Dependability Evaluation of a Blockchain-as-a-Service Environment

Carlos Melo*, Jamilson Dantas*, Danilo Oliveira*, Iure Fé*, Rubens Matos§,
Renata Dantas*, Ronierison Maciel* and Paulo Maciel*
*Informatics Center, Federal University of Pernambuco, Recife, Brazil
§Institute of Education, Science, and Technology of Sergipe
{casm3, jrd, dmo4, isf, rcspd, rsm4, prmm}@cin.ufpe.br*, rubens.matos@gmail.com§

***Abstract*—The blockchain shared ledger emerged as an alternative to the bureaucratic banking system that may take days to confirm a payment or a transfer between clients. The blockchain concept evolved and became viable for various applications beyond the domain of financial transactions. Blockchains become a way to reach better relationships through contract validation, documents transfer, and personal and business data security. Recently, the blockchain-as-a-service has debuted on Microsoft Data Centers, and now many share an infrastructure that can change and improve their security routines. This paper evaluates the feasibility of a blockchain-as-a-service infrastructure and helps those who plan to deploy or sell blockchains. A modeling methodology based on Dynamical Reliability Block Diagrams (DRBD) is adopted to evaluate two dependability attributes: system's reliability and availability. The proposed infrastructure contains the minimum requirements to deploy the Hyperledger Cello, a platform to create and manage blockchains. The availability results pointed out a system downtime of 121 hours per year and reliability issues that must be addressed when building blockchain-as-a-service infrastructures.**

I. Introduction

Through the years the monetary system evolved since grains barter until coins, paper, titles, plastic cards and so on. All of them aim buyers and sellers protection [1], but for modern times, with malware proliferation, spam, robbery, and where one needs to take their money until banks reveal the need of an improved system. The blockchains enable one to leave their homes without cash or pay high maintenance taxes for bank services and with an even less bureaucratic system that makes the transactions take less time than it usually should [2].

Cloud computing environments may host and manage blockchains, as well as the people who will have access to this service, whether to conduct transactions or merely monitor them. However, this environment may present failures and difficulties in the service provisioning that can lead to a more significant amount of time to carry out a transaction, which is in contravention of the blockchain precepts. The availability and reliability of cloud computing systems have great importance to those who plan to contract or deliver shared ledgers through these environments, most to keep up with stringent Service Level Agreements (SLA).

This paper has as the primary objective the dependability evaluation of a blockchain-as-a-service provider through Dynamical Reliability Block Diagrams (DRBDs). The service deployment happens into an architecture that contains the minimum amount of resources to provide blockchains. The chosen platform was the Hyperledger Cello one of the many projects hosted by Hyperledger and managed by Linux Foundation. The behavioral models describe the infrastructure components, as well as their relationship. The research contributions are subsidies and artifacts to people or organizations that plan to offer or acquire blockchains. By knowing the systems availability and reliability beforehand, those service providers may apply high availability techniques such as redundancy or preventive maintenance and improve their SLA. Among our main **contributions** those who deserve highlights are listed below:

- Blockchain-as-service dependability models proposition and evaluation;
- A blockchain-as-a-service contextualization;
- A sensitivity analysis evaluation to identify which components impacts the most on system's availability.

This paper is organized as follows. The Section II shows this research's related works and their main differences to ours. The Section III presents dependability and modelling concepts. Already the Section IV presents the Blockchain-as-a-service concepts, as well as the proposed evaluation scenario based on Hyperledger Cello deployment. Section VI shows the dependability models that represents the proposed architecture. The Section VII shows the feasibility of the proposed models through a case study. Finally on Section VIII we set the final remarks about the obtained results.

II. Related Works

This Section presents the works that underlie this paper. Some of them are blockchain related, while others are related to system's modeling. In [3] they combined blockchain storage in an attempt to construct a personal platform based on privacy. An automatic access-control protocol that enables blockchain management implementation able to run transactions and system's instructions that do not need to be trusted or reevaluated by third parties.

In [4] a review of blockchains and smart contracts as mean to facilitate the data sharing in an Internet of Things (IoT) environment is proposed, they also point out the main issues and the advantages of this combination. Already in [5] a framework called Hawk was implemented to improve the way that smart contracts work in a blockchain environment.

In [6] the relationship between deployment cost and system's downtime of a synchronization service hosted by a cloud computing platform was evaluated; Reliability Block Diagrams (RBD) and Continuous Time Markov Chains (CTMC) were used to represent the interaction between the resources.

