Cloud infrastructure planning considering the impact of maintenance and self-healing routines over cost and dependability attributes

Carlos Melo, Jamilson Dantas, Paulo Pereira, Felipe Oliveira, and Paulo Maciel

Universidade Federal de Pernambuco, Recife, Brazil
E-mail: {casm3, jrd, prps, fdo, prmm}@cin.ufpe.br

Jean Araujo

Universidade Federal do Agreste de Pernambuco, Garanhuns, Brazil
E-mail: jean.teixeira@ufape.edu.br

Abstract: Cloud Computing is the main trend regarding Internet service provisioning. This paradigm that emerged from distributed computing gains more adepts every day. For those who provide or aim at providing a service or a private infrastructure, much has to be done, costs related to acquisition and implementation are common, and an alternative to reduce expenses is to outsource resources' maintenance. Outsourcing tends to be a better alternative for those who provide small infrastructures than monthly pay some employees to keep the service life cycle. This paper evaluates infrastructure reliability and the impact of outsourced maintenance over the availability of private infrastructures. Our baseline environments focus on Blockchain as a service; however, by modeling both service and maintenance routines, this study can be applied to most cloud services. The maintenance routines evaluated by this paper considers a set of Service Level Agreements and some particularities related to reactive, preventive, and self-healing methods. The goal is to point out which one has the best cost-benefit for those with small infrastructures, but that still plans to provide services over the Internet. Preventive and self-healing repair routines provided a better cost-benefit solution than traditional reactive maintenance routines, but this scenario may change according to the number of available resources that the service provider has.

Keywords: Maintenance; Reliability; Availability; Modeling; Cloud Computing; Blockchain; Container; Services; SLA.

Reference to this paper should be made as follows: C Melo, J Araujo, J Dantas, P Pereira, F Oliveira and P Maciel. Cloud infrastructure planning considering the impact ofmaintenance and self-healing routines over cost anddependability attributes. *Int. J. of Grid and Utility Computing*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes:
Carlos Melo received his bachelor's degree in computer science from Federal Rural University of Pernambuco – UFRPE, Brazil in 2014, and the M.Sc. degree in computer science from the Federal University of Pernambuco - UFPE in 2016. Today he is a Ph.D. Candidate in the same research area at Federal University of Pernambuco. He is a participant of MoDCS Research Group.

Jean Araujo is graduated in Information Systems from Faculdade Sete de Setembro (2008), Specialization in Computer Network Security from Universidade Gama Filho (2012), M.Sc. and Ph.D. in Computer Science from Centro de Informática at Universidade Federal de Pernambuco (2012 and 2017, respectively), and post-doctoral researcher at the Faculdade de Ciências de Tecnologias from Universidade NOVA de Lisboa (2018). Currently, he develops activities as Adjunct Professor and Institutional Coordinator of the Academic Incentive Scholarship Program at the Universidade Federal do Agreste de Pernambuco. He is also a founding member of the BCC Coworking Research and Innovation Lab., and member of the board of the Multidisciplinary Laboratory of Social Technologies. He is also a permanent member of the Post-Graduate Program in Computer Science at the Universidade Federal de Sergipe. In addition, he is the founder of the UNAME Research Group, and a participant of the MoDCS Research Group. He works mainly in the area of Computer Networks and Distributed Systems, with experience in the areas of software development, cloud computing, systems performance evaluation, software aging and rejuvenation, Linux systems, simulation and analytical models, and intellectual property.

Jamilson Dantas received his B.S. degree in Information Systems from the UNIRIOS - Centro Universitário do Rio São Francisco, Brazil, in 2009. He earned his M.Sc. and his PhD in Computer Science from the Centro de Informática da Universidade Federal de Pernambuco, in 2013 and 2018 respectively. He is a professor at the Universidade Federal de Pernambuco - CIn, working in Computer Networks and Distributed Systems. He is a participant of MoDCS Research Group.

Paulo Pereira is a PhD candidate at Universidade Federal de Pernambuco - UFPE, working mainly with computer networks, cloud, fog and edge computing, IoT, time series forecast, performance and availability evaluations, and analytical modeling. He is also a participant in the Modeling of Distributed and Concurrent Systems research group - MoDCS, and UNAME research group.

Felipe Oliveira received the B.S. degree in computer science from the UFRPE - Universidade Federal Rural de Pernambuco, in 2019. He is M.Sc. candidate at UFPE - Universidade Federal Rural de Pernambuco. In addition, he has worked on research projects encompassing software aging, cloud computing, performance evaluation, and computational modeling. His research interests include performance and dependability evaluation, Markov chains, Petri nets, and communication systems. Mr. Oliveira is a member of the MoDCS and UNAME Research Groups, also of the IEEE Society.

Paulo Maciel received the degree in electronic engineering in 1987 and the M.Sc. and Ph.D. degrees in electronic engineering and computer science from the Federal University of Pernambuco, Recife, Brazil, respectively. Since 2001, he has been a member of the Informatics Center, Federal University of Pernambuco (UFPE), where he is currently an Associate Professor.

1 Introduction

Service provisioning evolved and still has a long way to run before it reaches its plateau. Many have been done through the years; we moved from mainframes to supercomputers, then a set of big servers deployed on an even massive data canter, and now most can be done with a personal computer. The last step in this evolution is based mainly on the resource virtualization technology that puts cloud computing in our daily lives. The cloud paradigm [28] emerged at the beginning of the 2000s and now is full of adepts, such as Amazon, Google, Microsoft, and IBM. However, cloud computing is not about financial power, like those big companies that stand worldwide. It is more about technology.

Today, the technology that was present only in the hands of a few enables anyone to provide their infrastructures, using personal computers and open source tools to accomplish tasks 30 years ago were not even expected to exist. Now, private cloud computing infrastructures are gaining terrain, and service provisioning can be done closer to the user than never had been, as can be seen in edge, and fog computing [24], as well as in other cloud-related paradigms. Nevertheless, besides the free-to-go option based on open-source tools, providing a service is not an easy task, and dealing with others' money may not be a cheap alternative, and that is when outsourcing arises.

Many companies (and freelancers) around the world focus their expertise on keeping the services and infrastructures of others running smoothly [12]. This outsourcing method is an alternative for those who cannot afford the expenses to keep a fixed staff with monthly incomes and responsible for repairing a system only when it enters into a failure state, which may take a while [11].

This paper evaluates the system's overall reliability and proposes a set of maintenance policies that were modeled, altered, and evaluated, aiming to identify their respective impact on the availability and costs of a provided cloud service. These policies were based on the process of outsourcing resources repair and were modeled through Stochastic Petri Nets (SPN), considering as baseline a blockchain as a service environment. The cost-benefit of each evaluated maintenance policy was analyzed. We point out a ranking containing 36 architectures with different parameters used by those who aim to provide or provide services through private cloud computing infrastructures.

