An Approach to Support the Design and the Dependability Analysis of High Performance I/O Intensive Distributed Systems

Lucas Bressan(✉), Laércio Pioli, Mario A. R. Dantas, Fernanda Campos, and André L. de Oliveira

Programa de Pós Graduação Em Ciência da Computação, UFJF, Juiz de Fora, Brazil
lucasbressan@ice.ufjf.br

Abstract. Frequent service down times and poor system performance can affect aspects such as the availability, quality of experience and generate millions of dollars in lost revenue. High Performance Computing (HPC) environments are often required to comply with performance and dependability requirements. The CHESS methodology provides support for the design and the evaluation of dependability and performance system attributes. In this paper we extend the CHESS methodology to support the design and the dependability analysis of HPC environments. The proposed approach was employed in the Grid'5000, a highly distributed and I/O intensive HPC environment. The application of the proposed approach provided key information for demonstrating dependability, deriving project decisions, agreeing on new design choices and resource allocation strategies.

1 Introduction

Dependability is the ability of a system to operate as intended and to deliver its services when required and in a trusted manner [17]. It is broken down into availability, reliability, safety, security and resilience [21]. Fault tolerance relates to the capability of a system to continue operating as intended, after encountering a failure [13]. Availability is directly related to fault tolerance and refers to the ability of a system to operate continuously by either protecting itself against or quickly recovering from failures [19].

Distributed architectures such as High Performance Computing (HPC) environments are often required to attend to performance and dependability requirements. In certain domains (e.g.: industrial, military, banking and e-health) long service response times, failures and momentary service down times can affect their provided Quality of Experience (QoE) and generate undesirable or even contribute to catastrophic consequences. Thus, HPC environments must ensure their dependability, performance and are sometimes required to implement redundancy, error detection, fault recovery capabilities [6] and provide low I/O times and data exchange latency [22].

L. Barolli et al. (Eds.): 3PGCIC 2020, LNNS 158, pp. 29–40, 2021.
https://doi.org/10.1007/978-3-030-61105-7_4

Analyzing and demonstrating these requirements can prove to be very challenging when dealing with big and complex distributed HPC environments. Thus, compositional analysis and simulation techniques are applied when designing these kinds of environments [9]. CHESS is a methodology and toolset that enables the high level specification (i.e. using UML and SysML constructs) and analysis of system models [7]. Unlike other model-based design (e.g. Matlab & Simulink) and compositional analysis (e.g. Hip-Hops [3]) techniques, the CHESS Toolset is completely free and partially open-source thus, allowing developers to easily extend and adapt it to their specific needs. Due to its maturity and technological readiness, the methodology is used in the industry and applied in domains such as IoT [11], automotive [7] and petroleum [15]. The CHESS Toolset and its State-Based Analysis extension, provide extensive support for the analysis of dependability related attributes such as availability and reliability [5] via the discrete-event simulations of Stochastic Petri Nets (SPN). These results can be further attached to dependability requirements as evidence for their compliance. Furthermore, the results can also be used as basis for agreeing on project decisions and achieving satisfactory levels of performance and dependability. Although SPN-based availability and reliability estimation techniques are constantly applied in the context of distributed architectures [4], most of them do not provide a general purpose, high level system architecture and error behavior specification interface like CHESS does.

In previous work [7], we have extended and applied CHESS towards the generation of safety evidence for the certification of aerospace and automotive systems. Similarly, in this paper, we extend the CHESS methodology even further to support the design and dependability evaluation of HPC environments. The approach is intended to support engineers on deriving new design and additional project decisions by considering dependability analysis results and performance attributes. The feasibility of the approach was evaluated considering the Grid'5000 [2], a highly distributed and I/O intensive HPC environment. The approach successfully supports the demonstration and evaluation of the impacts different environment configurations have on dependability and performance. Furthermore, it also effectively contributes with agreeing on design choices and resource allocation strategies based on dependability, performance and cost attributes.

The rest of this paper is organized as follows: In Sect. 2 we present the related work. Section 3 contains the background. A description of the proposed approach is presented in Sect. 4. Section 5 illustrates the evaluation of the application of the proposed approach in the Grid'5000. Finally, Sect. 6 presents the concluding remarks and future work.

