Blockchain as a service environment: a dependability evaluation

Leonel Correia (leonelfeitosa@ufpi.edu.br)
Federal University of Piauí
Jamilson Ramalho (jrd@cin.ufpe.br)
Federal University of Pernambuco
Francisco Airton Silva (faps@ufpi.edu.br)
Federal University of Piauí

Research Article

Keywords:

DOI: https://doi.org/

Additional Declarations: No competing interests reported.

J. of Supercomputing manuscript No.
(will be inserted by the editor)

Blockchain as a service environment: a dependability evaluation

Leonel Feitosa Correia ± · Jamilson Ramalho Dantas ∗ · Francisco Airton Silva ±

Received: DD Month YEAR / Accepted: DD Month YEAR

Abstract Blockchain has become an important processing paradigm in recent years. The blockchain supports financial transactions and validates contracts, documents and data. However, the evolution of blockchain has become viable for many applications. The servers' availability and reliability (dependence) are required in the data processing. The contract will only be signed if there are enough components to form the blockchain blocks. This paper analyses the dependency between project components that use blockchain. We present a model based on stochastic Petri net (SPN) for evaluating the dependency of the blockchain architecture. The Design of Experiments (DoE) method was used to analyze this model's factors, seeking to know which ones had the higher impact on the system. The sensitivity analysis showed that the MongoDB component has a greater impact on the system dependency and the need to upgrade such a component. Also, for reliability, making component improvements is unnecessary if the system has fewer than 36,000 hours of runtime.

Keywords smart city, dependability, fault tree, Markov chain, resource redundancy

Leonel Feitosa Correia (L. C.) and Francisco Airton Silva(F.A.S.) (corresponding author) are with
Laboratory of Applied Research to Distributed Systems (PASID), Federal University of Piauí (UFPI), Picos, Piauí, Brazil; R. Cícero Duarte, nº 905 - Junco, 64607-670 E-mail: leonelfeitosa,faps@ufpi.edu.br

Jamilson Ramalho (J.R.) is with
Centro de Informática, Universidade Federal de Pernambuco, Piauí, Brazil E-mail: jrd@cin.ufpe.br

1 Introduction

The monetary system has evolved through various forms of payment. All the payment methods aim to protect buyers and sellers, including various types of fraud such as malware proliferation, spam and theft [1, 2]. Blockchain allows you to store your money safely without paying high maintenance fees to the bank and with a less bureaucratic system [3]. However, this implementation process is complex, and this transformation poses considerable challenges, such as financial stability, supervision and regulation. In the context of financial technologies, blockchain has been adopted in several countries. Japan has adopted blockchain since 2017 for bidding processes to unify all property and land records in urban areas.

Usually, blockchain technology needs to use a cloud computing environment to host and manage data distribution services. However, this environment can present flaws and difficulties in providing services that can take a longer time to perform transactions [4]. However, the availability and reliability of cloud computing systems are of great importance to those planning to contract, deliver or share through these distributed system environments [5]. Thus, evaluating the availability and reliability of blockchain architectures is a complex task. This paper mainly aims to evaluate reliability and availability through the simulation of a blockchain network as a data distribution service through stochastic Petri net models with dependency.

Petri Nets [6] are a family of formalisms very well suited for modelling several system types since concurrency, synchronization, communication mechanisms, and deterministic and probabilistic delays are naturally represented. This work adopts a particular extension, namely, Stochastic Petri Nets [7], which allows the association of stochastic delays to timed transitions, and the respective state space can be converted into CTMC [8]. SPN models present a strong mathematical foundation and are suitable for representing and analyzing parallel systems with heterogeneous components that exhibit concurrency and synchronization aspects. In SPNs, Places are represented by circles, whereas transitions are depicted as filled rectangles (immediate transitions) or hollow rectangles (timed transitions). Therefore, this formalism represents a superior choice to model cloud computing systems.

We have evaluated the Hyperledger Cello, one popular project hosted by Hyperledger and managed by the Linux Foundation. Our behavioural models describe the entire infrastructure's components, relationships, and dependencies. The research contributions are relevant to project managers and organizations planning to offer a blockchain network for data distribution. However, knowing the limits of availability and reliability allows service providers to apply techniques to increase system availability and reliability, such as redundancy and preventive maintenance. Therefore, the main contribution of this paper is to present a proposal with analytical models to evaluate the availability and sensitivity analysis for a Hyperledger Cello platform based on the existing components in the proposed architecture.

The remaining of this paper is divided as follows: Section 2 describes the related works. Section 3 presents the architecture of the analyzed system. Section 4 shows the adopted methodology. Section 5 presents the extended SPN models. Section 6 presents the results obtained in the simulations. Finally, Section 7 concludes the work and presents future work.

2 Related Work

This section presents a state of art survey related to this work's proposal. Table 1 summarizes a comparison between the works that are close to this research, highlighting the main differences of our proposal. Ten works were raised, the oldest being from 2016. We looked for related work that would be related to the Blockchain field. Second, the criterion of analytical model usage was included. Finally, in a deeper way, we sought to select works that focused on architectural issues specifically linked to computational aspects (mainly data processing).

Regarding **context**, few works have explored the theme related to blockchain associated with analytical models. Therefore, works were selected that use stochastic Petri nets, generalized SPN and DRBD to evaluate computational communication systems that use reliability and performability metrics. Melo et al., [9] in their research conducted a feasibility assessment for a blockchain infrastructure as a service and helps those planning to deploy or sell blockchains. A modelling methodology based on Dynamic Reliability Block Diagrams (DRBD) is adopted to assess two reliability attributes: system reliability and availability. Rodrigues et al. [10] present an approach based on generalized stochastic Petri nets (GSPN) to evaluate the performance of private cloud computing environments that adopt NoSQL DBMS as a storage system. Models are presented to jointly estimate throughput and availability, which are prominent indicators of QoS. Liu et al. [11] present a new model of a generalized coloured stochastic Petri net (CGSPN) based on IT infrastructures, which reflects the dynamic behaviour and procedure processing service requests under the advanced active-active mechanism.

Jammal et al. [12] propose a cloud scoring system with the SPN model. In contrast, the Petri Net model assesses the availability of cloud application implementations. So illustrating the approach with a use case that shows how you can use the various deployment options to satisfy tenant and cloud provider needs. Zabala et al. [13] present the modelling of a virtual firewall based on SPN to analyze the performance in terms of throughput and delay. Mendonça et al. [14] present an integrated experience-model approach to evaluating cloud-based disaster recovery solutions. They have used SPNs and fault injection experiments to assess availability-related metrics. To demonstrate the approach's feasibility, distinct real-world cloud-based DR solutions (e.g. active/active and active/standby) were modelled and analyzed. Silva et al. [15] propose an SPN modelling strategy to represent method call executions of mobile cloud systems. This approach allows a designer to plan and optimize

MCC environments where SPNs represent system behaviour and drive parallelizable application execution time.

Pinheiro et al. [16] propose a formal framework based on SPN to represent application partitioning at the method call level. The framework considers the network bandwidth available to send and receive tasks to the cloud. Jammal et al. [17] propose a stochastic Petri Net model that captures the stochastic characteristics of cloud services. The model assesses the availability of cloud services and their deployments in geographically distributed data centres. Fé et al. [18] propose a stochastic model to assist cloud planning. The model was validated for a set of significant scenarios by comparing the results of the respective model with those obtained from real system measurements. This model takes as input the auto-scaling configuration parameters and the time between user requests. The proposed model calculates the throughput, the mean response time and the cost of configuring the cloud computing infrastructure. A sensitivity analysis was also performed to identify the impact of parameters on system performance.

