolute execution time or absolute data value, the average resilience of an application is computed by taking a geometric mean of the RFs. We call this measure the Resilience Factor Yield (RY), and it may be applied to summarize RF_{PE} and RF_{VE}. The geometric mean has the property that the geometric mean of the ratios is the same as the ratio of the geometric means, and it provides a measure of central tendency.

When an HPC application consists of a set of tasks T = {T_1, T_2, T_3 . . . T_N} and the RF for each individual task is computed as RF_{T_n}, the overall application resilience may be calculated using the RY:

$$RY = \sqrt[n]{RF_{T1}RF_{T2}...RF_{Tn}} \tag{4}$$

Since an HPC application is vulnerable to multiple types of fault events, it may be possible that several distinct resilience solutions are combined to enhance its overall resilience. The RY is an appropriate metric for understanding the overall improvement in application resilience on account of multiple solutions. For a set of solutions S={S_1, S_2, S_3 ... S_N} that each provide fault resilience for an application, the RY_{PE} provides a measure of their combined impact on the performance characteristics. These solutions may be implemented at any layer of the system stack.

Similarly, the RY_{VE} enables the measurement of aggregate effect of multiple fault models, or the combination of resilience solutions on the efficiency of an application data value:

$$RY_{VE} = \sqrt[n]{RF_{VE_1}RF_{VE_2}...RF_{VE_n}} \tag{5}$$

Since the RF_{VE} metric is used to measure the efficiency of a single application data value, the RY may also be used to average the RF of more than one application data value. For example, in a matrix data structure, the RY provides a single value efficiency measure for the matrix by taking the geometric mean of the RF_{VE} of each element of the matrix. When the RY is applied to several application data values, there is an implicit assumption of independence between the values. For a simple application that contains only two variables A and B, both of these are vulnerable to errors, and the RF_A and RF_B measures the impact of any perturbations of each value on the outcome of the application. The cumulative effect, i.e., the errors in A and B is computed by calculating the RY.

4 APPLYING THE RESILIENCE METRICS

In this section, we demonstrate the use of the resilience measures for a linear solver application.

4.1 Measuring Soft Error Resilience

We use the RF to measure the resilience of an iterative linear solver application to silent errors (also known as silent data corruptions). Since the SDCs typically remain undetected, the MTTI of the application does not capture the impact of the SDCs on the solver. In solving a linear system of equations A.x = B, the iterative solver begins with an approximation of the solution and progressively refines the solution until the residual error is below a specified threshold. The injection of a silent data corruption within the solver may impact the outcome of the application in different ways depending on the location of the error. The silent error may be benign and not affect the outcome of the application, may be detected by the system, cause abnormal termination of the solver, or may remain undetected but affect the correctness of the final outcome.

For our study, we injected the SDCs in only the solution vector 'x' of the solver during execution of the application. Therefore, the SDCs are unlikely to cause the application to fail, but they may affect its correctness and its time to solution. The RF_{PE} indicates the impact of SDCs on the performance of the iterative linear solver. Figure 2 (a) illustrates the RF_{PE} for application runs during which upto 8 SDCs were injected into the solution vector. The time to solution terms used in the calculation of the RF_{PE} are for runs that complete correctly. Despite the fact that there is no algorithm-based detection or mitigation available for these application runs, the $RF_{PE} > 0.9$ since the algorithm requires a limited number of additional iterations to converge to correct solutions due to the intrinsic resilience of the solver.

To improve the resilience of the iterative linear solver using algorithm-based fault tolerance techniques, we incorporate SDC detection by tracking the residual norm of the solver. For a correct solution, it is expected the residual monotonically reduces every iteration of the solver. Therefore, the detection algorithm flags the presence of a SDC when the montononicity condition is violated. For the recovery of the solver, the iterations performed until the detection of the error condition are discarded, and the solver is restarted from an initialized state. This avoids the need to maintain checkpoints of the variable as the solver progresses. Figure 2 (b) shows the impact of the ABFT detection and mitigation on the RF_{PE} of the solver. The RF_{PE} for the linear solver is still > 0.9 since the algorithmic detection is fairly inexpensive. The RF_{PE} being > 0.9 even for higher SDC injection rates suggests that not every SDC causes a violation of the monotonicity norm of the residual. Therefore, although the recovery entails discarding the iterations of the solver, such action is not invoked for every SDC.

