Towards Resilient EU HPC Systems: A Blueprint

April 2020.

Funded by the Horizon 2020 Framework Programme of the European Union

This work has received funding from the European Union's Horizon 2020 research and innovation programme under the projects **ECOSCALE** (grant agreement No 671632), **EPI** (grant agreement No 826647), **EuroEXA** (grant agreement No 754337), **Eurolab4HPC** (grant agreement No 800962), **EVOLVE** (grant agreement No 825061), **EXA2PRO** (grant agreement No 801015), **ExaNest** (grant agreement No 671553), **ExaNoDe** (grant agreement No 671578), **EXDCI-2** (grant agreement No 800957), **LEGaTO** (grant agreement No 780681), **MB2020** (grant agreement No 779877), **RECIPE** (grant agreement No 801137) and **SDK4ED** (grant agreement No 780572).

The work was also supported by the European Commission's Seventh Framework Programme under the projects **CLERECO** (grant agreement No 611404), the NCSA-Inria-ANL-BSC-JSC-Riken-UTK Joint-Laboratory for Extreme Scale Computing – **JLESC** (https://jlesc.github.io/), **OMPI-X** project (No ECP-2.3.1.17) and the Spanish Government through **Severo Ochoa programme** (SEV-2015-0493).

This work was sponsored in part by the U.S. Department of Energy's Office of Advanced Scientific Computing Research, program managers Robinson Pino and Lucy Nowell. This manuscript has been authored by UT-Battelle, LLC under Contract No DE-AC05-00OR22725 with the U.S. Department of Energy.

Editorial team	Affiliation
Petar Radojkovic	Barcelona Supercomputing Center
Manolis Marazakis	Foundation for Research and Technology - Hellas (FORTH)
Paul Carpenter	Barcelona Supercomputing Center
Reiley Jeyapaul	Arm
Dimitris Gizopoulos	University of Athens
Martin Schulz	Technical University of Munich Leibniz Supercomputing Centre

Contributors	Affiliation
Adria Armejach	Barcelona Supercomputing Center
Eduard Ayguade	Barcelona Supercomputing Center
François Bodin	European Extreme Data & Computing Initiative (EXDCI)
Ramon Canal	Barcelona Supercomputing Center Universitat Politècnica de Catalunya
Franck Cappello	Argonne National Laboratory
Fabien Chaix	Foundation for Research and Technology - Hellas (FORTH)
Guillaume Colin de Verdiere	Alternative Energies and Atomic Energy Commission (CEA)
Said Derradji	Bull/ATOS
Stefano Di Carlo	Politecnico di Torino
Christian Engelmann	Oak Ridge National Laboratory
Ignacio Laguna	Lawrence Livermore National Laboratory
Miquel Moreto	Barcelona Supercomputing Center
Onur Mutlu	ETH Zurich
Lazaros Papadopoulos	National Technical University of Athens
Olly Perks	Arm
Manolis Ploumidis	Foundation for Research and Technology - Hellas (FORTH)
Bezhad Salami	Barcelona Supercomputing Center
Yanos Sazeides	University of Cyprus
Dimitrios Soudris	National Technical University of Athens
Yiannis Sourdis	Chalmers University
Per Stenstrom	Chalmers University
Samuel Thibault	University of Bordeaux
Will Toms	University of Manchester
Osman Unsal	Barcelona Supercomputing Center

Involved projects
CLERECO: Cross-Layer Early Reliability Evaluation for the Computing Continuum https://www.clereco.eu/
ECOSCALE: Energy-efficient Heterogeneous Computing at Exascale http://www.ecoscale.eu/
EPI: European Processor Initiative https://www.european-processor-initiative.eu/
EuroExa: Co-designed Innovation and System for Resilient Exascale Computing in Europe: From Applications to Silicon https://euroexa.eu/
Eurolab4HPC https://www.eurolab4hpc.eu/
EVOLVE: Leading the Big Data Revolution https://www.evolve-h2020.eu/
EXA2PRO: Enhancing Programmability and Boosting Perfromance Portability for Exascale Computing Systems https://exa2pro.eu/
ExaNest: European Exascale System Interconnect and Storage http://exanest.eu/
ExaNoDe: European Exascale Processor & Memory Node Design http://exanode.eu/
EXDCI: European Extreme Data & Computing Initiative https://exdci.eu/
LEGaTO: Low Energy Toolset for Heterogeneous Computing https://legato-project.eu/
MB2020: European Modular and Power-efficient HPC Processor https://www.montblanc-project.eu/
RECIPE: Reliable Power and Time-constraints-aware Predictive Management of Heterogeneous Exascale Systems http://www.recipe-project.eu/
SDK4ED: Software Development ToolKit for Energy Optimization and Technical Debt Elimination https://sdk4ed.eu/

Table of Contents

Executive summary

This document aims to spearhead a Europe-wide discussion on HPC system resilience and to help the European HPC community define best practices for resilience. We analyse a wide range of state-of-the-art resilience mechanisms and recommend the most effective approaches to employ in large-scale HPC systems. Our guidelines will be useful in the allocation of available resources, as well as guiding researchers and research funding towards the enhancement of resilience approaches with the highest priority and utility. Although our work is focused on the needs of next generation HPC systems in Europe, the principles and evaluations are applicable globally.

This document is the first output of the ongoing European HPC resilience initiative (https://www.resilientHPC.eu/) and it covers individual nodes in HPC systems, encompassing CPU, memory, intra-node interconnect and emerging FPGA-based hardware accelerators. With community support and feedback on this initial document, we will update the analysis and expand the scope to include other types of accelerators, as well as networks and storage.