Table 1: *Related Work Comparison*

Work	**Model**	**Metrics**	**Sensitivity Analysis**	**Contribution**
[9]	DRBD	Reliability	Yes	Reliability assessment of a Blockchain-as-a-Service Environment
[11]	CGSPN	MRT, utilization	Yes	SPN for cloud service reliability assessment
[10]	GSPN	MRT, utilization, drop rate	No	Evaluation of NoSQL DBMS in a private cloud environment
[17]	SPN	MRT, reliability, service charge	Yes	Availability analysis of cloud-deployed applications
[18]	SPN	MRT, escalation policy, utilization	Yes	Stochastic performance and cost model for planning
[12]	SPN	MRT, utilization	No	High Availability Aware Deployments in the Cloud Context
[13]	SPN	MRT, utilization	Yes	Model of an SPN-Based Virtual Firewall
[14]	SPN	MRT, utilization	Yes	Availability analysis of a recovery solution
[15]	SPN	MRT, utilization	Yes	Mobile cloud benchmarking
[16]	SPN	MRT	Yes	Analysis of performance and data traffic of mobile cloud environments
This work	SPN	Reliability, availability	Yes	Availability and reliability assessment

3 Architecture and Base Models

This section presents the reference architecture for the blockchain system and the SPN model, with details on the execution flow and its base components. The SPN model was proposed to apply a simulation that integrates the formal description, proof of correction, and performance evaluation of the proposed context [15, 19, 20, 21, 22, 23, 24, 25**?**].

Figure 1 illustrates the reference architecture representing the Hyperledger Cello. The environment used to host Hyperledger Cello consists of two nodes, the master node and the worker node, each responsible for running a series of services. The flow starts with the Watchdog, responsible for monitoring the blockchain network service and the system's status. RestServer performs environment provisioning, orchestration and task management. The dashboard provides environment management for system administrators. Docker manages containers and provides the tools needed to run and virtualize applications. Nodes run Docker as a host for Hyperledger Cello. Python also runs on the host by supporting the Watchdog, RestServer, and Dashboard on the Master Nodes. We are considering using a service like Nginx, a reverse proxy used by Hyperledger Cello, to improve web performance. NodeJS is a JavaScript runtime used by Cello to improve provisioning. MongoDB is an open-source distributed database that allows you to query and index data. The Hardware used to run Hyperledger Cello can be a desktop or a virtual machine. The fundamental prerequisite is that the operating system is Linux.

Figure 2 presents two models of a Master and Worker architecture that represent a series of components in a blockchain network. This model deals with an architecture with the minimum requirements to provide a blockchain network on top of the Hyperledger Cello platform. If any of the components fail, the system will not be available, and the service running on the worker will not be accessible from the external infrastructure of the blockchain network.

Master Node The master node is the machine responsible for providing access management to the blockchain network. Through the master node, it is possible to create, delete and define who can share information or see what is coming from one user to another, representing some dependencies. The master contains the hardware, an operating system, MongoDB, Python, node.js, nginx, Docker, Dashboard, RestServer and Watchdog. The hardware component (HW) is the foundation of the entire blockchain architecture containing a direct dependency on the HW. If the HW fail, all software components will fail. To repair a machine that has had a hardware failure, the blockchain architecture repair routine starts with repairing it. After repairing the HW, the operating system (OS) becomes the next component to be repaired. Next, all other software is repaired. Docker should be repaired first, followed by Dashboard, RestServer and Watchdog. The model's focus is to help professionals choose the best architecture configuration for their blockchain system. Table 2 presents the adopted guard expressions.

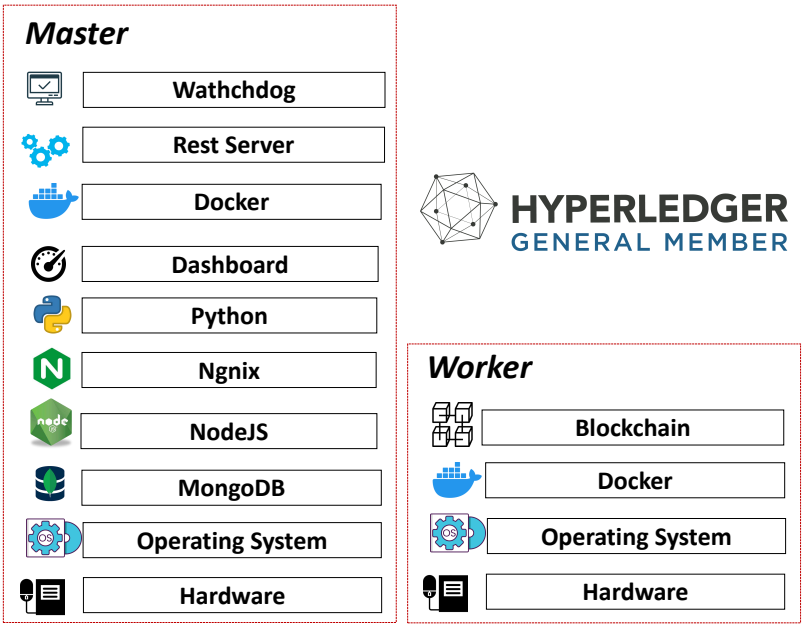

Fig. 1: Illustration of the architecture of a system that uses Hyperledger Cello.

Table 2: Condições de guarda.

Transition	Index	Expression	Description
PYTH_R	[C-01]	(#OS_U>0)	Enabled when OS is active.
NG_R	[C-02]	(#NOD_U>0)	Enabled when NodeJSesta is active.
DOC_R	[C-03]	(#OS_U>0) AND (#MONG_U>0)	Enables when OS and MongoDB are active.
DAS_R	[C-04]	(#NOD_U>0) AND (#MONG_U>0) AND (#NG_U>0)	Enables when the NodeJS, MongoDB and Ngnix are active.
REST_R	[C-05]	(#PYTH_U>0) AND (#NOD_U>0) AND (#NGN_U>0) AND (#DOC_U>0)	Enables when Python, NodeJS, Ngnix and Docker are active.
WAT_R	[C-06]	(#PYTH_U>0) AND (#DOC_U>0)	Enables when Python and Docker are active.

Worker Node Another SPN model is proposed for the Worker Node and presented in Figure 2. This model has fewer components, containing only three elements: the hardware (HW), operating system (OS) and Docker. All system elements have dependency characteristics with the previous component. If the hardware fails, all software components will fail. The failure and recovery times used in the Worker Node were the same used in the Master Node. A blockchain