Figure 3 summarizes the resilience factor in terms of the precision for the iterative linear solver. For these SDC injection experiments, we measure the impact on the residual value. The Figure 3 shows the RF_{VE} for the residual for multiple SDC injection rates. For these runs there is no algorithm-based detection or mitigation. The distribution of RF_{VE} illustrates the variance in the residual resulting from the silent data corruptions. For most runs, the RF_{VE} values are clustered above 0.9, which confirms inherent resilience of the solver algorithm and the limited impact of the multiple SDCs on the residual value.

4.2 Measuring Hard Error Resilience

We also use the RF to evaluate the resilience of the solver to hard errors that result in process failures. For an MPI implementation of the solver, we rely on the primitives supported by the User-level Failure Mitigation (ULFM) interface. ULFM extends the MPI standard with primitives that enable support for handling process failures, which enable parallel applications to recover from the failure of

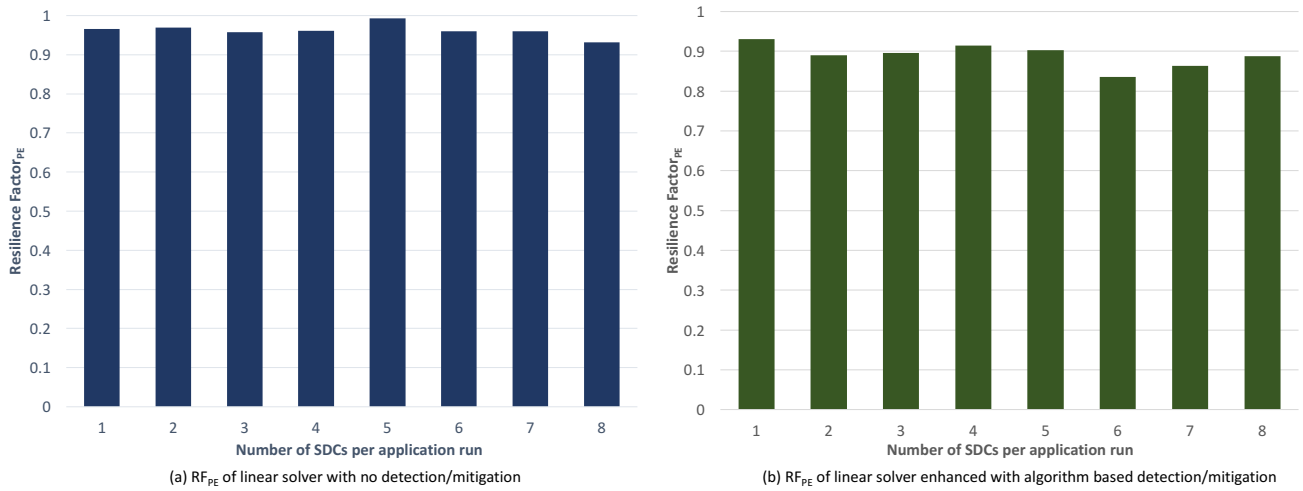

Figure 2: Resilience Factor$_{PE}$ for SDC affected linear solver

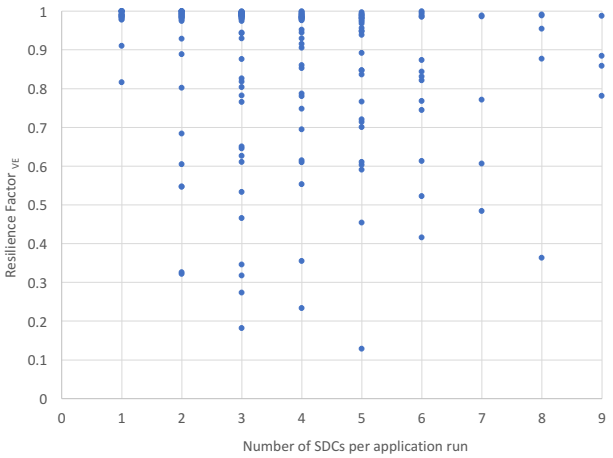

Figure 3: Resilience Factor$_{value}$ for linear solver residual

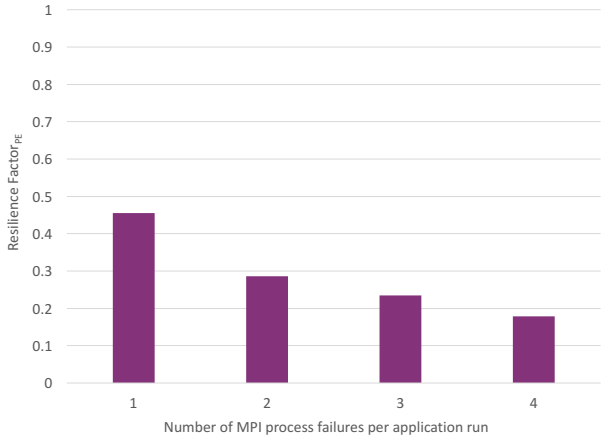

Figure 4: Resilience Factor$_{perf}$ for solver affected by MPI process failures