2 Related Work

A few authors have in the past applied or extended the CHESS Methodology, to support the design of systems belonging to various different domains [11]. They have also analyzed the feasibility of the Toolset and of some of its analysis capabilities for evaluating and demonstrating compliance with standards and project-specific dependability requirements [7].

Mazzini et. al. [11] performed a feasibility study of the CHESS Toolset regarding its support for the development, analysis, verification, operation, management and monitoring of mission-critical IoT systems. The authors provide an in depth analysis of the CHESS Toolset capabilities and how they comply with some of the main development life-cycle activities of mission-critical IoT architectures. The authors have concluded that the Toolset offers a holistic solution for the development of IoT environments and provides a meaningful view of the system and its architecture as a whole through its models and the traceability among their artifacts.

In previous work [7], we have analyzed the applicability of the CHESS Methodology for generating certifiable evidence for safety-critical embedded systems. We present a systematic process to support the use of the CHESS Methodology and Toolset. The process covers the production of dependability evidence necessary for the certification of systems of the automotive and aerospace domains. The approach considers the analysis techniques implemented in the CHESS Toolset for the production of such evidence and is evaluated through a realistic automotive Hybrid Braking system. Moreover, we have also provided a mapping between the requirements within the ISO26262, DO-331 and SAE ARP 4754A standards and the activities supported by both the approach and CHESS.

3 Background

3.1 High Performance Computing (HPC)

High Performance Computing (HPC) systems are designed to provide high processing power, low communication and data access latency. HPC environments comprise of complex combinations of hardware, software and large scale distributed and parallel applications e.g.: computing clusters, grids and supercomputers [10].

Due to the need of attending requirements such as low communication and data access latency, I/O performance is very important in HPC systems. Environments such as these are usually divided into computing and storage resources. The constant and massive movements of data between these layers and the different configuration scenarios implemented by computing and storage devices, can affect the I/O performance of these systems [18]. Furthermore, different configurations can also affect dependability attributes such as availability and reliability. Thus, it is very important to evaluate the impact of design choices on dependability, performance and costs before implementing them.

3.2 CHESS and the CHESS Dependability Analysis Plugin

The CHESS Toolset provides support for the specification of system models and the evaluation of their dependability attributes. The Toolset is completely free of charge and is available as an Eclipse IDE extension [5]. CHESS is built upon an open and highly extensible platform thus, allowing developers to easily extend and adapt it according to their specific needs. Initiatives such as the AMASS Project [1] for example, have recently integrated the CHESS Toolset with additional design, validation, certification and dependability assessment solutions.

The Toolset is built upon the CHESS Methodology [12]. The CHESS Methodology enables system design at different levels of abstraction and supports system development phases such as the definition of requirements, system and component level architectural specification and error modeling. CHESS models are specified using the CHESS Modeling Language (CHESS-ML). CHESS-ML extends the traditional UML and SysML modeling languages and provides profiles to enable the specification of high level architectural, error models and dependability analysis scenarios. CHESS supports the analysis of quantitative and qualitative dependability properties. Quantitative properties include metrics such as availability and reliability. Qualitative properties concern aspects such as fault tolerance. The CHESS-ML language alongside the CHESS Methodology ensure the traceability between the model elements thus, providing a meaningful overview of the system as a whole: model elements such as components, requirements, error models, evidence and analysis results can all be traced one to another.

The State-Based Analysis Plugin (CHESS-SBA) [14,16] gathers the information within CHESS models and converts it into Stochastic Petri Nets (SPN). The analysis considers the architectural model and error information within UML State Machines. The semantics of these State Machines are based on the Fault-Error-Failure model and enable the specification of random faults, error states, error modes propagation and repair times. Additional failure annotation strategies such as stochastic and Fault Propagation and Transformation Calculus (FPTC) are also supported by the plugin.

CHESS-SBA calculates dependability metrics such as Availability and Reliability, by performing discrete-event simulations [15]. Reliability describes the probability a system remains healthy continuously from $t = 0$ to $t = x$. Availability is divided into instantaneous and averaged. Instantaneous availability gives the probability a system is healthy at $t = x$. Averaged availability, denotes the fraction of time within an interval, a system remains healthy. CHESS-SBA does not require a complete model of the overall system to perform the analysis. Thus, models can be specified in different levels of detail according to project needs.

