Predictive Reliability and Fault Management in Exascale Systems: State of the Art and Perspectives

RAMON CANAL, Universitat Politècnica de Catalunya (UPC) and Barcelona Supercomputing Center (BSC)
CARLES HERNANDEZ and RAFA TORNERO, Universitat Politècnica de València (UPV)
ALESSANDRO CILARDO, Universitá degli studi di Napoli Federico II (UNINA)
GIUSEPPE MASSARI, FEDERICO REGHENZANI, and WILLIAM FORNACIARI, Politecnico di Milano (POLIMI)
MARINA ZAPATER and DAVID ATIENZA, Ecole Polytechnique Federale de Lausanne (EPFL)
ARIEL OLEKSIAK and WOJCIECH PIĄTEK, Poznan Supercomputing and Networking Center (PSNC)
JAUME ABELLA, Barcelona Supercomputing Center (BSC)

Performance and power constraints come together with Complementary Metal Oxide Semiconductor technology scaling in future Exascale systems. Technology scaling makes each individual transistor more prone to faults and, due to the exponential increase in the number of devices per chip, to higher system fault rates. Consequently, High-performance Computing (HPC) systems need to integrate prediction, detection, and recovery mechanisms to cope with faults efficiently. This article reviews fault detection, fault prediction, and recovery techniques in HPC systems, from electronics to system level. We analyze their strengths and limitations. Finally, we identify the promising paths to meet the reliability levels of Exascale systems.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → **Grid computing**; *Reliability*;

Additional Key Words and Phrases: HPC, supercomputing, exascale, reliability, prediction, survey, faults, failures

This work has received funding from the European Union's Horizon 2020 (H2020) research and innovation program under the FET-HPC Grant Agreement No. 801137 (RECIPE). Jaume Abella was also partially supported by the Ministry of Economy and Competitiveness of Spain under Contract No. TIN2015-65316-P and under Ramon y Cajal Postdoctoral Fellowship No. RYC-2013-14717, as well as by the HiPEAC Network of Excellence. Ramon Canal is partially supported by the Generalitat de Catalunya under Contract No. 2017SGR0962.
Authors' addresses: R. Canal, Universitat Politècnica de Catalunya (UPC) and Barcelona Supercomputing Center (BSC); email: rcanal@ac.upc.edu; C. Hernandez and R. Tornero, Universitat Politècnica de València (UPV); emails: carherlu@upv.es, ratorga@disca.upv.es; A. Cilardo, Universitá degli studi di Napoli Federico II (UNINA); email: acilardo@unina.it; G. Massari, F. Reghenzani, and W. Fornaciari, Politecnico di Milano (POLIMI); emails: {giuseppe.massari, federico.reghenzani, william.fornaciari}@polimi.it; M. Zapater and D. Atienza, Ecole Polytechnique Federale de Lausanne (EPFL); emails: {marina.zapater, david.atienza}@epfl.ch; A. Oleksiak and W. Piątek, Poznan Supercomputing and Networking Center (PSNC); emails: {ariel, piatek}@man.poznan.pl; J. Abella, Barcelona Supercomputing Center (BSC); email: jaume.abella@bsc.es.

0360-0300/2020/09-ART95 $15.00
https://doi.org/10.1145/3403956

ACM Reference format:
Ramon Canal, Carles Hernandez, Rafa Tornero, Alessandro Cilardo, Giuseppe Massari, Federico Reghenzani, William Fornaciari, Marina Zapater, David Atienza, Ariel Oleksiak, Wojciech Piątek, and Jaume Abella. 2020. Predictive Reliability and Fault Management in Exascale Systems: State of the Art and Perspectives. *ACM Comput. Surv.* 53, 5, Article 95 (September 2020), 32 pages.
https://doi.org/10.1145/3403956

1 INTRODUCTION AND MOTIVATION

Exascale-grade High-performance Computing (HPC) systems require dramatic improvements in energy efficiency to provide an unprecedented performance level within a strict power envelope. This implies the use of advanced Complementary Metal Oxide Semiconductor (CMOS) technologies only a few nanometers wide (i.e., each logic gate may consist of few atoms). Yet, device scaling comes together with a higher susceptibility to manufacturing process variations, environmental conditions, radiation and aging [41]. While these effects may be lower than in previous technologies at the transistor or singe-cell level [101], the reliability per area unit is similar or lower in latest technology nodes [127]. These dependability concerns together with the incredibly large number of electronic devices in future Exascale systems, puts reliability in par with energy efficiency and performance as design considerations in current and future HPC systems. As a consequence, Exascale systems will suffer dramatically higher fault rates [25, 93]. This is already observable in Petascale systems as we will see in Section 1.1.

In this scenario, classic error detection and correction mechanisms will fail to scale, since they have been devised to deal with relatively low fault rates. Therefore, Exascale systems cannot rely only on error detection and correction mechanisms acting once faults have happened, but need instead effective ways to maximize applications survivability and, consequently, making the system more efficient and predictable. Exascale systems will require ensuring reliable operation in the presence of very high fault rates, including transient and permanent faults, steadily degrading hardware while meeting stringent power constraints and achieving high performance.

1.1 Errors Reported in HPC Systems

Reliability has already been a concern for HPC systems for decades. As early as 2003, the Big Mac Virginia Tech's Advanced Computing facility failed to boot due to the high failure rate in non-ECC protected memory [142]. The impact of radiation was later confirmed in 2009, when the Jaguar supercomputer (number 1 in the Top500 list at that time) reported 350 ECC-corrected errors per minute [142]. However, ECC is not enough to deal with increasing fault rates. For instance, the Titan supercomputer at Oak Ridge National Lab recently reported a Mean Time Between Failure (MTBF), due to detected uncorrectable errors (DUE) caused by radiation, of only 44 hours [168]. The previous works provide DRAM-only errors, but an HPC system is susceptible to other sources (e.g., abnormal execution time, software bugs, I/O errors) [60]. When considering all possible sources, Reference [58] reported a Mean Time To Interruptions (MTTI) of 3.5 days; and Reference [120] reported a 20× increase in failure rate when an application moves from 10.000 to 22.000 CPU cores.

Reliability concerns affect not only supercomputers but also data centers. For instance, a recent study for Facebook data centers reveals that every month 3% of the servers experience errors corrected in memory, whereas 0.03% of the servers experience DUE in DRAM memory [122]. Thus, only memory errors, despite ECC protection, may make one out of every 3,000 servers to crash every month with 2014–2015 technology. Moreover, even recoverable errors have a non-negligible impact on performance [92]. More advanced technologies with higher susceptibility to radiation

and aging, the use of larger memories, as well as the effect of other semiconductor components, such as processors and GPUs, can only lead to much higher hardware-related failure rates in the future.

1.2 The Need for Reliability in HPC

Reliability has been acknowledged as a major roadblock for HPC applications in current supercomputers and data centers [25]. As reported in recent years, faults have already the power to cause frequent issues in nowadays HPC systems despite existing means for fault tolerance [63]. In fact, the reliability of HPC systems has recently gained particular attention [14, 93].

In particular, the authors in Reference [93] propose a methodology based on identifying patterns for faults and errors. These patterns define a flow from detection to containment and recovery. The methodology considers the full system stack and it is demonstrated for checkpoint-restore (for process failures), process migration (for error avoidance) and DRAM errors (for soft-error resilience). These works are orthogonal to ours. In this work, we focus on the hardware/middleware interactions for HPC resiliency. At the hardware level, the already known reliability problems (e.g., process variations, soft-errors) come along additional problems at different scales, such as thermal and application timing issues.

1.2.1 Thermal Issues. The increasing power density in post-Dennard chips increases on-chip temperature. In turn, on-chip peak temperatures and thermal gradients increase silicon device wear-out and they threaten the **long-term reliability** of chips (usually measured as Mean Time To Failure (MTTF)) [70]. Current techniques such as Dynamic Voltage and Frequency Scaling (DVFS) or core turn-off can potentially reduce the system's long-term reliability due to undesired collateral effects such as thermal cycling [39]. In metallic structures, if the thermal cycle amplitude increases from 10°C to 20°C, lifetime reliability can decrease up to 16× [38]. Furthermore, given that performance is highly affected by thermal aspects, the operational frequency can decrease more than 35% when working at 110°C instead of at 60°C [90].

The concerns above lead to the need for prediction-based (proactive) and emergency-based (reactive) thermal management with the goal of reducing hot-spots and maintaining temperature gradients within a 5°C limit. Otherwise, the large number of chips in Exascale systems together with highly heterogeneous thermal-related faults across chips (e.g., due to different utilization and different process variations) will lead to highly unpredictable and frequent faults. Similarly, thermal aspects also affect indirectly system performance. Namely, the vibrations induced by the fans may have a noticeable impact on the IO throughput, and thus decrease the performance of data-sensitive applications [29, 30]. One should underline that proper thermal management is crucial from the perspective of system reliability and delivered quality of service.

1.2.2 Application Timing Issues. Corrected errors (CE) may have collateral effects in timing, thus decreasing performance and QoS. Detected Unrecoverable Errors (DUE) impose the abnormal termination of applications and potential system reboots, which may lead to increased operational costs and lower end user satisfaction. Finally, Silent Data Corruption (SDC) can be even more challenging than DUE, since failures remain unnoticed, which may have catastrophic consequences depending on the type of application where they occur, since HPC applications are nowadays widespread in financial, engineering and scientific domains among others.

Applications have also shown high susceptibility to correctable errors in data centers depending on the means set to log those errors [78]. In particular, owners of data centers need to log information related to errors to diagnose systematic failures and replace faulty (or error prone) components. However, as shown in Reference [78], a fault rate of four errors per second is enough to increase execution time of HPC applications by 2.5× and decrease the quality-of-service (QoS)

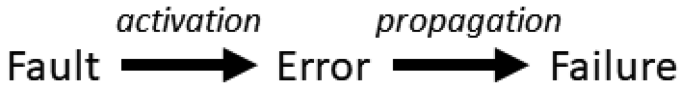

Fig. 1. Fundamental chain of dependability threats.

of web-based applications by 100×. Larger memories and advanced CMOS technologies can only worsen this trend.

A number of operations are intrinsically resilient to faults due to their heuristic nature or due to the characteristics of their output, so that upon a fault, the impact may be a slower convergence towards the solution, or a degradation of the output that, although different from the golden (i.e., error free) output, is semantically equivalent or sufficiently good. For instance, a fault altering the temperature or humidity of a data point in a large matrix used for weather forecasting, has negligible effects at the scale at which weather is forecast. In the particular case of HPC applications, this effect has been studied for multiple solvers [22, 23, 26, 66]. Therefore, although an increasing fault rate challenges correct execution of applications and/or their performance, some faults, even if uncorrectable, may be naturally tolerated for some applications. This has particular relevance in HPC systems where a single failure may impact the execution of an application running for many hours on a large set of computing resources, thus making full re-execution unaffordable and yielding ineffective rollback (e.g., checkpointing) if errors are frequent. In this context, ignoring some faults may be a suitable solution.

1.3 Summary

Increased fault rates in future HPC systems will naturally lead to higher CE, DUE, and SDC rates, thus causing unacceptable impact in performance, QoS, and operational costs, apart from unforeseeable consequences due to SDC. Therefore, error detection and correction techniques, while still needed, will not be enough to deal with increased fault rates. In this context, solutions to mitigate fault rates, and to keep temperature low and constant so that fault rates do not exacerbate, will be needed to complement fault tolerance. In this article, we specifically focus on solutions addressing HPC systems. While some of these approaches may be applicable to conventional data centers, the uniqueness of the HPC infrastructure requires its own analysis and particular solutions—as we will see in short. We first review the state of the art on relevant fault models for HPC systems (Section 2), fault prediction techniques (Section 3), and error detection and correction techniques for HPC systems (Section 4), putting all techniques in perspective with forthcoming HPC challenges. Finally, we provide some future directions for research and summarize this work in Section 5.

2 FAULT TAXONOMY AND MODELS FOR HPC

The introduction of the generic term *dependability* was probably the first attempt to introduce "a global concept that contains the attributes of reliability, availability, safety, maintainability, etc." [11]. In 1980, a joint committee on "Fundamental Concepts and Terminology" was formed by the IEEE Computer Society and the IFIP WG 10.4 [110]. The committee published in 1992 a book named *Dependability: Basic Concepts and Terminology* [109, 111]. In this work, we will follow their terminology. Especially significant is the distinction between fault, error, and failure. As defined in Reference [11], "the fundamental chain of dependability and security threats" is shown in Figure 1.

For instance, at circuit level faults occur due to many reasons (e.g., electromagnetic interference) and create a wrong transient or permanent state in combinational or sequential elements. Eventually, a fault may be activated by making it visible somehow at the output of the circuit (e.g., output latch) thus becoming an error. This could be the case of a particle strike (fault) that creates

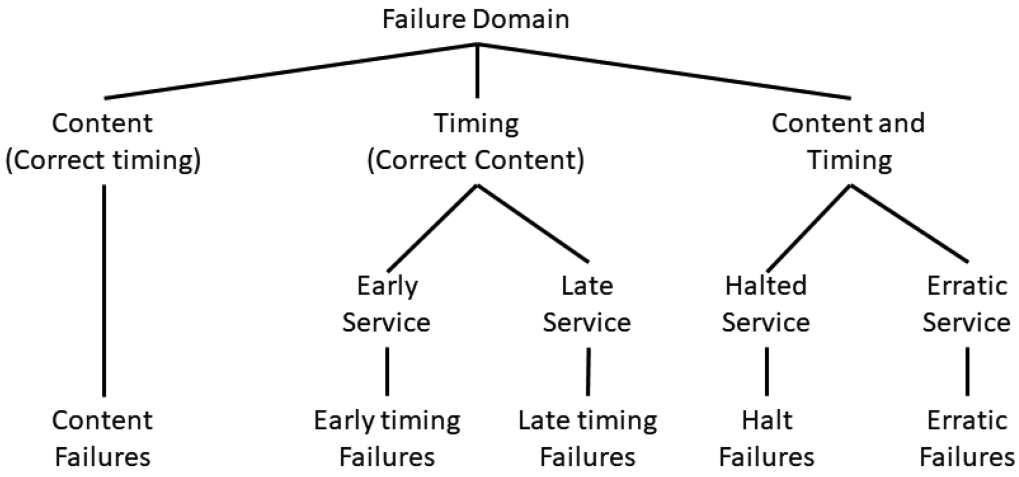

Fig. 2. Failure modes according to their domain.

a glitch in an internal signal of an adder. If such signal determines the output of the adder, then it becomes an error. Eventually, if such error is not managed properly (e.g., it is not detected), it can be propagated until becoming a failure at the specific scope (e.g., at microarchitectural level). For instance, the adder may deliver a wrong result for an instruction of the program. Such failure may be perceived as a fault at a coarser scope. For instance, such wrong value may only be used to be compared against another value and the output of the comparison is not changed, thus masking the fault at software level, or such fault may be propagated to the visible output of the program, thus potentially becoming a failure at a more general scope.

