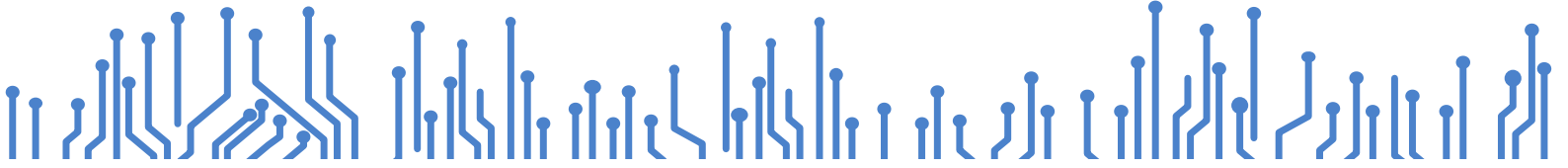


**REliable Power and time-Constraints-aware Predictive management of heterogeneous
Exascale systems**



RECOPE

**WP3 Predictive Reliability and QoS Enforcing
Methodologies**

D3.1 State of the Art on Predictive Reliability



<http://www.recipe-project.eu>



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 801137

Grant Agreement No.: 801137

Deliverable: D3.1 State of the Art on Predictive Reliability

Project Start Date: 01/05/2018

Duration: 36 months

Coordinator: *Politecnico di Milano, Italy*

Deliverable No:	D3.1
WP No:	3
WP Leader:	R. Canal
Due date:	31/01/2019
Delivery date:	31/01/2019

Dissemination Level:

PU	Public Use	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

DOCUMENT SUMMARY INFORMATION

Project title:	REliable Power and time-ConstraInts-aware Predictive management of heterogeneous Exascale systems
Short project name:	RECIPE
Project No:	801137
Call Identifier:	H2020-FETHPC-2017
Thematic Priority:	Future and Emerging Technologies
Type of Action:	Research and Innovation Action
Start date of the project:	01/05/2018
Duration of the project:	36 months
Project website:	http://www.recipe-project.eu

D3.1 State of the Art on Predictive Reliability

Work Package:	WP3 Predictive Reliability and QoS Enforcing Methodologies
Deliverable number:	D3.1
Deliverable title:	State of the Art on Predictive Reliability
Due date:	31/01/2019
Actual submission date:	31/01/2019
Editor:	J. Abella
Authors:	J. Abella, R. Canal, C. Hernandez, G. Massari, F. Reghenzani, M. Zapater
Dissemination Level:	PU
No. pages:	34
Authorized (date):	31/01/2019
Responsible person:	W. Fornaciari
Status:	Plan Draft Working Final SUBMITTED Approved

Revision history:

Version	Date	Author	Comment
v.0.1	14/12/2018		First outline, including a classification of the state-of-the-art
v.1.0	17/01/2019		First complete version
v.2.0	29/01/2019		Final version after internal review

Quality Control:

	Who	Date
Checked by internal reviewer	G. Agosta	25/01/2019
Checked by WP Leader	R. Canal	25/01/2019
Checked by Project Technical Manager	G. Agosta	31/01/2019
Checked by Project Coordinator	W. Fornaciari	31/01/2019

COPYRIGHT

©Copyright by the **RECIPE** consortium, 2018-2020.

This document contains material, which is the copyright of RECIPE consortium members and the European Commission, and may not be reproduced or copied without permission, except as mandated by the European Commission Grant Agreement no. 801137 for reviewing and dissemination purposes.

ACKNOWLEDGEMENTS

RECIPE is a project that has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 801137. Please see <http://www.recipe-project.eu> for more information.

The partners in the project are Universitat Politècnica de València (UPV), Centro Regionale Information Communication Technology srl (CeRICT), École Polytechnique Fédérale de Lausanne (EPFL), Barcelona Supercomputing Center (BSC), Poznan Supercomputing and Networking Center (PSNC), IBT Solutions S.r.l. (IBTS), Centre Hospitalier Universitaire Vaudois (CHUV). The content of this document is the result of extensive discussions within the RECIPE ©Consortium as a whole.

DISCLAIMER

The content of the publication herein is the sole responsibility of the publishers and it does not necessarily represent the views expressed by the European Commission or its services. The information contained in this document is provided by the copyright holders "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the members of the RECIPE collaboration, including the copyright holders, or the European Commission be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of the information contained in this document, even if advised of the possibility of such damage.

Contents

1	Introduction and Motivation	7
1.1	Errors reported in HPC systems	7
1.2	The need for reliability in HPC	7
2	Fault Taxonomy and fault models for HPC	9
2.1	Specific needs of HPC	10
2.1.1	Influence of thermal ageing in reliability	11
2.1.2	Thermal modelling and reliability in heterogeneous reconfigurable systems	11
2.1.3	Timing requirements in HPC	12
2.2	Predominant fault sources in HPC	12
2.2.1	Thermal gradients and thermal cycling	13
3	Fault-Prediction techniques for HPC	14
3.1	Techniques based on failure tracking	14
3.2	Techniques based on symptom monitoring	15
3.3	Techniques based on error reports	16
3.4	Timing analysis in HPC	17
3.5	Recapitulation	17
4	Fault detection and recovery in HPC systems	18
4.1	Fundamental hardware monitoring	18
4.2	Application level fault detection and recovery	19
4.3	System-level solutions	22
4.3.1	Task migration	22
4.3.2	Heterogeneous task migration	23
4.3.3	Power and thermal aware resource management	24
5	RECIPE contributions	24
6	Summary	25

1 Introduction and Motivation

Exascale systems require improving energy efficiency by orders of magnitude to provide unprecedented levels of performance within limited power envelopes. Meeting these strong energy efficiency requirements implies using advanced CMOS technologies with tiny devices (i.e. each gate may consist of few atoms), and thus with higher susceptibility to electromagnetic disturbances, environmental conditions, radiation and aging. These reliability concerns coupled with the fact that an incredibly large number of electronic devices mainly devoted for computation, connectivity, and storage, will be integrated in the future Exascale systems, puts resilience as one of the key aspects to be considered in current and future HPC systems.

As a consequence, Exascale systems will suffer dramatically higher fault rates. 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 failure 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 [86]. 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 [86]. However ECC is not enough to deal with increasing fault rates and, for instance, the Titan supercomputer at Oak Ridge National Lab reported a Mean Time Between Failure (MTBF) due to detected uncorrectable errors (DUE) caused by radiation of only 44 hours [101].

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 [74]. Thus, only memory errors, despite ECC protection, may make one every 3,000 servers to crash every month with 2014-2015 technology. Moreover, even correctable errors have a non-negligible impact in performance. 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 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, but it gains particular relevance with the advent of Exascale

computing [18]. As shown before, faults have already the power to cause frequent issues in nowadays HPC systems despite existing means for fault tolerance [33]. In fact, the reliability of HPC systems has recently gained particular attention [9].