The need for resilience features is analysed based on three guiding principles:

(1) The resilience features implemented in HPC systems should assure that the failure rate of the system is below an acceptable threshold, representative of the technology, system size and target application.
(2) Given the high cost incurred by uncorrected error propagation, if hardware errors occur frequently they should be detected and corrected at low overhead, which is likely only possible in hardware.
(3) Overheating is one of the main causes of unreliable device behaviour. Production HPC systems should prevent overheating while balancing power/energy and performance.

Based on these principles, the main outcome of this document is that the following features should be given priority during the design, implementation and operation of any large-scale HPC system:

- ECC in main memory
- Memory demand and patrol scrub
- Memory address parity protection
- Error detection in CPU caches and registers
- Error detection in the intra-node interconnect
- Packet retry in the intra-node interconnect
- Reporting corrected errors to the BIOS or OS (system software requirement)
- Memory thermal throttling
- Dynamic voltage and frequency scaling for CPUs, FPGAs and ASICs
- Over-temperature shutdown mechanism for FPGAs
- ECC in FPGA on-chip data memories as well as in configuration memories

The remaining state-of-the-art resilience features surveyed in this document should only be developed and implemented after a more detailed and specific cost–benefit analysis.

1. Introduction

One of the greatest challenges in High-Performance Computing (HPC) is the need to build resilient systems. Resilience is an especially difficult challenge in HPC, in which tightly coupled compute jobs typically execute across a large number of nodes for several hours or even days. Without effective provisions for resilience, in hardware and/or software, a single failure in one node or server typically causes a domino effect, resulting in the termination of the whole job. Such job termination normally results in no useful outcome, so the node hours that have already been expended on the job are wasted. Recently, the most common use of supercomputers and HPC systems are applications that solve large-scale real-world simulation problems [1, 2, 3]. Such applications, implemented as workflows or multi-scale systems are inherently fault-tolerant and have error-containment capabilities that eliminate the domino effects caused by failures. However, failure in a node executing a distributed thread leads to contained error recovery, which translates into decreased performance and throughput.

System reliability is also an important limiter on the ability to scale to larger systems. This is because increasing the number of nodes causes the overheads of error containment and error recovery to increase exponentially. Reliability is therefore an important requirement and challenge for exascale, as recognized by the HPC strategic research agendas of the European Technology Platform for High Performance Computing (ETP4HPC) [4, 5, 6], Eurolab4HPC [7], the U.S. Department of Energy [8, 9, 10, 11] and the U.S. National laboratories [12, 13].

Ensuring resilience of large-scale HPC systems is complex, and the development, analysis and evaluation of features to improve resilience requires investment in research and engineering. This document helps by analysing a wide range of state-of-the-art resilience mechanisms, and it selects the most effective approaches and identifies gaps in the design space. This prioritization is essential to properly allocate resources and to focus expertise in the European HPC community and beyond, in order to ensure that the most essential resilience features are addressed with the highest priority.

The initial scope of the analysis is influenced by the needs of the EuroEXA project,[1] which has funded the work so far. As such, in this document, we focus on resilience features of individual HPC nodes, covering CPU, memory, FPGAs and the intra-node (socket-to-socket) interconnect. As part of ongoing and future work, with community support and feedback based on this document, we will extend the analysis to GPUs, vector accelerators, NICs, interconnect networks and storage. This document is also mainly focused on the resilience features in hardware and low-level system software (e.g., the OS). Interaction among these features and resilience techniques at higher levels of the software stack is part of future work.[2]

[1] Co-designed Innovation and System for Resilient Exascale Computing in Europe: From Applications to Silicon (H2020 Grant Agreement number: 754337). https://euroexa.eu/

[2] We do mention checkpoint/restart and algorithm based fault tolerance (ABFT). However, a thorough analysis of additional software and cross-layer resilience techniques (e.g., distributed hierarchical checkpointing, cross-layer error containment) is part of future work.

The following paragraphs summarise the main steps we took in this study as well as the results of our analysis:

Step 1: Understanding the resilience-related characteristics of applications and systems in the HPC domain

Resilience is a general term that refers to a system's capability to recover from or prevent failures. System resilience is typically subdivided into three separate concepts: Reliability, Availability and Serviceability (RAS), which are defined in detail in Section 2.1.

It is not straightforward to determine what types and degrees of resilience should be provided in a specific HPC system. These depend on the environment and user requirements. It is also important to understand the specific characteristics of HPC systems, especially the impact of different errors and failures on system performance, reliability and availability. This discussion is summarised in Section 3.

Step 2: Analysing the resilience features of state-of-the-art HPC nodes

Current HPC on-node resilience features are defined mainly by US and Asian chip manufacturers and HPC system integrators with decades of experience in building resilient systems. These features, therefore, are the gold standard by which we measure the resilience of any novel HPC node architecture, including the ones developed in Europe. For this reason, we have to understand them well. In Section 4, we itemise state-of-the-art HPC resilience features with an explanation for each one of them.

Step 3: Setting a priority for the implementation for each identified resilience feature

Any resilience feature requires a cost–benefit analysis: comparing the cost of failures, whether avoided or mitigated, in relation to their likelihoods, against the cost of the resilience feature itself.[3] All these costs may be monetary costs, or they may be a loss in performance or an increase in power and energy. The cost of a potential failure can be estimated based on its likelihood and the time and expense of HPC job restart, node reboot and replacement of hardware components. The cost of a resilience feature is due to the additional engineering effort and time to develop it and the need for additional hardware to implement it, such as extra silicon or interconnect links. Such features also have indirect costs, such as runtime performance overheads as well as power and energy overheads, caused by system monitoring, error detection, error correction, checkpointing, etc. Cost–benefit analysis of each resilience feature should be used to prioritise development and implementation on production HPC platforms.