As classified in Reference [11] from the failure domain viewpoint, one can distinguish between:

- "*Content failures*. The content of the information delivered deviates from the golden (non-faulty) execution."
- "*Timing failures*. The arrival time or the duration of the execution deviates from the non-faulty execution."

These definitions can be specialized (i.e., content can be numerical, alphanumerical or any other (e.g., color), and a "timing failure" can be early, late, or simply not complete after a given time threshold). When both timing and content are incorrect, failures can be classified as:

- "*Halt* if the service is halted (the external state becomes constant, or the system activity is not perceptible)."
- "*Erratic*s or *degradation*. The system responds but it is erratic or under-performing."

Figure 2 summarizes the service failure modes with respect to the failure domain viewpoint.

Merging the previous classification, many recent works encapsulate both timing and/or content errors in the following categories:

- *Silent Data Corruption (SDC)*. No error is detected, but data (application code and/or application data) is corrupted. No change in timing is observed.
- *Detected Unrecoverable Error (DUE)*. An error is detected (in data and/or timing), but it cannot be corrected.
- *Corrected Error (CE)*. An error, either explicitly detected or not, is corrected so that content and timing are recovered.

2.1 HPC Specific Requirements

HPC centers have some specific requirements that are not present in conventional data centers. For instance, in data centers, the business will define the Tier level of the infrastructure. To the contrary, HPC data centers require a differentiated approach to maximize scientific output while minimizing operational and capital expenses. This requires a detailed analysis of the services/applications provided and customer profiles. Different applications or customer groups may require different levels of redundancy and reliability. System designers and managers must trade-off availability against the cost of potential failures. Another difference between HPC and data centers is the power density of a rack. In a conventional data center, it varies between less than 1 kW up to 6 kW, whereas HPC centers deal with much higher densities. It is not unusual to find machines, currently in the market, consuming over 30 kW per rack [183]. Consequently, in this survey, we focus solely on HPC centers and solutions.

Overall, lower power is desirable for cost reasons, as well as lower temperature. This, in fact, allows us to reduce cooling costs, with direct benefits in terms of infrastructure reliability and reduction of power needed to feed the cooling system. However, the need for increasing performance in HPC systems would reallocate any potential power saving or temperature reduction to increase computational power.

As the HPC centers get larger, the probability of errors increases proportionally to the number of components. This is particularly true as technology integration grows, thus using smaller devices (e.g., gates and wires), which are subject to increasing current densities at relatively high temperatures. Accordingly, those devices experience higher stress as the silicon technology shrinks. Therefore, HPC centers need to deploy suitable monitoring, prevention, and recovery tools to keep their systems operational.

2.1.1 Energy and Power Modelling. There are now several extensive research efforts focusing exclusively on power and energy models and techniques for the processors composing these extreme-scale computing systems. In Reference [134], the authors summarize these research efforts with a special focus on predictive power and energy models with an emphasis on node architecture (considering CPUs, GPUs, Intel Xeon Phis, and FPGAs). While Reference [134] presents power- and energy-related analytical models for high-performance computing systems and applications, Reference [43] focuses more on methods and tools as well as whole simulation environments that can make use of such models. Without power and energy models, it is not possible to perform thermal modeling and, thus, model the reliability aspects that are described in the next section.

2.1.2 Thermal Aging and Reliability. Bias Temperature Instability (namely, BTI), is a degradation effect that changes the threshold voltage of CMOS transistors, and has been regarded as one of the most relevant aging effects in CMOS technology [163]. From a technological perspective, the BTI occurs when, under a constant gate voltage, a stress in temperature (i.e., increasing temperature from ambient to 200°C) results in charges being trapped in the transistor gate oxide and reduces the voltage threshold of transistors [67]. These variations affect the switching characteristics of transistors and, therefore, the maximum frequency under which the circuit can work [157].

The BTI effect causes two main components: (i) a non-permanent effect that disappears once the system is switched off, and (ii) a semi-permanent effect that increases the effect of the previous one as the system ages. This partially recoverable nature of BTI poses some interesting challenges for the power/thermal/performance management of circuits as the duration of sleeping periods can impact BTI degradation and the overall system's reliability. Recent work shows the importance of the impact of temperature transients on BTI [65], whereas the previous observation implies that taking into account both, application characteristics and transient temperature on BTI modeling,

is of utmost importance. This has been mostly analyzed at the circuit but also chip level [28]. Yet, there start to be efforts to evaluate physical effects at the data center level [104], but limited to electromigration in the power delivery network.

Observation 1. *While application and circuit-level concerns have been kept separated in HPC, thermal effects on reliability call for the investigation of the impact of applications on temperature, as well as for appropriate techniques to minimize thermal transients with negligible impact on performance.*

2.1.3 Thermal Modeling and Reliability in Heterogeneous Reconfigurable Systems. One of the most important challenges brought by current heterogeneous and reconfigurable systems is thermal modeling. As pointed out in Reference [70]: "In traditional high-performance CPUs, the knowledge of the chip floorplan allows to identify, at design time, the hot-spots. However, this is not feasible if the system is equipped with reconfigurable fabrics (e.g., FPGAs) for application-specific acceleration purposes. The thermal behavior will depend on actual accelerators used, which are unknown during the design phase." As a result, the development of thermal and reliability-aware policies comes at the cost of either exploring a large variety of reconfigurable fabrics utilizations, or shifting thermal evaluation from the chip design phase to the end-user application development phase or the run-time system. In this context, thermal simulators as 3D-ICE [155] provide a solid base for runtime monitoring, prediction, and managing of the cooling mechanisms in an HPC context. Similarly, recent proposals point in the direction of mixed design-time/run-time models, enabling proactive thermal and reliability techniques for both conventional multicores and heterogeneous systems [99].

However, considering only the chip might not be sufficient to model the overall system thermal efficiency, especially in terms of HPC. Thus, there is a need to evaluate how the thermal behavior of the chip affects the whole infrastructure. This process starts from the computing node where it belongs, through the rack and runs up to the whole server room including cooling equipment. To this end, one should consider the power and thermal states of the resources in the HPC center hierarchy and their mutual impact to get the full overview of the HPC system at hand. In this context, simulators such as DCworms [106], that allows incorporating thermal and cooling models for the purpose of computing infrastructure simulations, can be applied to study the scalability of proposed solutions and overall performance and energy gains. The authors of References [6, 42, 133] addressed this issue by providing power models for both computing and non-computing equipment. In Reference [40], these models [42] were used to study the energy efficiency of the data centers for different workloads and management policies. The M2DC project [131] also took advantage of thermal models in developed heterogeneous servers with energy-aware fan management and providing them the power capping functionality. In particular, adopting thermal models for predicting CPU temperatures enabled proactive fan management (i.e., smooth changes in fan speed). The motivation is the fact that the proper thermal management does not only reduce energy costs but it also can increase the server's lifetime and performance, including the performance of running applications [29]. The same approach can be applied at the HPC center level as the cooling efficiency factors, like the coefficient of performance, are highly dependent on the measured temperatures. It must be noted that reliability and efficiency require—sometimes—opposite measures at the HPC center level. For example, higher room temperature leads to lower cooling costs allowing even free cooling scenarios, especially for Direct Liquid Cooling (DLC) that enables high temperature of coolant. However, this approach may have negative effect on reliability both at the chip level (degradation caused by high temperature operation) and the whole HPC system

(shorter operation time in case of cooling system failure). Thus, a trade-off between reliability and efficiency must be carefully balanced in future Exascale systems.

> Observation 2. *Thermal modelling is no longer a chip design phase concern to be evaluated just once. Instead, the use of reconfigurable fabrics calls for new approaches based on the run-time thermal models to optimize cooling solutions and application performance. A careful trade-off between reliability and efficiency must be found using thermal/cooling models for whole HPC/datacenter systems.*

2.1.4 Timing Requirements in HPC. Focusing on timing requirements, in recent years it is possible to notice an increasing trend of emerging HPC applications that need strict timing requirements, that are typical of embedded systems. In fact, soft real-time guarantees, intended as average Quality of Service (QoS) control, may be insufficient for certain classes of applications, such as natural disaster prediction algorithms, medical software, and real-time video transcoding tools [69, 135]. However, existing timing analysis tools delivering a certain degree of guarantees are intended for relatively simple systems and execution scenarios with limited parallelism [2, 179]. Instead, HPC platforms build upon high-performance CPUs able to run tens of threads simultaneously with unobvious interactions among them, thus challenging current practice for (worst-case) timing analysis [143].

> Observation 3. *Techniques for the estimation of the Worst-Case Execution Time (WCET) used in the embedded domain, where platforms and execution models are simple, need to be scaled up to the challenge of executing parallel applications on large-scale high-performance systems due to the increasing need for timing guarantees in HPC.*

2.1.5 Network Capabilities. To accommodate current HPC workloads, data center interconnects need to be "scalable, efficient, fault tolerant and easy-to-manage" [76]. A probe of this fact lays out in the number of architectures proposed in literature to improve scalability and performance in data center networks [5, 44, 79, 80, 113]. However, the issue of reliability has not been addressed to the same extent and the majority of studies have focused on understanding the impact of network failures on system operation [76, 100, 105]. These works collect events from network logs with the goal of analyzing and characterizing different types of interconnect faults and errors. However, the analysis of recorded data is still challenging due to several aspects [76]. The most remarkable issue is the inability to quantify precisely the impact of these types of failures on the running applications.

> Observation 4. *Network fault models are not mature enough to be used in fault-prediction techniques. On the one hand, a deeper understanding and characterization of interconnect failures is required. On the other hand, they need to be correlated with the application impact to close the existing gap.*

2.2 Predominant Fault Sources in HPC

For classification, we will distribute fault sources in two categories: *internal* and *external* faults. Internal faults are caused within the system itself (and its components). External faults are caused by the interaction of the system with the environment and/or surrounding systems.

Table 1. Internal Sources of Variability, Leading to Circuit Degradation and/or Faults

Proximity	Spatial	Temporal	
		Reversible	Irreversible
Inter-Die Variation	Parametric: Gate Length (Lg), Threshold Voltage (Vth), Oxide Thickness (tox)	Operating temperature, Activity factor	Hot-electron Effect, BTI shifts
Intra-Die Variation	Pattern Density/Layout induced	On-Die Hotspots	Hot-spot enhanced BTI
Device-to-Device Variation	Atomistic Dopant Variation, Line-Edge Roughness, Paramter Standard Deviatons	SOI (Silicon On Insulator) Body History, Self-Heating	BTI induced Vth shift, Time-Dependant Dielectric Breakdown (TDDB)

Fault caused by external sources originate in: (1) noisy input signals (among those power is the biggest concern) and external radiation; (2) operating ambient temperature (extreme temperature operations in uncontrolled environments); and (3) particle strikes (alpha particles from package decay, cosmic rays creating energetic neutrons and protons, and thermal neutrons). In an HPC system, noisy input signals (1) and, especially, power noise can be managed by extra power regulators on the board. On the other side, ambient temperature (2) is controlled in the long run. Yet, particular and localized temperature fluctuations can induce faults. Finally, particle strikes are the main external concern during operation. As pointed out in Section 1.1, fault rates caused by particle strikes produce a significant reduction of uptime and MTTF.

Internal faults are caused by mismatches in the manufacturing process and/or the degradation of the circuitry during its lifetime. Table 1 describes these sources of variability in the circuitry. The sources are classified according to three criteria:

- *Proximity*: inter-die (between different dies), intra-die (within a given die), and device-to-device (transistor to transistor).
- *Spatial*: affecting the dimensions or material density (time independent).
- *Temporal*: causing degradation when negative situations arise (time dependent).

Internal faults are present in the chips used in HPC systems. As most spatial internal sources follow a statistical distribution, manufacturers select the less affected chips for their high-end products. Such solutions are able to operate reliably at higher operating frequencies during the binning process prior to sell them. Those components are, therefore, sold at higher prices. Most HPC systems are built with these high-end products. While the effects of these sources of faults may be lower than in ordinary chips, as technology scales, distributions widen, so all chips now have extra features to ensure a functioning device (and a high yield), despite spatial internal sources of faults.

Temporal sources involve certain operation conditions that favor the appearance of faults. In this sense, (high) temperature as well as temperature transients trigger all temporal sources. Moreover, parameter variations manifesting as faults may be exacerbated at high temperatures. Consequently, it is of great interest to study these phenomena. Next, we describe more in detail the effect of the temperature on the system reliability.

2.2.1 Thermal Gradients and Thermal Cycling. Thermal stress is a rapid change (in time or space) in temperature. Thermal stress degrades the MTTF of the system [112]. Reducing hot spots is not enough to achieve an adequate thermal management of high-performance CPUs

and increase MTTF [98]. In the following paragraphs, we will briefly describe the thermal issues caused by temporal or spatial gradients and thermal cycling.

Temporal Temperature Gradient (TTG) can be defined as the rate that temperature changes over time. Given a point in time, the spatial temperature gradient (STG) is the temperature difference between two points in the circuit. Both STG and TTG pose a critical impact on the system lifetime reliability [38]. Yet, note that STG is mostly affected by power and thermal throttling applied at the processor level, i.e., the allocation of work and specific setup of all cores in the system need to be taken into consideration. In contrast, TTG is mostly dependent on the operating frequency and the workload characteristics of each core.

Finally, *Thermal Cycling (TC)* is the phenomenon of regularly increasing and decreasing temperature. [180]. Thermal cycling can be measured through Downing's rainflow-counting algorithms [64]. MTTF reduction due to thermal cycling occurs due to the mismatch on the expansion coefficient between the layers of the chip, which results in thermo-mechanical stresses. Thermal cycling reduces the MTTF as the number of cycles or the amplitude of the cycles increase. Large amplitudes are usually caused by the co-scheduling of very different thermal profile applications on a single core. Power saving techniques as DVFS or turning on and off cores can increase the number of thermal cycles [38].