In work done by [7], Mobile Backend-as-a-service was evaluated through hierarchical modeling in two scenarios: baseline one, without automatic repair mechanism, obtaining a lower availability than the second one with an automatic repair routine.

Differently, from the presented studies, this paper proposes a methodology, behavioral models, as well as a sensitivity analysis for blockchain-as-a-service infrastructure deployment. The proposed modeling methodology enables one to reproduce and improve the proposed models to obtain better dependability values to keep up with stringent SLA; already the sensitivity analysis technique points out what is the component that must be improved to obtain better availability values.

III. Background

This section describes the fundamental concepts of system modeling and dependability evaluation, which are necessary for the complete understanding of this paper.

A. Dependability Evaluation

Dependability is the ability of a computer system to provide services that can be both: justifiably and trusted [8]. 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 [9].

In this paper, the system's reliability and availability are the dependability attributes evaluated. The system's reliability is the probability of a system execute the programmed function into a predetermined time limit without any interruption [9]. The system's reliability does not consider the repair time, only the time to a failure occurrence based on a Random Variable **T**.

Already the system's steady-state availability can be represented by the ratio between the Mean Time To Failure (MTTF), which is defined by MTTF = $\int_0^\infty R(t)\partial t$ and Mean Time To Repair (MTTR), which can be find through MTTR = $MTTF \times (\frac{UA}{A})$. The system's unavailability (UA) is $1 - A$. The Equation 1 presents the system availability calculation.

$$A = \frac{MTTF}{MTTF + MTTR} \tag{1}$$

The period when the system is not available is called downtime. Usually, we present the system's downtime in units of time, as in hours per year: Downtime = $UA \times 8760h$, whereas the system's annual uptime, which represents the opposite is $8760h - Downtime$.

Behavioral modeling enables one to find system's dependability attributes. Modeling allows managers and administrators to see what truly matters through a high-end abstract view of complex environments and their relationship. Combinatorial (or non-state space) and state-space models are the dependability models broadly classification.

- Combinatorial models capture conditions that make a system fail (or to be working) regarding structural relationships between the system components.
- State-space models represent the system behavior (failures and repair activities) by its states and event occurrence expressed as rates or distribution functions.

Reliability Block Diagram (RBD) and Fault Tree (FT) are among the most prominent combinatorial modeling formalism, whereas Stochastic Petri Nets (SPN), Continuous Time Markov Chain (CTMC), and Stochastic Automata Networks are widely used state-space modeling formalism [10], [11].

However, despite the high abstraction level, with RBDs and Fault Trees, the priority between system's component and repairs routines are not possible to be modeled [12]. For more complex systems, a most powerful modeling tool is required, such as Dynamic Reliability Block Diagram (DRBD) and Petri Nets.

DRBD is an extension of the traditional RBDs that considers dependencies on dynamic systems, priority between repairs, and resource sharing [13]. Besides if the modeled system is too big for an SPN or the SPN is too complicated that a graphical notation loses its visual appeal, the DRBD becomes an alternative. The DRBD's attributes are known as modeling constructs: SDEP (state dependency) block and SPARE (spare component) block. In this work, we map DRBDs into SPNs, as in Figure 1 that represents the SDEP block behavior.

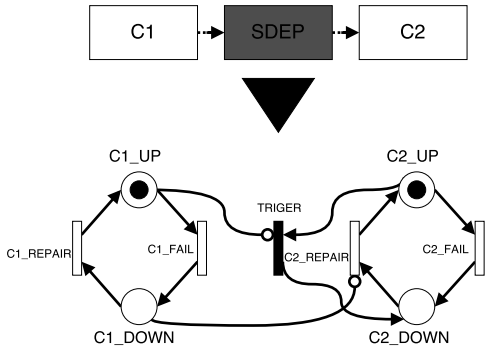

Fig. 1. SDEP block conversion to SPN

The SDEP block can connect a source block to one or more target blocks. If this block enters into failure state, then all target blocks will fail as well, which can be seen in Figure 1.

B. Sensitivity Analysis

To determine the most influential factors on interest metrics Sensitivity analysis techniques are helpful tools [14]. In this paper the percentage difference sensitivity analysis technique is employed, characterized by a sensitivity index known as $S_\theta(Y)$, which indicates the impact of a given measure Y with respect to a parameter θ.