4 The Proposed Approach

This section describes the approach to support the definition and evaluation of the dependability properties of HPC environments. The proposed approach is incremental and comprises of activities covering the definition of requirements, architectural design, error modeling and dependability analysis. Figure 1 shows the approach steps.

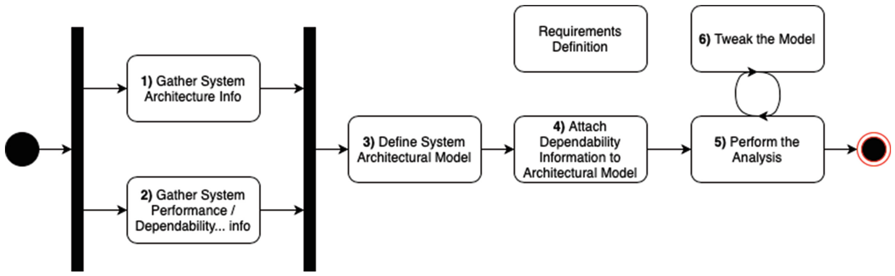
Requirements Definition
6) Tweak the Model
1) Gather System Architecture Info
2) Gather System Performance / Dependability... info
3) Define System Architectural Model
4) Attach Dependability Information to Architectural Model
5) Perform the Analysis

Fig. 1. The proposed approach.

The first two steps **(1,2)** of the proposed approach cover the collection of data regarding system architecture, dependability and performance properties. Additional data such as component costs and maintenance requirements may also be recovered during **step 2**. Such information does not necessarily have to cover the entire system since different parts of the architecture can be modeled and analyzed separately if desired.

The system architecture data, includes information such as the devices used in the architecture and how they communicate with each other e.g.: devices used in the computation and storage nodes. System dependability information covers the error information considered during the analysis and it can be gathered in a couple of different ways. One way to do it is by analyzing the data within system error logs. Such data requires the system to be up and running and therefore, must be collected from setups that are similar to the intended one. Information such as system up/down times and the number of times a certain piece of software/hardware has failed throughout a time span can be attached to model components as dependability information. Dependability information can also be gathered from device specifications e.g.: Mean Time Between Failures (MTBF) or experiments. Additional information such as performance, maintenance and implementation costs can also be gathered and used alongside dependability information later when agreeing on how to configure the system and allocate its resources.

Once having all the necessary information collected, the system model can now be specified using UML, SysML and CHESS-ML constructs. The next couple of steps cover the specification of the system architecture according to the CHESS Methodology and the attachment of failure information to model elements **(steps 3 and 4)**. System specification can be performed in many different ways and abstraction levels. As previously mentioned in Sect. 3.2, the CHESS Methodology provides means for the definition of architectures on system, component and deployment levels. The CHESS-SBA plugin work with models belonging to all those levels and thus, it is completely up to the system designer to decide the complexity and the level of detail of their to-be-analyzed system model.

The next two steps **(5,6)** include the analysis of the dependability attributes, their evaluation and the estimation of the model tweaks necessary to achieve the desired system requirements. Such requirements can be defined at any stage of the process and may relate to fault tolerance, availability, reliability, performance and costs. System designers can use the dependability information of different system configurations alongside performance and cost data to estimate the necessary model optimizations. Increasing dependability is usually costly and does not always translates into better system performance or an expressive gain in availability. Therefore, system engineers must analyze all these variables together and agree upon the tweaks necessary so the considered system satisfies its requirements. Furthermore, dependability analysis results can be attached to the model as dependability evidence and used to argue over and agree on project decisions e.g.: design choices, resource allocation and planned maintenance.

5 Evaluation

In order to analyze how the proposed approach can support the derivation of project decisions based on its provided information, we have performed an evaluation aimed towards answering the following research question: **RQ:** How does the information provided by the proposed approach can support project managers on identifying and agreeing upon project decisions based on dependability/performance requirements and analysis results? The evaluation was conducted considering the architectural, performance and dependability information gathered from the Grid'5000.

5.1 Experimental Environment

The Grid'5000 [2] is a highly distributed and High Performance Computing environment. It is spread across 8 different locations along France and comprises of a total of 800 nodes grouped into clusters. These clusters are variant-intensive and may implement a different set of solutions from each other e.g: CPUs, GPUs, storage and network communication devices.