Observation 5. *The dynamic behavior of temperature in terms of temporal and spatial gradients, as well as thermal cycles, has a direct impact on the MTTF of HPC processors. Therefore, processor configuration and utilization can no longer be unaware of those thermal concerns. Suitable techniques must be devised to leverage such issues along with other concerns, namely, performance, power, hot-spot management, and the like.*

3 FAULT-PREDICTION TECHNIQUES FOR HPC

A number of fault prediction mechanisms and analytical methods for estimating application's robustness have been proposed in the literature. Predicting faults, rather than detecting them, provides some additional time to react to recover from the fault (or even avoid it at all). Estimating application's robustness based on fault statistics and effective usage of resources minimizes application crashes and helps determine optimal resource utilization. This information can be exposed to the software orchestrator to drive efficiently the different recovery mechanisms and the utilization of the system to maximize resource efficiency.

In our analysis, we follow the taxonomy introduced in Reference [145], but we only keep those categories that apply to the problem at hand. For the sake of completeness, apart from the very few works targeting HPC systems, we include relevant works that could be applied to HPC systems.

3.1 Techniques Based on Failure Tracking

These techniques build upon the idea that past failures can be used to predict future failures. Therefore, one of the main limitations of this type of technique is that failures must have occurred to be able to predict future ones. Hence, while those techniques can be appropriate for failures due to transient and intermittent faults, they lack the ability to prevent permanent faults. In particular, the latter relates to the fact that, once a failure caused by a permanent fault has manifested, any corrective action may help preventing further failures due to such fault, but cannot do anything to mitigate the fault, since it is already permanent.

Some works—not specific for HPC systems—have analyzed statistical relationships and probability distributions of the time-between-failures [137]. These techniques may also be used in the

context of HPC systems, since their statistical nature makes them agnostic to the source of the failure data. Such an approach is, indeed, investigated in Reference [20], where failure prediction is performed based on the probabilistic analysis of job failures in an HPC system.

Other works, instead of looking for probability distributions, build upon dependencies and correlations to predict failures based on the occurrence of other failures [57, 58, 71, 116, 167]. In particular, Reference [167] notes that failures can occur either close in time or close in space. In general, close in time failures may relate to a single fault (e.g., a permanent fault or uncorrected transient fault) that leads to multiple failures, whereas close in space failures may relate to a broader set of conditions, such as a single fault that manifests in several components using the faulty (shared) component, or as multiple faults whose occurrence is not independent (e.g., due to high aging of an overused set of resources). Such ideas have been used by Reference [116] to predict failures of the IBM's BlueGene/L system. In particular, authors build upon event logs with reliability, availability, and serviceability (RAS) information to analyze whether some patterns exist in terms of time occurrence or space occurrence of failures. Then, upon the detection of a failure, if a positive space or time correlation has been found for that fault, then related faults are assumed to occur in the near future either in the same component (time dependence) or in neighbor components (space dependence). This work was extended for IBM's BlueGene/Q system [57, 58] for fatal system events. These concepts have also been applied to distributed systems, thus being of relevance for HPC systems [71].

Observation 6. *Since fault rates are expected to increase in future HPC systems, there will be more room to learn from already occurred faults, errors and failures. Indeed, rather than building upon failures, appropriate fault and error monitoring may allow developing these fault prediction techniques without requiring the occurrence of failures.*

3.2 Techniques Based on Symptom Monitoring

Symptom-based prediction builds upon system state information to predict whether a failure may occur in the future. Unlike failure tracking techniques, those based on symptom monitoring do not need any failure to occur in the system to predict the occurrence of future failures. Hence, they do not suffer from the same limitation as other techniques where failures need to occur to predict future failures, which plays against preventing permanent faults. Symptom monitoring may allow predicting failures due to future permanent faults, thus taking corrective actions before those faults actually occur, hence avoiding those faults. However, since correlation between symptoms and future failures may be weaker than the correlation between multiple failures, symptom monitoring techniques may have higher chances of raising false positives (i.e., predicting a failure that would not occur) and false negatives (i.e., failing to predict a future failure).

Most techniques do not target failures due to (electronics-related) faults but, instead, software concerns due to memory leaks, system performance degradation, and resource utilization that may lead to functional failures and decreased performance. For instance, function approximation has been used to estimate when performance of some servers may degrade due to serving large amounts of requests [8]. Machine learning has also been a popular approach for predicting failures based on symptom monitoring. Machine learning was already used successfully to predict failures of mechanical components in the early 1990s [170]. However, it has been used in a plethora of works targeting server-type and telecommunications systems [71, 86, 145]. In this context, Hoffmann et al. [87] showed that the selection of appropriate input variables for machine learning is the most relevant concern to maximize the accuracy of the approach.

A different approach within symptom monitoring consists of training a classifier with data related to systems prone to failure and systems that are not, using some variables of the system. Then, during operation, those variables are monitored and assessed by the classifier algorithm, which, based on the current state of the system, decides whether it matches better a failure-prone or a failure-free condition. This concept has been applied with discrete variables [81], with continuous variables [126] and with support vector machines [72, 171] for disk and server failure prediction, thus being of relevance for HPC systems. Such techniques are not restricted to specific components. Moreover, it has been shown that, since some of these techniques do not provide binary answers (failure versus non-failure) but continuous values, their outputs can be used to monitor slowly evolving system states that get closer to failure [15]. Similar approaches have also been used for fault classification across transient or permanent faults [138]. While this is not a failure prediction scheme per se, fault classification can be used to feed failure prediction, since transient faults can be eliminated, whereas permanent ones remain and may likely lead to failures in the future.

While symptom monitoring needs training data, other approaches build upon system models determining the range of values expected for multiple variables during failure-free operation. Hence, no training phase is needed, in general, for these approaches. During operation, the set of variables used for failure prediction are assessed against the system model to determine whether values are within failure-free ranges. If this is not the case, then a failure is predicted. This approach has been applied to hard disk failure prediction [91, 126] using models that use data during failure-free operation to predict failures. Similar implementations based on matrix representation of the variables monitored and residual deviations with respect to failure-free behavior could also be used for failure prediction in HPC systems [151]. Alternative models have been used either based on statistical distributions of the variables assessed (e.g., mean and variance) to predict failures [177] or defining grammars of failure-free sequences of values [33]. These models can be generally applied to failure prediction in HPC systems by monitoring appropriate variables of the system. Some other models build upon component utilization and interaction within a system to compare the set of components used against the different sets used during failure-free operation [34, 103]. At processor level, some authors show that specific variables (e.g., mispredicted branches, cache misses) vary noticeably upon the occurrence of a fault, so they can be used to determine whether a fault has occurred [128]. While authors do not use this technique for failure prediction, it could be used together with other techniques that, for instance, relate error frequency with failure chances, as discussed later. Those approaches, although not used explicitly for HPC systems, could be taken into account.

A number of techniques for failure prediction, build upon past history of monitored variables to predict their future values and, consequently, whether those values will fall into a range corresponding to a failure. Such prediction can be performed with different means:

- Regression: A function is adjusted to the data and future values used for prediction.
- Residual computation: A residual value of the measurements is computed and used for prediction.
- Time series prediction: Both stationary and non-stationary time series are used to predict future values.
- Signal processing techniques: Noise is removed from measurements for a better prediction.

In general, these techniques have not been used for failure prediction due to (electronics) faults, but their different incarnations can be applied to the problem at hand. For instance, regression-based methods have been used to predict time-to-exhaustion of a given resource [75]. A similar approach could be applied to failure prediction if appropriate variables to monitor are identified. Residuals for fractality and time series of Hölder exponents have also been used to

predict resource exhaustion [148]. Time series have been used to predict whether values will violate a threshold [83].

Observation 7. *While transient faults can be tolerated and, hence, fault prediction can build on past transient faults to predict future ones, permanent faults cannot be removed. Hence, symptom-based fault prediction is particularly useful to this aim. Since temperature is highly related with aging and thus with permanent faults, there is a huge potential to adopt symptom-based fault prediction techniques, already used in other domains, to predict permanent hardware faults and prevent the occurrence of unrecoverable faults.*

3.3 Techniques Based on Error Reports

These techniques build upon error events (i.e., error logs) to predict future failures. Unlike previous categories, these techniques neither need actual failures to have occurred, nor monitor specific variables periodically. Instead, error reports are monitored and decisions taken on an event-related basis.

Some authors use genetic algorithms to identify the rules to predict failures based on error reports [178], whereas others build upon identifying specific sequences of errors occurring before failures to anticipate those failures [173]. Fault trees and Markov Bayesian Networks have also been suggested as potential methods on which to build failure prediction solutions [145].

As for the case of occurred failures, some techniques aim at identifying dependencies and correlations between errors, either in time or in space, to predict failures. In particular, an observation common across multiple works is that the number of errors per time unit increases before a failure [114]. This observation has been corroborated in several works. For instance, in the IBM BlueGene/L supercomputer, a job experiencing two non-fatal events has a higher chance to experience a failure (above 5×) than if it only experiences one [116]. Increased error frequency has been the basis for several failure prediction methods. Some authors rely on changes in the distribution of error types to predict failures [150], whereas others study the error frequency and, if such frequency increases, an imminent failure is predicted [107, 129]. Error frequency has also been used, not only to predict failures, but to classify faults and failures as either transient or permanent [1, 117].

Beyond error frequency, some works also consider whether some patterns exist in the sequence of errors prior to a failure. In particular, error types and times are exploited for pattern identification [117, 146, 172]. In the case of patterns, a specific technique has been applied to HPC systems, building upon techniques from the signal processing domain [73]. Such technique is proven efficient to schedule checkpoints in failure-prone locations and to migrate tasks away from those nodes.

In this context, some authors investigate how to monitor error logs in distributed HPC systems and deliver information appropriately to software layers to build failure prediction mechanisms on top [140].

Observation 8. *While techniques based on error reports do not need those errors to become failures to use them, faults need to occur in both cases. Hence, limitations related to the occurrence of permanent faults are also a concern for this type of technique. However, expected increasing error rates enable the development of more accurate fault-prediction mechanisms due to the increased amount of information in the error logs.*

3.4 Timing Analysis in HPC

Dealing with application timing constraints in the HPC scenario is extremely challenging, due to the unpredictability of hardware and software layers, usually composed of Commercial-Off-The-Shelf (COTS) components that make the tasks timing analysis hard or even impossible [45]. Timing constraints are usually considered in the soft real-time sense in HPC. Several research works and tools to deal with the resource management problem have been developed in recent years and they are thoroughly reviewed in literature surveys [94, 152]. However, the number of works considering strict timing constraints for HPC is far much lower. In fact, even if hard real-time has been widely studied in recent decades for parallel architectures [51], the applicability of such techniques in HPC environments is limited—if at all possible—due to the previously discussed unpredictability challenges. Although some recent works on HPC timing predictability exist [61, 135], this problem is still open, and several challenges have to be tackled, since existing solutions are not general enough yet. In this context, probabilistic timing analysis that aims at reducing the amount of information needed for the timing analysis and relieves end users from having to exercise too much control on their system under analysis [27], offers an interesting venue to develop solutions for HPC systems.

Observation 9. *The emergence of HPC applications with strict real-time needs calls for the development of suitable WCET estimation techniques to guarantee with sufficient confidence that timing faults cannot occur. Building upon methods that allow modeling the system as a black box (or as close as possible to a black box) is a promising path to follow due to the portability and scalability of those approaches.*

3.5 Summary

As shown, there is a plethora of techniques that can be used for fault, error, and failure prediction. Most of them have not been applied to the particular case of HPC systems or do not target (electronics) fault-related failures. However, the number of possibilities to develop failure prediction techniques for HPC systems is huge, but appropriate techniques need to be devised and proven effective.

4 ERROR DETECTION AND RECOVERY IN HPC SYSTEMS

Literature on error detection and recovery is abundant, and many techniques are general enough to be applied to both HPC and non-HPC systems. Next, we review the most relevant techniques for the problem at hand, at hardware, application, and system level.

4.1 Fundamental Hardware Monitoring

Reliability, availability, and serviceability (RAS) is a term used in computer systems to include the design and implementation of appropriate means to ensure system reliability, high availability and serviceability. While other related concepts have been often considered in computer systems, such as security and maintainability, the term RAS is often used (and abused) to refer to the original three concepts, as well as to some others.

The term RAS was originally coined by IBM to refer to the robustness of their mainframe computers [88]. Nowadays, not only IBM provides support for RAS, but virtually all hardware vendors in the HPC domain provide support for it, including Intel [96], AMD [124], and ARM [9]. RAS support includes a number of interdependent features. The most common ones are as follows:

- The initial Machine Check Architecture capabilities. Some tests are performed to validate that hardware operates normally and without errors. For instance, in the case of memories, usual March tests are passed to validate that no permanent fault is in place [21]. Those tests consist of writing specific data patterns intended to trigger different fault types.
- Processor instruction error detection. For example, residue codes for data operated, and valid opcode checking are examples of error detection means in this category.
- Parity or ECC errors in caches, system memory, and memory bus have been shown effective to capture faults leading to bitflips, such as those caused by radiation, temperature disturbances, aging, and crosstalk.
- Input/Output: Checksums (like CRC) for data transmission and storage, with a nature similar to that of parity and ECC.
- Storage: Checkpointing or journaling file systems for file repair after crashes.
- Background scrubbing. This solution is often employed to ensure that single-event upsets (SEUs) are detected and corrected timely before multiple SEUs accumulate, thus becoming unrecoverable.
- Power/cooling: violation of nominal operating frequency, voltage, temperature, power envelope. For example, processors are usually halted on a temperature overrun to protect the physical integrity of the chip.

RAS support includes not only specific hardware support but also Operating System (OS) support to monitor errors (even if recovered by hardware means), and configure the system and trigger recovery actions if needed. For instance, Linux-based machines include the `mce-log` daemon to track RAS-related information by interacting periodically with the corresponding RAS hardware support. Similarly, the Windows Hardware Error Architecture (WHEA) performs a similar work for Windows machines. In both cases, the OS can trigger protective and/or remedial actions when necessary based on the predictive failure analysis (PFA) performed.

Nowadays, computers have many RAS features that guarantee a sustained (high) availability of the system. This relates to the fact that individual processors may be designed with a specific (very low) Failure in Time (FIT) rate.[1] For instance, a processor may have 200 FIT, so that, on average, a failure is expected every 5,000,000 hours (i.e., every 571 years). However, if we set up 100,000 such processors working cooperatively in a supercomputer or data center, then we can expect a failure in any of the processors every 50 hours (i.e., every 2 days), which may be unacceptable for applications lasting several days. Note that in those large systems, other components such as interconnects, memories, and so on, will also contribute to the overall FIT rate of the system.