The remainder of this paper is organized as follows. Section 2 presents the works that underlie this research and how these works differ from what we propose in this paper. Section 3 introduces basic concepts about cloud computing, blockchain technologies, reliability, availability, as well as the system's modeling and evaluation. While Section 4 presents the support methodology that enables the proposed models to be constructed, these models are presented in Section 5. Section 6 provides the case studies and applications demonstrating how feasible the proposed models are. Finally, Section 7 presents the final remarks and future works.

2 Related Works

Over the last few years, authors have devoted their efforts to study and evaluate the effects of maintenance on the service's availability. This section presents some of these works and how they related to what has been done by this paper.

The authors in Araujo et al. 2021 [1] evaluated the impact of the software aging phenomenon on a given system's general availability. They used Stochastic Petri Nets (SPNs) and modeled a maintenance routine based on a self-healing strategy. In this paper, we do not consider software aging issues and provide general maintenance models that can be used for a broader set of services that can be provided over private cloud computing infrastructures.

In Melo et al. 2020 [20], the authors evaluated the impact of maintenance routines over the dependability of data centers' electrical infrastructures. The authors also used Stochastic Petri Nets as modeling formalism to represent both corrective and preventive maintenance of electrical sites. The main difference between their paper to ours is that it does not present a self-healing strategy since it deals with physical resources like UPS and does not deal directly with cloud computing infrastructures and services, which is the focus of this research.

Following [20], the authors in Elusakin et al. 2020 [8] focused on the reliability evaluation of subsea blowout presenters (BOP), the authors also used Petri Nets aiming the identification or prediction of the best condition to perform maintenance on these pieces of equipment. Blancke et al. 2018 [5] focused on the predictive maintenance of a hydro generator. Once again, Petri Nets and its extensions were used as modeling formalism, focusing on physical components.

In Verhagen et al. 2018 [29] the authors proposed a set of proportional hazard models or expressions that can be used to evaluate predictive maintenance routines over aircraft and their impact on the general reliability. The focus of this paper differs from ours directly. Our models are for cloud services. Most cloud services are noncritical, while the Verhagen model can directly impact the life and death of persons and is a too specific model.

Lee et al. 2016 [14], the authors used stochastic modeling to identify the most opportune moment to perform either preventive maintenance or equipment replacement on a general reparable system, which can also be applied to both hardware and software. This approach also does not consider the self-healing mechanism and could be hard to adapt by those who are not familiar with it.

Finally, in Sousa et al. 2012 [26] Petri Nets were, once again, the chosen modeling formalism, the authors evaluated the impact of maintenance routines on the performability of Electronic Funds Transfers systems (EFT). The authors also described a maintenance procedure to be followed to reduce this impact, but they neither consider preventive maintenance nor self-healing approaches.

Table 1 presents a comparative between this paper and the presented one. All the present works deal with maintenance models at some level, as well as with dependability attributes. However, this paper is the only one that presents both reliability and availability evaluations.

3 Background

This section presents the fundamental concepts of dependability evaluation, including availability and reliability analysis, systems modeling, Blockchain, and cloud computing paradigm.

3.1 Dependability Evaluation

Dependability is the ability of a computer system to provide services that can be both: justifiably and trusted [3]. Evaluating the dependability of a computer system is usually the result of evaluating at least one of its five attributes: reliability, availability, maintainability, integrity, and safety [4].

In this paper, the system's reliability and availability are the dependability attributes evaluated. The system's reliability is the probability of a system to execute the programmed function into a predetermined time limit without any interruption [4]. In general, the system's availability means the probability of the system is operational at a given instant of time. The Steady-state availability, also called the long-run

Table 1 State of Art

Publication	Dependability Attribute	Costs	Maintenance	Models
Araujo et al. 2021 [1]	Availability	No	Yes	SPN
Melo et al. 2020 [20]	Availability	Yes	Yes	SPN
Elusakin et al. 2020 [8]	Reliability	No	Yes	SPN
Verhagen et al. 2018 [29]	Reliability	No	Yes	Expressions
Lee et al. 2016 [14]	Reliability	No	Yes	Expressions
Sousa et al. 2012 [26]	Availability	No	Yes	SPN
This Paper	Reliability and Availability	Yes	Yes	SPN and RBD

availability, limits the availability function as time tends to infinity (Equation 1).

$$A = \lim_{t\to\infty} A(t), t \geq 0 \tag{1}$$

Also, the system's availability may be represented by a ratio between the Mean Time To Failure (MTTF) and Mean Time To Repair (MTTR) of the system (Equation 2).

$$\text{A} = \frac{\text{MTTF}}{\text{MTTF+MTTR}} \tag{2}$$

The MTTF for a system may be computed from Equation 3, where **R(t)** is the reliability of that system as a function of elapsed time. The Equation 4 provides a way of computing the MTTR from the values of MTTF, availability (A), and unavailability (UA = $1 - A$).

$$\text{MTTF} = \int_0^\infty R(t)\partial t \tag{3}$$

$$\text{MTTR} = \text{MTTF} \times (\frac{UA}{A}), \tag{4}$$

As expected, the system's unavailability is the counterpart of the system's availability and can be measured by subtracting $(1 - A)$. The system's downtime, which corresponds to the amount of time where the system could not be accessed, is usually given in units of time, such as in hours per year: Unavailability $\times$ 8760h, relating both unavailability and the evaluated period.

The availability can also be represented by the number of nines [23], as shown in Table 2. For example, a system with four 9's of availability is classified as fault-tolerant, meaning an annual downtime of nearly 1 hour.

The system's reliability and availability-related metrics and values can usually be obtained through simulation, measurement, and analytical models. The latter was chosen due to the high-level system view provided and their great flexibility in adapting parameters and achieving results faster than the other two evaluation techniques. Regarding analytical modeling, two main trends are highlighted:

- **Combinatorial or non-state-space** models capture conditions that make a system fail (or be working) regarding structural relationships between the system's components [27];
- **State-space models** represent the system's behavior (failures and repair activities) by a set of states and the occurrence of events that can be expressed as rates or distribution functions. These models represent more complex relationships between the system's components than combinatorial models do [18].

Reliability Block Diagram (RBD) [16] and Fault Tree (FT) [17] are among the most prominent combinatorial modeling formalisms, whereas Petri Nets (PN), Continuous Time Markov Chain (CTMC) and Stochastic Automata Networks are well-known state-space modeling formalisms [9,

Table 2 Service availability in number of nines.

# of 9's	**Avail. (%)**	**System Type**	**Downtime (year)**
1	90	unmanaged	5 weeks
2	99	managed	4 days
3	99.9	well managed	9 hours
4	99.99	fault tolerant	1 hour
5	99.999	high availability	5 minutes
6	99.9999	very high availability	30 seconds
7	99.99999	ultra availability	3 seconds

16, 27]. In this paper, RBDs describe the system part that does not present any dependency between components.