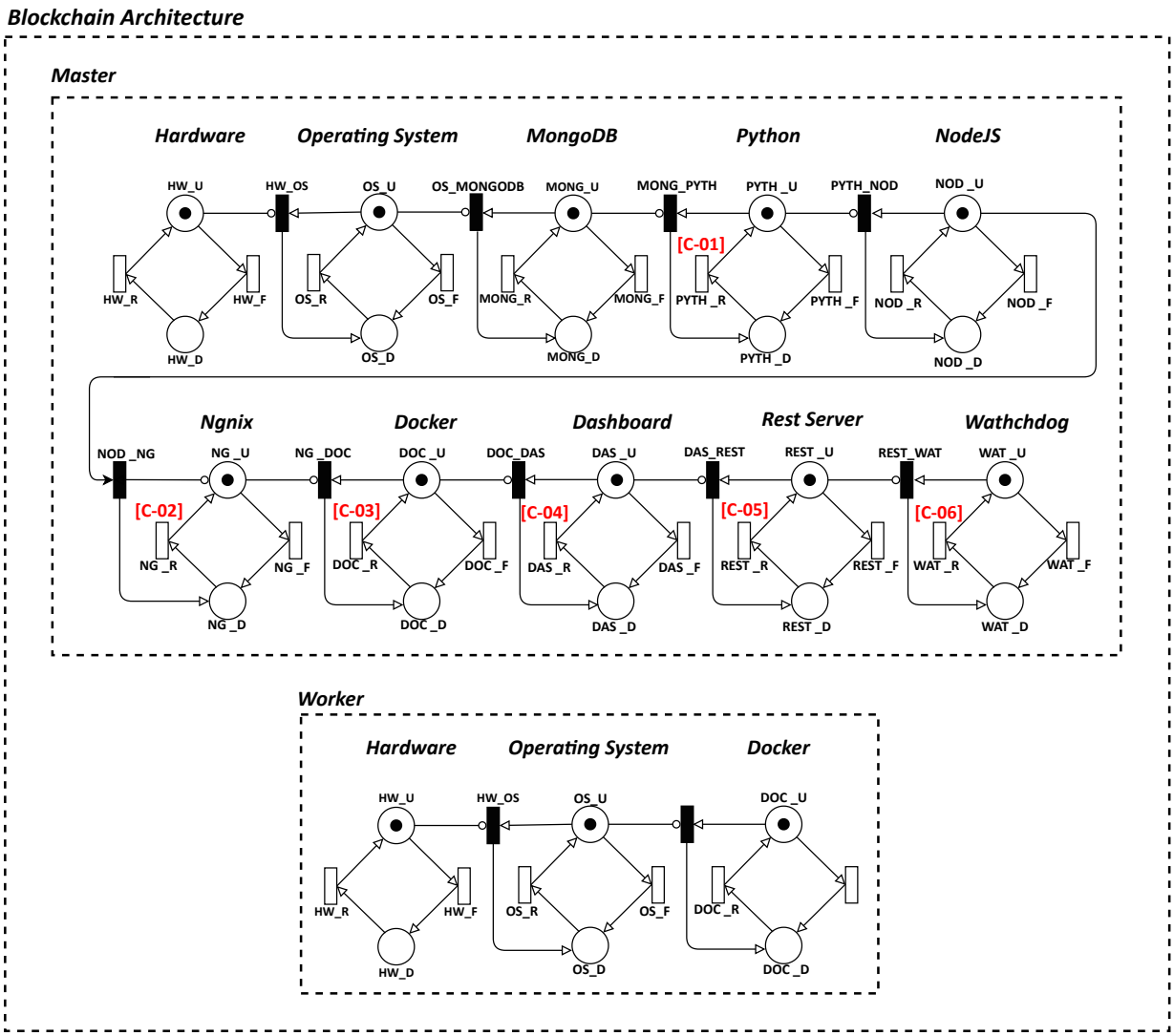

Fig. 2: Base SPN that is composed of the Master and Worker nodes.

system has the characteristics of a P2P system, where the client can also be a server [26].

4 Modeling Methodology

Figure 3 presents a flowchart that summarizes the strategy used in this work as a research methodology composed of eight steps.

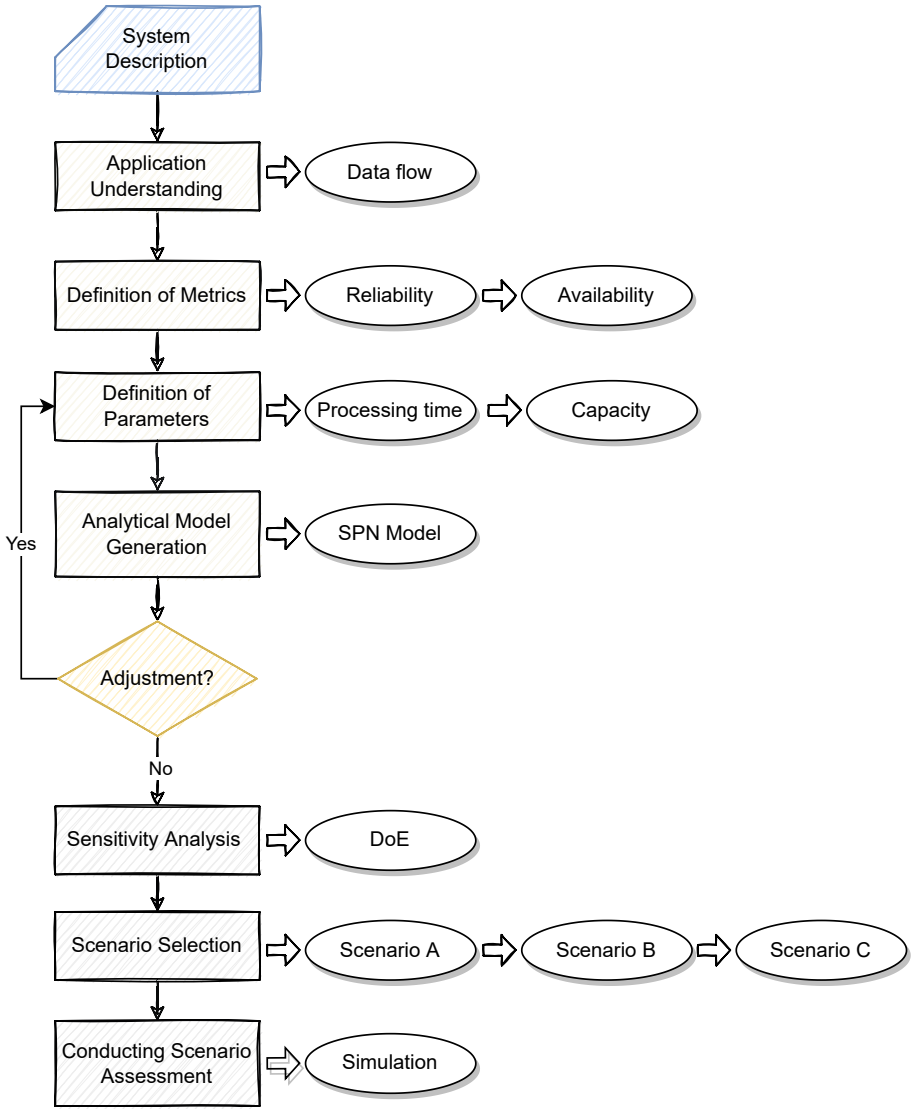

Fig. 3: Modeling Methodology.

Understanding the Application: It is important to understand how the application works, define how many components are involved, and the system's data flow, for example, where the data will be sent after passing through component 'x'. **Metric Definition:** The metrics of interest must be identified, considering the model information to diagnose the system performance. In this work, the selected metrics (reliability and availability) can be important in the end user's perception and useful for the system administrators. **Parameter**

Definition: The parameters that will be inserted in the model are defined here. These parameters define the behaviour and capability of each component's features. **Analytical Model Generation:** A performance model using a Petri net is developed. In this part, it is built considering the defined metrics and parameters and the expected results. The choice of the Petri model is given because the scenario has several components needing a specific level of abstraction. **Sensitivity Analysis:** Using DoE, the analysis presents impacts considering predefined factors and levels. DoE enables us to identify the most relevant factors for the results of the chosen metrics and how the interaction between the factors and variations in their levels impact performance. **Scenario Selection**: Some scenarios are created for performance analysis. This step defines which scenarios can represent the reality of a blockchain system. Scenarios will be chosen to analyze the most important factors considering the sensitivity analysis results. **Performing the Scenario Evaluation**: The constructed scenarios are evaluated using the Petri net model through simulation. In each scenario, the factors are varied, and the metrics will be analyzed, allowing observe which configurations the system performs satisfactorily.

4.1 Sensitivity Analysis

In this work, the Design of Experiments (DoE) was carried out, corresponding to a collection of statistical techniques that deepen the knowledge about the product or process under study [27]. The DoE can be defined by a series of tests in which the researcher changes the variables or input factors to observe output responses. The parameters to be changed are defined using an experiment plan. The objective is to generate the most significant amount of information with the fewest experiments possible. System behaviour based on parameter changes can be observed using output sets. Table 3 presents the factors used in constructing the DoE. The execution of the DoE seeks to identify the factors that most influence the system. In this analysis, MTTF and MTTR were chosen as the dependent variable because it is the most perceptive aspect for the end user.

Table 4 presents the MTTFs and MTTRs for the current case study. Equation 1 shows the expression to calculate the availability. P represents the probability of containing a token in WAT_U and W_DOC_U. DoE was applied considering twenty factors: HW-MTTF, HW-MTTR, OS-MMTF, OS-MMTR, MongoDB-MTTF, MongoDB-MTTR, Python-MTTF, Python-MTTR, NodeJS -MTTF, NodeJS-MTTR, Nginx-MTTF, Nginx-MTTR, Dashboard-MTTF, Dashboard-MTTR, RestServer-MTTF, RestServer-MTTR, Watchdog-MTTF, Watchdog-MTTR, Docker-MTTF, Docker-MTTR. The factors have two levels of variation, with 50% higher and lower.

$$A = P\left\{\left((\#WAT\_U > 0)\, AND\, (\#W\_DOC\_U > 0)\right)\right\} \tag{1}$$

Figure 4 presents the Pareto chart for the factors related to the availability metric. When a factor has a high impact on tests, very different values are

Table 3: Factors and Levels.