In this study we have considered the characteristics of the computation and storage nodes within the Dahu cluster, located in Grenoble. The considered group of nodes include Dell PowerEdge C6420 servers, interconnected via a Gigabit Ethernet network environment. These nodes implement a pair of Intel Xeon Gold 6130 2.10GHz/16 core CPUs and include a total of 192GB of RAM each. As for storage solutions, the nodes may contain 240GB/480 GB SATA SSDs and 4.0 TB SATA HDDs. The nodes are running under CentOS7 (kernel v3.10.0–957.21.2.el7.x86_64) and use the ext4 file system [18].

5.2 Execution

It is already known for a fact, that Solid State Drives (SSDs) generally provide better availability, reliability and I/O performance when compared to mechanical Hard Disk Drives (HDDs). This is however, only the tip of the iceberg.

Different data scheduler choices and storage configurations may also impact on these attributes. That being said, is it always worth it to adopt a SSD exclusive architecture for both data and meta-data storage? Do the dependability and performance gains related to the adoption of a SSD exclusive architecture always justify the extra costs compared to adopting a hybrid storage approach? When using a hybrid architecture, will certain services still require their data to be stored exclusively on SSDs due to their specific performance, availability and reliability needs?

For this evaluation, we have considered a total of 9 different Grid'5000 scenario configurations. Each one implementing a different set of storage device and I/O scheduler combinations. Three different approaches to store data and metadata were considered during the evaluation. In the first scenario, both data and metadata are stored in HDDs. In a different scenario, data is kept into HDDs while metadata is stored into SSDs. In the last scenario, both data and metadata are stored into SSDs. When it comes to the selection of different Linux I/O schedulers, different experiments were performed considering the Complete Fairness Queueing (CFQ), Deadline and Noop schedulers. Dependability properties such as availability, reliability and fault tolerance can all be evaluated through the State-Based Analysis (CHESS-SBA) CHESS Plugin. As previously mentioned in Sect. 3.2, the analysis considers error information specified in UML State Machines and other types of failure annotations.

Before annotating components with quantitative failure data, we must first analyze the different scenarios and the failure probabilities for each component depicted in the model. When it comes to storage devices for example, failures within them may also affect overall system dependability. Therefore information regarding their failure rate distributions are necessary to provide an appropriate description of their failure behavior. Such information can be gathered through previous experiments, system logs or through the device manufacturers themselves. Since we don't have data regarding the failure rates or component up/down times of the storage device arrays within the storage nodes, we will be relying on the failure rates provided by previously published experiments. Even though device manufacturers provide metrics such as the Mean Time Between Failures (MTFB) claiming that their hardware offers an average of five years of continuous operation, experiments have shown that storage device failure rates go way beyond that. According to previously published studies [8,20], Hard Disk failure rates follow a Weibull distribution and can present an annual failure rate of 5% after 2 years of continuous operation. Such value can go up as high as 10% after 5 and 18% after 10 years of use. When it comes to SSDs however, the failure probabilities tend to be constant and lay around 0.10% a year therefore, following an Exponential failure distribution. Figure 2 shows the failure information model describing the failure behavior of Mechanical Hard Disks.

The state machine attached to the HDD component describes its behavior upon encountering both internal and external faults. As earlier mentioned, the internal failure rates of mechanical disks follow a Weibull distribution. Therefore, the distribution wei(0.54,2E6) describes the probability of the component failing

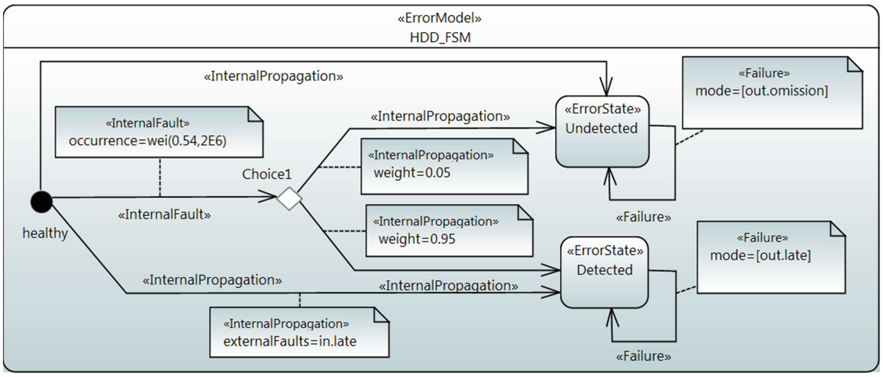

Fig. 2. Error model state machine describing the failure behavior of the HDD component.

randomly over time. Once failing, there is a 95% chance that the Hard Drive failure will be detected by the system and that a redundant backup drive will kick right into its place. Such automatic detection and device substitution may take a fraction of time and may cause delays in I/O requests. Furthermore, there is also a 5% probability that the failure goes undetected. In that case, the Hard Disk will stop responding to any requests until replaced or fixed manually.