Petri Nets [22] is a family of well-suited formalism for modeling several system types, including concurrency, synchronization, communication mechanisms, besides supporting deterministic and probabilistic delays. This paper adopts a particular extension of Petri nets, namely Stochastic Petri Nets (SPN) [21], which allows the association of stochastic delays to timed transitions using the exponential distribution. We present an SPN that is used to represent the baseline non-reparable environment. This SPN comprises the previously presented reliability block diagrams (RBDs) and is used to estimate the meantime to absorption (MTTA) later used on a set of maintenance models. The proposed maintenance models will provide the means to calculate the overall system's availability metrics, considering reactive, preventive, and self-healing routines.

3.2 *Docker, Blockchain and Ethereum Platform*

Docker is a set of related technologies that include a container image format and a runtime library, which manages containers' life cycle through APIs and a command-line tool [2]. Docker is the actual pattern for container creation and management; this technology enables users and companies to virtualize their resources without the need to virtualize an entire operating system or, by other means, by just virtualizing the application. This paper evaluates the use of Docker containers running Ethereum blockchain images to perform distributed service provisioning.

Vitalik Butterin proposed the Ethereum blockchain [6] back in 2013. It was the first blockchain platform to include a complete program language (Solidity) and the possibility to develop a distributed application (Dapp) that both users and service providers could maintain. The Ethereum platform is portable, such as the Java computing Language, where the developed applications may run in any environment due to the Java Virtual Machine (JVM). The Ethereum distributed applications (Dapps), written in Solidity, run over the Ethereum Virtual Machine (EVM).

An Ethereum node is the client that enables one to participate in an Ethereum network, and many Ethereum clients can turn a device into an Ethereum node. Some clients are provided by the Ethereum community, such as Geth and Trinity, while others are from third-party developers, which is Parity.

The Ethereum platform is provided as a cloud service in cloud computing well-known providers, which are Amazon AWS, Microsoft Azure, and Google Cloud Platform. Blockchain-as-a-Service became a trend among those service providers and will be even more present in daily lives since banking services, and other distributed applications will migrate from traditional cloud models to a more safe and secure mechanism Blockchain.

By general means, a blockchain application is based on shared ledgers used to store transactions' records into entities called blocks, inserted one after another into a kind of linked list [10]. Besides storing data about the performed transactions, each block also has a pointer to the immediately preceding block. Figure 1 shows how a typical blockchain works.

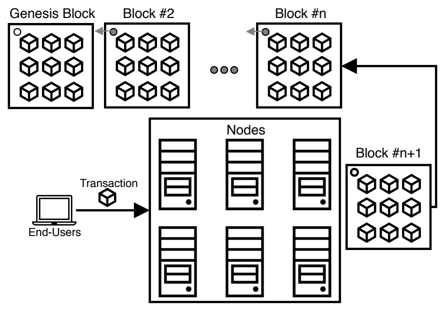

Figure 1: How does Blockchain Works?

End-users request a peer-to-peer network composed primarily of conventional machines or dedicated servers. These machines receive the names of nodes in a P2P architecture and are responsible for executing, ordering, and validating the requested transactions. Each transaction is inserted into a block with other transactions. Each block is then inserted into a chain of blocks. Such a chain is called a blockchain.

4 Modeling and Evaluation Methodology

This section presents how we had accomplished our primary goals and how this work can be replicated. At first, the required hardware and software resources are defined, as well as their expected behavior. The list of requirements apply to any computer system, including cloud platforms and their respective services, and corresponds to one of the steps of the system's understanding process [13].

The prerequisites to create a blockchain environment based on the Ethereum private platform image include any modern hardware running Linux, MacOSX, or Microsoft Windows 10 Operating System with support to Docker Engine 18+. The Docker Engine is responsible for creating and managing the Ethereum containers.

After listing our relevant requirements and understanding how they interact with each other, we may determine how to evaluate the system, which includes a path to obtain the needed information about it [13]. Figure 2 shows an organization chart that summarizes our strategy.

The rectangles represent each step of the methodology, and arrows connecting the rectangles define the execution order. The evaluator only advances to the next step after the completion of the current one. The diamond represents a step that can lead to two different paths, that is, if the results are considered satisfactory, the evaluation proceeds; otherwise, it returns to any of the previous steps, where adjustments will be made until it shows a behavior close to the one expected from a real system. The dashed circles distinguish the subcategories of the methodology steps.

4.1 Preliminary Study

The preliminary study covers the first five steps of the supporting methodology: (1) system understanding; (2) deployment requirements; (3) survey of the parameters and metrics that will be evaluated; (4) a survey of the input values for the models, which are proposed in (5).

System understanding: this step consists of understanding the Ethereum platform and the relation between the Docker container engine and the Ethereum image.

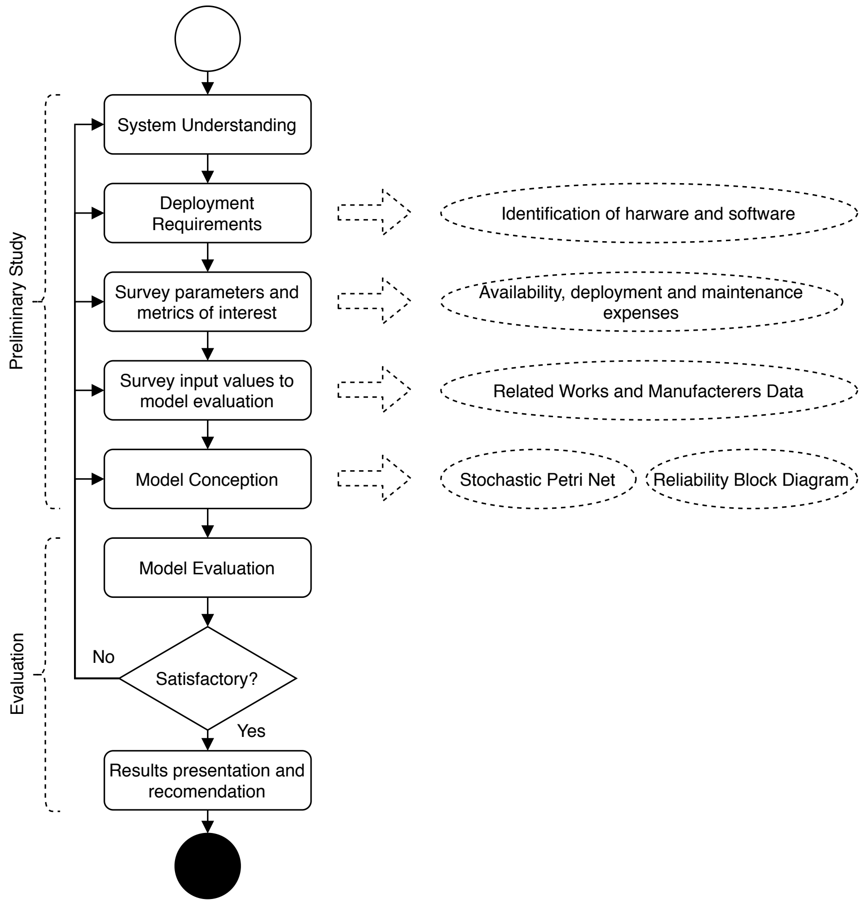
Preliminary Study
Evaluation
System Understanding
Deployment Requirements
Survey parameters and metrics of interest
Survey input values to model evaluation
Model Conception
Model Evaluation
Satisfactory?
No
Yes
Results presentation and recomendation
Identification of harware and software
Availability, deployment and maintenance expenses
Related Works and Manufacterers Data
Stochastic Petri Net
Reliability Block Diagram

Figure 2: Supporting Methodology

Deployment Requirements: identification of the main requirements for service deployment.