Reference	Factors	Levels	
A	HW-MTTF	50%	-50%
B	HW-MTTR	50%	-50%
C	OS-MMTF	50%	-50%
D	OS-MMTR	50%	-50%
E	MongoDB-MTTF	50%	-50%
F	MongoDB-MTTR	50%	-50%
G	Python-MTTF	50%	-50%
H	Python-MTTR	50%	-50%
J	NodeJS-MTTF	50%	-50%
K	NodeJS-MTTR	50%	-50%
L	Nginx-MTTF	50%	-50%
M	Nginx-MTTR	50%	-50%
N	Dashboard-MTTF	50%	-50%
O	Dashboard-MTTR	50%	-50%
P	RestServer-MTTF	50%	-50%
Q	RestServer-MTTR	50%	-50%
R	Watchdog-MTTF	50%	-50%
S	Watchdog-MTTR	50%	-50%
T	Docker-MTTF	50%	-50%
U	Docker-MTTR	50%	-50%

Table 4: Input Values for Blockchain Architecture Components.

Component	MTTF	MTTR
HW	8760 h	100 min
OS	2893 h	15 min
MongoDB	1440 h	20 min
Python, NodeJS, nginx Dashboard, RestServer, Watchdog	788.4 h	1 h
Docker	2990 h	1 h

obtained when changing its level. According to the p-values found, the effects of the MongoDB-MTTF factor have the greatest impact among the factors in this study, followed by Docker-MTTR and OS-MMTF. Therefore, choosing the database with the shortest failure time to use is important for the impact of system availability. Watchdog-MTTF and MongoDB-MTTR have the least impact on the system.

Figure 5 shows the main effects graph for availability. The graph represents the availability to carry out the tests at each level. In this graph, the more horizontal the line, the less influence the factor has, as it means that the different levels of the factor influence the final result similarly. The MongoDB-MTTF, Docker-MTTR, OS-MTTF, Dashboard-MTTR and Python-MTTR factors had the greatest impact.

Figure 6 presents the interaction graph. An interaction occurs when a difference in another factor changes the influence of a certain component on the result. If the lines of the graphs are parallel, there is no interaction between the factors. In general, there was little interaction between the factors. However,

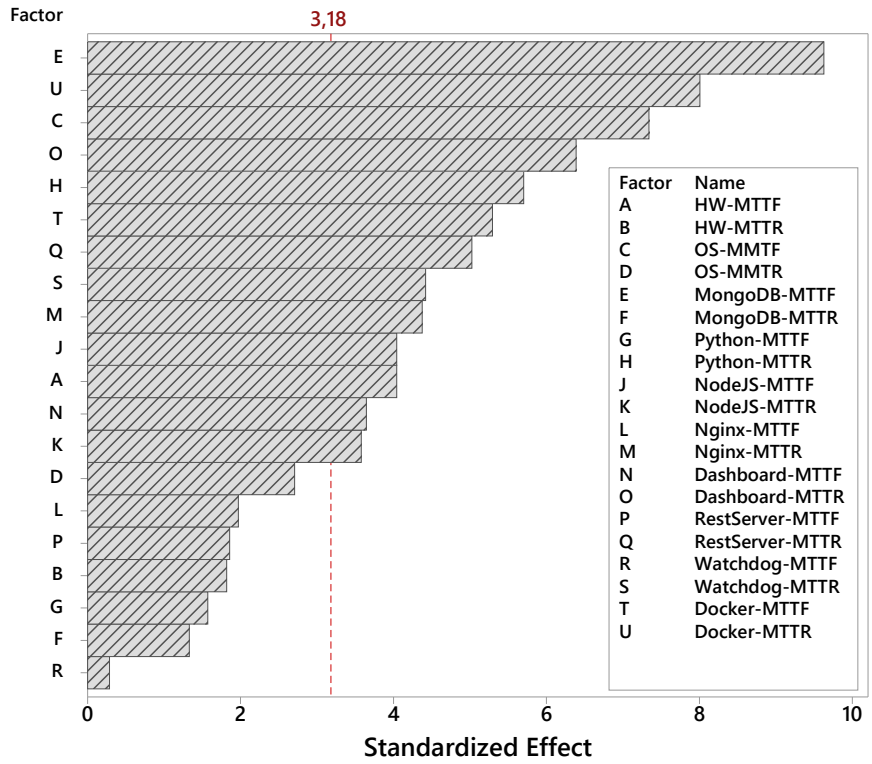

Fig. 4: Influence of MTTR and MTTF Factors

we can highlight the interaction between the MongoDB-MTTF and Python-MTTR factors, characterizing itself with greater interaction, reaching the level of -50%, even if in a minimal way. However, this demonstrates that if the evaluator opts for the MongoDB database, taking into account the MTTF, the best choice is the MTTF of the database with more than 50%. The interaction between MongoDB-MTTF and Docker-MTTR was similar to the one mentioned above. The same choice criterion applies if the database choice is MongoDB. For the interaction between Docker-MTTR and OS-MMTF, if the evaluator considers Docker MTTR, the best choice falls within the +50%.

5 Extended Models

This section presents the structure of the extended model applying Cold, Warm and Hot Standby redundancies [28, 29, 30]. The MTTF and MTTR values for the extended models are the same as in Table 4 presented earlier. The time that triggered the redundant server in the SWITCH_TIME transition was 0.0833333 hours, extracted from [31]. The characteristic of redundant models is presented in Table 5.

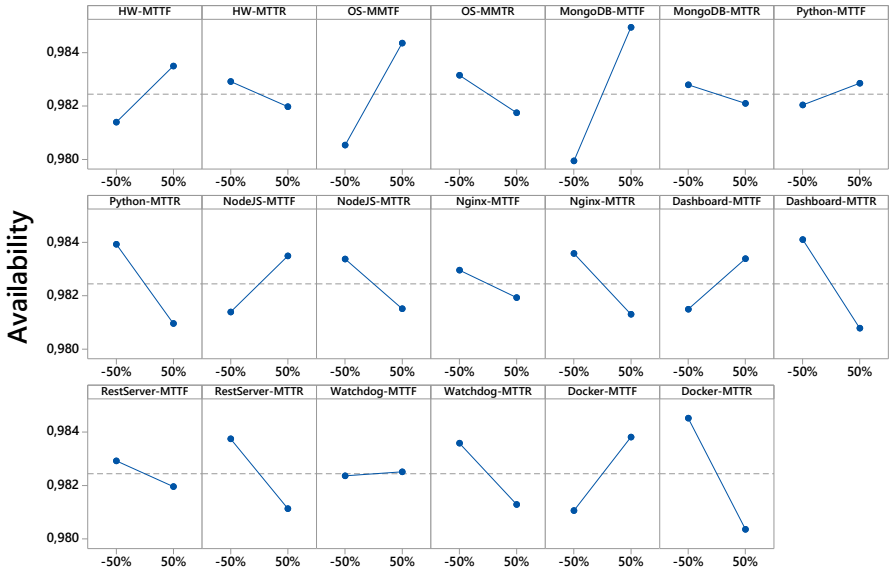

Fig. 5: Main Effects for Availability

Table 5: Features of Redundant Models.

Redundancy	**Description**
Cold Standby	The Cold Standby redundancy applied to MongoDB provides increased availability because a component with the same characteristics is activated if the database fails. However, after failure, all components are disconnected until the redundant component is triggered.
Warm Standby	Warm Standby works with similar availability as Cold Standby, but both components work simultaneously. If the main component fails, another machine is activated to ensure more availability for the system.
Hot Standby	Hot Standby feature a feature of doubling the component's capacity, resulting in a reduction in component failure.

Model Two - Cold Standby A major limitation of the base proposal is that if one of the two servers fails, the entire system will stop working. We have only considered the redundancy of the MongoDB component to assess whether there is an improvement in the availability. MongoDB_01 and MongoDB_02 are always connected; however, MongoDB_02 is instantiated only if MongoDB_01 fails. Therefore, the cold standby redundancy mechanism is applied when the main component fails, providing system operation after a component fails.