MPI ranks. The solver uses error codes returned from MPI routines to detect the presence of failed ranks. The MPI_Comm_revoke revokes the communicator, and for recovery of the application, the MPI_Comm_shrink primitive is used to shrink a communicator, which excludes all known failed MPI ranks. The MPI process failure recovery mechanism requires the solution vector state on each processor rank to be copied to a neighbor rank during normal operation.

The results for the RF$_{PE}$ for the linear solver in Figure 4 compare the resilience factor, i.e., the performance outcome ratio between a failure free execution of the solver to the performance in the presence of multiple MPI process rank failures. The results indicate the inefficiency of the ULFM-based recovery and the performance impact of continuing the solver on partially failed communicator.

4.3 Measuring Hard and Soft Error Resilience

One of the key benefits of the RF is that it enables the quantification of an application's reliability and performance characteristics independent of the fault model. Therefore, the metric captures the combined impact on these application attributes in a single number. In order to evaluate the combined effect of solutions that provide hard and soft error resilience for the linear solver application, we rely on the algorithm-based detection and mitigation for transient silent errors and the ULFM MPI interface for tolerating process failures. The RF$_{PE}$ for the linear solver shown in Figure 5 shows the results for application execution runs, which are injected with SDCs and hard errors that result in MPI rank failures. For these experiments, the application uses 32 MPI ranks and each run is injected with 1 hard error and multiple SDCs. Despite the single