Observation 10. *While RAS support will still be needed in future HPC systems to cope with most of the errors, the increased fault rates per processor together with potentially higher processor counts per HPC system will lead to much higher error rates, thus exceeding the capabilities of RAS support to keep the HPC system available. Hence, RAS support on its own will not scale up to the challenge.*

4.2 Application-level Fault Detection and Recovery

4.2.1 Checkpoint/Restore. Checkpointing has been often used as a means for efficient fault recovery. A checkpoint consists of a snapshot of the execution at a certain point in time, which can be used to resume the execution from that point without starting over. In general, a checkpoint must reflect the architectural state of the application at a given instant, thus

[1]The FIT rate is defined as the number of failures expected per 10^9 hours of operation.

including its architectural registers and memory state. Unfortunately, checkpoints introduce some non-negligible overheads in terms of timing and storage, since saving the state requires freezing execution and storing potentially large amounts of data, with crucial implications for exascale applications [47]. Because of the above overheads, several solutions support incremental checkpointing (i.e., only new or modified data, since last checkpoint is stored [77]), or rely on a hierarchical and multi-level checkpoint organization for improved scalability [48, 123]. Some solutions take an application-specific approach for optimizing the checkpointing performance (e.g., family of solvers [36]) or at the library level [123, 158].

Due to the tradeoff between checkpointing frequency and fault rate, a number of works aim at identifying optimal checkpoint intervals [19, 46, 48, 53, 55, 59]. Interestingly, checkpointing operations may even have significant impacts on energy consumption, calling for specific optimization techniques aimed at the combined optimization of checkpoint rate and performance/energy efficiency, addressing both traditional checkpointing and multilevel solutions [46]. Recent contributions suggest that resource management for exascale systems should expressly support the selection of the optimal checkpointing strategy, depending on each application's execution characteristics, as well as scheduling decisions that are resilience-aware and make use of accurate time predictions [49]. For example, the work in Reference [48] addresses the relationship between system failure rates, checkpoint/restart overheads, and checkpoint intervals and develops a prediction model for finding the optimal time duration between successive checkpoints. Finally, checkpointing can be performed hierarchicaly. In References [53, 55], authors consider multi-level checkpointing as a baseline and they provide insight on how to optimize the selection of checkpoint levels based on failure distributions observed in a system, and how to compute the optimal checkpoint intervals for each of the checkpointing levels. Later on, in Reference [59], authors propose a family of hierarchical mechanisms consisting of two checkpointing levels: level 1 deals with errors with low checkpoint/recovery overheads such as transient memory errors, while checkpoint level 2 deals with hardware crashes such as node failures.

The complications brought by checkpointing techniques are exacerbated by the massive adoption of accelerators in HPC, particularly GPUs. In fact, as more and more HPC workloads rely on accelerators, an increasingly large part of the application execution state reside outside the host processor and memory. This offloading of computation poses new issues related to both the access to the execution state and the particular reliability characteristics of acceleration devices. For example, GPUs tend to have higher DUEs per GB than CPUs [74, 156, 169] and GPUs may come with large memory ports (e.g., 128 bit for High-Bandwidth Memory 2 technologies) as well as reduced correction capabilities [132]. As an example, the work in Reference [60] shows that the DUE rate per GB for GDDR5 memory in NVIDIA Kepler GPUs can be as high as five times the DUE rate of CPU memory equipped with state-of-the-art error checking and correction support. The availability of effective and scalable checkpointing techniques for accelerators is thus essential for emerging exascale systems. Initial contributions [130, 149, 166] do not support features that are normally available in recent devices, such as NVIDIA Unified Virtual Addressing (UVA), as pointed out in Reference [74]. The decoupled CPU-GPU architecture poses additional technical challenges for effective application-wide checkpointing. Various works rely on a proxy-based architecture, including CRCUDA [164] and CheCL [165], which targets the OpenCL nonproprietary programming model [160]. OpenCL support is also featured by VOCL-FT [136], which provides efficient soft error recovery capabilities while reducing data exchanges between the device and the host as well as disk traffic. Other works, introduce some form of GPU checkpointing like HiAL-Ckpt [181], HeteroCheckpoint [102], and cudaCR [139], taking an application-specific approach to provide GPU-side checkpointing. Last, the CRUM framework presented in Reference [74], which also relies on a proxy-based approach along with new shadow page synchronization mechanisms,

directly addresses the support for CUDA's unified virtual memory (UVM) available in the latest device generations, enabling fast asynchronous checkpointing for large-memory CUDA UVM applications and significantly reducing checkpointing overheads. While all the above contributions address GPU devices, FPGAs have emerged during recent years as an alternative for dedicated acceleration matching a few specific types of HPC workloads. For this type of accelerator, checkpointing is a nearly unexplored area, outside embedded systems [147, 185].

Observation 11. *Increasing error rates in future HPC systems will lead to increasing checkpoint costs. Therefore, there is a need for effective fault prediction, also extended to heterogeneous accelerators, to prevent errors occurrence while containing the cost of checkpointing. Further reducing the cost of checkpointing opportunistically will also allow tolerating increasing error rates.*

4.2.2 Algebraic- and Data-based Detection and Recovery. On a different strand, some works build upon the algebraic properties of the algorithms being executed to extend them for fault recovery. In particular, solutions build upon mathematical relationships, adding software redundancy and/or data interpolation to recover from faults without needing to store checkpoints and, instead, using the data of fault-free threads to recover the data for the faulty one [3, 4, 35, 108]. Algebraic properties have also been used for error detection for algorithms such as CG, Fourier transform, QR and LU factorizations, and matrix multiplication among others [17, 36, 50, 85, 89, 115].

Recently, other authors started exploring data-based approaches such as linear prediction methods or machine learning to detect SDCs [54, 56, 161, 162]. Eventually, the key component of any infering mechanism is the feature extraction (or identifying what are the key parameters to monitor). In Reference [56], authors provide a detailed analysis on multiple available features. Several prediction methods have been proposed: linear [54], or quadratic [56]. On the machine learning side, the proposal in Reference [161]—extended in Reference [162]—relies on the end user declaring some state variables for monitoring. The fault detector is trained for the specific program and later, during operation, is able to detect whether the values for those variables are abnormal and, hence, a SDC may have occurred.

Observation 12. *This type of technique needs to address three key challenges in future HPC systems: (1) lack of generality, (2) increasing error rates, and (3) atomicity of some computations on reconfigurable fabric or accelerators. In particular, the latter, apart from being a challenge, also opens the opportunity to configure those fabrics so that they perform computations and check for errors simultaneously with virtually negligible recovery costs.*

4.2.3 n-Modular Redundancy. One of the most used techniques for error detection and recovery consists of using n-modular redundancy, where n refers to the number of redundant copies of the system executed [119]. This scheme is based on the redundant execution of the program and the comparison of the outputs across the redundant instances to detect errors, based on the assumption that a single fault will not lead to errors in multiple instances or, at least, if this was the case, the error would be different in the faulty instances. This would guarantee error detection every time outputs are compared. Correction, instead, may be built in different ways, among which we name the following:

- *Majority voting recovery.* On an error, if $n \geq 3$ and the error probability is low enough, then it is almost guaranteed that only up to 1 instance can be faulty. Hence, there will be a higher number of (identical) correct outputs than the number of faulty outputs. By comparing

outputs and voting, the correct output can be determined. Then, the state of the faulty instance can be replaced by the state of a fault-free one before resuming execution. However, this solution is only valid as long as the number of fault-free outputs is strictly higher than $n/2$. For instance, if $n = 2$, then such a recovery mechanism is not possible.
- *Checkpoint rollback.* On an error detection, execution can be rolled back to the last fault-free checkpoint for all instances, regardless of the value of n and the number of faulty instances.
- *Restart.* An even simpler mechanism to recover consists of simply restarting the faulty task. This solution can be regarded as appropriate as long as tasks are short enough, so that their re-execution does not involve too many redundant computations.

Usual implementations of n-modular redundancy include Triple Modular Redundancy (TMR) and Dual Modular Redundancy (DMR). For instance, the HP NonStop architecture [16] builds upon fully redundant boards whose outputs are compared at the Sphere of Replication (SoR) of the full board, thus detecting errors only when requests are sent out of the board. Other SoR schemes exist comparing the outputs of redundant computations at pipeline stage level, which allows quick detection and recovery by simply reexecuting not-yet committed faulty instructions [153].

An important consideration in n-modular redundant systems is the independence of redundant instances so that a single fault does not lead multiple instances to the same erroneous output, since this would defeat the purpose of redundancy. This concern has been considered in the safety-critical domain (e.g., in the automotive domain [97]) and faults of interest include voltage drops, crosstalk, and so on. The usual solution consists of introducing some form of diversity across redundant instances, which can be attained by different means:

- Using independent devices, e.g., using independent boards with independent power supplies.
- Diverse hardware implementations, e.g., using two different processors, for instance, an Intel and an AMD processor.
- Diverse software implementations, e.g., different implementations of the algorithm or different compilations of the same algorithm.
- Time diversity, e.g., executing redundantly identical binaries on identical hardware, but with some time slack in between so that a fault does not affect the same instructions in redundant instances.

The scheduling of redundant tasks in parallel systems has been an important concern [176] as well as whether to enable only partial redundancy [18, 95], so that it is used only for those nodes or computations more vulnerable to faults. Finally, n-modular redundancy has also been implemented by software means by making programs perform all computations redundantly. Such concept is known as n-version programming [10]. Particular redundant MPI implementations have been evaluated with success [68].

Observation 13. *n-Modular redundancy looses effectiveness as error rates increase, which is the case of future HPC systems. Therefore, these solutions are expected to remain effective as long as fault-prediction techniques can keep error rates low enough.*

4.2.4 Data Representation. Error detection mechanisms, such as n-modular redundancy among others, rely on the comparison of results against those of redundant computations, or against some form of reference value or data check. In theory, these techniques are effective. However, their practical implementation on actual computers with limited data representation may pose some issues. In particular, processors implement finite-precision numbers (e.g., 32-bit

or 64-bit) that naturally fail to cover the spectrum of any number field, such as Integer or Real numbers. Normally, this is not a big concern for integer numbers, since all numbers in a range (e.g., $[-2^{63}, 2^{63} - 1]$) can be represented and hence, as long as the program does not need numbers beyond this range, the actual implementation is accurate with respect to the abstract algorithm. However, in the case of real numbers, limited representation leads to limited precision, which imposes some form of rounding for computations. Hence, rounding can easily bring deviations with respect to the expected (theoretical) result and discrepancies across redundant computations if operations can occur in different order. In particular, the latter concern relates to the fact that the Associative Property does not hold for real numbers with limited precision. In other words, if limited precision is used for real numbers, in general, we have that

$$(A + B) + C \neq A + (B + C).$$

Therefore, either it is guaranteed that the same computations are performed strictly in the same order across redundant executions (or in an appropriate order for comparison against a golden reference), or some degree of tolerance is allowed in the comparison so that small discrepancies potentially caused by rounding effects do not alter the result of the comparison. Note that the latter may allow some error tolerance if errors do not cause deviations larger than those already introduced by rounding effects.

Observation 14. *The increasing use of reconfigurable fabrics and accelerators in future HPC systems must also be aware of data representation problems, since different implementations can help mitigate this problem or exacerbate it.*

4.2.5 Non-determinism. At a different abstraction level, we find that some algorithms may be intrinsically non-deterministic, thus challenging error detection, since a single correct result may not exist. For instance, algorithms based on pseudo-random search and/or optimization, such as those based on genetic algorithms or simulated annealing, may take different choices based on both different (random) initial values and different (random) choices during algorithm execution. For instance, a genetic algorithm may pair individuals randomly, choose points for crossover of individuals randomly and apply mutation of some genes randomly. Simply modifying the random seed of the pseudo-random number generator (PRNG) or performing different actions calling the PRNG in a different order, may lead to different random choices and, thus, different results. Hence, determining whether a partial or final result is fault-free is particularly challenging for non-deterministic algorithms regardless of whether *n*-modular redundancy is used or not.

Observation 15. *Semantic error checks based on assessing whether specific properties hold for partial results may open the door to tolerate not only non-determinism but also data representation variations and even some errors. Many algorithms whose output quality does not depend on specific values but, globally, on all data, may greatly benefit from this type of approach in future HPC systems with increased error rates. Thus, research on this topic becomes highly relevant.*

4.3 System-level Solutions

4.3.1 Task Migration. **Task migration** is a recovery action that can be used as an alternative or as a support to Checkpoint/Restore (C/R). It consists of moving the code of a running task among processing resources, as well as moving the allocated memory pages among different memory nodes.

The operating system or the resource managers can operate both in *reactive* and *proactive* mode. In the former case, the task migration would consist of changing the set of resources allocated to the given task, before relaunching it or performing a rollback. In the latter instead, the task migration would be performed whenever a prediction of fault has been provided. In such a case, the OS or the resource manager would check if the affected hardware is currently used to run some tasks, and if so, change the resource allocation preventing such tasks from experiencing the expected faults.

Task migration can be exploited in the Restore phase of a C/R protocol, thus resuming the execution of the target application or tasks on a different set of computational resources. Depending on the scope under which task migration is performed, in HPC we can distinguish among:

- Inter-node task migration
- Intra-node task migration

The *Inter-node migration* consists of moving the tasks (or the entire application) from one computational node to another. If the application has been implemented by using the Message Passing Interface (MPI) programming model, then this may typically require the movement of MPI processes (*Process migration*) among nodes. Some authors proposed an extension of the OpenMPI runtime to perform this in a transparent manner [144].

In large clusters, process migration is therefore useful to add reliability and to balance the resource allocation across the cluster [94]. The migration request can be managed by a centralized entity (e.g., by a global resource manager) or by the single node (a local resource manager), that, for instance, may require processes to migrate in case of overload or a predicted fault. In this regard, whatever the entity in charge of triggering the migration, we must take into account the considerable cost due to the interruption, the migration of code and data to another node and the restore procedure.

In 1996, the *Cocheck* environment was proposed, being the first migration mechanism implemented in MPI [159]. This environment was built on top of the MPI framework and not inside (actually small modifications to MPI framework were applied). The global consistent state is achieved by imposing no message in-flight over the network. Then, checkpoint or a migration of a subset of processes is performed according to what is required.

Process migration can also be used to support classical C/R approaches, as presented in Reference [175]. The technique minimizes the number of checkpoints. To do so, the system is extended with a health monitor for each computing node. In case of imminent fault prediction, all running processes are migrated to another node.