Survey parameters and metrics of interest: the third step corresponds to the establishment of the system metrics that will be evaluated. Choosing metrics and evaluation parameters that have a low impact on the user and administrator perception will lead to a waste of time and resources [13].

Survey input values to models evaluation: based on the system understanding and as a result of previous activities, we may obtain the input values that the proposed models will use by consulting previous knowledge, related works, and manufacturers' specifications.

Models Conception: after defining the parameters, metrics of interest, and the main components required to run the service, it is possible to propose high-level models that allow obtaining a set of preliminary results.

4.2 Evaluation

The evaluation process has two main steps: (I) model evaluation and (II) the presentation and recommendations based on the obtained results.

Model Evaluation: the proposed models were evaluated through the Mercury tool [15], right after it be fed with an input of the obtained mean time to failure (MTTF) and mean time to repair (MTTR) values; as an output, this step provides the required artifacts to accomplish the data presentation process.

Results presentation and recommendation: this step is characterized by data representation, which can be done through graphs and tables and will depend on the audience to which the analyst wishes to present the results. The results will include the proposed models, the reliability and availability evaluation of these models. The models, evaluations, and raw data can be seen and understood by designers, analysts, and administrators, but hardly by top management; hence it important to provide graphical and textual interpretation available to support the decision-making process.

5 Proposed Architectures and Models

This section presents the evaluated environment. A baseline architecture managed by a single organization, this environment can host several Ethereum containers that could handle or perform transactions for one or more distributed applications. Figure 3 shows a high-level view of the proposed environment.

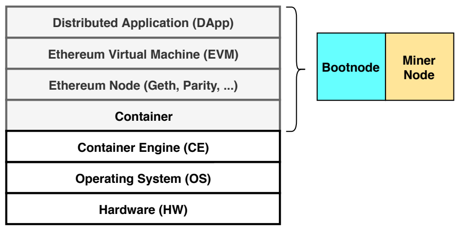
Distributed Application (DApp)
Ethereum Virtual Machine (EVM)
Ethereum Node (Geth, Parity, ...)
Container
Container Engine (CE)
Operating System (OS)
Hardware (HW)
Bootnode
Miner Node

Figure 3: Service Provisioning Stack

There are four different components in the proposed environment: the server's hardware (HW), operating system (OS), container engine (Docker Engine), and the deployed containers. There are two types of containers, the boot node and the miner node. The boot node is a dummy node that can be used only to connect the pairs nodes or other containers to exchange data, transactions, and blocks between themselves. The boot node does not execute any job that requires considerable computational power. However, it is essential because when an application needs to start, the miner nodes need to connect to the boot node. The miner node is the node responsible for executing computationally intensive jobs and performing transactions by solving mathematical puzzles to obtain the required credentials to write a block on the Blockchain.

5.1 Primary Models

To represent this architecture and the fact that all components must be operational to perform service provisioning, we considered two-stage hierarchical modeling. In the first stage, RBDs are adopted to the main system components. The RBD presented in Figure 4 depicts the basic server components (hardware, operating system, and container engine (Docker Engine)).

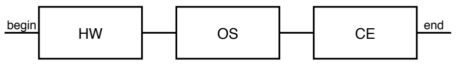

Figure 4: RBD for non-dependent components

This RBD's components are in series since if one of its components fails, the whole subsystem will fail. After evaluating the server RBD, the respective MTTF is obtained. The MTTR values will be discussed later on. The next step on the system's modeling deals with container type, the miner, and the boot node, which was done through another RBD. Figure 5 presents the containers RBD, which can be used to represent both types of Ethereum nodes.

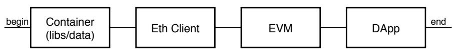

Figure 5: RBD for Miner and Bootnode containers

Finally, Figure 6 presents the top-level environment model, which hierarchically relates both RBDs previously depicted and represented each of the main components in an Ethereum environment. The Service Infrastructure is represented by a Stochastic Petri Net (SPN) due to the dependency between the miner, the boot node, and the container engine.

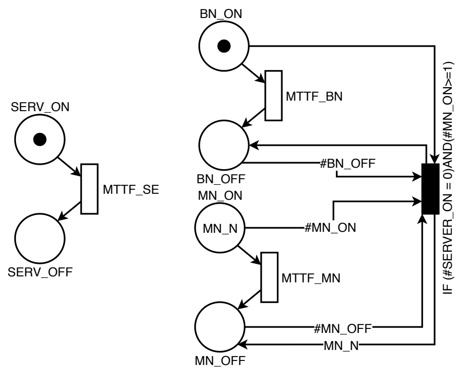

Figure 6: Baseline non-reparable Environment

This SPN depicts three components. The first component is the basic server, whose MTTF_SE is obtained by evaluating the previously presented RBD (See Figure 4). The other two components presented in Figure 3 are represented (each one) by the RBD presented on Figure 5. The places (circles) SERV_ON, BN_ON (Boot Node On), and MN_ON (Miner Node On) mark the upstate for each component, respectively the Server, and both Bootnode and Miner Node containers, while their counterpart, that means a mark on places SERV_OFF, BN_OFF, and MN_OFF indicates that the component is in a downstate.

Initially, every component is operational (upstate); each component has a mean time to fail (MTTF) associated. These times follow an exponential distribution. They are represented in the model by timed transitions (white rectangle) connected through arcs (arrows) to the place that corresponds to the current component state (UP or DOWN).

There are two types of transitions, the timed transition that depends on time to activate and the black rectangle, which is the transition called immediate transition. The immediate transition may be activated if the required amount of tokens are present in the connected place. The proposed model has one immediate transition, used to change the state of a component if this component is dependent on another that failed. For example, if the server enters into a failure state, all containers (miners and boot node) will fail immediately as well, meaning that in this case, the system does not wait for the meantime to fail associated with the timed transitions in order to perform a failure. Table 3 describes what happens when we have a token in a specific place and specific transition fires due to the passage of time.