Figure 7 presents an overview of the model in cold standby. Given the DoE analysis, it was identified that MongoDB has the greatest impact, so MongoDB was the component chosen to do the redundancy. When the component fails,

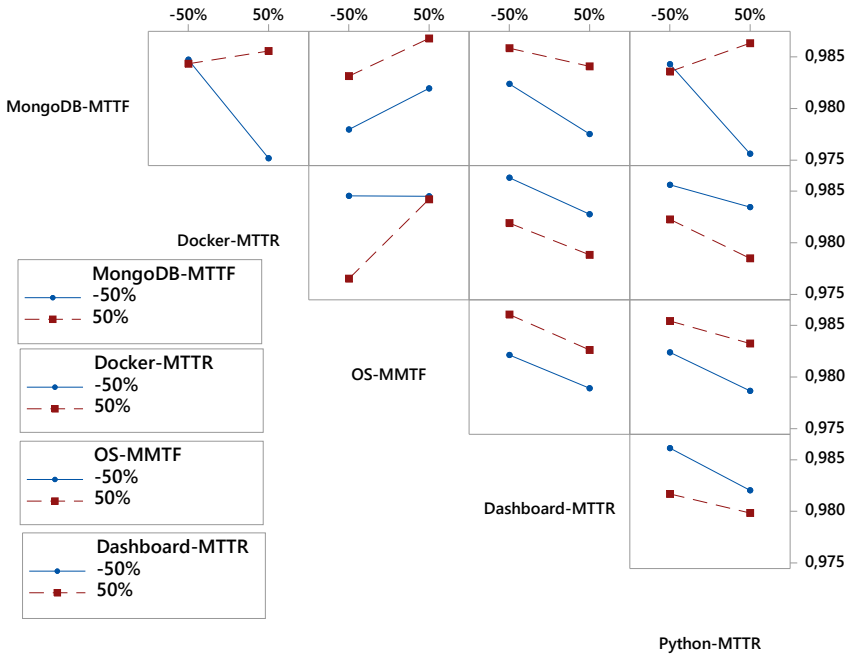

Fig. 6: Influence of factors in relation to availability

the redundant MongoDB will be triggered so that the system continues to be fed with the database and can carry out storing data normally. When the first MongoDB is repaired, the redundancy will be disabled as it is only needed in case the main component fails. Only if both groups fail will the system become unavailable. Table 6 presents the guard expressions of the extended model.

Table 6: Extended model guard expressions using Cold Standby

Transition	**Index**	**Expressão**	**Description**
SWITCH_TIME	[C-09]	(#MONGODB_U=0)	Enables when MongoDB is disabled
T1	[C-08]	(#MONGODB_U>0)	Enabled when MongoDB is enabled
T2	[C-08]	(#MONGODB_U>0)	Enabled when MongoDB is enabled

Model Three - Warm Standby Here, the unique redundancy of MongoDB was considered to evaluate availability. We have MongoDB_01, W_MongoDB and W_MongoDB_01. Both components are always connected. However, when

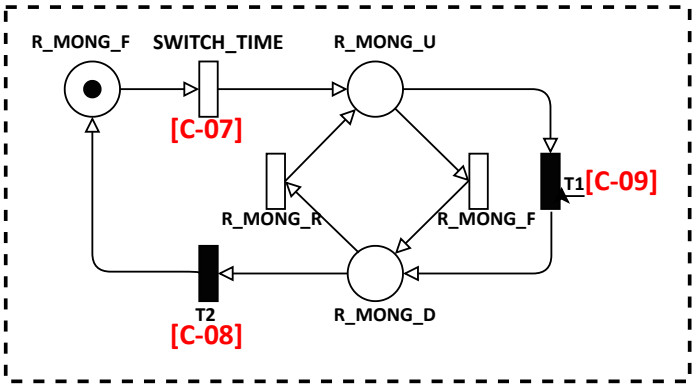

Fig. 7: SPN Model with Cold Standby Redundancy

W_MongoDB crashes, nothing happens. The system continues to be fed with the data present in MongoDB_01. However, if MongoDB_01 goes down, W_MongoDB_01 is automatically activated. Thus, the system is fed with the data provided by the redundancy of the database. However, if W_MongoDB_01 fails, the system becomes idle.

Figure 8 presents an overview of the SPN model with the proposed extension of the base model. However, when MongoDB fails, redundancy will be triggered so that the system continues to feed the database. When group 1 MongoDB is repaired, the opposite component (group 2) will be disabled. If both MongoDB fails, the system will become unavailable. Table 7 presents the guard conditions used for system operation in the extended model. In this case, using guard conditions was of great help in avoiding visual pollution of the model since several connections had to be made.

Table 7: Extended model guard conditions using Warm Standby

Transition	**Index**	**Expressão**	**Description**
MONGODB_PYTHON	[C-010]	(#W_MONGODB_U1=0)	Enables when MongoDB is disabled
T4	[C-11]	(#MONGODB_U>0)	Enabled if MongoDB is enabled
T5	[C-12]	(#MONGODB_U=0)	Enabled if MongoDB is disabled

Model Four - Hot Standby In the model based on Hot standby, it is necessary to double the number of tokens of that component to become redundant. In Hot Standby redundancy, the faulty module is replaced without significant delay, as the resilient modules are also powered. Figure 9 shows the model

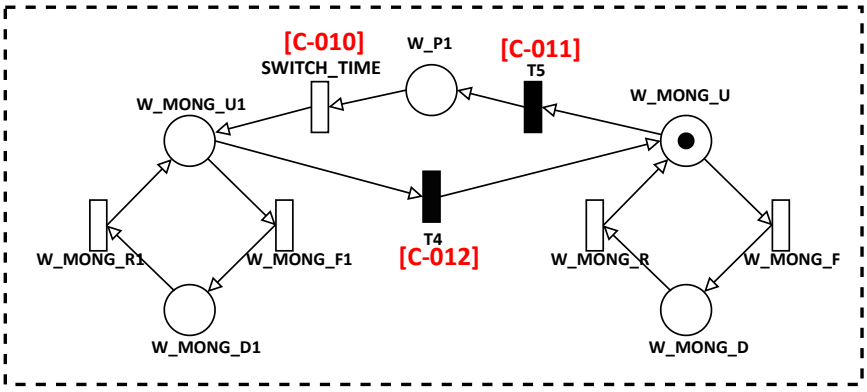

Fig. 8: SPN Model with Warm Standby Redundancy

with Hot Standby of the MongoDB component with increased capacity, where MONG_U works with double capacity.

Model Five - Reliability Figure 10 presents the SPN reliability model for the blockchain architecture of the baseline scenario. This model is composed of thirteen system components present in the system. The MTTF transitions trigger each component's change from active to inactive status. Each component can operate independently if the number of tokens in UP equals the markup value. This model is a variation of the base SPN model by removing MTTR transitions from all components. Once components fail, they cannot be repaired. All input parameters are the same as in the base model. This model aims to show the system's confidence level to continue working as a function of time. Equation $R = 1 - (WH\_D > 0)\, OR\, (OS\_D > 0)$
$OR\,(MONGODB\_D > 0)\, OR\,(PYTHON\_D > 0)$
$OR\,(NODEJS\_D > 0)\, OR\,(NGNIX\_D > 0)$
$OR\,(DOCKER\_D > 0)\, OR\,(DASHBOARD\_D > 0)$
$OR\,(RESTSERVER\_D > 0)\, OR\,(WATCHDOG\_D > 0)$
$OR\,(W\_WH\_D > 0)\, OR\,(W\_OS\_D > 0)$
$OR\,(W\_DOCKER\_D > 0)$ defines the reliability (R) of the model.