For *Intra-node migration*, things are much simpler, since the resources are typically under the control of a single instance of the OS. Moreover, the overheads are much lower with respect to the inter-node case. In the scope of the single node case, a reliability-oriented resource management policy can exploit the isolation mechanisms provided by the OS (e.g., Linux control groups or containers) to implement this recovery action.

> Observation 16. *Increasing fault rates in future HPC, as well as thermal considerations—which ultimately impact also fault rates—pose additional requirements and constraints on task migration for future HPC systems.*

4.3.2 Heterogeneous Task Migration. Task migration, in the context of heterogeneous hardware, poses additional challenges that do not exist when operating on homogeneous resources, where any task can be potentially migrated to any hardware context. Instead, migration on heterogeneous platforms is much more complex if it needs to occur across heterogeneous computing resources [94].

In 1998, the Tui System [154], an experimental framework to perform process migration between heterogeneous machines, was introduced. The article was well received by the scientific community as it shows several issues affecting heterogeneous migration. The main problem, in fact, is given by the conversion between different ISAs, that may require different instructions, register numbers, register size, and so on. For this reason, the compiler is necessarily involved, because it must produce a code that matches one-to-one between different architectures. As a consequence, to simplify the problem, the Tui System introduced strong constraints regarding the two architectures involved, which leads to a migration looking like *quasi-homogeneous* rather than heterogeneous.

Other solutions proposed an Open MPI middleware to provide migration mechanisms in heterogeneous systems [24]. In this case, the process is not directly migrated but a new process is started in the destination machine. Unfortunately, this middleware provides dedicated MPI calls, violating the standard and requiring substantial rewriting of all MPI applications.

Observation 17. *Heterogeneity expected in future HPC systems, with reconfigurable fabrics and other accelerators (e.g., GPUs), challenges existing task migration solutions for these environments. Thus, transparent and efficient task migration solutions are needed, to react timely if faults are predicted to occur soon. This must be achieved while preserving application timing constraints as much as possible.*

4.3.3 Power and Thermal Aware Resource Management. Peak temperature control through power management [141] was one of the first attempts to enable temperature control in a system through available tools. However, even if power management techniques can have an influence on the thermal hot spots across the chip, these techniques are nowadays insufficient to deal with hot spots. Novel policies centered on thermal behavior have appeared for both design-time [31] and run-time [125].

Thermal management strategies may exhibit conflicting goals between peak temperature reduction and thermal stress reduction [37]. Yet, they do not consider power management or thermal cycling. In addition, Reference [182] proposes task scheduling methods for reducing temporal temperature gradients but disregards thermal cycling and spatial gradient.

Holistic policies, such as Reference [98], are able to manage efficiently all thermal reliability aspects and pave the way to collaborative hardware (e.g., DVFS) software (e.g., workload allocation and application configuration) techniques to enhance the reliability of the system while providing the adequate power/performance/QoS. However, there is still a lack of research works that tackle efficient thermal management not only with the goal of controlling current thermal emergencies but also preventing the consequences of thermal stress, thereby focusing on long-term reliability both at the CPU and the overall server level.

Observation 18. *The increasing importance of thermal concerns, which impact reliability, in future HPC systems, call for appropriate task scheduling methods that consider all concerns holistically. Those approaches that mitigate aging rather than simply predicting an imminent fault are of particular interest to minimize already high fault rates expected.*

4.4 Network-level Solutions

Redundancy is the main characteristic provided by any reliable network that has been shown absolutely useful to recover from transient and permanent failures [7, 62]. In this context different spatial and temporal redundancy techniques have been applied [121]. On the one hand, spatial

redundancy techniques replicate components or data in the system, e.g, error-correction codes and transmission over multiple paths through the network, adopting in some cases 1:1 redundancy schemes [7]. On the other hand, temporal redundancy techniques consist of implementing sliding window protocols, as Transmission Control Protocol (TCP). Link Control Blocks (LCBs) is another example of implementing sliding window protocols to provide reliable delivery of network traffic at hardware level, used in Gemini Interconnect [7], among others.

Observation 19. *As future HPC systems grow in size, mechanisms that dynamically re-route packets when a device or link is unreliable are very expensive, hard to design/verify, and hard to manage. Therefore, models for analyzing the impact of redundancy schemes on overall system performance must be revised taking into account not only performance but also cost.*

4.5 Programming Models and Runtime Managers

Several programming models include now resilience support. In Reference [82], authors provide a detailed analysis of the resilience features of the different programing languages, grouped by paradigm: message passing (e.g., MPI-ULFM [118]), partitioned global address space (e.g., UPC++ [12]), asynchronous partitioned global address space (e.g., X10 [32]), actor (e.g., Erlang [174]), dataflow (e.g., Legion [13]). Table 2 summarizes the comparison developed in Reference [82]. The insertion of resilience (fault tolerance) features is clearly visible. Beyond its original performance oriented purpose, at this moment, the programming models and the respective runtime managers include some of the features listed in the previous sections (i.e., task migration, energy/power/thermal resource management, checkpoint/recovery, reliability monitoring). Similarly, standalone resource managers such as SLURM [184], PBS [84], or Cobalt [52] (the three cover—at this moment—the top five HPC systems) are also progressing in this direction.

Observation 20. *As future HPC systems grow in size, runtime managers—especially—and programming models—in cooperation—need to be an active part of the resilience stack and contribute towards reliability prediction, error detection and recovery.*

5 CONCLUSIONS

Exascale systems will suffer from high fault-rates. This projection, coupled with the fact that it is not possible to recover from all faults—once they happen—asks for effective ways of maximizing applications survivability and, consequently, making the system more efficient and predictable.

Given the reliability needs shown in Section 1.2 and the current state of the art, we need to explore fault prediction mechanisms and analytical methods for estimating application's robustness. Predicting faults will provide the time to react to recover from the fault and will allow error detection and correction mechanisms to scale properly. Statistical and machine learning techniques will likely have prominent importance to predict faults and leverage application and run-time layers. Estimating application's robustness based on fault statistics and effective usage of resources will minimize application crashes and help determining optimal resource utilization. This information can be exposed to both the local and the global resource managers to drive efficiently the different recovery mechanisms (including checkpointing), the proactive reliability policies, and the utilization of the system to maximize resources efficiency.

The co-running applications have a significant impact on the reliability and efficiency of the system. Since they are executed at the same time, they compete for the shared resources.

Table 2. Programming Model Fault Tolerance Features [82]

Distributed System	Adaptability		Fault Tolerance					Performance
	Resource Allocation d=dynamic f=fixed f*=fixed (+spare)	Resource Mapping e=explicit i=implicit	Fault Type h=hard h*=hard (design only) s=soft	Fault Level t=task p=process	Recovery Level u=user s=system	Fault Detection hb=heartbeat ce=comm/err ex=external p=prediction di=data inspect.	Fault Tolerance c/r=checkpoint restart rep=replication mig=migration tr=task restart tx=transaction ab=algorithm-based	Performance Recovery sh=shrinking nsh=non-shrinking glb= global load-balancing spec=speculative exec.
MPI								
MPI-1	f	e	-	-	-	-	-	-
MPI-2/3	d	e	-	-	-	-	-	-
MPICH-V	d	e	h	p	s	hb	c/r	-
FMI	f*	e	h	p	s	ce	c/r	nsh
rMPI	f	e	h	p	s	ex	rep	nsh
RedMPI	f	e	s	p	s	di	rep	-
AMPI	f	e	h	p	s	ce, p	c/r, mig	glb
FT-MPI	d	e	h	p	u	ce	ab	sh/nsh
MPI-ULFM	d	e	h	p	u	hb/ ce	ab	sh/ nsh
FA-MPI	d	e	h/ s	t/p	u	ex	tx	sh/nsh
PGAS								
UPC	f	e	-	-	-	-	-	-
F2008 Coarrays	f	e	-	-	-	-	-	-
F201 8 Coarrays	f	e	h	p	u	ce	ab	-
GASPI	d	e	h	p	u	ce	ab	sh/nsh
GASPI (C/R)	f *	e	h	p	s	ce	c/r	nsh
OpenSHMEM	d	e	h	p	u/s	ce	ab, c/r	sh/nsh
APGAS								
Chapel	f	e	-	-	-	-	-	-
Chapel (prototype)	f	e	h*	p	s	ce/ex	rep	-
X10	f	e	h	p	u	ce	ab	-
X10-FT	d	e	h*	p	s	hb	c/r	nsh
Actor								
Charm++	f		h	p	s	ce	c/r	glb
Charm++ ACR	f *		h/ s	p	s	ce & di	c/r & rep	glb/nsh
Erlang	d	e	h	p	u	ex	ab	sh/nsh
Akka	d		h	p	u	hb	ab, c/r	glb
Orleans	d		h	p	s	hb	c/ r, tx, rep	glb
Dataflow Systems								
OCR	f	i/e	-	-	-	-	-	-
Legion	f	i/e	h*	t / p	s	ex	tr	glb, spec
PaRSEC	f	i/e	s	t	u/s	di	ab, c/ r, tr	-
NabBIT	f		s	t	s	di	tr	glb
Spark	d		h	p	s	hb	c/ r, tr	spec

Understanding how the applications affect each other might help to schedule them more efficiently, especially when timing requirements must be considered.

To achieve these goals, there is a need for combining expertise on thermal and reliability modeling, as well as on reliability-aware workload management techniques.

In this article, we reviewed the main reliability concerns for future HPC systems, and the state-of-the-art predictive solutions for fault mitigation, as well as error detection and correction techniques for HPC systems. As presented, some valuable solutions exist mainly for error detection and correction, whereas predictive reliability and QoS is a less mature area requiring further investigation and elaboration of practical solutions.

REFERENCES

[1] J. Abella, P. Chaparro, X. Vera, J. Carretero, and A. González. 2008. On-line failure detection and confinement in caches. In *Proceedings of the 14th IEEE International On-line Testing Symposium*. 3–9. DOI: https://doi.org/10.1109/IOLTS.2008.15

[2] J. Abella, C. Hernandez, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-askasua, J. Perez, E. Mezzetti, and T. Vardanega. 2015. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *Proceedings of the 10th IEEE International Symposium on Industrial Embedded Systems (SIES'15)*. 1–10. DOI: https://doi.org/10.1109/SIES.2015.7185039

[3] E. Agullo, L. Giraud, A. Guermouche, J. Roman, and M. Zounon. 2013. Towards resilient parallel linear Krylov solvers: Recover-restart strategies. INRIA, Research Report RR-8324.

[4] E. Agullo, L. Giraud, P. Salas, and M. Zounon. 2016. Interpolation-restart strategies for resilient eigensolvers. *SIAM J. Sci. Comput.* 38, 5 (2016), C560–C583. DOI: https://doi.org/10.1137/15M1042115 arXiv:https://doi.org/10.1137/15M1042115

[5] M. Al-Fares, A. Loukissas, and A. Vahdat. 2008. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM Conference on Data Communication (SIGCOMM'08)*. ACM, New York, NY, 63–74. DOI: https://doi.org/10.1145/1402958.1402967

[6] A. M. Al-Qawasmeh, S. Pasricha, A. A. Maciejewski, and H. J. Siegel. 2015. Power and thermal-aware workload allocation in heterogeneous data centers. *IEEE Trans. Comput.* 64, 2 (Feb. 2015), 477–491. DOI: https://doi.org/10.1109/TC.2013.116

[7] R. Alverson, D. Roweth, and L. Kaplan. 2010. The Gemini system interconnect. In *Proceedings of the 18th IEEE Symposium on High Performance Interconnects*. 83–87. DOI: https://doi.org/10.1109/HOTI.2010.23

[8] A. Andrzejak and L. Silva. 2007. Deterministic models of software aging and optimal rejuvenation schedules. In *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management*. 159–168. DOI: https://doi.org/10.1109/INM.2007.374780

[9] ARM. 2017. *ARM Reliability, Availability, and Serviceability (RAS) Specification—ARMv8, for the ARMv8-A Architecture Profile*. White paper. Retrieved from https://developer.arm.com/docs/ddi0587/latest.

[10] A. Avizienis. 1995. *Software Fault Tolerance. Chapter 2: The Methodology of N-Version Programming*. Wiley.

[11] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Depend. Secure Comput.* 1, 1 (Jan. 2004), 11–33. DOI: https://doi.org/10.1109/TDSC.2004.2

[12] J. Bachan, S. B. Baden, S. Hofmeyr, M. Jacquelin, A. Kamil, D. Bonachea, P. H. Hargrove, and H. Ahmed. 2019. UPC++: A high-performance communication framework for asynchronous computation. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'19)*. 963–973.

[13] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken. 2012. Legion: Expressing locality and independence with logical regions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. 1–11.

[14] L. Bautista-Gomez, F. Zyulkyarov, O. Unsal, and S. McIntosh-Smith. 2016. Unprotected computing: A large-scale study of DRAM raw error rate on a supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'16)*. 645–655. DOI: https://doi.org/10.1109/SC.2016.54

[15] H. R. Berenji, J. Ametha, and D. Vengerov. 2003. Inductive learning for fault diagnosis. In *Proceedings of the 12th IEEE International Conference on Fuzzy Systems (FUZZ'03)*, Vol. 1. 726–731 vol.1. DOI: https://doi.org/10.1109/FUZZ.2003.1209453

[16] D. Bernick, B. Bruckert, P. D. Vigna, D. Garcia, R. Jardine, J. Klecka, and J. Smullen. 2005. NonStop/spl reg/ advanced architecture. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'05)*. 12–21. DOI: https://doi.org/10.1109/DSN.2005.70

[17] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello. 2015. Lightweight silent data corruption detection based on runtime data analysis for HPC applications. In *Proceedings of the 24th International Symposium on*

High-Performance Parallel and Distributed Computing (HPDC'15). Association for Computing Machinery, New York, NY, 275–278. DOI: https://doi.org/10.1145/2749246.2749253

[18] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello. 2017. Toward general software level silent data corruption detection for parallel applications. *IEEE Trans. Parallel Distrib. Syst.* 28, 12 (Dec. 2017), 3642–3655. DOI: https://doi.org/10.1109/TPDS.2017.2735971

[19] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien. 2011. Checkpointing strategies for parallel jobs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. ACM, New York, NY, Article 33, 11 pages. DOI: https://doi.org/10.1145/2063384.2063428

[20] J. Brandt, F. Chen, V. De Sapio, A. Gentile, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. Wong. 2010. Quantifying effectiveness of failure prediction and response in HPC systems: Methodology and example. In *Proceedings of the International Conference on Dependable Systems and Networks Workshops (DSN-W'10)*. 2–7. DOI: https://doi.org/10.1109/DSNW.2010.5542629

[21] M.-A. Breuer and A. D. Friedman. 1976. *Diagnosis & Reliable Design of Digital Systems*. Springer.