We classify some of the listed resilience features as "MUST HAVE" in production HPC systems. These features are considered essential to be incorporated in a given component or technology in order to make that component or technology a viable building block for a reliable large-scale HPC system. These resilience features are classified as "MUST HAVE" in production systems based on three main criteria:

[3] The cost of a particular resilience feature could be also compared with other resilience mechanisms that would mitigate the failure impact. Re-execution of a failure-affected HPC job could also be considered a resilience mechanism with the corresponding cost.

(1) The resilience features implemented in HPC systems should assure that the failure rate of the system is below an acceptable threshold, representative of the technology, system size and target application.

(2) Given the high cost of the uncorrected errors, if hardware errors occur frequently they should be corrected, at low cost, preferably (where possible) in hardware.

(3) Overheating is one of the main causes of unreliable device behavior. Production HPC systems should monitor the temperature of their components and include mechanisms that prevent overheating while balancing power/energy and performance.

Outcome: The list of the most important HPC resilience features

The main outcome of this document is the selection of the most important resilience features that should have the highest priority during the design and implementation of any EU HPC system. These conclusions are described in Section 5.

2. Terminology

Terms such as *error, failure, fault* or *system repair* have a specific meaning when used in the context of the HPC system resilience. These terms, however, may be used differently in distinct studies, and in academia and industry. Therefore, to mitigate any terminology-related confusion or misunderstanding, we start with a brief summary of the resilience-related terminology that we use in this document.

2.1. Resilience, reliability, availability, serviceability

Resilience is a general term that refers to a system's capability to recover from or prevent failures. System resilience is typically subdivided into three separate concepts: Reliability, Availability and Serviceability (RAS). There are many ways to define RAS; an intuitive definition in the context of an HPC system is as follows:

- **Reliability:** How infrequently is a failure seen in a system?
- **Availability:** How infrequently is the functionality of a system or application impacted by failures?
- **Serviceability:** How well are system failures and their impact communicated to users and service personnel, and how efficiently and non-disruptively can the HPC system or its components be repaired and brought back into service?

2.2. Errors, failures, faults, defects

Most of the previous studies use the following definitions of computing system failures, errors, faults and defects [14, 15]:

- **Failure** is an event that occurs when the delivered service deviates from correct service. For example, it is expected that a data read from memory delivers correct data stored on a given address; and any deviation from this service is a *failure of the memory device or interface*. This failure, however, does not have to imply disruption in service of the affected node. In case of a failing memory device, for example, Error Correcting Code (ECC) can correct the errors so the HPC node keeps delivering correct service with no negative impact on the behavior of the running applications. Failures can be *detected* or *undetected,* as detailed in Section 3.3.
- **Error** is the deviation of the system state (seen externally) from its correct service state. For example, a memory device (e.g., DIMM) can deliver data to the memory controller that is inconsistent with the stored ECC. In this case, the memory controller will flag a memory error. Errors can be further divided into two groups: *corrected* and *uncorrected,*[4] as detailed in Section 3.2.

[4] The errors that are not corrected can be referred to as uncorrected (Arm) or uncorrectable (Intel, HPC system integrators). In this document we use the term uncorrected errors.

- **Fault** is the adjudged or hypothesized root cause of an error. For example, the cause of a DRAM error could be a particle impact or a defect in the memory cell or circuit. Not all faults lead to errors — errors get manifested only if faults change the system state, e.g. an arithmetic operation or a data accessed.

- **Defects** are physical sources of faults and errors.

3. The cost of system errors

In order to quantify the impact of system errors, it is important to distinguish between:

(1) Scheduled and unscheduled system outages and repairs

(2) Corrected and uncorrected errors

(3) Detected and undetected failures

3.1. Scheduled vs. unscheduled system outages and repairs

Scheduled system outage repairs are scheduled ahead of time based on failure prediction or observed reduced performance caused by corrected errors. The scheduled system repairs typically require replacement of a single hardware component, most frequently a CPU or a memory DIMM. In this scenario a working component that is predicted to fail or whose performance is below expectations, is replaced with a spare one. The cost of this hardware replacement is in the range of a few node hours of downtime. After the hardware component is replaced, the node is tested. Standard HPC node tests are based on running stress benchmarks over a time period that ranges between a few hours and a few days.

Uncorrected hardware errors lead to **unscheduled outages and system repairs**. Similar to scheduled repairs, unscheduled system repairs usually involve the substitution of a single hardware component with a spare one, and testing of the affected node. The penalty of unscheduled outages, however, comes from unplanned HPC job termination. Since usually such job termination provides no useful outcome, all the node hours from the start of the job until the unscheduled outage is lost.[5] In HPC, a single tightly-coupled job may execute for hours or even days on a large number of nodes, and therefore the penalty of an unscheduled job termination can be significant.

3.2. Corrected and uncorrected errors

It is important to distinguish between corrected and uncorrected system errors and their impact.

Corrected errors: The nodes that are used in HPC systems incorporate advanced Error Correcting Codes (ECC) and protocols that can detect and correct hardware errors. For example, main memory ECC is able to detect and correct multiple corrupted bits in a data word and even handle cases where an entire DRAM chip is corrupted. Data correction is performed in parallel with data reads, **so corrected errors effectively have no impact on system performance;** although energy consumption may be negatively impacted.