The increasing power density in modern post-Dennard multicores raises chip temperature concerns, specially due to on-chip peak temperatures and thermal gradients, leading to a wear-out of silicon devices and putting at stake the **long-term reliability** of chips, also defined as the Mean Time To Failure (MTTF). Current power-saving techniques such as DVFS or core turn-off can potentially lead to reduction of the system long-term reliability due to undesired collateral effects, such as thermal cycling [28]. For instance, in metallic structures, when a thermal cycle amplitude increases from 10C to 20C, lifetime reliability can decrease up to 16x [26]. Furthermore, given that performance is highly affected by thermal aspects, the speed of specific structures may drop by more than 35% when working at 110C instead of at 60C [53].

The above-mentioned concerns lead to the need of proposing both thermal modeling and thermal optimization techniques tailored to current multicore chips. In particular, thermal modelling and control within the RECIPE project will allow **proactive (prediction-based) and reactive (emergency-based) thermal management** with the goal of reducing hot-spots and maintaining temperature gradients within the 5-degree limit, enhancing lifetime reliability. To achieve this goal, we need to combine expertise on thermal and reliability modelling, as well as on reliability-aware workload management techniques. Most of these techniques will be inspired from the embedded systems world[96], being scaled and ported to the heterogeneous HPC domain.

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 also have shown high susceptibility to correctable errors in data centers depending on the means set to log those errors [45]. 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 [45], a fault rate of 4 errors per second is enough to increase execution time of HPC applications by 2.5x and decrease the quality-of-service (QoS) of web-based applications by 100x. 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 (e.g. media), 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 [15, 16, 19, 36]. 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.

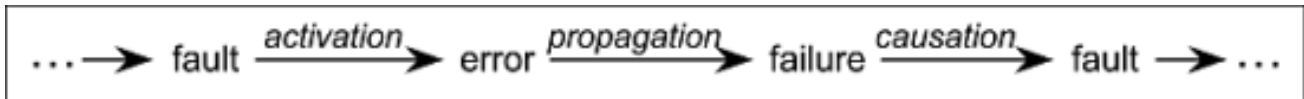


Figure 1: Fundamental chain of dependability threats.

This has particular relevance in supercomputers 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 reexecution unaffordable and yielding ineffective rollback (e.g. checkpointing) if errors are frequent. In this context, ignoring some faults may be a suitable solution.

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 error rates will be needed to complement fault tolerance. In this document, we review the state of the art on relevant fault models for HPC systems, fault prediction techniques, and error detection and correction techniques for HPC systems.

2 Fault Taxonomy and fault models for HPC

The introduction of the generic term dependability was probably the first attempt to introduce a global concept and terminology that subsumes the attributes of reliability, availability, safety, maintainability, etc.[6]. The origin of this effort dates back to 1980, when a joint committee on Fundamental Concepts and Terminology was formed by the Technical Committee on Fault-Tolerant Computing of the IEEE Computer Society and the IFIP WG 10.4 [65]. Continued intensive discussions led to the 1992 book Dependability: Basic Concepts and Terminology [66, 67]. In this work, we will follow their terminology. Specially significant is the distinction between fault, error and failure. As defined in [6], the fundamental chain of dependability and security threats is shown in Figure 1. Similarly, this work classifies faults from the failure domain viewpoint. In this case, we can distinguish between:

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

These definitions can be specialized: 1) the content can be in numerical or nonnumerical sets (e.g., alphabets, graphics, colors, sounds), and 2) a timing failure may be early or late, depending on whether the service is delivered too early or too late. Failures when both information and timing are incorrect fall into two classes:

- Halt failure, or simply halt, when the service is halted (the external state becomes constant, i.e., system activity, if there is any, is no longer perceptible to the users); a special case

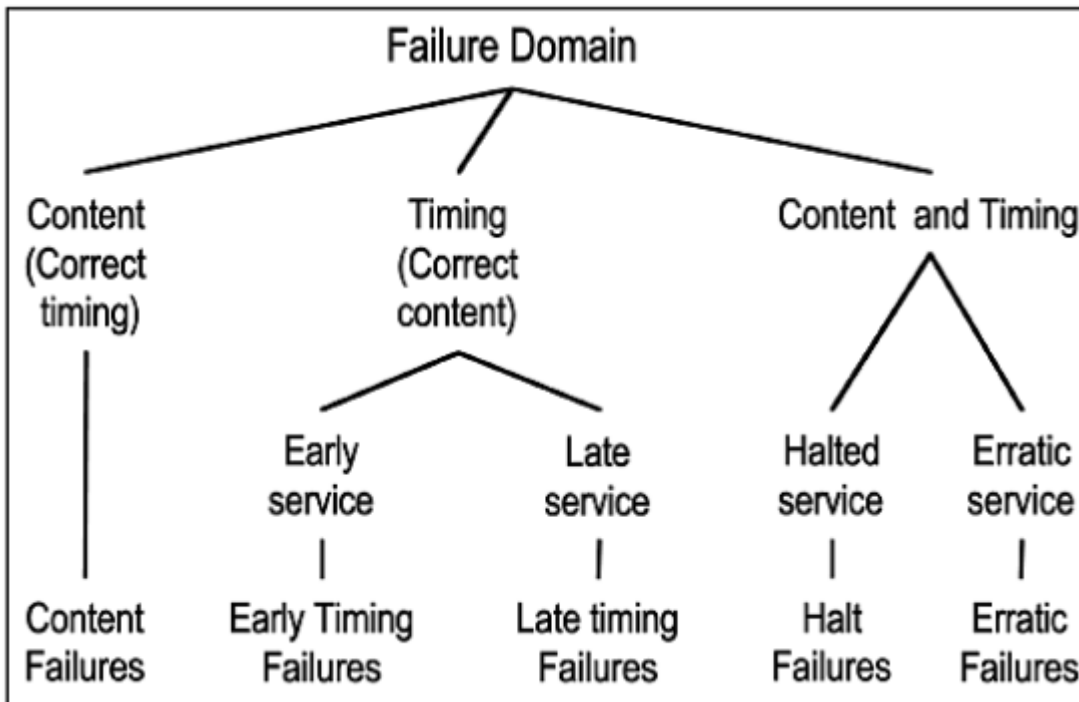


Figure 2: Failure modes with respect to the failure domain viewpoint

of halt is silent failure, or simply silence, when no service at all is delivered at the service interface (e.g., no messages are sent in a distributed system).

- Erratic failures otherwise, i.e., when a service is delivered (not halted), but is erratic (e.g., babbling).

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 and it may (or may not) lead to a change in timing.
- Detected Unrecoverable Error (DUE). An error is detected (in data and/or timing) but it cannot be corrected.