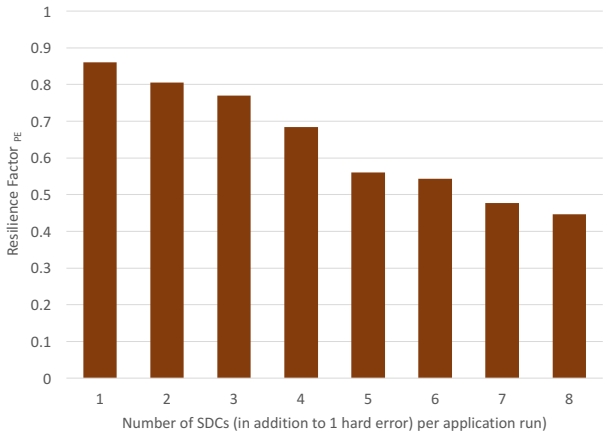

Figure 5: Resilience Factor$_{perf}$ for linear solver with hard and soft errors

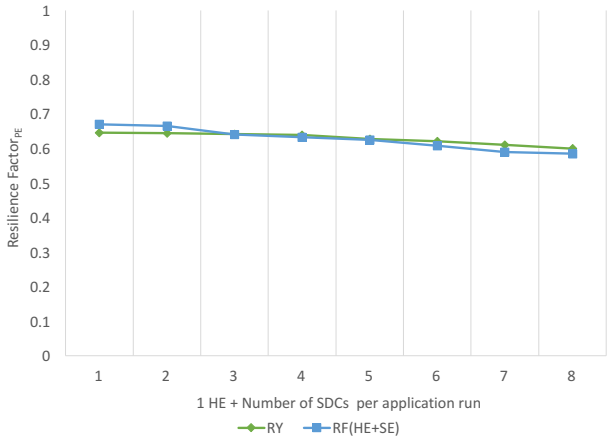

Figure 6: Correlating the Resilience Factor Yield for hard and soft errors

failure per run, the solver shows rapid decrease in the RF$_{PE}$ as the number of SDCs injected is increased.

4.4 Calculating the Resilience Factor Yield

The RF$_{PE}$ provides a measure of the impact of incorporating a resilience solution for distinct fault types on the application performance. Similarly the RF$_{VE}$ measures the improvements in an application's reliability due to a specific solution. However, calculating the RF separately provides an incomplete assessment of the overall application resilience. To understand the combined effect of using the algorithmic solution for SDC detection and mitigation and the ULFM-based solution for handling process failures, we apply our Resilience Factor Yield metric. We calculate the RF$_{PE-SE}$ using experiments in which the solver application is enhanced with the algorithm-based detection and mitigation and the application execution is subjected to only SDCs. Similarly, the RF$_{PE-HE}$ is calculated using experiments in which the ULFM primitives are used for the detection of process failures and recovery of the MPI communicator. The RY, which is calculated as the geometric mean of the RF$_{PE-SE}$ and RF$_{PE-HE}$, provides a single figure of merit for application resilience to SDC and fail-stop errors. We correlate this calculated RY with the RF$_{PE-HE+SE}$, which is measured using experiments in which the application code contains the algorithm-based mechanisms as well as the ULFM primitives and the application is simultaneously subjected to hard and soft errors. For the measurement of the RF$_{PE-HE+SE}$, the application is injected with a single hard error and multiple SDCs. For the calculation of the RY, the application runs are subjected to the same number of SDCs for the individual measurement of the RF$_{PE-SE}$ and injected with a single hard error per run for calculating the RF$_{PE-HE}$. Figure 6 shows the correlation between the RY calculated using the equation 5 and the actual RF$_{PE-HE+SE}$. The strong correlation between the RY and RF$_{PE-HE+SE}$ for similar fault injection rates justifies the use of our resilience factor yield to measure the overall impact on application resilience of multiple disparate fault types and the composition of resilience solutions implemented at multiple layers of the system.

5 RELATED WORK

While the MTTF and MTTR are the most widely used metrics for quantifying the reliability of a system, several variations of these metrics have been developed. For example, with the recognition that not every anomalous event in a system results in failure, the measure of mean time to interrupt (MTTI) is used in place of MTTF for intervals between errors. However, the MTTI does not for account for silent errors, which escape detection but result in incorrect application outcomes or premature termination of an application. Other variations of the MTTF are the job MTTF (JMTTF), the system MTTF (SMTTF), which accounts for the full HPC system and node MTTF (NMTTF) for individual nodes [16]. These reliability metrics provide a measure of the continuous service accomplishment, whereas availability provides a measure of the service accomplishment as a ratio that includes the time to repair a system and resume service. Note that reliability and availability are quantifiable metrics, rather than synonyms for dependability.