Equation 2 shows how the percentage difference index is calculated for a metric $Y(\theta)$, where $max\{Y(\theta)\}$ and

$min\{Y(\theta)\}$ are the maximum and minimum output values, respectively, computed when varying the parameter θ over the range of its n possible values of interest. If $Y(\theta)$ is known to vary monotonically, so only the extreme values of θ (i.e., θ_1 and θ_n) may be used to compute $max\{Y(\theta)\}$; $min\{Y(\theta)\}$, and subsequently $S_\theta(Y(\theta))$ [15].

$$S_\theta(Y) = \frac{max\{Y(\theta)\} - min\{Y(\theta)\}}{max\{Y(\theta)\}} \quad (2)$$

The $S_\theta(Y)$ computation is done while the values of the model parameters are fixed, which is carried out for each input parameter until all parameters have been analyzed and the one with the most significant impact be found.

IV. Blockchain-as-a-Service

The blockchain-as-a-service concept is characterized by the possibility to create and manage a blockchain network over cloud computing environments [16]. The blockchain concept, as well as the environment used to host and provide this mechanism, are described in this section.

A. Blockchain

The blockchain is a distributed ledger used to record transactions (blocks) over a peer-to-peer network [1]. The Figure 2 shows how the blockchain works, the transactions cannot be canceled or undone and are always forward.

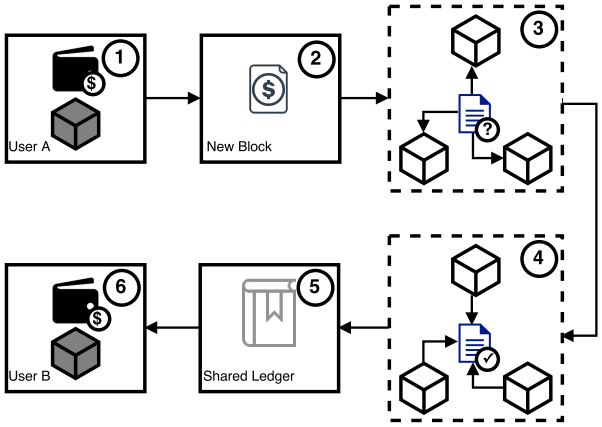

Fig. 2. How does a blockchain works?

1) User A wants to transfer "something" to User B;
2) A new block is created and sent to validation;
3) The new block must be validated by the blockchain network members;
4) The block is validated by the network;
5) The new block is inserted on the shared ledger;
6) User B receives the "something" from User A and the transaction is finished.

This "something" may be a cryptocurrency, a smart contract, a document, or anything that is supported by the provided blockchain. The platform manager defines who and what users can exchange. The platform chosen for this work is the Hyperledger Cello.

B. Proposed Environment with Hyperledger Cello

The Hyperledger is an open consortium hosted by Linux Foundation that aims to improve the advances in blockchain technologies [17]. The Hyperledger Cello is a blockchain module able to provide on-demand blockchain [17]. The environment used to host the Hyperledger Cello consists of two nodes: Master Node and Worker Node. Each of them is responsible to run a series of services as shown in Figure 3.