5.2 Maintenance Models

Three different maintenance routines are considered in this paper. The first one is Reactive and comprises one of the most common maintenance routines, which is characterized by the fact that a system tends to fail, and this failure must be detected to call be made to the maintenance teams. Figure 7 presents an SPN that represents this maintenance routine.

At first, the SERV, which includes all the components presented on the baseline model (see Figure 6), is operational (SERV_ON) and has an MTTF associated. After the required time has passed, the SERV enters the failure state (SERV_OFF). A detection time (DTC) is required to pass in order for the failure to be detected. This time is considered administrative time; a client must contact the service provider or, at least, an administrator must take note of the current system state to perform a call to the maintenance team. A maintenance team (MAIN_TEAM) must be available when the call is performed, so it can move to the system's site or data center (M2S) and perform the required repair according to an expected amount of time (MTTR). Table 4 describes how tokens move from a place to another based on the timed transitions.

The next maintenance routine is an improvement from the reactive maintenance policy and includes the possibility to perform preventive maintenance. This new policy is also one of the most commons when dealing with the life cycle of both hardware and software. Figure 8 presents the SPN that represents this routine.

This routine includes the preventive transition (PREV) and considers a certain amount of time before performing preventive maintenance; this time has to be great enough not to impact the system's availability. Being important to point out that when applying preventive maintenance, the system goes down. Table 5 presents the next state function for this model.

The third and last maintenance routine evaluated by this paper considers a self-healing mechanism associated with reactive maintenance routines. This mechanism is based on a monitor application that checks if the SERV is operational, and if it is not, the mechanism tries to reboot it. The reboot may or may not happen, and we must ROLL a probability rate to reach a previous state. This approach is based on the fact that most failures will be software failures and not hardware ones, meaning that the maintenance team may not be required. However, the monitor itself may fail. All these possibilities are considered in this model, presented in Figure 9.

The Table 6 shows the next place function for this model. It is important to mention that the immediate transition SUCCESS has a success probability of 80%, against 20% of the FAIL transition (these values have been estimated). When reactive maintenance is performed, it also repairs the monitor to a previous state, meaning it can keep working.

6 Case Studies

This section provides three case studies that demonstrate how feasible are the proposed models: (1) model evaluation, (2) service level agreements definition, (3) cost-benefit evaluation. At first, we need to obtain the required system's input values to feed our models and perform the availability evaluation. Some of these values were

Table 3 $\delta-$function for each place on baseline model

Place/Trans.	MTTF_SE	MTTF_BN	MTTF_MN
SERV_ON	SERV_OFF	-	-
SERV_OFF	-	-	-
BN_ON	BN_OFF	BN_OFF	-
BN_OFF	-	-	-
MN_ON	MN_OFF	-	MN_OFF
MN_OFF	-	-	-

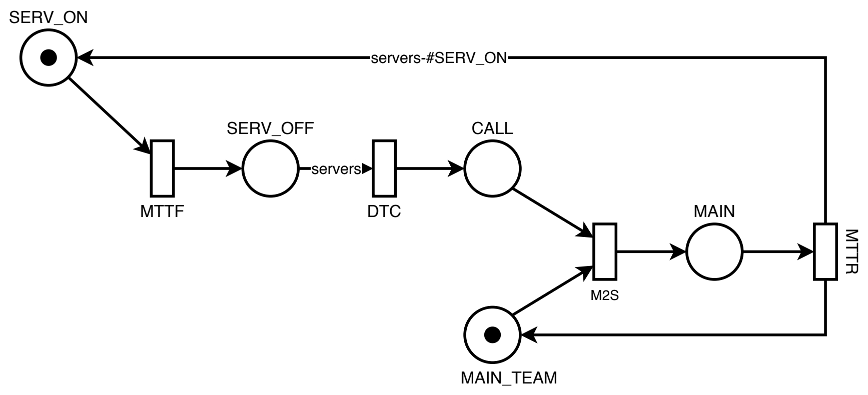

Figure 7: Reactive Maintenance Routines

Table 4 $\delta-$function for each place on reactive model

Place/Trans.	MTTF	DTC	M2S	MTTR
SERV_ON	SERV_OFF	-	-	
SERV_OFF	-	CALL	-	
CALL	-	-	MAIN.	
MAIN_TEAM	-	-		
MAIN.	-	-	-	SERV_ON /MAIN_T

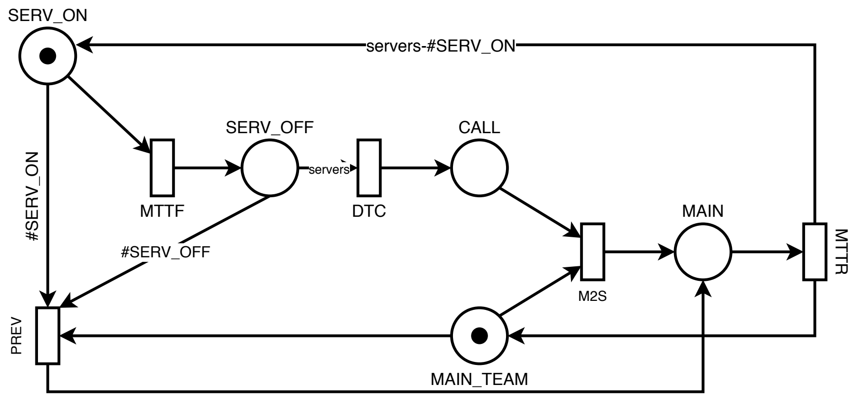

Figure 8: Reactive + Preventive Maintenance Routines

Table 5 $\delta-$function for Reactive+Preventive routine

Place/Trans	**MTTF**	**DTC**	**PREV.**	**M2S**	**MTTR**
SERV_ON	SERV_OFF	-	MAIN.	-	-
SERV_OFF	-	CALL	MAIN.	-	-
CALL	-	-	-	MAIN.	-
MAIN_T.	-	-	MAIN.		-
MAIN.	-	-	-	-	SERV_ON /MAIN_T