2.1 Specific needs of HPC

Contrary to the traditional data centre world, where your business is likely to define the Tier level your infrastructure should provide and your livelihood depends on the level of availability being met at all times, HPC data centres require a more differentiated approach in order to maximise scientific output and minimise infrastructure investment and operational costs. This requires a detailed analysis of the services run and the customers each HPC center serves. Different services or customer groups may require different levels of redundancy and reliability and a trade-off needs to be made between the cost of providing higher availability against the cost of

potential failures. For instance, traditional data centres rack density will vary between less than 1kW to around 6kW, HPC centres are dealing with much higher densities. It is not unusual to see machines currently on the market consuming over 30kW per rack and these numbers are expected to rise with the next generation of systems [112]. This increases the importance of an efficient infrastructure and makes cooling much more complex. Most of the large HPC centres across the world are government funded. For this reason, the costs for infrastructure and operation will always have to compete with the investment costs in new HPC hardware, as this is where the results come from. For this reason an HPC centre must minimize its infrastructure and operational costs in order to be able to maximise its investments in its core business. Electricity will be one of the largest cost contributors to the HPC centres budget. In order to control this expense, it is critical to maximise the energy efficiency of the data centre. Reducing the redundancy and reliability requirements on power and cooling can help improve energy efficiency of the overall operation thus reducing operational costs as well as infrastructure investments. Therefore providing adequate availability with an acceptable number of failures and interrupts and maximum compute power is the key. As the HPC centres get larger in computation power and storage space, the probability of errors increases proportionally to the number of parts. Early detection and errors prevention is therefore becoming ever more important in order to keep the HPC centre operational. This requires an HPC centre to deploy good monitoring, prevention and recovery tools on their systems.

2.1.1 Influence of thermal ageing in reliability

Bias Temperature Instability (namely, BTI), is a degradation effect that changes the threshold voltage of CMOS transistors. From a technological perspective, the BTI occurs when, under a constant gate voltage, a stress in temperature (i.e., increasing temperature from ambient to 200C) results in charges being trapped in the transistor gate oxide and reduces the voltage threshold of transistors [37]. These variations affect the switching characteristics of transistors and, therefore, the maximum frequency under which the circuit can work [97].

The BTI effect consists to 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 from the power/thermal/performance management of circuits as the duration of sleeping periods can impact BTI degradation and overall reliability of the system. Recent work shows the importance of the impact of temperature transients on BTI [35], whereas the previous observation imply that taking into account both application characteristics and transient temperature on BTI modeling is of utmost importance. These two aspects, however, have been traditionally neglected in previous work, which is generally focused on the development of circuit-level BTI models.

2.1.2 Thermal modelling and reliability in heterogeneous reconfigurable systems

One of the most important challenges brought by current heterogeneous and reconfigurable systems is thermal modelling. In traditional MPSoCs, the knowledge of the chip floorplan allows to identify at design time the location of the hot spots. However, this approach is no longer possible

in heterogeneous MPSoCs equipped with reconfigurable fabric, as the thermal distribution depends on the accelerators implemented, which are unknown during the design phase. As a result, the development of thermal and reliability-aware policies comes at the cost of shifting thermal evaluation from the chip design phase to the run-time management. In this context, accurate, fast and flexible thermal simulators, such as 3D-ICE [96], help understand the power dissipation requirements, tailoring the cooling to the chip requirements to best utilize HPC infrastructures while keeping cooling costs at a minimum and enabling run-time management. 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 [61].

2.1.3 Timing requirements in HPC

Focusing on timing requirements, in last 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 QoS control, may be not sufficient for certain classes of applications, such as natural disaster prediction algorithms, medical software, and real-time video transcoding tools [39, 81].

2.2 Predominant fault sources in HPC

For classification, we will distribute fault sources in two categories. 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.

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, specially, 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 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 this 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 situation arises (time dependent)

Table 1: Internal sources of variability, leading to circuit degradation and/or faults

Proximity	Spatial	Temporal	
		Reversible	Irreversible
Inter-Die Variation	Parameter (Lg, Vth, 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 Body History, Self-Heating	BTI induced Vth shift, Time-Dependant Dielectric Breakdown (TDDB)

Internal faults are present in the chips present in HPC systems. As most spatial internal sources follow an statistical distribution, manufacturers select the less affected chips for their high-end products which sell at higher prices. Most HPC systems are built with these high-end products. While the effects of these sources may be lower than in ordinary chips, as technology scales, distributions widen, all chips now have extra features to ensure a functioning device (and a high yield). Temporal sources involve certain operation conditions which favor the appearance of faults. In this sense (high) temperature triggers all temporal sources as well as parameter variations may be exacerbated at high temperatures. Consequently, it is of great interest to study this phenomena. Next sections, describe in more detail the effect of temperature on the system reliability.

2.2.1 Thermal gradients and thermal cycling

As stated above, thermal stress influences system reliability, impacting the MTTF of the system [68]. Thus, reducing the thermal hot spots is not enough to achieve adequate thermal management for MPSoCs and increase MTTF. In this work, any rapid temperature change, in either time or space, can be considered a thermal stress situation. In what follows, we will briefly focus on the thermal issues due to temporal or spatial gradients and thermal cycling.

Temporal Temperature Gradient (TTG) can be defined as the rate of temperature changes over time. For a given time, the rate of the temperature changes from one point to another indicates the spatial temperature gradient (STG). Both STG and TTG pose a critical impact on the system lifetime reliability [27]. However, it has to be noted that STG is mostly affected by the power and thermal management techniques applied at the overall MPSoC system, i.e., the allocatoin and specific setup of all cores in the system need to be taken into consideration. In contrast, TTG is mostly affected by the core frequency and the workload running in each specific core.

Finally, thermal cycling is the phenomena which takes place when the temperature rises up (or drops down) and goes back to the initial value (which can be defined as a thermal cycle) frequently [110]. Thermal cycling can be counted by Dowining simple rainflow-counting algorithms[34]. MTTF reduction due to thermal cycling occurs due to the mismatch on the expansion coefficient between the layers of the chip, which results in thermomechanical stresses. Thermal cycling (TC) tends to reduce the whole system MTTF as the number of cycles or amplitudes increases. Large amplitudes are normally induced due to improper task scheduling on a

single core. Number of thermal cycles increases especially by the power management techniques which frequently turn cores on and off [27].

3 Fault-Prediction techniques for HPC

In the context of RECIPE, we will explore fault prediction mechanisms and analytical methods of estimating application's robustness. Predicting faults will give us the time to react in order to recover from the fault. 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 will be exposed to the software orchestrator to drive efficiently the different recovery mechanisms and the utilization of the system to maximize resources efficiency.

In our analysis, we follow the taxonomy introduced in [89], but only keeping 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 can be applied to the problem at hand.

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 techniques is that failures must have occurred in order 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 produced due to 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 [82]. These techniques may also be used in the context of HPC systems since their statistical nature makes them agnostic of the source of the failure data. Such an approach is, indeed, investigated in [13], where failure prediction is performed based on the probabilistic analysis of job failures in the 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 [100, 71, 40]. In particular, [100] 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 [71] 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, related faults are assumed to occur in

the near future either in the same component (time dependence) or in neighbor components (space dependence). These concepts have also been applied to distributed systems, thus being of relevance for HPC systems [40].