Reliability was also assessed using DoE. Figure 4 shows the Pareto chart. The HW, OS, MongoDB, NodeJS, Docker and Dashboard components impact the reliability the most. Figure 11 shows the reliability by varying the MTTF of those components with the greatest impact. The MTTF of the components was varied between base value, base value plus 25%, base value plus 50% and base value plus 75%. The increase in execution time is directly proportional to reliability. The longer the execution time, the lower the system's reliability. In both configurations, reliability started at 0, but the highest angle of fall occurred until Time = 50000 hs. Reliability starts to decrease and tends to stabilize towards the end of the experiment. We can also observe that the

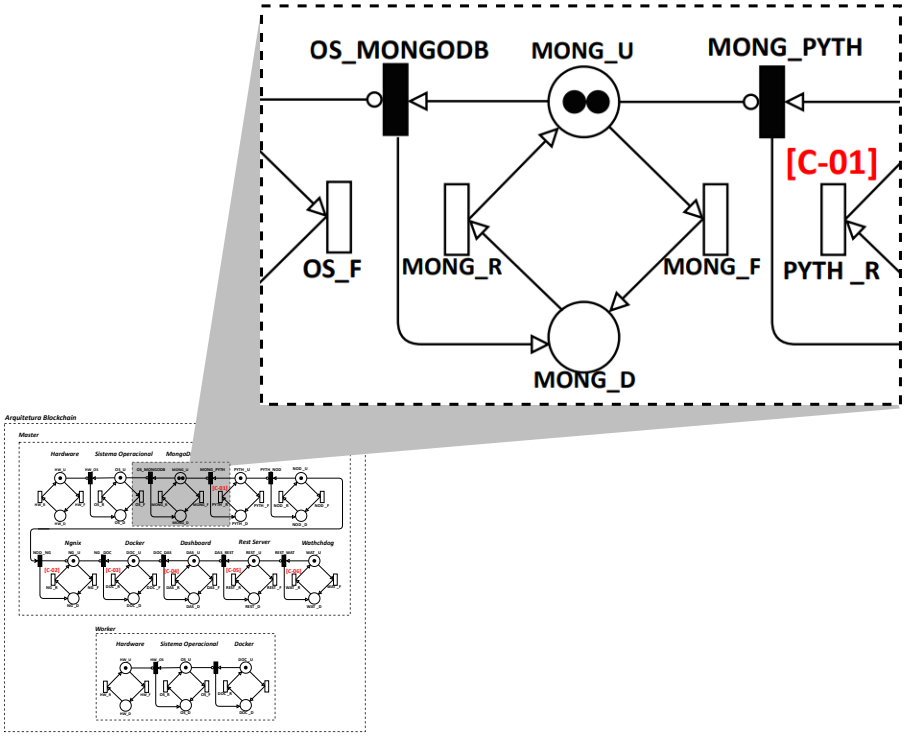

Fig. 9: SPN Model with Hot Standby Redundancy

points where the reliability tends to a value smaller than 0.099 are: (i) base configuration = 45000 hs; (ii) base configuration plus 25% = 52500 hs; (iii) base configuration plus 50% = 61500 hs; and (iV) base value configuration plus 75 % = 73500 hrs. For the starting point, the entire system started its execution with reliability at 100%. Therefore, the longer the failure time, the greater the reliability, as the system will operate longer.

Figure 12 presents the bar graph for reliability. Four cuts (time window) are performed at runtime for better visualisation, so we can evaluate the behaviour over time. T1 goes from 0h to 36000hs, T2 goes from 36001hs to 73500hs, T3 goes from 73501hs to 111000hs, and T4 goes from 111001hs to 150000hs. The base model has the highest reliability because the system has little execution time, and thus the probability of system failure is much lower. However, the longer the system remains active, the base model shows itself as inefficient in terms of reliability. However, improvements in HW, OS, MongoDB, NodeJS, Docker and Dashboard components are not relevant for a time lower than 36000hs. In order to consider the need for improvements in the components, a run greater than 36000hs must be taken into account. It was also observed that as the running time of the system increases, the reliability values are

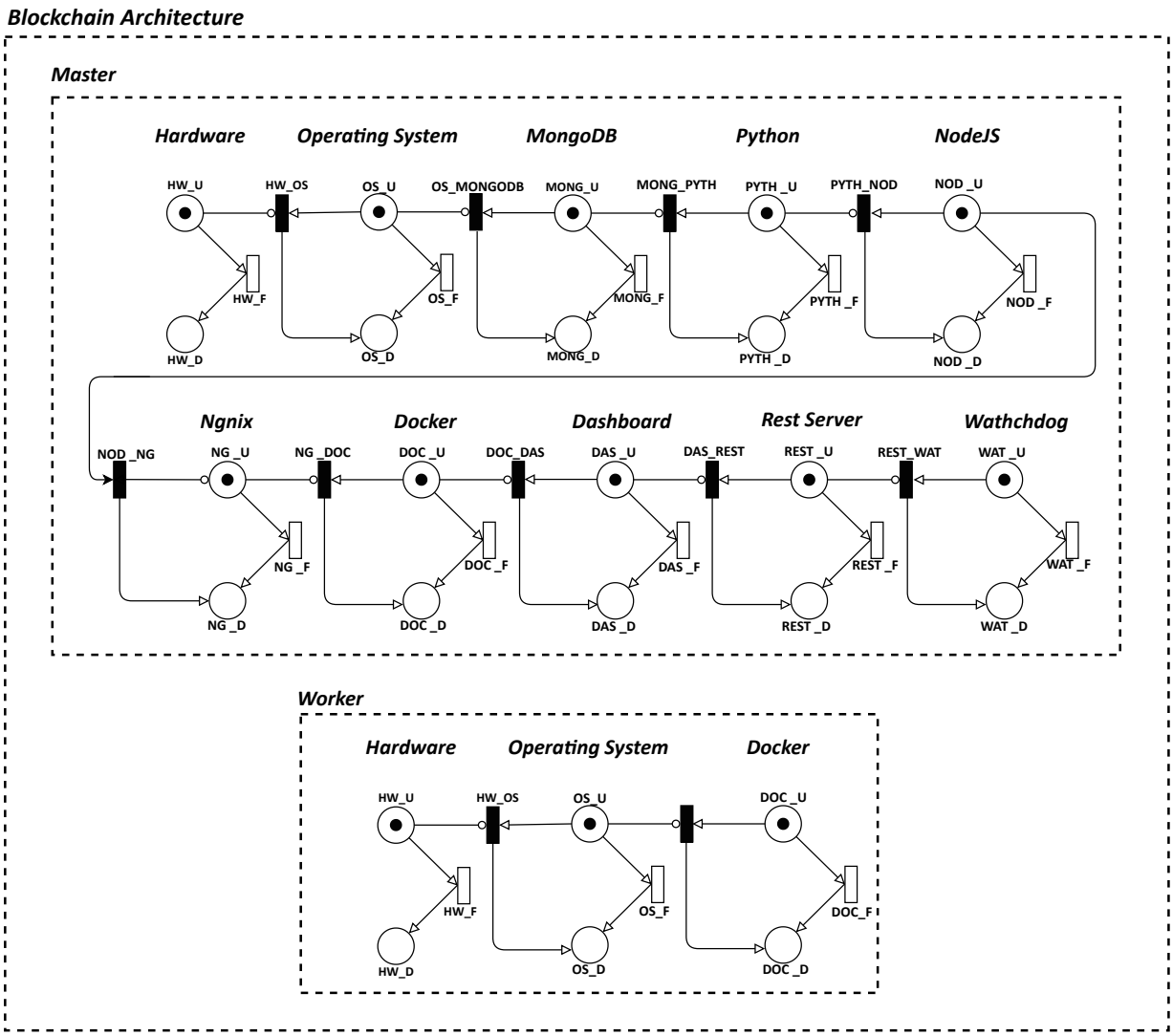

Fig. 10: Reliability model.

inverted. Therefore, the base model at the beginning of the execution showed the highest reliability at the end of the experiment with the lowest reliability.

6 Model Comparison

This section presents the availability analysis of the four models presented in the paper. The variation in the availability of each model was observed, as well

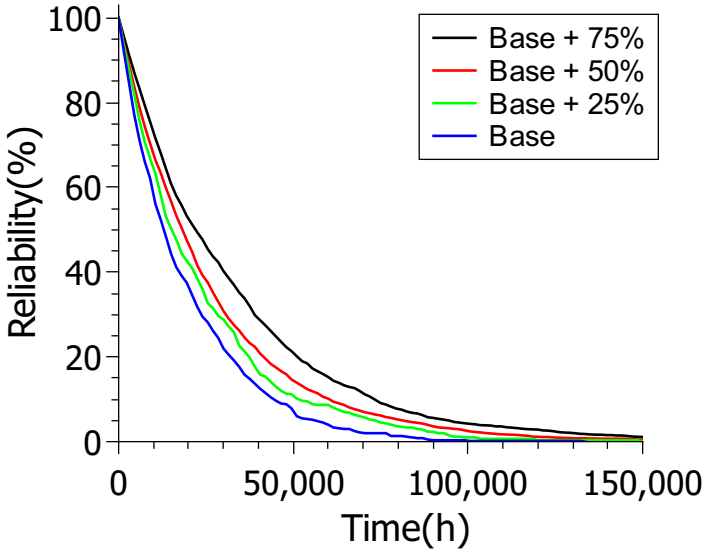

Fig. 11: Reliability levels varying the MTTF of HW, OS, MongoDB, NodeJS, Docker and Dashboard components, ranging from 25%, 50% and 75%.

as the impact of each component on the architecture. The proposed model was evaluated using the Mercury Script Language [32] tool. In this study, the test was performed to assess the reliability and availability of the presented models. During the simulation, the Hot Standby redundancy proved superior to the other models in terms of availability.