[5] Checkpoint/restart approach would reduce the cost of job termination; the penalty would cover only the node hours from the last saved checkpoint until the unscheduled outages.

Uncorrected errors: If the detected error in a component cannot be corrected with available error correction mechanisms, typically the HPC job has to be terminated with no useful outcome from the execution. The consequence of this is that the node-hours used on the job until error detection are lost. The penalty of such a job termination may be significant (in terms of lost server hours); amounting to monetary, node availability and energy costs.

Also, when an uncorrected error is detected, the node is typically shut down and removed from production until the faulty part has been replaced and the node has been tested.[6] In current systems, the total penalty of uncorrected errors, including the lost server hours, is in the order of tens or hundreds of node hours.

Deferred Errors: A deferred error is a detected error that has not been corrected (i.e. that remains uncorrected), but has been allowed to propagate. However, the propagation is not silent, since the system knows about the error and is able to track it and its impact. Propagation of deferred errors is also referred to as data poisoning.

Overall, uncorrected errors can have a **significant negative impact on system performance, reliability and availability**.

3.3. Detected and undetected failures

Failures that manifest as errors, i.e., deviation of the system state (seen externally) from its correct expected system state, are **detected failures.**

Some of the failures may not be detected with state-of-the-art error detection mechanisms. The **undetected failures** may lead to several outcomes:

- **No impact:** Undetected failures may contaminate obsolete data, i.e., data not used by the application after the failure's occurrence. In this case, the undetected failures have no impact on application outcome or the overall service provided by the HPC system.

- **Incorrect system operation:** Undetected failure may lead to incorrect data being used by the application, e.g., affecting addresses of even binary code. Use of such incorrect data addresses or code may lead to service interruption, e.g., because the application tries to access an address that is invalid in its address space. For such uncorrected errors, the cost caused by service interruptions, system repair and testing is high.

- **Silent Data Corruption (SDC):** Undetected failures may lead to unexpected erroneous application outcomes — the application may execute to the end and give an answer that could even look plausible, but is actually wrong. This scenario may occur if the failure affects data that is later used by the application. Producing undetected incorrect scientific results is considered **more damaging than a service interruption** [16]**.**

[6] Advanced resilience features enable running the node in a degraded mode, e.g. with blacklist memory regions in which uncorrected errors were detected.

4. HPC resilience features

This section summarizes the state-of-the-art of HPC resilience features. The presented material is based on experiences of HPC service providers and on the extensive review of publicly-available industrial documents from **Intel** [17, 18, 19, 20, 21, 22, 23], **Arm** [24, 25, 50, 51], **HPC system integrators** [26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41] and **FPGA vendors** [42, 43, 44, 45, 46, 47].

4.1. Error detection and correction

Error detection: In the context of the EU HPC initiative, we have to guarantee that our error detection coverage is similar to, or better than state-of-the-art HPC solutions. If this would not be the case, it would be very difficult to convince any HPC service provider that the EU HPC system is indeed stable and reliable.

Error detection is important as it is a main pre-requirement of any further error analysis. If we want to perform any quantitative analysis of errors in a production setting, it is strongly preferred that error detection mechanisms have a high coverage. *Collection and analysis of statistics related to failures* is one of the milestones set by the ETP4HPC [5].[7]

In addition, error detection is the main prerequisite for prediction of any future failures; e.g., corrected DRAM errors could be used to predict uncorrected ones. *Prediction of failures and fault prediction algorithms* are two of the milestones set by the ETP4HPC [5, 6].[8,9] A recent survey of the existing pre-failure techniques, however, still shows significant room for improvement in this research area [48, 49].

Error correction: In general, it is important to consider the return-on-investment analysis of any error correction mechanism. However, given the typically high cost of uncorrected errors, it is safe to conclude that highly-probable hardware errors should be detected and corrected at low overhead, preferably (if possible) in hardware. Examples of highly probable errors are DRAM errors that occur at the rate of hundreds of errors per year, per DIMM. For this reason, current HPC systems incorporate advanced hardware mechanisms that correct single-bit and detect multi-bit DRAM errors, so that they do not impact reliability, with only a minimal impact on performance, power and energy and modest cost in additional area on chip.

It is sometimes difficult to distinguish between error detection and error correction mechanisms. The same RAS feature can correct some errors (e.g., single-bit DRAM errors) and only detect some others (e.g., multi-bit DRAM error). Whether the RAS feature performs error detection or error correction depends on the complexity of the error. Therefore, in Table 1, we summarize both error detection and correction features. In addition to error detection and correction, it is important to report errors to the BIOS and operating system, as we summarize in Table 2.

[7] Milestone M-ENR-FT-5; Availability date: 2018.
[8] Milestone M-ENR-FT-6; Availability date: 2019.
[9] Milestone M-ENR-FT-6; Availability date: 2021.