[22] P. Bridges, K. Ferreira, M. Heroux, and M. Hoemmen. 2012. Fault-tolerant linear solvers via selective reliability. ArXiv e-prints, June 2012. arXiv:1206.1390 [math.NA].

[23] G. Bronevetsky and B. De Supinski. 2008. Soft error vulnerability of iterative linear algebra methods. In *Proceedings of the 22nd Annual International Conference on Supercomputing (ICS'08)*. ACM, New York, NY, 155–164. DOI: https://doi.org/10.1145/1375527.1375552

[24] U. Cabello, J. Rodríguez, A. Meneses, S. Mendoza, and D. Decouchant. 2014. Fault tolerance in heterogeneous multi-cluster systems through a task migration mechanism. In *Proceedings of the 11th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE'14)*. IEEE, 1–7.

[25] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. 2014. Toward exascale resilience: 2014 update. *Supercomput. Front. Innovat.* 1, 1 (2014). http://superfri.org/superfri/article/view/14.

[26] M. Casas, B. de Supinski, G. Bronevetsky, and M. Schulz. 2012. Fault resilience of the algebraic multi-grid solver. In *Proceedings of the 26th ACM International Conference on Supercomputing (ICS'12)*. ACM, New York, NY, 91–100. DOI: https://doi.org/10.1145/2304576.2304590

[27] F. J. Cazorla, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, and T. Vardanega. 2019. Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. *ACM Comput. Surv.* 52, 1, Article 14 (Feb. 2019), 35 pages. DOI: https://doi.org/10.1145/3301283

[28] S. Cha, C. Chen, and L. S. Milor. 2014. System-level estimation of threshold voltage degradation due to NBTI with I/O measurements. In *Proceedings of the IEEE International Reliability Physics Symposium*. PR.1.1–PR.1.7. DOI: https://doi.org/10.1109/IRPS.2014.6861168

[29] C. S. Chan, Y. Jin, Y. K. Wu, K. Gross, K. Vaidyanathan, and T. S. Rosing. 2012. Fan-speed-aware scheduling of data intensive jobs. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'12)*. Association for Computing Machinery, New York, NY, 409–414. DOI: https://doi.org/10.1145/2333660.2333753

[30] C. S. Chan, B. Pan, K. Gross, K. Vaidyanathan, and T. S. Rosing. 2014. Correcting vibration-induced performance degradation in enterprise servers. *SIGMETRICS Perform. Eval. Rev.* 41, 3 (Jan. 2014), 83–88. DOI: https://doi.org/10.1145/2567529.2567555

[31] T. Chantem, X. S. Hu, and R. P. Dick. 2011. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. *IEEE Trans. Very Large Scale Integr. Syst.* 19, 10 (Oct 2011), 1884–1897. DOI: https://doi.org/10.1109/TVLSI.2010.2058873

[32] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. 2005. X10: An object-oriented approach to non-uniform cluster computing. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'05)*. Association for Computing Machinery, New York, NY, 519–538. DOI: https://doi.org/10.1145/1094811.1094852

[33] M. Y. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. 2004. Path-based faliure and evolution management. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation (NSDI'04)*. USENIX Association, Berkeley, CA, 23–23. http://dl.acm.org/citation.cfm?id=1251175.1251198

[34] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. 2002. Pinpoint: Problem determination in large, dynamic Internet services. In *Proceedings International Conference on Dependable Systems and Networks*. 595–604. DOI: https://doi.org/10.1109/DSN.2002.1029005

[35] Z. Chen. 2011. Algorithm-based recovery for iterative methods without checkpointing. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC'11)*. ACM, New York, NY, 73–84. DOI: https://doi.org/10.1145/1996130.1996142

[36] Z. Chen. 2013. Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'13)*. ACM, New York, NY, 167–176. DOI: https://doi.org/10.1145/2442516.2442533

[37] J. Choi, C. Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose. 2007. Thermal-aware task scheduling at the system software level. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'07)*. ACM, New York, NY, 213–218. DOI: https://doi.org/10.1145/1283780.1283826

[38] A. K. Coskun, T. S. Rosing, and K. C. Gross. 2008. Temperature management in multiprocessor socs using online learning. In *Proceedings of the 45th Annual Design Automation Conference (DAC'08)*. ACM, New York, NY, 890–893. DOI: https://doi.org/10.1145/1391469.1391693

[39] A. K. Coskun, T. S. Rosing, K. Mihic, G. De Micheli, and Y. Leblebici. 2006. Analysis and optimization of MPSoC reliability. *J. Low Power Electron.* 2, 1 (2006), 56–69. DOI: https://doi.org/doi:10.1166/jolpe.2006.007

[40] G. Da Costa, A. Oleksiak, W. Piatek, J. Salom, and L. Sisó. 2015. Minimization of costs and energy consumption in a data center by a workload-based capacity management. In *Energy Efficient Data Centers*, S. Klingert, M. Chinnici, and M. Rey Porto (Eds.). Springer International Publishing, Cham, 102–119.

[41] IEEE IRDS Technical Council. 2018. *International Roadmap for Devices and Systems*. IEEE.

[42] L. Cupertino, G. Da Costa, A. Oleksiak, W. Piatek, J.M. Pierson, J. Salom, L. Siso, P. Stolf, H. Sun, and T. Zilio. 2015. Energy-efficient, thermal-aware modeling and simulation of data centers: The CoolEmAll approach and evaluation results. *Ad Hoc Netw.* 25 (2015), 535–553. DOI: https://doi.org/10.1016/j.adhoc.2014.11.002

[43] P. Czarnul, J. Proficz, and A. Krzywaniak. 2019. Energy-aware high-performance computing: Survey of state-of-the-art tools, techniques, and environments. *Sci. Program.* 2019, 4 (2019), 8348791. DOI: https://doi.org/10.1155/2019/8348791

[44] W. J. Dally. 1991. Express cubes: Improving the performance of k-ary n-cube interconnection networks. *IEEE Trans. Comput.* 40, 9 (Sep. 1991), 1016–1023. DOI: https://doi.org/10.1109/12.83652

[45] D. Dasari, B. Akesson, V. Nélis, M. A. Awan, and S. M. Petters. 2013. Identifying the sources of unpredictability in COTS-based multicore systems. In *Proceedings of the 8th IEEE International Symposium on Industrial Embedded Systems (SIES'13)*. 39–48. DOI: https://doi.org/10.1109/SIES.2013.6601469

[46] D. Dauwe, R. Jhaveri, S. Pasricha, A. A. Maciejewski, and H. J. Siegel. 2018. Optimizing checkpoint intervals for reduced energy use in exascale systems. In *Proceedings of the 8th International Green and Sustainable Computing Conference (IGSC'17)*. 1–8. DOI: https://doi.org/10.1109/IGCC.2017.8323598

[47] D. Dauwe, S. Pasricha, A. A. Maciejewski, and H. J. Siegel. 2017. An analysis of resilience techniques for exascale computing platforms. *Proceedings of the IEEE 31st International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2017*. 914–923. DOI: https://doi.org/10.1109/IPDPSW.2017.41

[48] D. Dauwe, S. Pasricha, A. A. Maciejewski, and H. J. Siegel. 2018. An analysis of multilevel checkpoint performance models. *Proceedings of the IEEE 32nd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018*. 783–792. DOI: https://doi.org/10.1109/IPDPSW.2018.00125

[49] D. Dauwe, S. Pasricha, A. A. Maciejewski, and H. J. Siegel. 2018. Resilience-aware resource management for exascale computing systems. *IEEE Trans. Sustain. Comput.* 3, 4 (2018), 332–345. DOI: https://doi.org/10.1109/tsusc.2018.2797890

[50] T. Davies and Z. Chen. 2013. Correcting soft errors online in LU factorization. In *Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing (HPDC'13)*. ACM, New York, NY, 167–178. DOI: https://doi.org/10.1145/2493123.2462920

[51] R. I. Davis and A. Burns. 2011. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.* 43, 4, Article 35 (Oct. 2011), 44 pages. DOI: https://doi.org/10.1145/1978802.1978814

[52] Narayan Desai. 2005. Cobalt: An open source platform for HPC system software research. Edinburgh BG/L System Software Workshop.

[53] S. Di, L. Bautista-Gome, and F. Cappello. 2014. Optimization of a multilevel checkpoint model with uncertain execution scales. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*. 907–918. DOI: https://doi.org/10.1109/SC.2014.79

[54] S. Di, E. Berrocal, and F. Cappello. 2015. An efficient silent data corruption detection method with error-feedback control and even sampling for HPC applications. In *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 271–280. DOI: https://doi.org/10.1109/CCGrid.2015.17

[55] S. Di, M. S. Bouguerra, L. Bautista-Gomez, and F. Cappello. 2014. Optimization of multi-level checkpoint model for large scale HPC applications. In *Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium*. 1181–1190. DOI: https://doi.org/10.1109/IPDPS.2014.122

[56] S. Di and F. Cappello. 2016. Adaptive impact-driven detection of silent data corruption for HPC applications. *IEEE Trans. Parallel Distrib. Syst.* 27, 10 (Oct. 2016), 2809–2823. DOI: https://doi.org/10.1109/TPDS.2016.2517639

[57] S. Di, H. Guo, R. Gupta, E. R. Pershey, M. Snir, and F. Cappello. 2019. Exploring properties and correlations of fatal events in a large-scale HPC system. *IEEE Trans. Parallel Distrib. Syst.* 30, 2 (Feb. 2019), 361–374. DOI: https://doi.org/10.1109/TPDS.2018.2864184

[58] S. Di, H. Guo, E. Pershey, M. Snir, and F. Cappello. 2019. Characterizing and understanding HPC job failures over the 2K-day life of IBM BlueGene/Q system. In *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'19)*. 473–484. DOI: https://doi.org/10.1109/DSN.2019.00055

[59] S. Di, Y. Robert, F. Vivien, and F. Cappello. 2017. Toward an optimal online checkpoint solution under a two-level HPC checkpoint model. *IEEE Trans. Parallel Distrib. Syst.* 28, 1 (Jan. 2017), 244–259. DOI: https://doi.org/10.1109/TPDS.2016.2546248

[60] C. Di-Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer. 2014. Lessons learned from the analysis of system failures at petascale: The case of blue waters. In *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 610–621. DOI: https://doi.org/10.1109/DSN.2014.62

[61] P. Dinda, X. Wang, J. Wang, C. Beauchene, and C. Hetland. 2018. Hard real-time scheduling for parallel run-time systems. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC'18)*. ACM, New York, NY, 14–26. DOI: https://doi.org/10.1145/3208040.3208052

[62] J. Domke, T. Hoefler, and S. Matsuoka. 2014. Fail-in-place network design: Interaction between topology, routing algorithm and failures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*. 597–608. DOI: https://doi.org/10.1109/SC.2014.54

[63] J. Dongarra, T. Herault, and Y. Robert. 2015. *Fault Tolerance Techniques for High-Performance Computing*. Springer.

[64] S. D. Downing and D. F. Socie. 1982. Simple rainflow counting algorithms. *Int. J. Fatigue* 4, 1 (1982), 31–40. DOI: https://doi.org/10.1016/0142-1123(82)90018-4

[65] B. Eghbalkhah, M. Kamal, H. Afzali-Kusha, A. Afzali-Kusha, M. B. Ghaznavi-Ghoushchi, and M. Pedram. 2015. Workload and temperature dependent evaluation of BTI-induced lifetime degradation in digital circuits. *Microelectron. Reliabil.* 55, 8 (2015), 1152–1162.

[66] J. Elliott, M. Hoemmen, and F. Mueller. 2014. Evaluating the impact of SDC on the GMRES iterative solver. In *Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium*. 1193–1202. DOI: https://doi.org/10.1109/IPDPS.2014.123

[67] R. Entner. 2007. *Modeling and Simulation of Negative Bias Temperature Instability*. Ph.D. Dissertation. Technische Universitaet Wien, Institut fur Mikroelektronik.

[68] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell. 2012. Detection and correction of silent data corruption for large-scale high-performance computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. 1–12. DOI: https://doi.org/10.1109/SC.2012.49

[69] J. Flich, G. Agosta, P. Ampletzer, D. A. Alonso, C. Brandolese, E. Cappe, A. Cilardo, L. Dragic, A. Dray, A. Duspara, W. Fornaciari, G. Guillaume, Y. Hoornenborg, A. Iranfar, M. Kovac, S. Libutti, B. Maitre, J. M. Martinez, G. Massari, H. Mlinaric, E. Papastefanakis, T. Picornell, I. Piljic, A. Pupykina, F. Reghenzani, I. Staub, R. Tornero, M. Zapater, and D. Zoni. 2017. MANGO: Exploring manycore architectures for next-GeneratiOn HPC systems. In *Proceedings of the Euromicro Conference on Digital System Design (DSD'17)*. 478–485. DOI: https://doi.org/10.1109/DSD.2017.51

[70] W. Fornaciari, G. Agosta, D. Atienza, C. Brandolese, L. Cammoun, L. Cremona, A. Cilardo, A. Farres, J. Flich, C. Hernandez, M. Kulchewski, S. Libutti, J.M. Martinez, G. Massari, A. Oleksiak, A. Pupykina, F. Reghenzani, R. Tornero, M. Zanella, M. Zapater, and D. Zoni. 2018. Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems. In *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS'18)*. ACM, New York, NY, 187–194. DOI: https://doi.org/10.1145/3229631.3239368

[71] S. Fu and C. Xu. 2007. Quantifying temporal and spatial correlation of failure events for proactive management. In *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS'07)*. 175–184. DOI: https://doi.org/10.1109/SRDS.2007.18

[72] E. W. Fulp, G. A. Fink, and J. N. Haack. 2008. Predicting computer system failures using support vector machines. In *Proceedings of the First USENIX Conference on Analysis of System Logs (WASL'08)*. USENIX Association, Berkeley, CA, 5–5. Retrieved from http://dl.acm.org/citation.cfm?id=1855886.1855891.