The base architecture had 141 hours of unavailability, equivalent to 98.3% of stationary availability. With Cold redundancy, it is possible to observe a result with a longer availability time, totalling 126,775 hours of unavailability. Cold redundancy presented an availability of 98.4%. The Warm redundancy presented a better result than the Cold redundancy and the base model, having an unavailability time of approximately five days, with 98.5% of availability. However, the Hot one presented the highest performance, with only 77 hours of unavailability, and availability = 99%.

6.1 Factors Impact Evaluation

In the previous scenarios, the MongoDB failure time factor was analyzed individually, creating redundancies and testing system performance and availability. Such analyses allowed us to observe the factor with the greatest impact that interferes with all metrics in a very detailed way. However, in addition to having an isolated impact on the behaviour of the system, the DoE analysis showed that there is a strong interaction between the two factors in the average response time, as shown in the Pareto chart (Figure 4) and graph of

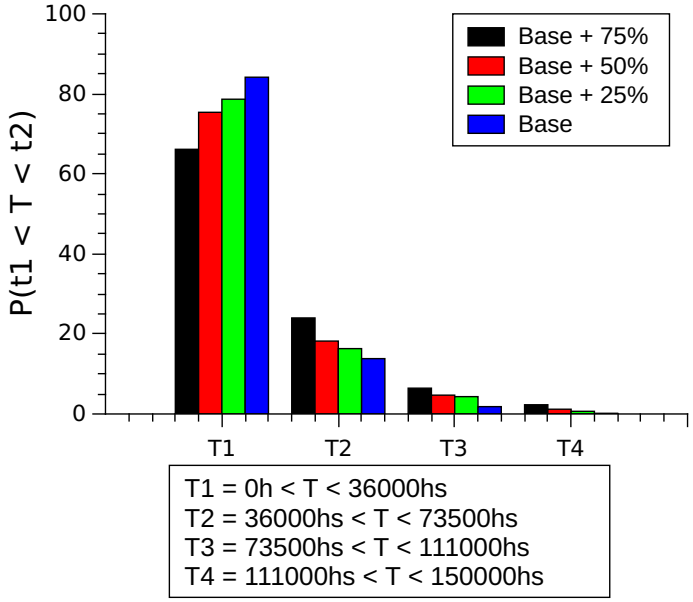

Fig. 12: Variation of Reliability levels varying the MTTF of the HW, OS, MongoDB, NodeJS, Docker and Dashboard components, variation of 25%, 50% and 75%.

interaction (Figure 6). These graphs only indicate the existence and magnitude of the interaction. Therefore, this section shows the variation between the two factors. Table 8 presents the combinations between the factors analyzed in this scenario.

Figure 14 presents a 3D surface graph to show the system behaviour considering system availability, varying two factors with a high impact on performance. Colours are related to the result of availability. The bar on the right indicates the magnitude of the results. The upper part indicates the highest availability, and the lower indicates the lowest availability. Therefore, purple represents the lowest availability, and red represents the highest availability. In the graph, it is worth highlighting the presence of a projection at the top that facilitates the interaction of factors.

Changing the MongoDB MTTF has a greater impact than changing the Docker MTTR. The red colour is present in most of the projections, indicating a high availability of the system. The purple colour corresponds to availabilities at the bottom of the chart. If a MongoDB failure time and a higher Docker recovery time are adopted, the system availability drops, showing that the Docker recovery factor is relevant to the system availability. Therefore, the result indicates that it is often more beneficial to invest in Docker recovery time and thus improve availability.

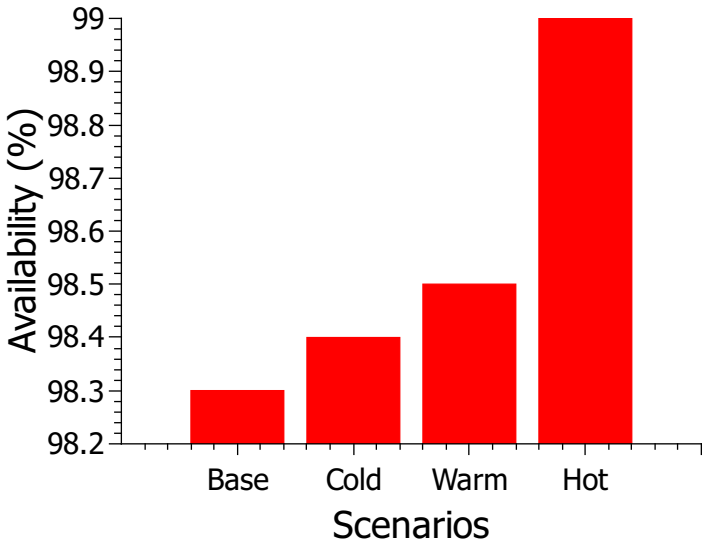

Fig. 13: Availability of the four models varying redundancy in the MongoDB component.

7 Conclusion

This paper proposed stochastic Petri net models for a blockchain architecture to help system administrators plan computer system architectures. The models consider several factors that influence the total availability of the system. Among the factors presented in the sensitivity analysis, it was noted that the MongoDB component has a greater impact on the availability and reliability of the system. There were significant components in the sensitivity analysis: HW, OS, NodeJS, Docker and Dashboard. Significant improvements in these components will increase the availability and reliability of the blockchain network. Modifications include using more powerful hardware or components with greater processing power. It was also noted that regarding reliability, making improvements to the components is unnecessary if the system has a runtime of fewer than 36,000 hours.

The models provide accurate availability and reliability metrics. The models were demonstrated by carrying out four case studies. The case studies provide a practical guide that shows how a system administrator can apply the model to perform assessments of various configurations for a blockchain architecture.

Combination	MongoDB MTTF (Time)	Docker MTTR (Time)	Availability (%)
#1	129600	90	0,98634
#2	129600	75	0,98693
#3	129600	60	0,98753
#4	129600	45	0,98813
#5	129600	30	0,98873
#6	108000	90	0,98562
#7	108000	75	0,98624
#8	108000	60	0,98686
#9	108000	45	0,98749
#10	108000	30	0,98811
#11	86400	90	0,98455
#12	86400	75	0,98521
#13	86400	60	0,98586
#14	86400	45	0,98652
#15	86400	30	0,98717
#16	64800	90	0,98277
#17	64800	75	0,98348
#18	64800	60	0,98419
#19	64800	45	0,9849
#20	64800	30	0,98562
#21	43200	90	0,97922
#22	43200	75	0,98004
#23	43200	60	0,98086
#24	43200	45	0,98169
#25	43200	30	0,98251

Table 8: Interaction between MongoDB-MTTF and Docker-MTTR.

8 Declarations

Ethical Approval Not applicable

Competing interests The authors have no relevant financial or non-financial interests to disclose.

Authors' contributions L.F. wrote the paper. F.A.S and J.R. reviewed the manuscript.

Funding No funding was received for conducting this study.

Availability of data and materials Data sharing not applicable.

References

1. Pedro W Abreu, Manuela Aparicio, and Carlos J Costa. Blockchain technology in the auditing environment. In *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE, 2018.
2. Tiana Laurence. *Blockchain for dummies*. John Wiley & Sons, 2019.