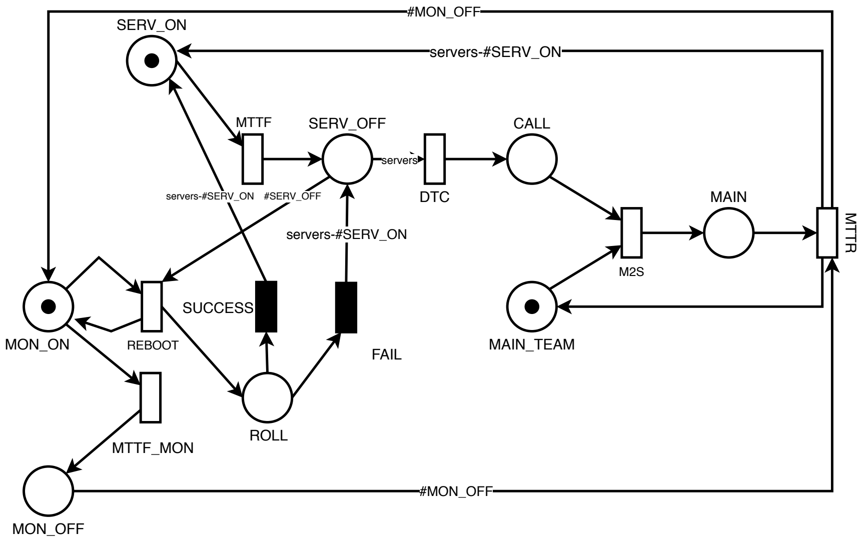

Figure 9: Reactive and Self-Healing Maintenance Routines

Table 6 $\delta-$function for each place on reactive/self

Place/Trans.	**MTTF**	**DTC**	**M2S**	**REBOOT**	**SUCCESS**	**FAIL**	**MTTR**
SERV_ON	SERV_OFF	-	-	-	-	-	-
SERV_OFF	-	CALL	-	ROLL	-	-	-
CALL	-	-	MAIN.	-	-	-	-
MAIN_TEAM	-	-		-	-	-	-
MAIN.	-	-	-	-		-	SERV_ON /MON_ON
MON_ON	-	-	-	ROLL /MON_ON	-	-	-
MON_OFF	-	-	-	-	-	SERV_OFF	-
ROLL	-	-	-	-	SERV_ON	-	-

obtained from a literature review [7, 19, 25], while other ones were extracted from manufacturer charts and white papers. The values used as an input to evaluate the RBDs and the SPN baseline model are presented in Table 7.

Table 7 Input Parameters for Primary Models

Component	MTTF (h)
Hardware (HW)	8760
Operating System (OS)	2893
Docker Engine (DE)	2516
Container	1258
EVM	788.4
Eth Client	788.4
DApp	788.4

6.1 Models Evaluation

The first set of models to be evaluated were the RBDs (See Figures 4 and 5), they provide the required input value to the SPN that corresponds to the baseline environment (See Figure 6). These models were evaluated on Mercury Tool [15] and are considered the primary models; the obtained values from their evaluation are presented in Table 8.

Table 8 Values from Primary Models Evaluation

Model	MTTF (h)
RBD Server	1166
RBD Container	272
Baseline SPN	1170

The system's reliability has been evaluated from the baseline SPN model (See Figure 6). Figure 10 presents the result of this evaluation considering 60 days or 1440 hours. At the initial moment (instant 0), the reliability is 1, meaning that the probability of the system to be working is a 100%, this probability varies until it reaches close to 0, as time tends to infinity, after only five days, the reliability is equal to 0.4932, meaning that probability of a failure occurs is more than 50%. From the ninth day forward, the probability of a failure to occur is higher than 99%, meaning that the maintenance team will be called to the site any time soon.

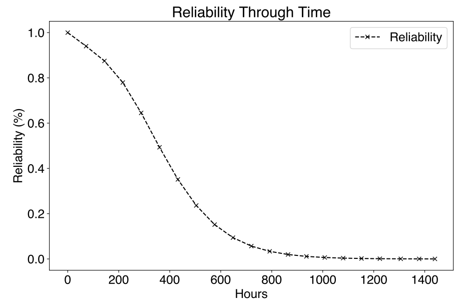

Figure 10: System's Reliability through time

The next step of our analysis comprehend the evaluation of the availability considering the maintenance models (See Figures 7, 8 and 9).

The values obtained from the primary models' evaluation are just a part of the required inputs to the maintenance models. It is essential to remember that the maintenance models require an expected repair time and a time to detect a system failure to perform preventive maintenance. These values are estimated and presented in Table 9.

Table 9 Input values to the Maintenance Models

Transition	Delay (h)
MTTF	1170
DTC	0.5
M2S	[4, 2, 1]
MTTR	1
MTTF_MON	788.4
PREV	2634

It is important to mention that transition M2S's delay varies according to the SLA, as we will show in the next subsection, including the availability for each of the maintenance models.

6.2 Service Level Agreements

A survey characterizes the deployment and maintenance expenses evaluation. At first, we list all the required components to accomplish the service provisioning. As both applications and operating systems are open-source technologies, we must focus on the servers that will host these applications.

After listing the resources, we have surveyed the server acquisition cost. An analysis based on web search, focusing on e-commerce sites such as Amazon, HPE, and DELL, provided us with a server that can be powerful but not too expensive. The HPE ProLiant DL360 Gen10 was chosen and cost US$1,884, which was the lowest value found in June of 2020.

We define a set of Service Level Agreements (SLA). Each one differs from the other in the value of the contract and the amount of time required to reach the site and perform the required maintenance routine. As parameterizable resources (See Figures 7, 8 and 9), the M2S transition, as well as any other transition on the proposed models, can have their delays redefined to meet any required value and achieve an aimed availability value.

As expected, we considered the same set of architectures previously evaluated in Section 6. Then, by doing a full combination of architectures, maintenance routines, and SLAs, we achieved Table 10.

Table 10 presents three different SLAs applied to the ten different scenarios evaluated. Each SLA has different characteristics, such as the cost of the contract in dollars, the temporal cost of moving to the location of the failure, and expected availability. which presents the possibilities, general availability for each maintenance model, and the values for each SLA. Also, it is important to mention that the value chosen for reactive maintenance is 25% percent of the server acquisition value, while the value for preventive maintenance is 20% of the value of reactive maintenance.

6.3 Cost-Benefit Evaluation

Knowing how much cost a server and having defined three possible SLAs, three possible maintenance routines, and a set of architectures, we aim to identify the best cost-benefit relationship. To determine the architectures with the best cost-benefit relationship, we must define the benefit we are seeking. We consider an excellent value between the obtained annual downtime and the deployment and maintenance expenses for this work. However, they are of different greatness, and a normalization process can be used to put them into the same interval, something between 0 and 1. Equation 5 presents the normalization process conducted.

$$\text{NormXY} = \frac{\text{NumX - MinNumX}}{\text{MaxNumX - MinNumX}} \tag{5}$$

Where X stands for Cost and Downtime, and Y for the respective architecture, MinNumX is the Minimum X value for the architectures, and MaxNumX is its counterpart.

Now in possession of the normalized cost and downtime values, we must relate them somehow, so we use the Euclidean distance. The architecture with less distance to the origin tends to be the best one. The Euclidean Distance calculation may be done through the following equation: DistanceZ = $\sqrt{\text{NMCost}^2 + \text{NMDowntime}^2}$, where Z stands for architecture and NM for the respectively associated metric, either Normalized Cost or Normalized Downtime.