When it comes to performance regarding the Grid'5000, different storage device/scheduler configurations may generate different average I/O latency values. Long response times regarding I/O operations can affect not only attributes related to system performance but also, those related to dependability. Depending on the application domain, a system might as well be considered unfeasible if it is not capable to provide its requested services with short time response delays. Previous experiments regarding the Grid'5000 have analyzed and made public the average read and write times per storage device/scheduler configuration [18]. Table 1 lists the average read and write times considering the different scenarios within the Grid'5000.

Table 1. Average I/O times per storage node and scheduler configurations.

Conf	CFQ(R)	CFQ(W)	Deadline(R)	Deadline(W)	Noop(R)	Noop(W)
HDDHDD	0.3543	2.0151	0.3468	2.0183	0.3637	2.2576
HDDSSD	0.2560	2.0871	0.2707	1.9146	0.2493	1.8570
SSDSSD	0.2815	0.6827	0.2841	0.7274	0.3087	0.6839

Once having the availability of the clusters containing different storage node configurations (i.e.: HDDs, SSDs and SSDs and HDDs), analyzing and comparing the results obtained through the State-Based Analysis, we were able to observe that the cluster that achieved the highest overall availability was the one storing both data and metadata on SSDs. Figure 3 shows the analysis results for each considered scenario. The graph shows the fraction of time each configuration has not failed within a year.

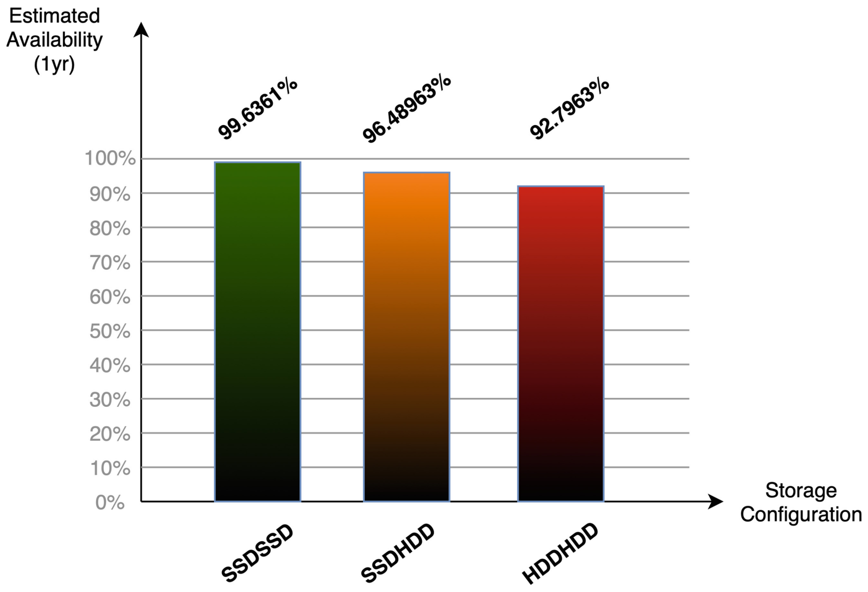

Fig. 3. State-based analysis results.

The obtained availability results however, can be improved by planning recurrent system checks and maintenance. Figure 4 shows the impact planned recurrent maintenance has on the availability of each configuration.