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. Differently to 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 those other techniques where failures need to occur to predict future failures, which plays against preventing permanent faults to occur. Symptom monitoring may allow predicting future failures due to future permanent faults, thus taking corrective actions before those faults actually occur, hence avoiding those faults. On the other hand, 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 having to serve large amounts of requests [4]. Machine learning has also been a popular approach to predict failures based on symptom monitoring. Machine learning was already used successfully to predict failures of mechanical components in the early 90's [102]. However, it has been used in a plethora of works targeting server-type and telecommunications systems [49, 40, 89]. In this context, Hoffmann et al. [50] 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, either with discrete variables [46], with continuous variables [78] and with support vector machines [103, 41], to disk and server failure prediction, thus being of relevance for HPC systems, since the techniques are not restricted to specific components. In fact, it has been shown that, since some of these techniques provide non-binary answers (failure vs non-failure) but, instead, continuous values, outputs of these techniques can be used to monitor slowly evolving system states that get closer to failure [10]. Similar approaches have also been used for fault classification across transient or permanent faults [83]. 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, a failure is predicted. This approach has been applied to hard disk failure prediction [54, 78] 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 [92]. Alternative models have been used either based on statistical distributions of the variables assessed (e.g. mean and variance) to predict failures [108] or defining grammars of failure-free sequences of values [22]. These models can be generally applied to failure prediction in HPC systems by monitoring appropriate variables of the system. Some other models build in component utilization and interaction within a system to compare the set of components used against the different sets used during failure-free operation [21, 62]. 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 [79]. 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 also be used.

A number of techniques for failure prediction build upon past history of monitored variables to predict their future values and hence, 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 [43]. 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 [90]. Time series have been used to predict whether values will violate a threshold [47].

3.3 Techniques based on error reports

These techniques build upon error events (i.e. error logs) to predict future failures. Differently to 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 [109], whereas others build upon identifying specific sequences of errors occurring before

failures to anticipate those failures [105]. Fault trees and Markov Bayesian Networks have also been suggested as potential methods on which to build failure prediction solutions [89].

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 [69]. This observation has been corroborated in several works. For instance, some authors show that in the IBM BlueGene/L supercomputer, on average, a job experiencing two non-fatal events has much higher chances to experience a failure (above 5x) than if it only experiences one [71]. 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 [91], whereas others study the error frequency and, if such frequency increases, an imminent failure is predicted [80, 63]. Error frequency has also been used, not only to predict failures, but to classify faults and failures as either transient or permanent [72, 1].

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 [104, 72, 88]. In the case of patterns, a specific technique has been applied to HPC systems, building upon techniques from the signal processing domain [42]. 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 [84].

3.4 Timing analysis in HPC

Dealing with application timing constraints in 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 [29]. Timing constraints are usually considered in soft real-time sense in HPC. Several research works and tools to deal with the resource management problem have been developed in last years and they are thoroughly reviewed in literature surveys [56] [93]. Much less works are available on strict timing constraints for HPC. In fact, even if hard real-time has been widely studied in last decades for parallel architectures [31], the applicability of such techniques in HPC environment is limited, due to the previously discussed unpredictability challenges. Despite some recent works on HPC timing predictability exist [32, 81], this problem is still open, several challenges have to be tackled and the current solutions are definitely immature.

3.5 Recapitulation

As shown, there is a plethora of techniques that can be used for 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, which is part of the work of RECIPE.

4 Fault detection and recovery in HPC systems

4.1 Fundamental hardware monitoring

Reliability, availability and serviceability (RAS) is a term used in computer systems that includes 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 used by International Business Machines (IBM) as to refer to the robustness of their mainframe computers [51]. Nowadays, not only IBM provides support for RAS, but virtually all hardware vendors in the HPC domain provide support for it, including Intel [58], AMD [76] and ARM [5]. RAS supports 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 [14]. Those tests consist of writing specific data patterns intended to trigger different fault types.
- Processor instruction error detection. E.g. 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.
- I/O: Cyclic redundancy check (CRC) checksums for data transmission/retry and data storage, with a nature similar to that of parity and ECC.
- Storage: Journaling file systems for file repair after crashes.
- Checksums on both data and metadata.
- 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 operating ranges of clock frequency, temperature, voltage, vibration. E.g. 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.

Computers designed with higher levels of RAS have many features that protect data integrity and help them stay available for long periods without failure. This relates to the fact that individual processors may be designed with a specific (very low) Failure in Time (FIT) rate¹. 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, etc. will also contribute to the overall FIT rate of the system.

4.2 Application level fault detection and recovery

Checkpointing Checkpoints have been often used as a means for efficient fault recovery. Checkpoints consist of a snapshot of the execution that can be used to resume the execution from that point without having to restart execution from the beginning. In general, a checkpoint must reflect the architectural state of the application at a given time instant, thus including its architectural registers and memory state. However, checkpoints introduce some non-negligible overheads of terms of both, timing and storage, since saving the state requires freezing execution and storing potentially large amounts of data, which may be time consuming, and may require large storage space to hold the full checkpoint. Therefore, what to checkpoint and when are key concerns.

Some libraries offer capabilities for checkpoint and rolling back execution upon a fault detection, such as [75] and [8]. Since there is a tradeoff between how often checkpointing must occur and how often faults occur, some works aim at identifying the optimal checkpoint rate [12]. As a way to decrease checkpoint cost, some works opt for using incremental checkpoints, thus storing only the data changed since the last checkpoint and performing full checkpoints seldom [44]. Other algorithm-specific works (i.e. for a family of solvers) build on adapting the application so that the amount of data needed to recover to restart execution upon a fault is significantly lowered [24].

Algebraic-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 [23, 64, 2, 3].

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 [24, 70, 52, 30, 48].

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

Alternatively, other authors build on machine learning to detect SDCs [99]. In particular, authors rely 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.

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 [73]. 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, 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 $n = 2$ 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 reexecution 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 [11] 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 [94].

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 [59]), and faults of interest include voltage droops, crosstalk, etc. 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 or identical hardware, but with some time slack in between so that a fault does not affect the same instructions in redundant instances.

How to schedule redundant tasks in parallel systems has been an important concern [107] as well as whether to enable only partial redundancy [57], 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 [7]. Particular redundant MPI implementations have been evaluated with success Detection and correction of silent data corruption for large-scale high-performance computing [38].

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.

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 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.

4.3 System-level solutions

4.3.1 Task migration

Task migration is a recovery action that can be used as 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 different set of computational resources.

Now, 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, this may typically require the movement of MPI processes (*Process migration*) among nodes. *Reghenzani et al.* proposed an extension of the OpenMPI runtime to perform this in a transparent manner [87].