Table 11 provides a ranking with the top-5 architectures considering their Euclidean Distance and pointing out their respective SLA, maintenance policy, annual downtime, as well as their deployment and maintenance expenses.

The ranking showed that both architecture (Arc.) 8, under SLA 1,

Table 10 SLA

Value					
Reactive (US$)	471	**Preventive (US$)**	94.2		
		SLA 1			
Expected Availability:	99%	**Contract Value (US$):**	2,000	**Time to Move to Site**	4h
Maintenance	**Reactive Calls**	**Preventive Calls**	**#Servers**	**Annual Expenses (US$)**	**Obtained Av. (%)**
Reactive	7.6	-	1	5,579	99.53
Reactive	3.4	-	2	3,601	99.76
Reactive	2.2	-	3	3,036	99.84
Reactive	1.5	-	4	2,706	99.88
Preventive+Reactive	7.4	3.1	1	5,777	99.51
Preventive+Reactive	2.9	3.2	2	3,667	99.76
Preventive+Reactive	1.4	3.1	3	2,951	99.84
Preventive+Reactive	0.8	3.2	4	2,678	99.90
Self+Reactive	4.6	-	1	4,166	99.68
Self+Reactive	2.9	-	2	3,365	99.82
Self+Reactive	1.8	-	3	2,847	99.86
Self+Reactive	1.5	-	4	2,706	99.89
		SLA 2			
Expected Availability:	99,9%	**Contract Value (US$):**	2,500	**Time to Move to Site**	2h
Maintenance	**Reactive Calls**	**Preventive Calls**	**#Servers**	**Annual Expenses (US$)**	**Obtained Av. (%)**
Reactive	7.6	-	1	6,079	99.70
Reactive	3.4	-	2	4,101	99.85
Reactive	2.2	-	3	3,536	99.90
Reactive	1.5	-	4	3,206	99.92
Preventive+Reactive	7.4	3.1	1	6,277	99.68
Preventive+Reactive	2.9	3.2	2	4,167	99.83
Preventive+Reactive	1.4	3.1	3	3,451	99.90
Preventive+Reactive	0.8	3.2	4	3,178	99.92
Self+Reactive	4.6	-	1	4,666	99.79
Self+Reactive	2.9	-	2	3,865	99.90
Self+Reactive	1.8	-	3	3,347	99.92
Self+Reactive	1.5	-	4	3,206	99.94
		SLA 3			
Expected Availability:	99,99%	**Contract Value (US$):**	3,000	**Time to Move to Site**	1h
Maintenance	**Reactive Calls**	**Preventive Calls**	**#Servers**	**Annual Expenses (US$)**	**Obtained Av. (%)**
Reactive	7.6	-	1	6,579	99.78
Reactive	3.4	-	2	4,601	99.89
Reactive	2.2	-	3	4,036	99.92
Reactive	1.5	-	4	3,706	99.94
Preventive+Reactive	7.4	3.1	1	6,777	99.72
Preventive+Reactive	2.9	3.2	2	4,667	99.88
Preventive+Reactive	1.4	3.1	3	3,951	99.91
Preventive+Reactive	0.8	3.2	4	3,678	99.93
Self+Reactive	4.6	-	1	5,166	99.85
Self+Reactive	2.9	-	2	4,365	99.91
Self+Reactive	1.8	-	3	3,847	99.94
Self+Reactive	1.5	-	4	3,706	99.95

Table 11 Architecture Ranking

Rank	**Arc.**	**Main. Policy**	**SLA**	**Cost US$**	**Down. (h)**	**Distance**
1st	8	Prev.+Reactive	1	2,678	8.76	0.1136
2nd	12	Self+Reactive	2	3,206	5.25	0.1363
3rd	12	Self.+Reactive	1	2,706	9.36	0.1365
4th	8	Prev.+Reactive	2	3,178	7.00	0.1397
5th	4	Reactive	2	3,206	7.00	0.1457

presented the best cost-benefit relationship; it considers a Preventive+Reactive maintenance policy. This architecture presented the same annual downtime (Down. (h)) as the 2nd place, that goes to architecture 12 under a self+reactive policy, but with a lower deployment and maintenance expense (Cost US$), meaning that the main difference can be seen in the Euclidean Distance (Distance). It is essential to mention that both architectures contain four servers running the provided service. Plus, considering that the maintenance team is only called when the service is down, there is a lower number of reactive maintenance given the number of servers, their general expense is reduced.

The third place goes to architecture 12, under the SLA 1, over a Self+Reactive maintenance routine. This architecture has a higher annual downtime than the previous ones but a lower deployment and maintenance expense. This architecture is followed by architecture 8, under the SLA 2 with preventive+reactive maintenance policy. Finally, we have architecture 4, which is under SLA 2, and a Reactive policy. As can be seen, none of the top five environments runs under SLA 3, meaning that the extra US$500 does not guarantee any improvement on availability, even considering a lower time to move to the site. Also, it is important to highlight that the top 4 architectures deal with preventive and self-healing maintenance routines, and they also present a better cost-benefit relationship given the higher expenses to accomplish reactive maintenance.

Figure 11 presents the general results of the sensitivity analysis considering all the scenarios evaluated. We see that in most of cases, SLAs 2 and 3 showed better results in terms of cost-benefit and stand out considerably over scenarios whose maintenance takes place only in a reactive manner with 4+ hours to move to the site.

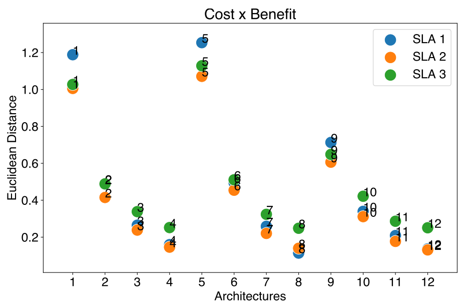

Figure 11: Cost-Benefit Evaluation

7 Conclusion and Final Remarks

This paper proposed a set of models to evaluate the reliability and the impact of maintenance routines on the system's general availability considering three different SLAs. The evaluated system considers Blockchain-as-a-service provisioning, which was also initially modeled as a non-reparable baseline environment. Three case studies demonstrate the feasibility of the proposed models and help those interested in planning Blockchain-as-a-service and many other services and applications that can be provided over private cloud computing environments.

The first case study dealt with the proposed models' evaluation through a set of input values used in the Mercury modeling tool. We could find other values used as input for the maintenance models, including the system's reliability evaluation. After that, we evaluated the system's availability in the second case study considering these maintenance routines under three different SLAs.

In the third and last case study, we have evaluated the cost-benefit considering both deployment and maintenance expenses against the value of the maintenance routines with a focus on the annual downtime. They were normalized in order to be compared since they are of different greatness. As a highlight, we may point out that Preventive and Self-Healing routines had a significant impact on overall availability without raising the maintenance expenses, which could be seen through the Euclidean Distance applied to a set of 36 architectures.