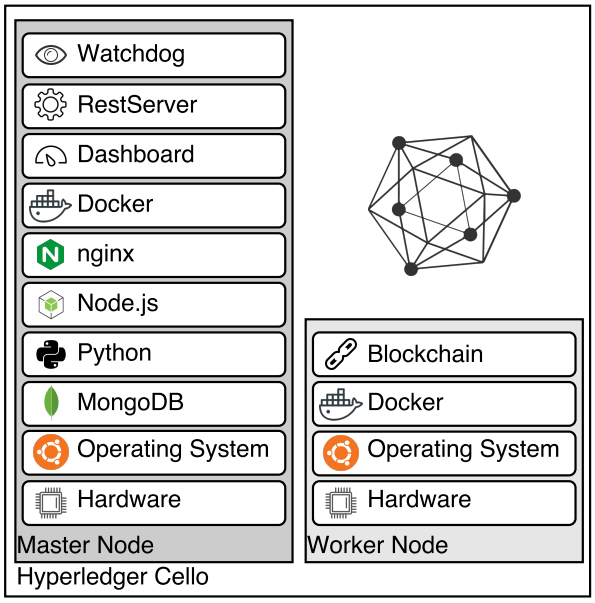

Fig. 3. Proposed Environment with Hyperledger Cello

- The Watchdog is the responsible for monitors the blockchain's service and the system's status [17];
- RestServer makes the environment provision, orchestration, and task management [17];
- Dashboard provides the environment management for system's administrators [17];
- Docker is a container that provides the necessary tools to run and virtualize software and apps over the host Operating System's Kernel [18]. Both Masters and Worker Nodes run the Docker as a host for Hyperledger Cello components;
- nginx is reverse proxy used by the Hyperledger Cello to web performance improvement [17];
- Node.js is a lightweight JavaScript runtime used by Cello to improve provisioning;
- Python runs over the host and provide the means to execute Watchdog, RestServer and the Dashboard into Masters Node [17];
- MongoDB is an open-source distributed database that enables one to query and to index their data [19];
- The Operating System for the proposed environment was the one recommended by Hyperledger Cello development team: Ubuntu Server 16.04 LTS;

V. Modeling Methodology

The first step to availability and reliability evaluation of the proposed environment It is to survey all the hardware and software resources available and needed to deploy a baseline architecture, as presented in Section IV. After listing the most relevant hardware and software resources to do the service deployment, we propose models that represents the environment's behavior. The Figure 4 shows an organization chart that summarizes the used strategy.

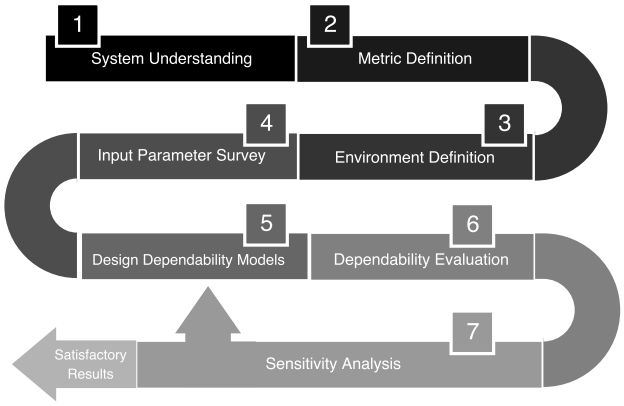

Fig. 4. System's Evaluation Methodology

1) **System Understanding** characterized by the identification of system components, applications, functionalities, as well as the required resources;
2) **Metric Definition** by knowing how the system works, we can choose the metrics of interest to evaluate (Reliability and Availability);
3) **Environment Definition** an architecture containing the minimum components necessary for the blockchain-as-a-service deployment;
4) **Input Parameter Survey** survey of mean time to failure (MTTF) and mean time to repair (MTTR) of the components and applications inherent to the proposed infrastructure;
5) **Design Dependability Models** proposition of combinatorial and state-based models for availability evaluation based on the architectures behavior;
6) **Dependability Evaluation** with the behavioral models that represent the proposed environment, we can evaluate our metrics of interest;
7) **Sensitivity Analysis** based on the dependability values obtained through dependability evaluation a sensitivity analysis was applied to identify which component impacts the most on the metrics of interest.

VI. Dependability Models

DRBDs were used to represent the relationship between the hardware and software components shown on Section IV. This section describes the models used for reliability and availability evaluation.

A. Master's Node

The Master's Node is the machine responsible for providing access and blockchains management. Through master's node, one can create, delete and define who can share the ledger or even who can see what is coming from a user to another. The DRBD that represents the master's node has some dependencies as shown in Figure 3. The master's node as shown in Figure 5 contains hardware, an operating system, MongoDB, Python, node.js, nginx Docker, Dashboard, RestServer, and Watchdog.

The hardware (HW) component is the base; all the software components run over it. If HW enters into a failure state, then all the software components will fail as well. To repair the machine that had a hardware failure the repair routine start by It. After the HW repair the operating system (OS) becomes the next component to be repaired, after that, repairs all the other software running over in SDEP order, which means that the Docker must be repaired first then the Dashboard, RestServer, and Watchdog, but the repair happens after the MongoDB. Dynamical Reliability Block Diagrams works with this concept: Priority and Shared resources. The operational mode (OM) that describes the Master Node is expressed by the relationship between the reliability OR the availability of all components HW $\wedge$ OS $\wedge$ MongoDB $\wedge$ Python $\wedge$ NodeJS $\wedge$ nginx $\wedge$ Docker $\wedge$ Dashboard $\wedge$ RestServer $\wedge$ Watchdog, to manage the service normally, all these components must be working.