The Architectural Vulnerability Factor (AVF) [10] was defined as the probability that a fault in a microarchitectural structure will result in a visible error in the final output of a program. The Timing Vulnerability Factor (TVF) provides a measure of the fraction of time a circuit node or device is susceptible to soft error upsets. The Program Vulnerability Factor (PVF) is a program-level metric that allows insight into the vulnerability of a software resource to hardware faults [15]. The Data Vulnerability Factor (DVF) was proposed as a metric to model the vulnerabilities of individual application data structures to support quantifying the impact of algorithm optimization [17]. The vulnerability factors provide an assessment of how sensitive a system is to a specific type of event. The derating factor was defined as the inverse of total error rate to represent an application's resilience against transient faults in memory [4]. Therefore, a derating factor of 10 implies that one out of every

10 transient faults in the system memory would lead to system failure. The normalized memory derating factor (NMDF) metric was developed to account for the size of the memory footprint, and is calculated as the inverse of total error rate divided by the size of the footprint [9].

Performability metrics have been developed to study the reliability and performance indices of a system in a composite manner [12]. Workload efficiency is a system-centric metric, which is a ratio of the wall clock running time for an idealized failure-free execution of a mix of several jobs to the actual wall clock time that includes scheduled and unscheduled system downtime [14]. The Total Productivity Factor (TPF), which is inspired by the definition of productivity and utility in economics, is defined as the ratio of the total cost of ownership of a HPC system to the value of productive scientific research achieved from the lifetime of the system [13]. The Failure Index (FI) has been proposed to study HPC application log files since it highlights the fluctuations in the failure intervals of an HPC system in contrast to statistical mean of all failure intervals provided by MTTF, which tends to ignore the outlier values. The FI is inspired by the Gini and Atkinson indices, which provide a measure of dispersion and inequality rather measure of central tendency of failure rate [11].

6 CONCLUSION

The resilience of high-performance computing applications to the presence of frequent faults, errors and failures in the system is one of the key challenges as systems use components manufactured using deeply scaled transistor devices, which are inherently less reliable than previous generations, and as system architectures and the software stack becomes progressively more complex. The HPC community has defined term resilience using an application centric paradigm, which places greater emphasis on the correctness of an application and the performance efficiency of managing any fault, error or failure events during its execution. However, we have continued to rely on system-based dependability metrics for reliability and availability. In this paper, we proposed metrics for quantifying resilience. In developing these new metrics, we advocate for an outcome-based approach to quantify resilience in terms of the performance and reliability implications of fault events on HPC applications. Outcome metrics attempt to consider the overall impact of fault events on an application's performance and the reliability of its computation. The resilience factor (RF) provides a measure of the impact of events in terms of performance and reliability outcomes. When applications are protected using resilience mechanisms, the RF metric provides a consistent method to measure the overall improvement in the reliability, and the penalty for detection, containment and mitigation of events, independent of the fault model and whether the resilience mechanism is a hardware or software-based solution. The resilience yield (RY) provides a measure of the effective resilience by combining the RF_{PE} and RF_{VE} of the individual aspects of an HPC application. We demonstrated the utility of the designed metrics by applying them to a linear solver application subjected to transient errors in the form of SDCs and hard errors that result in MPI process failures. The RF and RY were shown to provide simple, consistent ways to quantify the resilience of HPC applications.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, program manager Lucy Nowell, under contract number DE-AC05-00OR22725.

REFERENCES

[1] 2009. ResiliNets Strategy for Resilient and Survivable Networking. (2009). wiki.ittc.ku.edu/resilinetswiki/index.php

[2] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable Secure Computing* (January 2004), 11–33.

[3] P. Cholda, J. Tapolcai, T. Cinkler, K. Wajda, and A. Jajszczyk. 2009. Quality of resilience as a network reliability characterization tool. *IEEE Network* 23, 2 (March 2009), 11–19. DOI: http://dx.doi.org/10.1109/MNET.2009.4804331

[4] Kypros Constantinides, Stephen Plaza, Jason Blome, Bin Zhang, Valeria Bertacco, Scott Mahlke, Todd Austin, and Michael Orshansky. 2005. Assessing SEU Vulnerability via Circuit-Level Timing Analysis. In *In Proceedings of the 1st Workshop on Architectural Reliability.*

[5] J. T. Daly, L. A. Pritchett-Sheats, and S. E. Michalak. 2008. Application MTTFE vs. Platform MTBF: A Fresh Perspective on System Reliability and Application Throughput for Computations at Scale. In *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID).* 795–800. DOI: http://dx.doi.org/10.1109/CCGRID.2008.103

[6] N DeBardeleben, J Laros, JT Daly, SL Scott, C Engelmann, and B Harrod. 2009. High-End Computing Resilience: Analysis of issues facing the HEC community and path-forward for research and development. *Whitepaper* (December 2009).