[73] A. Gainaru, F. Cappello, M. Snir, and W. Kramer. 2012. Fault prediction under the microscope: A closer look into HPC systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. 1–11. DOI: https://doi.org/10.1109/SC.2012.57

[74] R. Garg, A. Mohan, M. Sullivan, and G. Cooperman. 2018. CRUM: Checkpoint-restart support for CUDA's unified memory. *Proceedings of the IEEE International Conference on Cluster Computing (ICCC'18)*. 302–313. DOI: https://doi.org/10.1109/CLUSTER.2018.00047arxiv:1808.00117

[75] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi. 1998. A methodology for detection and estimation of software aging. In *Proceedings of the 9th International Symposium on Software Reliability Engineering*. 283–292. DOI: https://doi.org/10.1109/ISSRE.1998.730892

[76] P. Gill, N. Jain, and N. Nagappan. 2011. Understanding network failures in data centers: Measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM Conference (SIGCOMM'11)*. ACM, New York, NY, 350–361. DOI: https://doi.org/10.1145/2018436.2018477

[77] R. Gioiosa, J. C. Sancho, S. Jiang, and F. Petrini. 2005. Transparent, incremental checkpointing at kernel level: A foundation for fault tolerance for parallel computers. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'05)*. 9–9. DOI: https://doi.org/10.1109/SC.2005.76

[78] M. Gottscho, M. Shoaib, S. Govindan, B. Sharma, D. Wang, and P. Gupta. 2017. Measuring the impact of memory errors on application performance. *IEEE Comput. Architect. Lett.* 16, 1 (Jan. 2017), 51–55. DOI: https://doi.org/10.1109/LCA.2016.2599513

[79] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. 2011. VL2: A scalable and flexible data center network. *Commun. ACM* 54, 3 (Mar. 2011), 95–104. DOI: https://doi.org/10.1145/1897852.1897877

[80] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. 2009. BCube: A high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM Conference on Data Communication (SIGCOMM'09)*. ACM, New York, NY, 63–74. DOI: https://doi.org/10.1145/1592568.1592577

[81] G. Hamerly and C. Elkan. 2001. Bayesian approaches to failure prediction for disk drives. In *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 202–209. http://dl.acm.org/citation.cfm?id=645530.655825

[82] S. Hamouda. 2019. *Resilience in High-Level Parallel Programming Languages*. Ph.D. Dissertation. The Australian National University. DOI: https://doi.org/10.25911/5d0cb264c1c22

[83] J. L. Hellerstein, F. Zhang, and P. Shahabuddin. 1999. An approach to predictive detection for service management. In *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management*. 309–322. DOI: https://doi.org/10.1109/INM.1999.770691

[84] R. L. Henderson. 1995. Job scheduling under the portable batch system. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (IPPS'95)*. Springer-Verlag, Berlin, 279–294.

[85] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. 2005. An overview of the trilinos project. *ACM Trans. Math. Softw.* 31, 3 (Sept. 2005), 397–423. DOI: https://doi.org/10.1145/1089014.1089021

[86] G. A. Hoffmann. 2006. *Failure Prediction in Complex Computer Systems: A Probabilistic Approach*. Shaker Verlag.

[87] G. A. Hoffmann, K. S. Trivedi, and M. Malek. 2007. A best practice guide to resource forecasting for computing systems. *IEEE Trans. Reliabil.* 56, 4 (Dec. 2007), 615–628. DOI: https://doi.org/10.1109/TR.2007.909764

[88] M. Y. Hsiao, W. C. Carter, J. W. Thomas, and W. R. Stringfellow. 1981. Reliability, availability, and serviceability of IBM computer systems: A quarter century of progress. *IBM J. Res. Dev.* 25, 5 (Sept. 1981), 453–468. DOI: https://doi.org/10.1147/rd.255.0453

[89] K.-H. Huang and J. A. Abraham. 1984. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Comput.* C-33, 6 (June 1984), 518–528. DOI: https://doi.org/10.1109/TC.1984.1676475

[90] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusamy. 2004. Compact thermal modeling for temperature-aware design. In *Proceedings of the 41st Design Automation Conference, 2004.* 878–883.

[91] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan. 2002. Improved disk-drive failure warnings. *IEEE Trans. Reliabil.* 51, 3 (Sep. 2002), 350–357. DOI: https://doi.org/10.1109/TR.2002.802886

[92] S. Hukerikar, P. C. Diniz, R. F. Lucas, and K. Teranishi. 2014. Opportunistic application-level fault detection through adaptive redundant multithreading. In *Proceedings of the International Conference on High Performance Computing Simulation (HPCS'14)*. 243–250. DOI: https://doi.org/10.1109/HPCSim.2014.6903692

[93] S. Hukerikar and C. Engelmann. 2017. Resilience design patterns: A structured approach to resilience at extreme scale. *Supercomput. Front. Innov.* 4, 3 (2017). DOI: https://doi.org/10.14529/jsfi170301

[94] H. Hussain, S. Malik, A. Hameed, S. U. Khan, G. Bickler, N. Min-Allah, M. B. Qureshi, L. Zhang, W. Yongji, N. Ghani, J. Kolodziej, A. Y. Zomaya, C. Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. E. Pecero, D. Kliazovich, P. Bouvry, H. Li, L. Wang, D. Chen, and A. Rayes. 2013. A survey on resource allocation in high performance distributed computing systems. *Parallel Comput.* 39, 11 (2013), 709–736. DOI: https://doi.org/10.1016/j.parco.2013.09.009

[95] Z. Hussain, T. Znati, and R. Melhem. 2018. Partial redundancy in HPC systems with non-uniform node reliabilities. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'18)*. IEEE Press, Piscataway, NJ, Article 44, 11 pages. http://dl.acm.org/citation.cfm?id=3291656.3291715

[96] Intel Corporation. [n.d.]. *Intel Xeon Processor E7 Family: Reliability, Availability, and Serviceability*. White paper. https://www.intel.com/content/www/us/en/processors/xeon/xeon-e7-family-ras-server-paper.html.

[97] ISO/DIS 26262: Road Vehicles—Functional Safety. 2011. International Organization for Standardization.

[98] A. Iranfar, M. Kamal, A. Afzali-Kusha, M. Pedram, and D. Atienza. 2018. TheSPoT: Thermal stress-aware power and temperature management for multiprocessor systems-on-chip. *IEEE Trans. Comput.-Aid. Design Integr. Circ. Syst.* 37, 8 (Aug. 2018), 1532–1545. DOI: https://doi.org/10.1109/TCAD.2017.2768417

[99] A. Iranfar, F. Terraneo, W. A. Simon, L. Dragić, I. Piljić, M. Zapater, W. Fornaciari, M. Kovač, and D. Atienza. 2017. Thermal characterization of next-generation workloads on heterogeneous MPSoCs. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS'17)*. 286–291.

[100] S. Jha, V. Formicola, C. Di Martino, M. A. Dalton, W. T. Kramer, Z. T. Kalbarczyk, and R. K. Iyer. 2017. Resiliency of HPC interconnects: A case study of interconnect failures and recovery in blue waters. *IEEE Trans. Depend. Secure Comput.* 15 (2017), 915–930.

[101] M. Jin, C. Liu, J. Kim, J. Kim, H. Shim, K. Kim, G. Kim, S. Lee, T. Uemura, M. Chang, T. An, J. Park, and S. Pae. 2016. Reliability characterization of 10 nm FinFET technology with multi-VT gate stack for low power and high performance. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'16)*. 15.1.1–15.1.4. DOI: https://doi.org/10.1109/IEDM.2016.7838420

[102] S. Kannan, N. Farooqui, A. Gavrilovska, and K. Schwan. 2014. HeteroCheckpoint: Efficient checkpointing for accelerator-based systems. In *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 738–743. DOI: https://doi.org/10.1109/DSN.2014.76

[103] E. Kiciman and A. Fox. 2005. Detecting application-level failures in component-based Internet services. *IEEE Trans. Neural Netw.* 16, 5 (Sep. 2005), 1027–1041. DOI: https://doi.org/10.1109/TNN.2005.853411

[104] T. Kim, Z. Sun, C. Cook, H. Zhao, R. Li, D. Wong, and S. X. . Tan. 2016. Invited: Cross-layer modeling and optimization for electromigration induced reliability. In *Proceedings of the 53rd ACM/EDAC/IEEE Design Automation Conference (DAC'16)*. 1–6. DOI: https://doi.org/10.1145/2897937.2905010

[105] M. Kumar, S. Gupta, T. Patel, M. Wilder, W. Shi, S. Fu, C. Engelmann, and D. Tiwari. 2018. Understanding and analyzing interconnect errors and network congestion on a large scale HPC system. In *Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'18)*. 107–114. DOI: https://doi.org/10.1109/DSN.2018.00023

[106] K. Kurowski, A. Oleksiak, W. Piątek, T. Piontek, A. Przybyszewski, and J. Węglarz. 2013. DCworms – A tool for simulation of energy efficiency in distributed computing infrastructures. *Simul. Model. Pract. Theory* 39 (2013), 135–151. DOI: https://doi.org/10.1016/j.simpat.2013.08.007 S. I. Energy efficiency in grids and clouds.

[107] R. Lal and G. Choi. 1998. Error and failure analysis of a UNIX server. In *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium*. 232–239. DOI: https://doi.org/10.1109/HASE.1998.731618

[108] J. Langou, Z. Chen, G. Bosilca, and J. Dongarra. 2008. Recovery patterns for iterative methods in a parallel unstable environment. *SIAM J. Sci. Comput.* 30, 1 (2008), 102–116. Retrieved from arXiv:https://doi.org/10.1137/040620394

[109] J. C. Laprie (Ed.). 1995. *Dependability—Its Attributes, Impairments and Means*. Springer-Verlag, Berlin.

[110] J. C. Laprie. 1995. Dependable computing and fault tolerance: Concepts and terminology. In *Proceedings of the 25th International Symposium on Fault-Tolerant Computing*. DOI: https://doi.org/10.1109/FTCSH.1995.532603

[111] J. C. C. Laprie, A. Avizienis, and H. Kopetz (Eds.). 1992. *Dependability: Basic Concepts and Terminology*. Springer-Verlag, Berlin.

[112] C. J. M. Lasance. 2003. Thermally driven reliability issues in microelectronic systems: Status-quo and challenges. *Microelectron. Reliabil.* 43, 12 (2003), 1969–1974. DOI: https://doi.org/10.1016/S0026-2714(03)00183-5

[113] C. E. Leiserson. 1985. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.* 34, 10 (Oct. 1985), 892–901. http://dl.acm.org/citation.cfm?id=4492.4495

[114] D. Levy and R. Chillarege. 2003. Early warning of failures through alarm analysis a case study in telecom voice mail systems. In *Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE'03)* 271–280. DOI: https://doi.org/10.1109/ISSRE.2003.1251049

[115] X. Liang, J. Chen, D. Tao, S. Li, P. Wu, H. Li, K. Ouyang, Y. Liu, F. Song, and Z. Chen. 2017. Correcting soft errors online in fast Fourier transform. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'17)*. ACM, New York, NY, Article 30, 12 pages. DOI: https://doi.org/10.1145/3126908.3126915

[116] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo. 2006. BlueGene/L failure analysis and prediction models. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*. 425–434. DOI: https://doi.org/10.1109/DSN.2006.18

[117] T. T. Y. Lin and D. P. Siewiorek. 1990. Error log analysis: Statistical modeling and heuristic trend analysis. *IEEE Trans. Reliabil.* 39, 4 (Oct. 1990), 419–432. DOI: https://doi.org/10.1109/24.58720

[118] N. Losada, P. González, M. J. Martín, G. Bosilca, A. Bouteiller, and K. Teranishi. 2020. Fault tolerance of MPI applications in exascale systems: The ULFM solution. *Future Gen. Comput. Syst.* 106 (2020), 467–481. DOI: https://doi.org/10.1016/j.future.2020.01.026

[119] R. E. Lyons and W. Vanderkulk. 1962. The use of triple-modular redundancy to improve computer reliability. *IBM J. Res. Dev.* 6, 2 (Apr. 1962), 200–209. DOI: https://doi.org/10.1147/rd.62.0200

[120] C. D. Martino, W. Kramer, Z. Kalbarczyk, and R. Iyer. 2015. Measuring and understanding extreme-scale application resilience: A field study of 5,000,000 HPC application runs. In *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 25–36. DOI: https://doi.org/10.1109/DSN.2015.50

[121] M. Médard and S. S. Lumetta. 2003. *Network Reliability and Fault Tolerance*. American Cancer Society. Retrieved from arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471219282.eot281.

[122] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. 2015. Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field. In *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 415–426. DOI: https://doi.org/10.1109/DSN.2015.57

[123] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. 2010. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*. 1–11. DOI: https://doi.org/10.1109/SC.2010.18

[124] Moor Insights & Strategy. 2017. *AMD EPYC Brings New RAS Capability*. White paper. Retrieved from https://www.amd.com/system/files/2017-06/AMD-EPYC-Brings-New-RAS-Capability.pdf.

[125] F. Mulas, D. Atienza, A. Acquaviva, S. Carta, L. Benini, and G. De Micheli. 2009. Thermal balancing policy for multiprocessor stream computing platforms. *IEEE Trans. Comput.-Aid. Design Integr. Circ. Syst.* 28, 12 (Dec. 2009), 1870–1882. DOI: https://doi.org/10.1109/TCAD.2009.2032372

[126] J. Murray, G. Hugues, and K. Kreutz-Delgado. 2003. Hard drive failure prediction using non-parametric statistical methods. In *Proceedings of the Conference on Artificial Neural Networks and Neural Information Processing (ICANN/ICONIP'03)*.

[127] B. Narasimham, S. Gupta, D. Reed, J. K. Wang, N. Hendrickson, and H. Taufique. 2018. Scaling trends and bias dependence of the soft error rate of 16 nm and 7 nm FinFET SRAMs. In *Proceedings of the IEEE International Reliability Physics Symposium (IRPS'18)*. 4C.1–1–4C.1–4. DOI: https://doi.org/10.1109/IRPS.2018.8353583

[128] S. Narayanasamy, A. K. Coskun, and B. Calder. 2007. Transient fault prediction based on anomalies in processor events. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition*. 1–6. DOI: https://doi.org/10.1109/DATE.2007.364448

[129] F. A. Nassar and D. M. Andrews. 1985. A methodology for analysis of failure prediction data. In *Proceedings of the 6th IEEE Real-Time Systems Symposium (RTSS'85)*. 160–166.