Table 1: Summary of error detection and correction features

Parameter	Description
ECC: Main memory	ECC protects main (system) memory (typically in the form of DRAM) from transient faults that can corrupt program data. The increasing density of DRAM devices (and consequently main memory capacity) increases the likelihood of such faults. An extensive overview of advanced ECC schemes, including single-device data correction (SDDC or Chipkill), double-device data correction (DDDC) and DDDC+1, can be found at the following document: *Memory RAS Configuration. User's Guide. Super Micro Computer Inc,. 2017* [41]. **Category:** Considered "MUST HAVE" in any production HPC system.
Memory Demand and Patrol Scrub	Combined with detection features, like the aforementioned ECC, these features provide the ability to find and correct memory errors, either reactively (demand) or proactively (patrol) addressing memory problems. In all cases, whenever the system detects an ECC error, it will attempt to correct the data and write it back, if possible. When correcting the data is not possible, the corresponding memory is tagged as failed or poisoned. Demand scrubbing is the attempt to correct a corrupted read transaction. Patrol scrubbing involves proactively sweeping and searching system memory and attempting to repair any errors found. Scrubbing also helps prevent the accumulation of single-bit errors to multi-bit errors. Patrol scrubbing errors may activate the Machine Check Architecture Recovery (MCA recovery) mechanism described in Table 4. **Category:** Considered "MUST HAVE" in any production HPC system.
Memory Address Parity Protection	Enables the correction of transient errors on address lines of the memory channel. Traditional parity is limited to detecting and recovering single-bit errors. **Category:** Considered "MUST HAVE" in any production HPC system.
Memory Lockstep	Memory Lockstep lets two memory channels work as a single channel, moving a data word two channels wide and providing eight bits of memory correction. Memory Lockstep provides protection against both single-bit and multi-bit errors. **Category:** Depends on workload, requires additional cost–benefit analysis.

Error detection: CPU caches and registers	Error detection in processor register processor caches. The increasing density, increasing cache sizes and increasing depth of the memory hierarchy in modern processors increases the likelihood of such faults. **Category:** Considered “MUST HAVE” in any production HPC system.
ECC: On-chip memories of FPGAs	Built-in or soft-core ECC is used to protect on-chip memories or Block RAMs (BRAMs) of FPGAs from transient faults. The increasing density of FPGAs increases the likelihood of such faults. In addition, it has been shown that BRAM ECC protects data against soft errors and undervolting faults, too. BRAMs of modern FPGAs are usually equipped with Single-Error Detection and Multiple-Error Correction (SECDED) type of ECC, which inherently fits the type of transient faults, soft errors, and undervolting faults. **Category:** Considered “MUST HAVE” in any production HPC systems.
Protecting Configuration Memory of FPGAs	In SRAM-based FPGAs (most common type), configuration memories are susceptible to different types of faults like Single Event Upset, which can lead to configuration and with that program changes on the FPGA. Dedicated ECC, patrol scrubbing, and redundancy techniques, like Triple Modular Redundancy, can protect configuration bits. **Category:** ECC: Considered “MUST HAVE” in any production HPC systems. Patrol scrubbing and redundancy techniques: Depends on system/workload, requires additional cost–benefit analysis.
Intra-node interconnect: Error detection	Intra-node interconnect refers to hardware and the associated protocols that connect components located on a single node, such as CPUs, hardware accelerators and the I/O hub. **Category:** Considered “MUST HAVE” in any production HPC system.
Intra-node interconnect: Packet Retry	Automatically retransmits packets containing errors. This supports recovery from transient errors. **Category:** Considered “MUST HAVE” in any production HPC systems. Depending on system/workload the packet retry can be performed in hardware or in software (would require additional cost–benefit analysis).

Table 2: Reporting of corrected errors

Parameter	Description
Reporting of corrected hardware errors	This feature reports hardware errors, which the RAS system successfully detected and corrected, to the OS. This reporting is a prerequisite for any quantitative analysis of the system resilience, and predictive failure analysis that would anticipate and avoid future uncorrected errors. **Category:** Considered "MUST HAVE" (or at least "HIGHLY DESIRABLE") in any production HPC system.

4.2. Prevention of uncorrected errors

The objective of the prevention techniques is to avoid unscheduled outages and system repairs due to uncorrected errors. Typical examples are degraded operation modes that reduce the probability of uncorrected errors at the cost of device performance, as summarised in Table 3. Most of the prevention techniques manage device temperature since overheating is one of the main causes for unreliable device behaviour. State-of-the-art production HPC systems monitor component temperature and employ mechanisms to prevent overheating while balancing power/energy and performance. We recommend that HPC systems developed in Europe deploy equivalent techniques.

Table 3: Uncorrected error prevention. Device degraded modes until repairs can be made.

Parameter	Description
Memory Thermal Throttling	The processor monitors memory temperature and can temporarily slow down the memory access rates to reduce temperatures, if needed. It can prevent DIMMs from overheating while balancing power and performance. Optionally, in conjunction with the systems management solution (provided by the HPC system integrator), the system may increase fan or pump speeds as needed to keep memory components operating within acceptable thermal limits. **Category:** Considered "MUST HAVE" in any production HPC system.
Core (Socket) Disable for Fault Resilient Boot	This feature disables a failing core (or socket) at boot time, allowing the system to power on despite the core (socket) failure. **Category:** Depends on system/workload, requires additional cost–benefit analysis.

Dynamic voltage and frequency scaling	Voltage and frequency under-scaling is an effective power reduction technique, applicable for CPUs, FPGAs, ASICs; however, it needs to be dynamically controlled to prevent the system reliability affects. **Category:** Considered “MUST HAVE” in any production HPC system.
Over-temperature shutdown mechanism for FPGAs	The environmental temperature might be out of control in the harsh environments, especially in edge devices where accelerators like FPGAs are commonly used. In these scenarios, over-temperature can potentially damage the device. Technologies like Xilinx Analog-to-Digital Converter in FPGAs provide opportunity to prevent system damage by monitoring the temperature and issuing a safe shutdown signal when temperature goes beyond the limit. **Category**: Considered “MUST HAVE” in any production HPC system.