[7] J. Laprie. 2005. Resilience for the Scalability of Dependability. In *Fourth IEEE International Symposium on Network Computing and Applications.* 5–6. DOI: http://dx.doi.org/10.1109/NCA.2005.44

[8] Jean-Claude Laprie. 1995. Dependable Computing: Concepts, Limits, Challenges. In *Proceedings of the Twenty-Fifth International Conference on Fault-tolerant Computing (FTCS'95).* IEEE Computer Society, Washington, DC, USA, 42–54.

[9] Mojtaba Mehrara and Todd Austin. 2008. Exploiting Selective Placement for Low-cost Memory Protection. *ACM Trans. Archit. Code Optim.* 5, 3, Article 14 (Dec. 2008), 24 pages. DOI: http://dx.doi.org/10.1145/1455650.1455653

[10] Shubhendu S. Mukherjee, Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin. 2003. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 36).* IEEE Computer Society, Washington, DC, USA, 29–.

[11] Andrei Păun, Clayton Chandler, Chokchai Box Leangsuksun, and Mihaela Păun. 2016. A failure index for {HPC} applications. *J. Parallel and Distrib. Comput.* 93-94 (July 2016), 146 – 153. DOI: http://dx.doi.org/10.1016/j.jpdc.2016.04.009

[12] R. M. Smith, K. S. Trivedi, and A. V. Ramesh. 1988. Performability analysis: measures, an algorithm, and a case study. *IEEE Trans. Comput.* 37, 4 (April 1988), 406–417. DOI: http://dx.doi.org/10.1109/12.2184

[13] Marc Snir and David A. Bader. 2004. A Framework for Measuring Supercomputer Productivity. *International Journal of High Performance Computing Applications* 18, 4 (Nov. 2004), 417–432. DOI: http://dx.doi.org/10.1177/1094342004048535

[14] Marc Snir, Robert W Wisniewski, Jacob A Abraham, Sarita V Adve, Saurabh Bagchi, Pavan Balaji, Jim Belak, Pradip Bose, Franck Cappello, Bill Carlson, Andrew A Chien, Paul Coteus, Nathan A DeBardeleben, Pedro C Diniz, Christian Engelmann, Mattan Erez, Saverio Fazzari, Al Geist, Rinku Gupta, Fred Johnson, Sriram Krishnamoorthy, Sven Leyffer, Dean Liberty, Subhasish Mitra, Todd Munson, Rob Schreiber, Jon Stearley, and Eric Van Hensbergen. 2014. Addressing failures in exascale computing. *International Journal of High Performance Computing Applications* 28, 2 (2014), 129–173. DOI: http://dx.doi.org/10.1177/1094342014522573

[15] Vilas Sridharan and David R. Kaeli. 2008. Quantifying Software Vulnerability. In *Proceedings of the 2008 Workshop on Radiation Effects and Fault Tolerance in Nanometer Technologies (WREFT '08).* ACM, New York, NY, USA, 323–328. DOI: http://dx.doi.org/10.1145/1366224.1366225

[16] Jon Stearley. 2005. Defining and measuring supercomputer Reliability, Availability, and Serviceability (RAS). In *In Proceedings of the Linux Clusters Institute Conference.*

[17] Li Yu, Dong Li, Sparsh Mittal, and Jeffrey S. Vetter. 2014. Quantitatively Modeling Application Resilience with the Data Vulnerability Factor. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14).* IEEE Press, Piscataway, NJ, USA, 695–706. DOI: http://dx.doi.org/10.1109/SC.2014.62