In large clusters, process migration is therefore useful to add reliability and to balance the resource allocation across the cluster [55]. 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 migration in case of overload or predicted fault, as we said. In this regard, whatever is the entity in charge of triggering the migration, we must take in account the considerable cost due to the interruption, the migration of code and data on another node and the restore procedure.

In 1996, *Stellner* proposed the first migration mechanism implemented in MPI: the *Cocheck* environment [98]. This environment was built on top 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 be used also to support classical C/R approaches, as presented by *Wang et al.* [106]. Their work introduced a process-level migration that allows us to potentially achieve a higher utilization of the system resources, with respect to virtualization based approaches. The basic idea was to try to minimize the number of C/R by using a proactive method: health monitoring of the computing node state and migration of all the running processes on a different node, in case of imminent fault prediction. Their solution actually reduces the number of performed C/R with respect to periodical checkpoint based techniques. However this requires to synchronize all the running processes into a global consistent state, before stopping and migrating them to a different node. This can represent an issue in case of imminent faults that require short time to act. The approach was implemented in LAM/MPI (predecessor of Open MPI) using the BLCR tool.

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

4.3.2 Heterogeneous task migration

A special mention should be reserved to task migration in a context of heterogeneous hardware, like RECIPE. When we talk about Process migration, typically we are assuming that we are operating on a homogeneous clusters. For several reasons, migration on heterogeneous platform is much more complex [55].

In 1998, *Smith et al.* presented the Tui System [95], an experimental framework to perform process migration between heterogeneous machines. The article was well received by the scientific community and it shows the several issues affecting the heterogeneous migration. The main problem in fact is given by the conversion between different ISA, that may require different instructions, register numbers, register size, etc. For this reason, the compiler is compulsory involved, because it must produce a code that matches one-to-one between different architecture. As a consequence, in order to simplify the problem, the Tui System introduced strong constraints regarding the two architectures involved, that leads to a migration much more *quasi-homogeneous* than heterogeneous.

Other few solutions were proposed, e.g. *Cabello et al.* [17] proposed an Open MPI middleware to provide migration mechanism in heterogeneous systems. 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 rewrite of all MPI applications.

In RECIPE, we have to deal with heterogeneity at node level, where other than having CPUs, we

must deal with GPUs and custom processing resources on FPGAs. This is an emerging scenario, for which the project gives us the chance of investigating novel solutions. In this regard, one of our goals is to integrate the possibility of migrating tasks among heterogeneous processors, in the software management stack (resource managers and programming libraries). This, trying to minimize the involvement of the application developer in the management of the migration actions.

4.3.3 Power and thermal aware resource management

Power and thermal management of MPSoCs is a topic that is quite rich in previous literature. As power consumption started to become one of the most significant challenges of multicore chips, researchers focused on controlling peak temperature indirectly by means of power management policies [85]. However, although power management approaches alleviate to some extent the thermal hot spots across the chip, these techniques are insufficient to deal with hot spots, and therefore require specific thermal management policies at both design [20] and run time [77].

Regarding thermal stress, which is an important factor that affects multicore chips' reliability, power and thermal management exhibit contradictory goals between peak temperature reduction techniques and thermal stress reduction approaches. Even though several works consider thermal stress, they rarely provide a comprehensive solution to cope with all thermal stress mechanisms along with power constraints. For instance, although in some works the tradeoffs between temporal and spatial thermal gradient mitigation schemes are investigated [25], power management and thermal cycling are not considered. In addition, other works [111] propose task scheduling methods for reducing temporal temperature gradients but disregard thermal cycling and spatial gradient.

Holistic policies, such as [60], are able to manage efficiently all thermal reliability aspects are paving the way to collaborative hardware (DVFS) software (workload allocation and application configuration) techniques to enhance the reliability of the system while providing the adequate power/performance/QoS.

5 RECIPE contributions

Exascale systems will suffer from higher 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 preceding state-of-the-art, RECIPE will explore fault prediction mechanisms and analytical methods of estimating applications robustness. Predicting faults will give us the time to react in order to recover from the fault. We will use statistical and machine learning techniques (eg., support vector machines) to predict faults and leverage application and runtime layers. Estimating applications robustness based on fault statistics and effective usage of resources will minimize application crashes and help determining optimal resource utilization. This information will 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. Understanding how the applications affect each other might help to schedule them more efficiently. We will propose a mathematical model of applications executed in parallel that will be used to schedule them on the available resources in order to increase the reliability and efficiency of the system. Data from the statistical and machine learning techniques described earlier will be used to feed the model. The model will be valid regardless of the types of processes - they may be applications executed in the operating system, virtual machines or containers in a virtual environment.

The key contributions of RECIPE will be:

1. Developing a combined reliability and timing modelling approach aimed at quantify the impact of failures in the system with the aim of achieving a predictive system behaviour accounting for the interferences of the applications co-hosted in the system and despite the presence of faults.
2. Reliability modelling will be extended by a thermal modeling infrastructure to enable the transient modeling of heterogeneous systems under thermal stress to provide a more accurate MTTF of chips.
3. Develop a reliability concious runtime manager, first of its class, that will consider holistically performance, energy and reliability when taking scheduling decisions.

6 Summary

In this report, we review 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.

Finally, we point out the roadmap of the RECIPE project and how it will contribute to advance the state-of-the-art.

References

- [1] J. Abella, P. Chaparro, X. Vera, J. Carretero, and A. Gonzalez. On-line failure detection and confinement in caches. In *2008 14th IEEE International On-Line Testing Symposium*, pages 3–9, July 2008.
- [2] E. Agullo, L. Giraud, P. Salas, and M. Zounon. Interpolation-restart strategies for resilient eigensolvers. *SIAM Journal on Scientific Computing*, 38(5):C560–C583, 2016.
- [3] Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Jean Roman, and Mawussi Zounon. Towards resilient parallel linear krylov solvers: recover-restart strategies. *INRIA, Research Report RR-8324*, 07 2013.
- [4] A. Andrzejak and L. Silva. Deterministic models of software aging and optimal rejuvenation schedules. In *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 159–168, May 2007.
- [5] ARM. *ARM Reliability, Availability, and Serviceability (RAS) Specification - ARMv8, for the ARMv8-A architecture profile*, 2017. White paper. <https://developer.arm.com/docs/ddi0587/latest>.
- [6] A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, Jan 2004.
- [7] Algirdas Avizienis. *Software Fault Tolerance. Chapter 2: The Methodology of N-Version Programming*, pages 23–. Wiley Editors, 01 1995.
- [8] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka. Fti: High performance fault tolerance interface for hybrid systems. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, Nov 2011.
- [9] L. Bautista-Gomez, F. Zyulkyarov, O. Unsal, and S. McIntosh-Smith. Unprotected computing: A large-scale study of dram raw error rate on a supercomputer. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 645–655, Nov 2016.
- [10] H. R. Berenji, J. Ametha, and D. Vengerov. Inductive learning for fault diagnosis. In *The 12th IEEE International Conference on Fuzzy Systems, 2003. FUZZ '03.*, volume 1, pages 726–731 vol.1, May 2003.
- [11] D. Bernick, B. Bruckert, P. D. Vigna, D. Garcia, R. Jardine, J. Klecka, and J. Smullen. Nonstop/spl reg/ advanced architecture. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 12–21, June 2005.
- [12] Marin Bougeret, Henri Casanova, Mikael Rabie, Yves Robert, and Frédéric Vivien. Checkpointing strategies for parallel jobs. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11*, pages 33:1–33:11, New York, NY, USA, 2011. ACM.