B. Worker Node

For the Worker Node, another DRBD is proposed and presented in Figure 6. This DRBD consists of hardware (HW), operating system (OS), and the Docker.

This DRBD describes the dependencies between worker node's components, as previously presented, if the hardware enters into a failure state, then all software components go down as well, as for the operating system it can take with it the Docker and the Blockchains running over It. The OM that describes the Worker Node is expressed by the relationship between the reliability OR the availability of all components running over it HW $\wedge$ OS $\wedge$ Docker all these components must be working as well.

C. Proposed Environment

The proposed environment model is an RBD that combines the Master and Worker DRBDs hierarchically. This model represents the architecture with the minimum requirements to provide a blockchain over the Hyperledger Cello platform. The RBD with two blocks is presented in Figure 7. All blocks must be operational for service provisioning.

In the proposed environment, if any of the components fail the system will not be available, also, any service that was running on the worker will not be accessible from the infrastructure outside. The OM that describes the entire environment depends on the relationship between the reliability OR the availability of Master and Worker nodes.

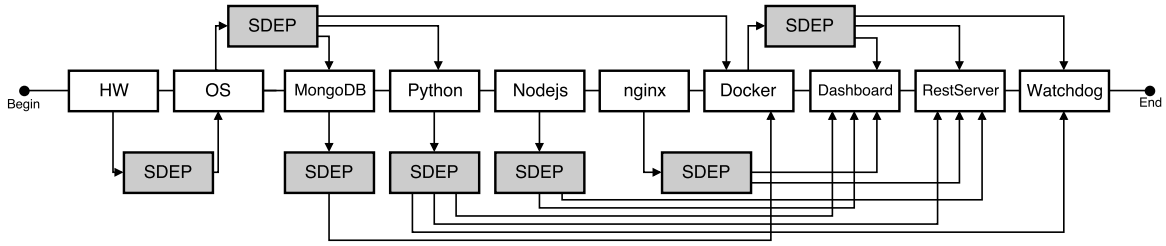

Fig. 5. Master Node's DRBD

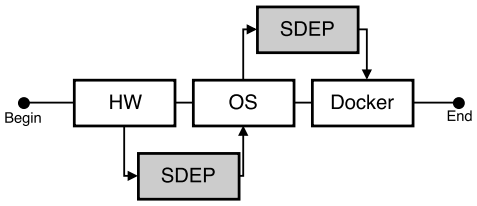

Fig. 6. Worker Node's DRBD

Fig. 7. Proposed Environment's RBD

VII. Case Study

This section provides the case study that demonstrates the feasibility of the proposed models. A list of input values are needed to evaluate our models; Those are in Table I.

TABLE I
Input values for Environment's 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

The proposed model's evaluation was made through Mercury Script Language [20]. The models that represented the proposed environment, as well as the Master and Worker subsystem have their availability results presented in Table II.

TABLE II
Availability Results

Model	Availability (%)
Master Node	98.72
Worker Node	99.89
Environment	98.61

The results pointed out an availability of 98.72% for the Master Node's DRBD, which is the node with the higher amount of components, in comparison with the Worker Node that has an availability of 99.89%. Already the RBD composed by the two DRBD representing the proposed environment, had an availability of 98.61, meaning an annual downtime of 121 hours or more than five entire days.

Since the steady-state availability is not time dependent we need to show the reliability values for a predetermined period, In our case a month (720 hours). Figure 8 shows the variable results for the system's reliability.

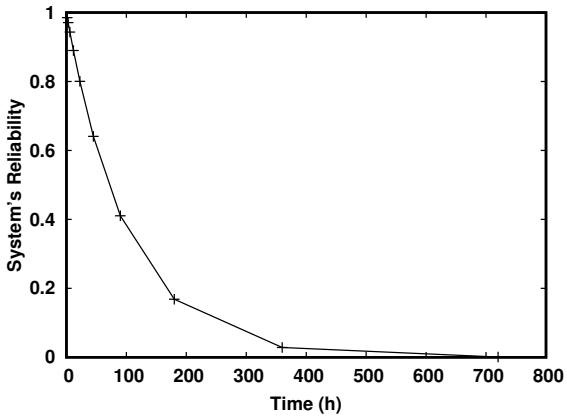

Fig. 8. System's Reliability

The reliability results for thirty days is 0.0008, which happens due to the environment components serialization, the failure of one may imply in the failures of many. After four days the reliability is already less than 50%, implying the need for preventive maintenance or the application of a redundancy technique.