[130] A. Nukada, H. Takizawa, and S. Matsuoka. 2011. NVCR: A transparent checkpoint-restart library for NVIDIA CUDA. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. 104–113. DOI: https://doi.org/10.1109/IPDPS.2011.131

[131] A. Oleksiak, M. Kierzynka, W. Piatek, G. Agosta, A. Barenghi, C. Brandolese, W. Fornaciari, G. Pelosi, M. Cecowski, R. Plestenjak, J. Činkelj, M. Porrmann, J. Hagemeyer, R. Griessl, J. Lachmair, M. Peykanu, L. Tigges, M. vor dem Berge, W. Christmann, S. Krupop, A. Carbon, L. Cudennec, T. Goubier, J. M. Philippe, S. Rosinger, D. Schlitt, C. Pieper, C. Adeniyi-Jones, J. Setoain, L. Ceva, and U. Janssen. 2017. M2DC—Modular Microserver DataCentre with heterogeneous hardware. *Microprocess. Microsyst.* 52 (2017), 117–130. DOI: https://doi.org/10.1016/j.micpro.2017.05.019

[132] D. A. G. Oliveira, P. Rech, L. L. Pilla, P. O. A. Navaux, and L. Carro. 2014. GPGPUs ECC efficiency and efficacy. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'14)*. 209–215. DOI: https://doi.org/10.1109/DFT.2014.6962085

[133] M. A. Oxley, E. Jonardi, S. Pasricha, A. A. Maciejewski, H. J. Siegel, P. J. Burns, and G. A. Koenig. 2018. Rate-based thermal, power, and co-location aware resource management for heterogeneous data centers. *J. Parallel Distrib. Comput.* 112 (2018), 126–139. DOI: https://doi.org/10.1016/j.jpdc.2017.04.015 Parallel Optimization using/for Multi and Many-core High Performance Computing.

[134] K. O'brien, I. Pietri, R. Reddy, A. Lastovetsky, and R. Sakellariou. 2017. A survey of power and energy predictive models in HPC systems and applications. *ACM Comput. Surv.* 50, 3, Article 37 (June 2017), 38 pages. DOI: https://doi.org/10.1145/3078811

[135] S. Park and M. Humphrey. 2011. Predictable high-performance computing using feedback control and admission control. *IEEE Trans. Parallel Distrib. Syst.* 22, 3 (Mar. 2011), 396–411. DOI: https://doi.org/10.1109/TPDS.2010.100

[136] A. J. Peña, W. Bland, and P. Balaji. 2015. VOCL-FT: Introducing techniques for efficient soft error coprocessor recovery. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'15)*. 1–12. DOI: https://doi.org/10.1145/2807591.2807640

[137] J. D. Pfefferman and B. Cernuschi-Frias. 2002. A nonparametric nonstationary procedure for failure prediction. *IEEE Trans. Reliabil.* 51, 4 (Dec. 2002), 434–442. DOI: https://doi.org/10.1109/TR.2002.804733

[138] M. Pizza, L. Strigini, A. Bondavalli, and F. Di Giandomenico. 1998. Optimal discrimination between transient and permanent faults. In *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium*. 214–223. DOI: https://doi.org/10.1109/HASE.1998.731615

[139] B. Pourghassemi and A. Chandramowlishwaran. 2017. cudaCR: An in-kernel application-level checkpoint/restart scheme for CUDA-enabled GPUs. In *Proceedings of the IEEE Cluster Conference (CLUSTER'17)*.

[140] R. Rajachandrasekar, X. Besseron, and D. K. Panda. 2012. Monitoring and predicting hardware failures in HPC clusters with FTB-IPMI. In *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium*. 1136–1143. DOI: https://doi.org/10.1109/IPDPSW.2012.139

[141] K. K. Rangan, G. Y. Wei, and D. Brooks. 2009. Thread motion: Fine-grained power management for multi-core systems. *SIGARCH Comput. Architect. News* 37, 3 (June 2009), 302–313. DOI: https://doi.org/10.1145/1555815.1555793

[142] Paolo Rech. [n.d.]. Reliability Issues in Current and Future Supercomputers. Retrieved from http://energysfe.ufsc.br/slides/Paolo-Rech-260917.pdf.

[143] F. Reghenzani, G. Massari, and W. Fornaciari. 2019. The real-time Linux kernel: A survey on PREEMPT_RT. *Comput. Surveys* 52, 1, Article 18 (Feb. 2019), 36 pages. DOI: https://doi.org/10.1145/3297714

[144] F. Reghenzani, G. Pozzi, G. Massari, S. Libutti, and W. Fornaciari. 2016. The MIG framework: Enabling transparent process migration in open MPI. In *Proceedings of the 23rd European MPI Users' Group Meeting (EuroMPI'16)*. ACM, New York, NY, 64–73. DOI: https://doi.org/10.1145/2966884.2966903

[145] F. Salfner, M. Lenk, and M. Malek. 2010. A survey of online failure prediction methods. *ACM Comput. Surv.* 42, 3, Article 10 (March 2010), 42 pages. DOI: https://doi.org/10.1145/1670679.1670680

[146] F. Salfner, M. Schieschke, and M. Malek. 2006. Predicting failures of computer systems: A case study for a telecommunication system. In *Proceedings of the 20th IEEE International Parallel Distributed Processing Symposium*. DOI: https://doi.org/10.1109/IPDPS.2006.1639672

[147] A. Sari, M. Psarakis, and D. Gizopoulos. 2013. Combining checkpointing and scrubbing in FPGA-based real-time systems. In *Proceedings of the IEEE 31st VLSI Test Symposium (VTS'13)*. 1–6. DOI: https://doi.org/10.1109/VTS.2013.6548910

[148] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu. 2003. Software aging and multifractality of memory resources. In *Proceedings of the International Conference on Dependable Systems and Networks*.721–730. DOI: https://doi.org/10.1109/DSN.2003.1209987

[149] L. Shi, H. Chen, J. Sun, and K. Li. 2012. vCUDA: GPU-accelerated high-performance computing in virtual machines. *IEEE Trans. Comput.* 61, 6 (June 2012), 804–816. DOI: https://doi.org/10.1109/TC.2011.112

[150] D. P. Siewiorek and R. S. Swarz. 1998. *Reliable Computer Systems*, 3rd ed. A. K. Peters, Ltd.

[151] R. M. Singer, K. Gross, J. Herzog, R. W. King, and S. Wegerich. 1997. Model-based nuclear power plant monitoring and fault detection: Theoretical foundations. In *Proceedings of the Conference on Intelligent System Application to Power Systems (ISAP'97)*.

[152] S. Singh and I. Chana. 2016. A survey on resource scheduling in cloud computing: Issues and challenges. *J. Grid Comput.* 14, 2 (June 2016), 217–264. DOI: https://doi.org/10.1007/s10723-015-9359-2

[153] T. J. Slegel, R. M. Averill, M. A. Check, B. C. Giamei, B. W. Krumm, C. A. Krygowski, W. H. Li, J. S. Liptay, J. D. MacDougall, T. J. McPherson, J. A. Navarro, E. M. Schwarz, K. Shum, and C. F. Webb. 1999. IBM's S/390 G5 microprocessor design. *IEEE Micro* 19, 2 (Mar. 1999), 12–23. DOI: https://doi.org/10.1109/40.755464

[154] P. Smith and N. C. Hutchinson. 1998. Heterogeneous process migration: The Tui system. *Softw.-Pract. Exper.* 28, 6 (1998), 611–640.

[155] A. Sridhar, M. M. Sabry, and D. Atienza. 2014. A semi-analytical thermal modeling framework for liquid-cooled ICs. *IEEE Trans. Comput.-Aid. Design Integr. Circ. Syst.* 33, 8 (Aug. 2014), 1145–1158. DOI: https://doi.org/10.1109/TCAD.2014.2323194

[156] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi. 2015. Memory errors in modern systems. *ACM SIGARCH Comput. Architect. News* 43, 1 (2015), 297–310. DOI: https://doi.org/10.1145/2786763.2694348

[157] J. H. Stathis. 2018. The physics of NBTI: What do we really know? In *Proceedings of the IEEE International Reliability Physics Symposium (IRPS'18)*. 2A.1–1–2A.1–4. DOI: https://doi.org/10.1109/IRPS.2018.8353539

[158] G. Stellner. 1996. CoCheck: Checkpointing and process migration for MPI. In *Proceedings of International Conference on Parallel Processing*. 526–531. DOI: https://doi.org/10.1109/IPPS.1996.508106

[159] G. Stellner. 1996. CoCheck: Checkpointing and process migration for MPI. In *Proceedings of the 10th International Parallel Processing Symposium (IPPS'96)*. IEEE, 526–531.

[160] J. E. Stone, D. Gohara, and Guochun Shi. 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. *Comput. Sci. Eng.* 12, 3 (May 2010), 66–73.

[161] O. Subasi, S. Di, L. Bautista-Gomez, P. Balaprakash, O. Unsal, J. Labarta, A. Cristal, and F. Cappello. 2016. Spatial support vector regression to detect silent errors in the exascale era. In *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'16)*. 413–424. DOI: https://doi.org/10.1109/CCGrid.2016.33

[162] O. Subasi, S. Di, L. Bautista-Gomez, P. Balaprakash, O. Unsal, J. Labarta, A. Cristal, S. Krishnamoorthy, and F. Cappello. 2018. Exploring the capabilities of support vector machines in detecting silent data corruptions. *Sustain. Comput.: Inform. Syst.* 19 (2018), 277–290. DOI: https://doi.org/10.1016/j.suscom.2018.01.004

[163] K. Sutaria, A. Ramkumar, R. Zhu, R. Rajveev, Y. Ma, and Y. Cao. 2014. BTI-induced aging under random stress waveforms: Modeling, simulation and silicon validation. In *Proceedings of the 51st Annual Design Automation Conference (DAC'14)*. ACM, New York, NY, Article 203, 6 pages. DOI: https://doi.org/10.1145/2593069.2593101

[164] T. Suzuki, A. Nukada, and S. Matsuoka. 2016. Transparent checkpoint and restart technology for CUDA applications. In *Proceedings of the GPU Technology Conference (GTC'16)*. https://tinyurl.com/ycb7y8xw.

[165] H. Takizawa, K. Koyama, K. Sato, K. Komatsu, and H. Kobayashi. 2011. CheCL: Transparent checkpointing and process migration of OpenCL applications. In *Proceedings of the IEEE International Parallel Distributed Processing Symposium*. 864–876. DOI: https://doi.org/10.1109/IPDPS.2011.85

[166] H. Takizawa, K. Sato, K. Komatsu, and H. Kobayashi. 2009. CheCUDA: A checkpoint/restart tool for CUDA applications. In *Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies*. 408–413. DOI: https://doi.org/10.1109/PDCAT.2009.78

[167] D. Tang and R. K. Iyer. 1993. Dependability measurement and modeling of a multicomputer system. *IEEE Trans. Comput.* 42, 1 (Jan. 1993), 62–75. DOI: https://doi.org/10.1109/12.192214

[168] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux, L. Carro, and A. Bland. 2015. Understanding GPU errors on large-scale HPC systems and the implications for system design and operation. In *Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. 331–342. DOI: https://doi.org/10.1109/HPCA.2015.7056044

[169] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. Debardeleben, P. Navaux, L. Carro, and A. Bland. 2015. Understanding GPU errors on large-scale HPC systems and the implications for system design and operation. In *Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. 331–342. DOI: https://doi.org/10.1109/HPCA.2015.7056044

[170] T. Troudet and W. Merrill. 1990. A real time neural net estimator of fatigue life. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'90)*. 59–64 vol. 2. DOI: https://doi.org/10.1109/IJCNN.1990.137695

[171] D. Turnbull and N. Alldrin. 2003. *Failure Prediction in Hardware Systems*. Tech. rep. University of California, San Diego, CA. Retrieved from http://www.cs.ucsd.edu/ dturnbul/Papers/ServerPrediction.pdf.

[172] R. Vilalta, C. V. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss. 2002. Predictive algorithms in the management of computer systems. *IBM Syst. J.* 41, 3 (2002), 461–474. DOI: https://doi.org/10.1147/sj.413.0461

[173] R. Vilalta and S. Ma. 2002. Predicting rare events in temporal domains. In *Proceedings of the IEEE International Conference on Data Mining*. 474–481. DOI: https://doi.org/10.1109/ICDM.2002.1183991

[174] S. Vinoski. 2007. Reliability with Erlang. *IEEE Internet Comput.* 11, 6 (2007), 79–81.

[175] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott. 2008. Proactive process-level live migration in HPC environments. In *Proceedings of the ACM/IEEE Conference on Supercomputing*. IEEE Press, 43.

[176] J. P. Wang and S. M. Shatz. 1988. Reliability-oriented task allocation in redundant distributed systems. In *Proceedings of the 12th Annual International Computer Software Applications Conference (COMPSAC'88)*. 276–283. DOI: https://doi.org/10.1109/CMPSAC.1988.17186

[177] A. Ward, P. Glynn, and K. Richardson. 1998. Internet service performance failure detection. *SIGMETRICS Perform. Eval. Rev.* 26, 3 (Dec. 1998), 38–43. DOI: https://doi.org/10.1145/306225.306237

[178] G. M. Weiss. 1999. Timeweaver: A genetic algorithm for identifying predictive patterns in sequences of events. In *Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 718–725.

[179] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. 2008. The worst-case execution-time problem—Overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.* 7, 3, Article 36 (May 2008), 53 pages. DOI: https://doi.org/10.1145/1347375.1347389

[180] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, and L. Shang. 2010. System-level reliability modeling for MPSoCs. In *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS'10)*. ACM, New York, NY, 297–306. DOI: https://doi.org/10.1145/1878961.1879013

[181] X. Xu, Y. Lin, T. Tang, and Y. Lin. 2010. HiAL-Ckpt: A hierarchical application-level checkpointing for CPU-GPU hybrid systems. In *Proceedings of the 5th International Conference on Computer Science Education*. 1895–1899. DOI: https://doi.org/10.1109/ICCSE.2010.5593819

[182] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin. 2008. Dynamic thermal management through task scheduling. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'08)*. 191–201. DOI: https://doi.org/10.1109/ISPASS.2008.4510751

[183] E. Yilmaz and L. Gilly. 2012. Redundancy and Reliability for an HPC Data Centre. Retrieved from http://www.prace-ri.eu/IMG/pdf/HPC-Centre-Redundancy-Reliability-WhitePaper.pdf.

[184] A. B. Yoo, M. A. Jette, and M. Grondona. 2003. SLURM: Simple Linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer Berlin, 44–60.

[185] Z. Yuan, Y. Liu, J. Li, J. Hu, C. J. Xue, and H. Yang. 2017. CP-FPGA: Energy-efficient nonvolatile FPGA with offline/online checkpointing optimization. *IEEE Trans. Very Large Scale Integr. Syst.* 25, 7 (July 2017), 2153–2163. DOI: https://doi.org/10.1109/TVLSI.2017.2680464

Received October 2019; revised April 2020; accepted May 2020