-
- [13] J. Brandt, F. Chen, V. De Sapio, A. Gentile, J. Mayo, P. Pbay, D. Roe, D. Thompson, and M. Wong. Quantifying effectiveness of failure prediction and response in hpc systems: Methodology and example. In *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 2–7, June 2010.
 - [14] Melvin A. Breuer and Arthur D. Friedman. *Diagnosis & Reliable Design of Digital Systems*. Springer, 1976.
 - [15] Patrick Bridges, Kurt Ferreira, Michael Heroux, and Mark Hoemmen. Fault-tolerant linear solvers via selective reliability. *arXiv:1206.1390*, 06 2012.
 - [16] Greg Bronevetsky and Bronis de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proceedings of the 22Nd Annual International Conference on Supercomputing, ICS '08*, pages 155–164, New York, NY, USA, 2008. ACM.
 - [17] Uriel Cabello, José Rodríguez, Amilcar Meneses, Sonia Mendoza, and Dominique Decouchant. Fault tolerance in heterogeneous multi-cluster systems through a task migration mechanism. In *Electrical Engineering, Computing Science and Automatic Control (CCE), 2014 11th International Conference on*, pages 1–7. IEEE, 2014.
 - [18] Franck Cappello, Al Geist, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. Toward exascale resilience: 2014 update. *Supercomputing Frontiers and Innovations*, 1(1), 2014.
 - [19] Marc Casas, Bronis R. de Supinski, Greg Bronevetsky, and Martin Schulz. Fault resilience of the algebraic multi-grid solver. In *Proceedings of the 26th ACM International Conference on Supercomputing, ICS '12*, pages 91–100, New York, NY, USA, 2012. ACM.
 - [20] T. Chantem, X. S. Hu, and R. P. Dick. Temperature-aware scheduling and assignment for hard real-time applications on mpsoCs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(10):1884–1897, Oct 2011.
 - [21] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: problem determination in large, dynamic internet services. In *Proceedings International Conference on Dependable Systems and Networks*, pages 595–604, June 2002.
 - [22] Mike Y. Chen, Anthony Accardi, Emre Kiciman, Jim Lloyd, Dave Patterson, Armando Fox, and Eric Brewer. Path-based failure and evolution management. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1, NSDI'04*, pages 23–23, Berkeley, CA, USA, 2004. USENIX Association.
 - [23] Zizhong Chen. Algorithm-based recovery for iterative methods without checkpointing. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing, HPDC '11*, pages 73–84, New York, NY, USA, 2011. ACM.
 - [24] Zizhong Chen. 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*, pages 167–176, New York, NY, USA, 2013. ACM.
 - [25] Jeonghwan Choi, Chen-Yong Cher, Hubertus Franke, Hendrik Hamann, Alan Weger, and Pradip Bose. Thermal-aware task scheduling at the system software level. In *Proceedings*

- of the 2007 International Symposium on Low Power Electronics and Design, ISLPED '07, pages 213–218, New York, NY, USA, 2007. ACM.
- [26] A. K. Coskun, T. S. Rosing, and K. C. Gross. Temperature management in multiprocessor socs using online learning. In *2008 45th ACM/IEEE Design Automation Conference*, pages 890–893, June 2008.
- [27] Ayse Kivilcim Coskun, Tajana Simunic Rosing, and Kenny C. Gross. Temperature management in multiprocessor socs using online learning. In *Proceedings of the 45th Annual Design Automation Conference, DAC '08*, pages 890–893, New York, NY, USA, 2008. ACM.
- [28] Ayse Kivilcim Coskun, Tajana Simunic Rosing, Kresimir Mihic, Giovanni De Micheli, and Yusuf Leblebici. Analysis and optimization of mp soc reliability. *Journal of Low Power Electronics*, 2(1):56–69, 2006.
- [29] D. Dasari, B. Akesson, V. Nlis, M. A. Awan, and S. M. Petters. Identifying the sources of unpredictability in cots-based multicore systems. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 39–48, June 2013.
- [30] Teresa Davies and Zizhong Chen. Correcting soft errors online in lu factorization. In *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing, HPDC '13*, pages 167–178, New York, NY, USA, 2013. ACM.
- [31] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, October 2011.
- [32] Peter Dinda, Xiaoyang Wang, Jinghang Wang, Chris Beauchene, and Conor Hetland. 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*, pages 14–26, New York, NY, USA, 2018. ACM.
- [33] Jack Dongarra, Thomas Herault, and Yves Robert. *Fault Tolerance Techniques for High-Performance Computing*. Springer, 2015.
- [34] S.D. Downing and D.F. Socie. Simple rainflow counting algorithms. *International Journal of Fatigue*, 4(1):31 – 40, 1982.
- [35] Behzad Eghbalkhah, Mehdi Kamal, Hassan Afzali-Kusha, Ali Afzali-Kusha, Mohammad Bagher Ghaznavi-Ghouschi, and Massoud Pedram. Workload and temperature dependent evaluation of bti-induced lifetime degradation in digital circuits. *Microelectronics Reliability*, 55(8):1152 – 1162, 2015.
- [36] J. Elliott, M. Hoemmen, and F. Mueller. Evaluating the impact of sdc on the gmres iterative solver. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 1193–1202, May 2014.
- [37] R. Entner. *Modeling and Simulation of Negative Bias Temperature Instability*. PhD thesis, Technische Universitaet Wien, Institut fur Mikroelektronik, 2007.
- [38] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell. Detection and correction of silent data corruption for large-scale high-performance computing. In

- SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, Nov 2012.
- [39] 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. Papat Stefanakis, T. Picornell, I. Piljic, A. Pupykina, F. Reghenzani, I. Staub, R. Tornero, M. Zapater, and D. Zoni. Mango: Exploring manycore architectures for next-generation hpc systems. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 478–485, Aug 2017.
- [40] S. Fu and C. Xu. Quantifying temporal and spatial correlation of failure events for proactive management. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, pages 175–184, Oct 2007.
- [41] Errin W. Fulp, Glenn A. Fink, and Jereme N. Haack. Predicting computer system failures using support vector machines. In *Proceedings of the First USENIX Conference on Analysis of System Logs, WASL'08*, pages 5–5, Berkeley, CA, USA, 2008. USENIX Association.
- [42] A. Gainaru, F. Cappello, M. Snir, and W. Kramer. Fault prediction under the microscope: A closer look into hpc systems. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, Nov 2012.
- [43] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi. A methodology for detection and estimation of software aging. In *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No.98TB100257)*, pages 283–292, Nov 1998.
- [44] R. Gioiosa, J. C. Sancho, S. Jiang, and F. Petrini. Transparent, incremental checkpointing at kernel level: a foundation for fault tolerance for parallel computers. In *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pages 9–9, Nov 2005.
- [45] M. Gottscho, M. Shoaib, S. Govindan, B. Sharma, D. Wang, and P. Gupta. Measuring the impact of memory errors on application performance. *IEEE Computer Architecture Letters*, 16(1):51–55, Jan 2017.
- [46] Greg Hamerly and Charles Elkan. Bayesian approaches to failure prediction for disk drives. In *Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01*, pages 202–209, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
- [47] J. L. Hellerstein, Fan Zhang, and P. Shahabuddin. An approach to predictive detection for service management. In *Integrated Network Management VI. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management. (Cat. No.99EX302)*, pages 309–322, May 1999.
- [48] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An overview of the trinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, September 2005.
- [49] G. A. Hoffmann. *Failure Prediction in Complex Computer Systems: A Probabilistic Approach*. Shaker Verlag, 2006.

-
- [50] G. A. Hoffmann, K. S. Trivedi, and M. Malek. A best practice guide to resource forecasting for computing systems. *IEEE Transactions on Reliability*, 56(4):615–628, Dec 2007.
- [51] M. Y. Hsiao, W. C. Carter, J. W. Thomas, and W. R. Stringfellow. Reliability, availability, and serviceability of ibm computer systems: A quarter century of progress. *IBM J. Res. Dev.*, 25(5):453–468, September 1981.
- [52] Kuang-Hua Huang and Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers*, C-33(6):518–528, June 1984.
- [53] Wei Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusamy. Compact thermal modeling for temperature-aware design. In *Proceedings. 41st Design Automation Conference, 2004.*, pages 878–883, July 2004.
- [54] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan. Improved disk-drive failure warnings. *IEEE Transactions on Reliability*, 51(3):350–357, Sep. 2002.
- [55] Hameed Hussain, Saif Ur Rehman Malik, Abdul Hameed, Samee Ullah Khan, Gage Bickler, Nasro Min-Allah, Muhammad Bilal Qureshi, Limin Zhang, Wang Yongji, Nasir Ghani, et al. A survey on resource allocation in high performance distributed computing systems. *Parallel Computing*, 39(11):709–736, 2013.
- [56] Hameed Hussain, Saif Ur Rehman Malik, Abdul Hameed, Samee Ullah Khan, Gage Bickler, Nasro Min-Allah, Muhammad Bilal Qureshi, Limin Zhang, Wang Yongji, Nasir Ghani, Joanna Kolodziej, Albert Y. Zomaya, Cheng-Zhong Xu, Pavan Balaji, Abhinav Vishnu, Fredric Pinel, Johnatan E. Pecero, Dzmityr Kliazovich, Pascal Bouvry, Hongxiang Li, Lizhe Wang, Dan Chen, and Ammar Rayes. A survey on resource allocation in high performance distributed computing systems. *Parallel Computing*, 39(11):709 – 736, 2013.
- [57] Zaeem Hussain, Taieb Znati, and Rami Melhem. 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*, pages 44:1–44:11, Piscataway, NJ, USA, 2018. IEEE Press.
- [58] Intel Corporation. *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>.
- [59] International Organization for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.
- [60] A. Iranfar, M. Kamal, A. Afzali-Kusha, M. Pedram, and D. Atienza. Thespot: Thermal stress-aware power and temperature management for multiprocessor systems-on-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(8):1532–1545, Aug 2018.
- [61] Arman Iranfar, Federico Terraneo, William Andrew Simon, Leon Dragic, Igor Pilji, Marina Zapater Sancho, William Fornaciari, Mario Kovac, and David Atienza Alonso. Thermal characterization of next-generation workloads on heterogeneous mpsoes. 2017.
- [62] E. Kiciman and A. Fox. Detecting application-level failures in component-based internet services. *IEEE Transactions on Neural Networks*, 16(5):1027–1041, Sep. 2005.

- [63] R. Lal and G. Choi. Error and failure analysis of a unix server. In *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (Cat. No.98EX231)*, pages 232–239, Nov 1998.
- [64] J. Langou, Z. Chen, G. Bosilca, and J. Dongarra. Recovery patterns for iterative methods in a parallel unstable environment. *SIAM Journal on Scientific Computing*, 30(1):102–116, 2008.
- [65] J. C. Laprie. Dependable computing and fault tolerance : Concepts and terminology. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'*, pages 2–, June 1995.
- [66] J.C. Laprie, editor. *Dependability Its Attributes, Impairments and Means*. Springer-Verlag, Berlin, Heidelberg, 1995.
- [67] J.C. C. Laprie, A. Avizienis, and H. Kopetz, editors. *Dependability: Basic Concepts and Terminology*. Springer-Verlag, Berlin, Heidelberg, 1992.
- [68] Clemens J.M. Lasance. Thermally driven reliability issues in microelectronic systems: status-quo and challenges. *Microelectronics Reliability*, 43(12):1969 – 1974, 2003.
- [69] D. Levy and R. Chillarege. Early warning of failures through alarm analysis a case study in telecom voice mail systems. In *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.*, pages 271–280, Nov 2003.
- [70] Xin Liang, Jieyang Chen, Dingwen Tao, Sihuan Li, Panruo Wu, Hongbo Li, Kaiming Ouyang, Yuanlai Liu, Fengguang Song, and Zizhong Chen. Correcting soft errors online in fast fourier transform. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17*, pages 30:1–30:12, New York, NY, USA, 2017. ACM.
- [71] Yinglung Liang, Yanyong Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo. Bluegene/l failure analysis and prediction models. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 425–434, June 2006.
- [72] T. . Y. Lin and D. P. Siewiorek. Error log analysis: statistical modeling and heuristic trend analysis. *IEEE Transactions on Reliability*, 39(4):419–432, Oct 1990.
- [73] R. E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200–209, April 1962.
- [74] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 415–426, June 2015.
- [75] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, Nov 2010.