As an answer to **RQ**, we can say that, by considering the dependability data listed on Figs. 3 and 4 and the I/O performance information listed on Table 1, project managers can evaluate and analyze their options when it comes to efficiently setting up the environment in many ways. If what they are looking for is high availability, very short I/O times and the reduced need of recurrent maintenance, then going for a SSD exclusive architecture and using the storage nodes with SSDs for both data and metadata and the CFQ scheduler could be a good idea. If they still need to provide a decent amount of availability and performance and are limited by costs, then they may be better off with the hybrid SSD / HDD approach. Even though it is known for a fact that SSDs generally provide better I/O response times and dependability, investing in 100% SSD storage architectures may not always the most efficient option especially, when considering costs. As the results have shown, it is still possible to get pretty decent I/O times and dependability, through the adoption of a hybrid HDDSSD storage architecture. Furthermore, a hybrid architecture may also allow a smarter

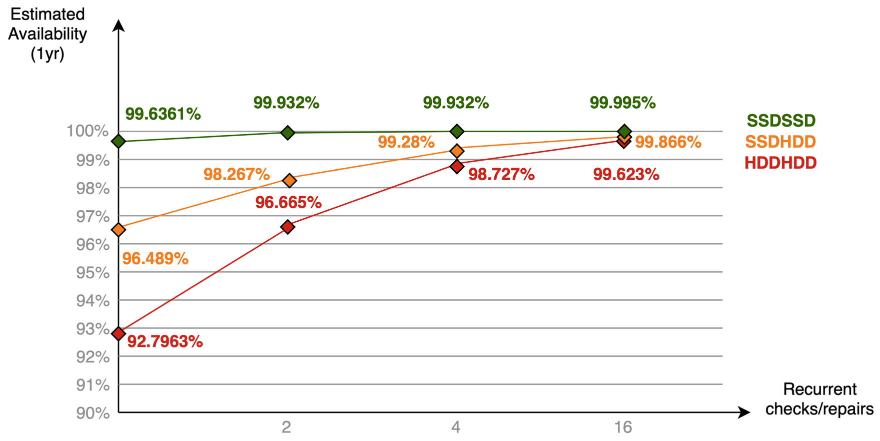

Fig. 4. Impact of recurrent maintenance on system availability.

resource allocation according to each service specific needs. If a certain job or service requires a high availability environment to execute and shorter amounts of time to perform its I/O operations, then such job, its data and metadata, could be allocated to SSD storage nodes only. If not, then it might as well have its data and metadata allocated to HDD and SSD storage nodes respectively.

6 Conclusions and Future Work

This paper has presented an approach to support the design and the dependability evaluation of HPC environments. The approach comprises of an extension of the CHESS methodology and was successfully applied and evaluated considering a highly distributed and I/O intensive HPC environment, the Grid'5000.

As a result, we observed that the methodology successfully provides support for the dependability evaluation of distributed systems. It does not only supports the estimation and demonstration of attributes such as failure behavior, availability and reliability, but also, the impact project choices may have upon them. Furthermore, we have also demonstrated how the analysis results can be combined with additional information such as performance data and provide assistance for deriving and agreeing upon new project decisions. These combined results can be used by engineers to derive new project decisions and agree on new design choices and resource allocation strategies.

As future work, we will gather further information from the Grid'5000, e.g. its up/down times and failure detection and mitigation mechanisms. This information is going to be used to enrich the current Grid'5000 model with more detailed data about its dependability properties. A more accurate model will provide us with more precise dependability metrics and help determining new strategies and requirements regarding risk mitigation and resource allocation.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, developed under the INRIA ALADDIN development action with the support of several Universities and funding bodies (see https://www.grid5000.fr for more info). We would also like to thank the Federal University of Juiz de Fora (UFJF), CNPq and CAPES for providing financial support to this study.