Some limitations of the current work is that the users of the proposed artifacts should have a minimum of prior knowledge about systems modeling, which can make it difficult to present results. In addition, the model validation process is based on estimated input parameters, such as the time required for the maintenance team to travel to the system failure site. However, this time is estimated and parameterized and the reality of the company can be updated.

As future works, we intend to evaluate the maintenability, as well as the impact of the maintenance routines over user-oriented performance metrics, such as system latency and transaction throughput for a set of different applications. Also, a performability study may highlight the impact of performance issues over the general availability of the system and the maintenance frequency.

Acknowledgment

The authors would like to thank the Brazilian Government for the financial support through the Fundação de Amparo à Ciência e Tecnologia de Pernambuco (FACEPE), and the Modeling of Distributed and Concurrent Systems (MoDCS) group for the help on improving this research.

References

[1] Araujo, J., Melo, C., Oliveira, F., Pereira, P., Matos, R.: A software maintenance methodology: An approach applied to software aging. In: 2021 Annual IEEE International Systems Conference (SysCon), pp. 1–8. IEEE (2021)

[2] Arundel, J., Domingus, J.: Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud. O'Reilly Media (2019)

[3] Avizienis, A., Laprie, J., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing **1**, 11–33 (2004)

[4] Avižienis, A., Laprie, J., Randell, B., of Newcastle upon Tyne. Computing Science, U.: Fundamental Concepts of Dependability. Technical report series. University of Newcastle upon Tyne, Computing Science (2001). URL `https://books.google.com.br/books?id=cDkmGwAACAAJ`

[5] Blancke, O., Combette, A., Amyot, N., Komljenovic, D., Lévesque, M., Hudon, C., Tahan, A., Zerhouni, N.: A predictive maintenance approach for complex equipment based on petri

net failure mechanism propagation model. In: PHM Society European Conference, vol. 4 (2018)

[6] Buterin, V., et al.: Ethereum white paper. GitHub repository **1**, 22–23 (2013)

[7] Callou, G., Ferreira, J., Maciel, P., Tutsch, D., Souza, R.: An Integrated Modeling Approach to Evaluate and Optimize Data Center Sustainability, Dependability and Cost. Energies **7**(1), 238–277 (2014). DOI 10.3390/en7010238. URL `http://www.mdpi.com/1996-1073/7/1/238`

[8] Elusakin, T., Shafiee, M.: Reliability analysis of subsea blowout preventers with condition-based maintenance using stochastic petri nets. Journal of Loss Prevention in the Process Industries **63**, 104,026 (2020)

[9] Garg, S., A, P., M, T., Trivedi, K.S.: Analysis of software rejuvenation using markov regenerative stochastic petri net. In: Proc. In: Sixth International Symposium on Software Reliability Engineering, (ISSRE'95), pp. 180–187. Paderborn (1995)

[10] Gupta, M.: Blockchain for DUMMIES. John Wiley & Sons, Inc. (2017)

[11] Hamidi, M., Liao, H.: Maintenance outsourcing contracts based on bargaining theory. In: Optimization and Dynamics with Their Applications, pp. 257–279. Springer (2017)

[12] Hou, L., Zheng, W.J.: A latent class regression approach to it maintenance outsourcing service management. IEEE Transactions on Engineering Management (2018)

[13] Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley Computer Publishing, John Wiley & Sons, Inc., New York (1991)

[14] Lee, H., Cha, J.H.: New stochastic models for preventive maintenance and maintenance optimization. European Journal of Operational Research **255**(1), 80–90 (2016)

[15] Maciel, P., Matos, R., Silva, B., Figueiredo, J., Oliveira, D., Fé, I., Maciel, R., Dantas, J.: Mercury: Performance and dependability evaluation of systems with exponential, expolynomial, and general distributions. In: 2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 50–57 (2017). DOI 10.1109/PRDC.2017.16

[16] Maciel, P., Trivedi, K., Matias, R., Kim, D.: Dependability modeling. In: Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions (2011)

[17] Malhotra, M., Trivedi, K.: Power-hierarchy of dependability-model types. Reliability, IEEE Transactions on **43**(3), 493–502 (1994). DOI 10.1109/24.326452

[18] Matos, R., Araujo, J., Oliveira, D., Maciel, P., Trivedi, K.: Sensitivity analysis of a hierarchical model of mobile cloud computing. Simulation Modelling Practice and Theory **50**, 151 – 164 (2015). DOI https://doi.org/10.1016/j.simpat.2014.04.003. URL `http://www.sciencedirect.com/science/article/pii/S1569190X14000616`. Special Issue on Resource Management in Mobile Clouds

[19] Melo, C., Dantas, J., Oliveira, D., Fé, I., Matos, R., Dantas, R.,

Maciel, R., Maciel, P.: Dependability evaluation of a blockchain-as-a-service environment. In: 2018 IEEE Symposium on Computers and Communications (ISCC), pp. 00,909–00,914. IEEE (2018)

[20] Melo, F.F.L., Junior, J.F.S., de Almeida Callou, G.R.: Evaluating the impact of maintenance policies associated to sla contracts on the dependability of data centers electrical infrastructures. Revista de Informática Teórica e Aplicada **27**(1), 13–25 (2020)

[21] Molloy, M.K.: On the integration of delay and throughput measures in distributed processing models. Ph.D. thesis, University of California, Los Angeles, USA (1981). AAI8201138

[22] Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (1989). DOI 10.1109/5.24143

[23] Öhmann, D., Simsek, M., Fettweis, G.P.: Achieving high availability in wireless networks by an optimal number of rayleigh-fading links. In: 2014 IEEE Globecom Workshops (GC Wkshps), pp. 1402–1407. IEEE (2014)

[24] Pereira, P., Araujo, J., Torquato, M., Dantas, J., Melo, C., Maciel, P.: Stochastic performance model for web server capacity planning in fog computing. The Journal of Supercomputing pp. 1–25 (2020)

[25] Sebastio, S., Ghosh, R., Mukherjee, T.: An availability analysis approach for deployment configurations of containers. IEEE Transactions on Services Computing (2018)

[26] Sousa, E., Maciel, P., Lins, F., Marinho, M.: Maintenance policy and its impact on the performability evaluation of eft systems. International Journal of Computer Science, Engineering and Applications **2**(2), 95 (2012)

[27] Trivedi, K.S., Hunter, S., Garg, S., Fricks, R.: Reliability analysis techniques explored through a communication network example (1996)

[28] Vaqueiro, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds: Towards a cloud definition. Computer Communication Review **39**, 50–55 (2009)

[29] Verhagen, W.J., De Boer, L.W.: Predictive maintenance for aircraft components using proportional hazard models. Journal of Industrial Information Integration **12**, 23–30 (2018)