4.3. Error containment: Limiting error impact

Even with advanced RAS features available in production systems, a failure of a single node (or a single application process) typically causes the termination of the whole job. Error containment is therefore a very important aspect of HPC system resilience and a key requirement for the reliable scale-out of HPC systems and applications. The state-of-the-art error containment features (summarized in Table 4) are, however, fairly complex and may introduce significant overheads. Therefore, before implementing these solutions in the EU HPC production system, it is important to perform a cost–benefit analysis.

Table 4: Uncorrected errors containment

Parameter	**Description**
Corrupt Data Containment Mode	Corrupt Data Containment mode, or tracking data poisoning prevents corrupt data from spreading through the system. Tracking data poisoning, i.e., tagging data that comes from a corrupt memory location, limits the corrupt data to the process currently running, thus preventing the data from affecting the rest of the system. The receiver of the data can check the poison tag and detect whether the data is corrupted. This mechanism requires enhancements in hardware and software layers. **Category:** Depends on system/workload, requires additional cost–benefit analysis.

Tracking Data Poisoning	Tracking data poisoning is a method to defer a detected error, by associating a poison state to the data. In a processor, a poisoned data value may be stored into a cache (e.g. L2 cache). It is considered "consumption" for a processor to use a poisoned data value to alter the non-speculative state. Any attempt to store a poisoned value into a control register or otherwise consume the value rather than storing it or transmitting it must result in an error. **Category:** Depends on system/workload, requires additional cost–benefit analysis.
Machine Check Architecture recovery (MCA recovery)	Allows higher-level software, such as the hypervisor, OS, or MCA recovery-aware application to recover from some data errors that cannot be corrected at hardware level. Memory Patrol Scrub or Last Level Cache Write Back detects these errors. MCA recovery reports the location of the error to the software stack, which then takes the appropriate action. For example, the OS might abort the task owner in response to an error, allowing the system to continue running. **Category:** Depends on system/workload, requires additional cost–benefit analysis.
Checkpointing/restart	Restarts the whole user application from a user-assisted coordinated checkpoint, in most cases taken at a synchronization point. Checkpoint/restart is a widely studied component of global system resilience. Achieving it in a traditional SPMD model is integrated in many applications. Alternatively, external libraries, such as BLCR or SCR can provide the necessary services. Achieving it at the level of tasks is being investigated by task-based programming libraries such as StarPU, OmpSs, and Charm++, seamlessly combining it with heterogeneous HPC system support. The knowledge of the task graph allows the runtime to automatize the selection of data to be saved and the restart at a given point of the graph. Careful design allows the system to restart only the failing nodes. **Category:** Depends on system/workload, requires additional cost–benefit analysis.

Algorithm based fault tolerance (ABFT)	ABFT approaches use algorithmic techniques to detect and correct data loss or data corruption during computation in order to improve the tolerance against faults in the applications themselves. Typical examples are iterative solvers, which can — under some circumstances — overcome wrong data caused by faults by smoothing them out across iterations. In some cases, this can replace or augment hardware approaches, but is typically restricted to specific application kernels and does not protect control structures. **Category:** Depends on workload, requires additional cost–benefit analysis.
Isolation Design Flow for FPGAs	The isolation design flow confirms that the individual regions of FPGA designs are isolated from each other in the event of failure. This can augment (but not fully replace) other resilience techniques for FPGAs and can enhance overall design reliability. Several techniques are embedded in this approach (confirmed for Xilinx) to guarantee the goal above: ● Isolation of test logic for safe removal ● Watchdog alarms ● Segregation by safety level ● Modular redundancy **Category:** Depends on system/workload, requires additional cost–benefit analysis.

4.4. Minimizing the system repair time

The objective of these features is to reduce the time needed to repair the system, and therefore to improve the system serviceability. However, system repairs must have a low-to-moderate impact on the HPC system down time (see Section 3). Hence, it is important to perform the cost–benefit analysis of these RAS features, see Table 5.

Table 5: Minimizing the system repair time

Parameter	**Description**
Failed DIMM Identification	Identifies specific failing DIMM(s), enabling IT support to replace only those DIMMS. **Category:** Depends on system/workload, requires additional cost–benefit analysis.

CPU Hot Add/Swap	Allows the addition of a physical CPU module to a running system. A new CPU can immediately replace a failing CPU via migration or be brought on-line later. **Category:** Depends on system/workload, requires additional cost–benefit analysis.
Memory Hot Add/Swap	Physical memory can be added while the system is running. Added memory can immediately replace failing memory via migration or be brought on-line later. **Category:** Depends on system/workload, requires additional cost–benefit analysis.
PCIe Express Hot Plug	Allows addition or removal of a PCIe card while the system is running. **Category:** Depends on system/workload, requires additional cost–benefit analysis.
Rewriting configuration memory of FPGAs	Using either an internal or external configuration controller to rewrite configuration memory without stopping device operation. **Category:** Depends on system/workload, requires additional cost–benefit analysis.
Dynamic Partial Reconfiguration for FPGAs	Dynamic Partial Reconfiguration can be used to replace part of the FPGA design without interrupting the functioning of other parts. **Category:** Depends on system/workload, requires additional cost–benefit analysis.

4.5. System redundancy

Systems may include spare components that can dynamically, at runtime, take over the operation of failing components, as shown in Table 6. System redundancy reduces the system repair time, assuming a low migration overhead from the failing to the spare component. Also, redundancy can mitigate the uncorrected error impact, e.g., if the failing and spare components work in mirroring mode. System redundancy, however, introduces significant overhead in hardware and operational costs as well as performance degradation. Therefore, redundancy is typically not applied on production HPC systems, at least not at the level of nodes.