- [76] Moor Insights & Strategy. *AMD EPYC Brings New RAS Capability*, 2017. White paper. <https://www.amd.com/system/files/2017-06/AMD-EPYC-Brings-New-RAS-Capability.pdf>.
- [77] F. Mulas, D. Atienza, A. Acquaviva, S. Carta, L. Benini, and G. De Micheli. Thermal balancing policy for multiprocessor stream computing platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(12):1870–1882, Dec 2009.
- [78] J. Murray, G. Hugues, and K. Kreutz-Delgado. Hard drive failure prediction using non-parametric statistical methods. In *Artificial Neural Networks and Neural Information Processing ICANN/ICONIP*, 2003.
- [79] S. Narayanasamy, A. K. Coskun, and B. Calder. Transient fault prediction based on anomalies in processor events. In *2007 Design, Automation Test in Europe Conference Exhibition*, pages 1–6, April 2007.
- [80] Fares A. Nassar and Dorothy M. Andrews. A methodology for analysis of failure prediction data. In *Proceedings of the 6th IEEE Real-Time Systems Symposium (RTSS '85), December 3-6, 1985, San Diego, California, USA*, pages 160–166, 1985.
- [81] S. Park and M. Humphrey. Predictable high-performance computing using feedback control and admission control. *IEEE Transactions on Parallel and Distributed Systems*, 22(3):396–411, March 2011.
- [82] J. D. Pfefferman and B. Cernuschi-Frias. A nonparametric nonstationary procedure for failure prediction. *IEEE Transactions on Reliability*, 51(4):434–442, Dec 2002.
- [83] M. Pizza, L. Strigini, A. Bondavalli, and F. Di Giandomenico. Optimal discrimination between transient and permanent faults. In *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (Cat. No.98EX231)*, pages 214–223, Nov 1998.
- [84] R. Rajachandrasekar, X. Besseron, and D. K. Panda. Monitoring and predicting hardware failures in hpc clusters with ftb-ipmi. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pages 1136–1143, May 2012.
- [85] Krishna K. Rangan, Gu-Yeon Wei, and David Brooks. Thread motion: Fine-grained power management for multi-core systems. *SIGARCH Comput. Archit. News*, 37(3):302–313, June 2009.
- [86] Paolo Rech. Reliability issues in current and future supercomputers. <http://energysfe.ufsc.br/slides/Paolo-Rech-260917.pdf>.
- [87] Federico Reghenzani, Gianmario Pozzi, Giuseppe Massari, Simone Libutti, and William Fornaciari. The mig framework: Enabling transparent process migration in open mpi. In *Proceedings of the 23rd European MPI Users' Group Meeting, EuroMPI 2016*, pages 64–73, New York, NY, USA, 2016. ACM.
- [88] F. Salfner, M. Schieschke, and M. Malek. Predicting failures of computer systems: a case study for a telecommunication system. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, pages 8 pp.–, April 2006.
- [89] Felix Salfner, Maren Lenk, and Mirosław Malek. A survey of online failure prediction methods. *ACM Comput. Surv.*, 42(3):10:1–10:42, March 2010.

-
- [90] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Yan Liu. Software aging and multifractality of memory resources. In *2003 International Conference on Dependable Systems and Networks, 2003. Proceedings.*, pages 721–730, June 2003.
 - [91] D. P. Siewiorek and R. S. Swarz. *Reliable Computer Systems, 3rd ed.* A. K. Peters, Ltd., 1998.
 - [92] R.M. Singer, Kenny Gross, Jim Herzog, R.W. King, and Stephan Wegerich. Model-based nuclear power plant monitoring and fault detection: Theoretical foundations. In *Conference on Intelligent System Application to Power Systems (ISAP)*, 1997.
 - [93] Sukhpal Singh and Inderveer Chana. A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of Grid Computing*, 14(2):217–264, Jun 2016.
 - [94] 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. Ibm’s s/390 g5 microprocessor design. *IEEE Micro*, 19(2):12–23, March 1999.
 - [95] Peter Smith and Norman C Hutchinson. Heterogeneous process migration: The tui system. *Software-Practice and Experience*, 28(6):611–640, 1998.
 - [96] Arvind Sridhar, Mohamed Mostafa Sabry Aly, and David Atienza Alonso. A semi-analytical thermal modeling framework for liquid-cooled ics. *IEEE T Comput Aid D*, 33(8):14. 1145–1158, 2014.
 - [97] J. H. Stathis. The physics of nbt1: What do we really know? In *2018 IEEE International Reliability Physics Symposium (IRPS)*, pages 2A.1–1–2A.1–4, March 2018.
 - [98] Georg Stellner. Cocheck: Checkpointing and process migration for MPI. In *Parallel Processing Symposium, 1996., Proceedings of IPPS’96, The 10th International*, pages 526–531. IEEE, 1996.
 - [99] O. Subasi, S. Di, L. Bautista-Gomez, P. Balaprakash, O. Unsal, J. Labarta, A. Cristal, and F. Cappello. Spatial support vector regression to detect silent errors in the exascale era. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 413–424, May 2016.
 - [100] D. Tang and R. K. Iyer. Dependability measurement and modeling of a multicomputer system. *IEEE Transactions on Computers*, 42(1):62–75, Jan 1993.
 - [101] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardleben, P. Navaux, L. Carro, and A. Bland. Understanding gpu errors on large-scale hpc systems and the implications for system design and operation. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 331–342, Feb 2015.
 - [102] T. Troudet and W. Merrill. A real time neural net estimator of fatigue life. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 59–64 vol.2, June 1990.
 - [103] D. Turnbull and N. Alldrin. *Failure prediction in hardware systems*, 2003. Tech. rep. University of California, San Diego, CA. <http://www.cs.ucsd.edu/~dturnbul/Papers/Server-Prediction.pdf>.

-
- [104] R. Vilalta, C. V. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss. Predictive algorithms in the management of computer systems. *IBM Systems Journal*, 41(3):461–474, 2002.
- [105] R. Vilalta and Sheng Ma. Predicting rare events in temporal domains. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 474–481, Dec 2002.
- [106] Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L Scott. Proactive process-level live migration in hpc environments. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 43. IEEE Press, 2008.
- [107] J. . Wang and S. M. Shatz. Reliability-oriented task allocation in redundant distributed systems. In *Proceedings COMPSAC 88: The Twelfth Annual International Computer Software Applications Conference*, pages 276–283, Oct 1988.
- [108] Amy Ward, Peter Glynn, and Kathy Richardson. Internet service performance failure detection. *SIGMETRICS Perform. Eval. Rev.*, 26(3):38–43, December 1998.
- [109] Gary M. Weiss. Timeweaver: a genetic algorithm for identifying predictive patterns in sequences of events. In *In Proceedings of the Genetic and Evolutionary Computation Conference*, pages 718–725. Morgan Kaufmann, 1999.
- [110] Yun Xiang, Thidapat Chantem, Robert P. Dick, X. Sharon Hu, and Li Shang. System-level reliability modeling for mpsoes. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES/ISSS '10*, pages 297–306, New York, NY, USA, 2010. ACM.
- [111] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin. Dynamic thermal management through task scheduling. In *ISPASS 2008 - IEEE International Symposium on Performance Analysis of Systems and software*, pages 191–201, April 2008.
- [112] Erhan Yilmaz and Ladina Gilly. Redundancy and reliability for an hpc data centre, 2012. <http://www.prace-ri.eu/IMG/pdf/HPC-Centre-Redundancy-Reliability-WhitePaper.pdf>.