References

1. AMASS: architecture-driven, multi-concern and seamless assurance and certification of cyber-physical systems. https://www.amass-ecsel.eu
2. Grid'5000. https://www.grid5000.fr/w/Grid5000:Home
3. Adachi, M., Papadopoulos, Y., Sharvia, S., Parker, D., Tohdo, T.: An approach to optimization of fault tolerant architectures using HiP-HOPS. Softw. Pract. Experience **41**(11), 1303–1327 (2011). https://doi.org/10.1002/spe.1044
4. Chattaraj, D., Sarma, M., Samanta, D.: Stochastic petri net based modeling for analyzing dependability of big data storage system. In: Abraham, A., Dutta, P., Mandal, J.K., Bhattacharya, A., Dutta, S. (eds.) Emerging Technologies in Data Mining and Information Security, pp. 473–484. Springer Singapore, Singapore (2019)
5. Cicchetti, A., Ciccozzi, F., Mazzini, S., Puri, S., Panunzio, M., Zovi, A., Vardanega, T.: CHESS: a model-driven engineering tool environment for aiding the development of complex industrial systems. In: Proceedings - 2012 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012, pp. 362–365 (2012). https://doi.org/10.1145/2351676.2351748
6. Dave, S.: Fault-tolerance techniques in distributed systems. J. Soc. Instrum. Control Eng. **31**(7), 775–780 (1992). https://doi.org/10.11499/sicejl1962.31.775
7. De Oliveira, A.L., Bressan, L., Montecchi, L., Gallina, B.: A systematic process for applying the CHESS methodology in the creation of certifiable evidence. In: Proceedings - 2018 14th European Dependable Computing Conference, EDCC 2018, pp. 49–56 (2018). https://doi.org/10.1109/EDCC.2018.00019
8. Gupta, V., Kaur, B.P., Jangra, S.: An efficient method for fault tolerance in cloud environment using encryption and classification. Soft Comput. **23**(24), 13591–13602 (2019). https://doi.org/10.1007/s00500-019-03896-6
9. Khanghahi, N., Ravanmehr, R.: Cloud computing performance evaluation: issues and challenges. Int. J. Cloud Comput. Serv. Architect. **3**(5), 29–41 (2013). https://doi.org/10.5121/ijccsa.2013.3503
10. Lucas, R., Ang, J., Bergman, K.: DOE Advanced Scientific Computing Advisory Subcommittee (ASCAC) Report: Top Ten Exascale Research Challenges. Technical report, United States (2014). https://doi.org/10.2172/1222713. https://www.osti.gov/servlets/purl/1222713
11. Mazzini, S., Favaro, J., Baracchi, L.: A model-based approach across the IoT lifecycle for scalable and distributed smart applications. In: Proceedings IEEE Conference on Intelligent Transportation Systems, ITSC October 2015, pp. 149–154 (2015). https://doi.org/10.1109/ITSC.2015.33
12. Mazzini, S., Favaro, J., Puri, S., Baracchi, L.: CHESS: an open source methodology and toolset for the development of critical systems. CEUR Workshop Proc. **1835**, 59–66 (2016)

13. Menychtas, A., Konstanteli, K.G.: Fault detection and recovery mechanisms and techniques for service oriented infrastructures, pp. 259–274 (2011). https://doi.org/10.4018/978-1-60960-827-9.ch014
14. Montecchi, L.: CHESS-SBA: state-based analysis. https://github.com/montex/CHESS-SBA
15. Montecchi, L., Gallina, B.: SafeConcert: a metamodel for a concerted safety modeling of socio-technical systems. In: 5th International Symposium on Model-Based Safety and Assessment (IMBSA 2017), LNCS, vol. 10437, pp. 129–144. Trento, Italy (2017)
16. Montecchi, L., Gallina, B.: SafeConcert: a metamodel for a concerted safety modeling of socio-technical systems. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) LNCS, vol. 10437, pp. 129–144 (2017). https://doi.org/10.1007/978-3-319-64119-5_9
17. Pan, Y., Hu, N.: Research on dependability of cloud computing systems. In: 2014 10th International Conference on Reliability, Maintainability and Safety (ICRMS), pp. 435–439 (2014). https://doi.org/10.1109/ICRMS.2014.7107234
18. Pioli, L., de Andrade Menezes, V.S., Dantas, M.A.R.: Research characterization on I/O improvements of storage environments. In: Lecture Notes in Networks and Systems, vol. 96, pp. 287–298 (2020). https://doi.org/10.1007/978-3-030-33509-0_26
19. Schmidt, K.: High Availability and Disaster Recovery: Concepts, Design, Implementation, 1st edn. Springer Publishing Company, Berlin (2010)
20. Schroeder, B., Gibson, G.A.: Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you. In: FAST 2007 - 5th USENIX Conference on File and Storage Technologies (2007)
21. Sommerville, I.: Software Engineering, 10th edn. Pearson (2015)
22. Wu, J., Liang, Q., Bertino, E.: Improving scalability of software cloud for composite web services. In: CLOUD 2009 - 2009 IEEE International Conference on Cloud Computing, pp. 143–146 (2009). https://doi.org/10.1109/CLOUD.2009.75