Table 6: System redundancy

Parameter	**Description**
Processor Sparing and Migration	Enables the dynamic and proactive reassignment of a CPU workload to a spare CPU in the system in response to failing memory or CPU components. The migration, which requires OS assistance, configures the state of a spare CPU socket to match the processor and memory state of the failing CPU. Once the migration is complete, the system can force the failing CPU offline for replacement in the next maintenance cycle. **Category:** Depends on system/workload, requires additional cost–benefit analysis.
Fine Grained Memory Mirroring	A method of keeping a duplicate (secondary or mirrored) copy of the contents of select memory that serves as a backup if the primary memory fails. The Intel Xeon processor E7 family supports more flexible memory mirroring configurations than previous generations allowing the mirroring of just a portion of memory, leaving the rest of memory un-mirrored. The benefit is more cost-effective mirroring for just the critical portion of memory versus mirroring the entire memory space. Failover to the mirrored memory does not require a reboot, and is transparent to the OS and applications. **Category:** Depends on system/workload, requires additional cost–benefit analysis.
Memory Sparing	Allows a failing DIMM or rank to dynamically failover to a spare DIMM or rank behind the same memory controller. When the firmware detects that a DIMM or rank has crossed a failure threshold, it initiates copying the failing memory to the spare. There is no OS involvement in this process. If the memory is in lockstep, the operation occurs at the channel pair level. DIMM and rank sparing is not compatible with mirroring or migration. **Category:** Depends on system/workload, requires additional cost–benefit analysis.

Memory Migration	Moves the memory contents of a failing DIMM to a spare DIMM, and reconfigures the caches to use the updated location so that the system can coherently use the copied content. This is necessary when a memory node fails or the memory node ceases to be accessible. The act of migrating the memory does not affect the OS or the applications using the memory. Typically, this operation is transparent to the OS. Because in some cases OS assistance improves performance, OS-assisted memory migration is also available. **Category:** Depends on system/workload, requires additional cost–benefit analysis.

5. Conclusions and future work

Building resilient systems is one of the greatest challenges of HPC in Europe. For this reason, the EU HPC community should carefully allocate the available resources and expertise to address the most important requirements for resilience.

This document aims to spearhead a Europe-wide discussion on HPC system resilience and to help the European HPC community to define best practices. We prioritise a wide range of approaches for resilience based on three main principles:

(1) The resilience features implemented in HPC systems should assure that the failure rate of the system is below an acceptable threshold, representative of the technology, system size and target application.
(2) Given the high cost of the uncorrected errors, if hardware errors occur frequently they should be corrected, and corrected at low cost, preferably (if possible) in hardware.
(3) Overheating is one of the main causes of unreliable device functioning. Production HPC systems should prevent overheating while balancing power/energy and performance.

The main outcome of this document is the recommendation of the following resilience features as “MUST HAVE” in production HPC systems:

- ECC in main memory
- Memory demand and patrol scrub
- Memory address parity protection
- Error detection in CPU caches and registers
- Error detection in the intra-node interconnect
- Packet retry in the intra-node interconnect
- Reporting corrected errors to the BIOS or OS (system software requirement)
- Memory thermal throttling
- Dynamic voltage and frequency scaling for CPUs, FPGAs and ASICs
- Over-temperature shutdown mechanism for FPGAs
- ECC in FPGA on-chip data memories as well as in configuration memories

Development and implementation of the remaining state-of-the-art resilience features should only be done based on an additional cost–benefit analysis. These types of analysis depend on the targeted workload and require significantly more work on modelling, simulation and measuring resilience.

This document is the first output of the ongoing European HPC resilience initiative. We focus on the resilience features of HPC nodes, covering the CPU, memory and intra-node interconnect, as well as emerging FPGA-based hardware accelerators. With community support and based on feedback, we will update the analysis and expand scope to include interactions among multiple levels of the software stack and to cover other types of accelerators, as well as networks and storage.

References

[1] S. Alowayyed, D. Groen, P. V. Coveney and A. G. Hoekstra. “Multiscale computing in the exascale era”. *Journal of Computational Science,* pp. 15-25, September 2017.

[2] F. Chen, W. Ge, L. Guo, X. He, B. Li, J. Li, X. Li, X. Wang and X. Yuan. “Multi-scale HPC system for multi-scale discrete simulation—Development and application of a supercomputer with 1 Petaflops peak performance in single precision”. *Particuology,* pp. 332-335, 2009.

[3] J. Luttgau, S. Snyder, P. Carns, J. M. Wozniak, J. Kunkel and T. Ludwig. “Toward Understanding I/O Behavior in HPC Workflows”. In *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, 2018.

[4] European Technology Platform for High Performance Computing. “ETP4HPC Strategic Research Agenda: Achieving HPC Leadership in Europe”. 2013.

[5] European Technology Platform for High Performance Computing. “Strategic Research Agenda: 2015 Update”. 2015.

[6] European Technology Platform for High Performance Computing. “Strategic Research Agenda 2017: European Multi-annual HPC Technology Roadmap”. 2017.

[7] Eurolab-4-HPC. “Eurolab-4-HPC Long-Term Vision on High-Performance Computing”. 2017.

[8] U.S. Department of Energy. “The Opportunities and Challenges of Exascale Computing: Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee”. 2010.

[9] U.S. Department of Energy. “Exascale Programming Challenges: Report of the 2011 Workshop on Exascale Programming Challenges”. 2011.