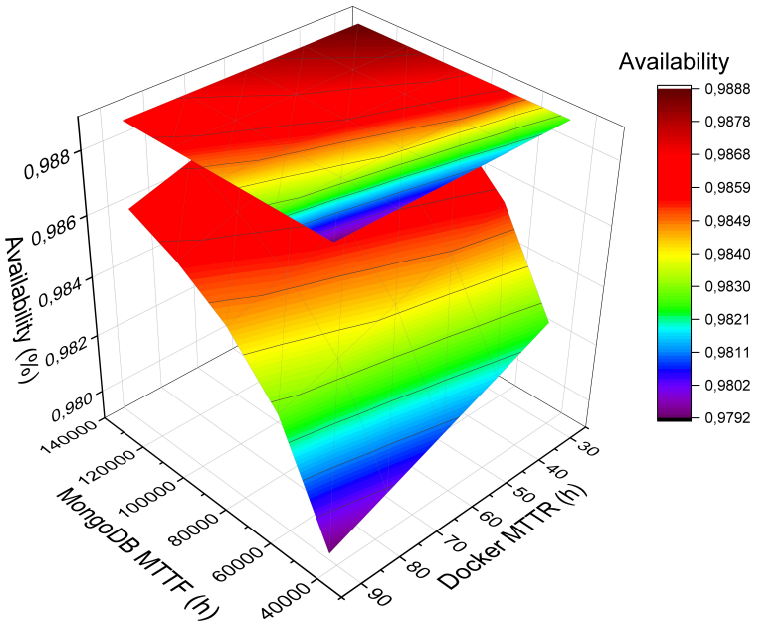

Fig. 14: Analysis of the average response time of MTTF-MongoDB and MTTR-Docker simultaneously.

3. Melanie Swan. *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.
4. William Mougayar. *Blockchain para negócios: promessa, prática e aplicação da nova tecnologia da internet.* Alta Books Editora, 2018.
5. Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business & Information Systems Engineering*, 59(3):183–187, 2017.
6. T. Murata. Petri nets: Properties, analysis and applications. *Proc. IEEE*, 77(4):541–580, April 1989.
7. A. Marsan. *Modelling with generalized stochastic Petri nets.* Wiley series in parallel computing. Wiley, 1995.
8. K Trivedi. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications.* Wiley Interscience Publication, 2 edition, 2002.
9. Carlos Melo, Jamilson Dantas, Danilo Oliveira, Iure Fé, Rubens Matos, Renata Dantas, Ronierison Maciel, and Paulo Maciel. Dependability evaluation of a blockchain-as-a-service environment. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 00909–00914. IEEE, 2018.
10. Matheus Rodrigues, Breno Vasconcelos, Carlos Gomes, and Eduardo Tavares. Evaluation of nosql dbms in private cloud environment: An approach based on stochastic modeling. In *2019 IEEE International Systems*

Conference (SysCon), pages 1–7. IEEE, 2019.
11. Yue Liu, Xiaoyang Li, Yanhui Lin, Rui Kang, and Lianghua Xiao. A colored generalized stochastic petri net simulation model for service reliability evaluation of active-active cloud data center based on it infrastructure. In *2017 2nd International Conference on System Reliability and Safety (ICSRS)*, pages 51–56. IEEE, 2017.
12. Manar Jammal, Ali Kanso, Parisa Heidari, and Abdallah Shami. Evaluating high availability-aware deployments using stochastic petri net model and cloud scoring selection tool. *IEEE Transactions on Services Computing*, 2017.
13. Luis Zabala, Ruben Solozabal, Armando Ferro, and Bego Blanco. Model of a virtual firewall based on stochastic petri nets. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–4. IEEE, 2018.
14. Júlio Mendonça, Ricardo Lima, Rubens Matos, João Ferreira, and Ermeson Andrade. Availability analysis of a disaster recovery solution through stochastic models and fault injection experiments. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 135–142. IEEE, 2018.
15. Francisco Airton Silva, Sokol Kosta, Matheus Rodrigues, Danilo Oliveira, Teresa Maciel, Alessandro Mei, and Paulo Maciel. Mobile cloud performance evaluation using stochastic models. *IEEE Transactions on Mobile Computing*, 17(5):1134–1147, 2017.
16. Thiago Pinheiro, Francisco Airton Silva, Iure Fe, Sokol Kosta, and Paulo Maciel. Performance and data traffic analysis of mobile cloud environments. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 4100–4105. IEEE, 2018.
17. Manar Jammal, Ali Kanso, Parisa Heidari, and Abdallah Shami. Availability analysis of cloud deployed applications. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 234–235. IEEE, 2016.
18. Iure Fe, Rubens Matos, Jamilson Dantas, Carlos Melo, and Paulo Maciel. Stochastic model of performance and cost for auto-scaling planning in public cloud. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2081–2086. IEEE, 2017.
19. Laécio Rodrigues, Patricia Takako Endo, and Francisco Airton Silva. Stochastic model for evaluating smart hospitals performance. In *2019 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6. IEEE, 2019.
20. Leylane Ferreira, Elisson da Silva Rocha, Kayo Henrique C Monteiro, Guto Leoni Santos, Francisco Airton Silva, Judith Kelner, Djamel Sadok, Carmelo JA Bastos Filho, Pierangelo Rosati, Theo Lynn, et al. Optimizing resource availability in composable data center infrastructures. In *2019 9th Latin-American Symposium on Dependable Computing (LADC)*, pages 1–10. IEEE, 2019.
21. Guto Leoni Santos, Demis Gomes, Judith Kelner, Djamel Sadok, Francisco Airton Silva, Patricia Takako Endo, and Theo Lynn. The internet of

things for healthcare: optimising e-health system availability in the fog and cloud. *International Journal of Computational Science and Engineering*, 21(4):615–628, 2020.
22. Francisco Airton Silva, Iure Fé, and Glauber Gonçalves. Stochastic models for performance and cost analysis of a hybrid cloud and fog architecture. *JOURNAL OF SUPERCOMPUTING*, 2020.
23. Daniel Carvalho, Laécio Rodrigues, Patricia Takako Endo, Sokol Kosta, and Francisco Airton Silva. Mobile edge computing performance evaluation using stochastic petri nets. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2020.
24. Francisco Airton Silva, Sokol Kosta, Matheus Rodrigues, Danilo Oliveira, Teresa Maciel, Alessandro Mei, and Paulo Maciel. Mobile cloud performance evaluation using stochastic models. *IEEE Transactions on Mobile Computing*, 17(5):1134–1147, 2018.
25. Thiago Felipe da Silva Pinheiro, Francisco Airton Silva, Iure Fé, Sokol Kosta, and Paulo Maciel. Performance prediction for supporting mobile applications' offloading. *The Journal of Supercomputing*, 74(8):4060–4103, 2018.
26. James F Kurose and Keith W Ross. Redes de computadores e a internet. *São Paulo: Person*, 28, 2006.
27. Leonel Feitosa, Glauber Gonçalves, Tuan Anh Nguyen, Jae Woo Lee, and Francisco Airton Silva. Performance evaluation of message routing strategies in the internet of robotic things using the d/m/c/k/fcfs queuing network. *Electronics*, 10(21):2626, 2021.
28. Lucas Santos, Benedito Cunha, Iure Fé, Marco Vieira, and Francisco Airton Silva. Data processing on edge and cloud: a performability evaluation and sensitivity analysis. *Journal of Network and Systems Management*, 29(3):1–24, 2021.
29. Francisco Airton Silva, Iure Fé, Carlos Brito, Gabriel Araújo, Leonel Feitosa, Eunmi Choi, Dugki Min, and Tuan Anh Nguyen. Supporting availability evaluation of a smart building monitoring system aided by fog computing. *Electronics Letters*, 58(12):471–473, 2022.
30. Gabriel Araújo, Laécio Rodrigues, Kelly Oliveira, Iure Fé, Razib Khan, and Francisco Airton Silva. Vehicular cloud computing networks: availability modelling and sensitivity analysis. *Int. J. Sens. Networks*, 36(3):125–138, 2021.
31. Eltton Araujo, Jamilson Dantas, Rubens Matos, Paulo Pereira, and Paulo Maciel. Dependability evaluation of an iot system: A hierarchical modelling approach. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 2121–2126. IEEE, 2019.
32. Danilo Oliveira, Rubens Matos, Jamilson Dantas, João Ferreira, Bruno Silva, Gustavo Callou, Paulo Maciel, and André Brinkmann. Advanced stochastic petri net modeling with the mercury scripting language. In *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 192–197, 2017.