[10] U.S. Department of Energy. “Preliminary Conceptual Design for an Exascale Computing Initiative”. 2014.

[11] U.S. Department of Energy. “Top Ten Exascale Research Challenges: DOE ASCAC Subcommittee Report”. 2014.

[12] F. Cappello, A. Geist, W. Gropp, L. Kale, B. Kramer and M. Snir. “Toward Exascale Resilience”. 2009.

[13] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer and M. Snir. “Toward Exascale Resilience: 2014 Update”. 2014.

[14] A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr. “Basic concepts and taxonomy of dependable and secure computing”. *IEEE Transactions on Dependable and Secure Computing,* pp. 11-33. 2004.

[15] S. Mukherjee. Architecture Design for Soft Errors. Morgan Kaufmann, 2008.

[16] A. Geist. “How To Kill A Supercomputer: Dirty Power, Cosmic Rays, and Bad Solder”. *IEEE Spectrum,* 2013.

[17] Intel Corporation. “Intel Xeon Processor E7 Family: Reliability, Availability, and Serviceability Advanced data integrity and resiliency support for mission-critical deployments”. 2011.

[18] Intel Corporation. “Intel Xeon Processor E7-8800/4800/2800 v2 Product Family Based Platform Reliability, Availability and Serviceability (RAS) Integration and Validation Guide”. 2014.

[19] Intel Corporation, “New Reliability, Availability, and Serviceability (RAS) Features in the Intel Xeon Processor Family,” 2017.

[20] Intel Corporation. “Reliability, Availability, & Serviceability (RAS) of Intel Infrastructure Management. Technologies Feature Support. Feature Brief”. 2017.

[21] Intel Corporation. “Intel Xeon Scalable Platform. Product Brief”. 2017.

[22] Intel Corporation. “Intel Product Quick Reference Matrix – Servers”. 2018.

[23] Intel Corporation. “Intel Xeon Processor Scalable Family. Datasheet, Volume One: Electrical”. 2018.

[24] Arm Limited. “ARM Reliability, Availability, and Serviceability (RAS) Specification ARMv8, for the ARMv8-A architecture profile”. 2017.

[25] Ampere Computing. “Ampere 64-bit Arm Processor. Product brief”. 2018.

[26] Bull/Atos Technologies. “Bullion S4 the most advanced workspace for fast data. Fact sheet”. 2015.

[27] Atos. “Bull Sequana S series. Technical specification”. 2017.

[28] Dell Inc., “Advanced Reliability for Intel Xeon Processors on Dell PowerEdge Servers, Technical White Paper”. 2010.

[29] Dell Inc., “PowerEdge R930”. 2016.

[30] Dell Inc., “Five Ways to Ensure Reliability, Availability, and Serviceability in Your Enterprise Environment”. 2016.

[31] Hewlett-Packard Development Company. “Avoiding server downtime from hardware errors in system memory with HP Memory Quarantine. Technology brief”. 2012.

[32] IBM Corp., “Reliability, Availability, and Serviceability. Features of the IBM eX5 Portfolio. Red Paper”. 2012.

[33] Lenovo Press. “Lenovo X6 Server RAS Features”. 2018.

[34] Lenovo Press. “RAS Features of the Lenovo ThinkSystem SR950 and SR850”. 2018.

[35] Lenovo. “ThinkSystem SR950. “Always-on” reliability on x86”. 2018.

[36] Oracle. “Oracle Server X5-4 System Architecture. White paper”. 2016.

[37] Lenovo Press. “Five Highlights of the ThinkSystem SR950”. 2018.

[38] Oracle “Oracle Server X7-2 and Oracle Server X7-2L System Architecture. White paper”. 2017.

[39] Oracle. “Oracle Server X7-2. Data sheet”. 2017.

[40] Oracle. “Oracle Server X7-8 Eight-Socket Configuration. Data sheet”. 2017.

[41] Super Micro Computer, Inc., “Memory RAS Configuration. User’s guide”. 2017.

[42] Xilinx. “Device Reliability Report. UG116 (v10.9)”. 2018.

[43] Xilinx. “7 Series FPGAs Memory Resources. UG473 (v1.12)”. 2016.

[44] Intel/Altera. “Intel Stratix 10 Embedded Memory User Guide, v18.1”. 2018.

[45] Intel/Altera. “AN 737: SEU Detection and Recovery in Intel Arria 10 Devices”. 2018.

[46] Intel/Altera. “AN 711: Power Reduction Features in Intel Arria 10 Devices”. 2018.

[47] Intel/Altera. “Reliability Report (MNL-1085)”. 2017.

[48] D. Jauk, D. Yang and M. Schulz. “Predicting Faults in High Performance Computing Systems: An In-Depth Survey of the State-of-the-Practice”. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2019.

[49] A. Frank, D. Yang, A. Brinkmann, M. Schulz and T. Süss. “Reducing False Node Failure Predictions in HPC”. In *The 26th IEEE International Conference on High Performance Computing, Data, and Analytics*, 2019.

[50] A. Rico, J. A. Joao, C. Adeniyi-Jones, and E. Van Hensbergen. “ARM HPC Ecosystem and the Reemergence of Vectors”. In *Proceedings of the Computing Frontiers Conference (Invited Paper), 2017*.

[51] J. Wanza Weloli, S. Bilavarn, M. De Vries, S. Derradji, and C. Belleudy. “Efficiency modeling and exploration of 64-bit ARM compute nodes for exascale”. *Microprocess. Microsyst.* 53. August 2017.