The sensitivity analysis technique evaluates the MTTF and MTTR for each component on the platform (see Table I). The input values for sensitivity analysis are chosen based on the rates and times between failures and repairs for each system component. Each parameter varied between +50% and -50% of its original value. The Table III shows the rank by sensitivity index.

The component with the most significant impact on the environment availability is the MongoDB Mean Time to Failure, ranked at the top, the failure of the MongoDB imply on the failure of Docker and applications running over it, as well as the access to the platform and blockchain creation and maintenance. The Figure 9 shows the impact of the MongoDB MTTF over the baseline availability or the primary availability measured through model evaluation (98.61%).

TABLE III
SENSITIVITY RANKING

Component	Rank	Value
$MTTF_{mongoDB}$	1st	2.92E-03
$MTTF_{os}$	2nd	2.41E-03
$MTTR_{rest,dash}$	3rd	1.90E-03
$MTTR_{docker}$	5th	1.86E-03
$MTTR_{watchdog}$	6th	1.84E-03
$MTTF_{python}$	7th	1.70E-03
$MTTF_{nodejs,nginx,rest,dash,watchdog}$	8th	1.69E-03
$MTTF_{docker}$	13th	1.62E-03
$MTTR_{python}$	14th	1.32E-03
$MTTR_{nodejs}$, $MTTF_{hw}$	15th	1.29E-03
$MTTF_{hw}$	17th	1.22E-03
$MTTR_{hw}$	18th	2.92E-04
$MTTR_{mongoDB}$	19th	2.68E-04
$MTTR_{os}$	20th	2.04E-04

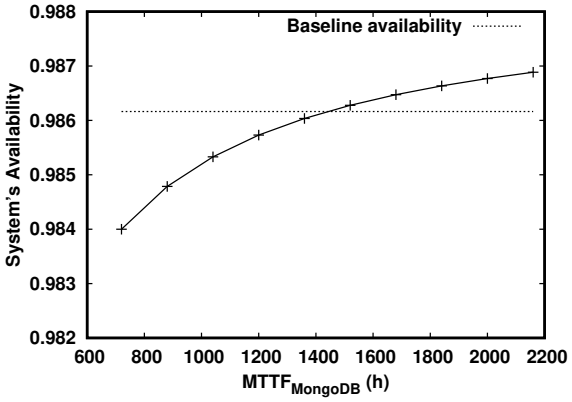

Fig. 9. MongoDB Sensitivity Analysis

The lower the value of MongoDB MTTF the bigger the system downtime, this component is a key for service provisioning, follow closed by Operating System MTTF. By knowing why these components impacts this much on the infrastructure availability, we may apply for redundancies on them and improve the obtained results.

VIII. FINAL REMARKS

This paper evaluated reliability and availability, which are system's dependability attributes, and presented the obtained results through Dynamic Reliability Block Diagrams (DRBDs) that model the system's behavior of an environment that provides blockchain-as-a-service. The availability evaluation results in 121 hours of annual downtime for the proposed architecture with two machines, one as master and another one as a worker.

Already the reliability evaluation shows that the system's has almost 100% of chance to suffer with a failure in less than 31 days. To mitigate this issue periodic maintenance must be done, and redundancy techniques can be applied to the proposed environment as an attempt to improve both availability and reliability results.

As a future work, we intend to evaluate high available environments through redundancy techniques and provide the deployment cost evaluation. The sensitivity analysis proposed here identify which components has the higher impact on availability, and pointed out which components must be improved. Also, a performance evaluation of the proposed architecture can help one to prove the viability of a blockchain transaction.

REFERENCES

[1] M. Gupta, *Blockchain for DUMMIES*. John Wiley & Sons, Inc., 2017.

[2] M. Swan, *Blockchain: Blueprint for a New Economy*. California: OŔeilly., Feb. 2015.

[3] G. Zyskind, O. Nathan, and A. . Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE Security and Privacy Workshops*, May 2015, pp. 180–184.

[4] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[5] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 839–858.

[6] C. Melo, J. Dantas, I. Fe, A. Oliveira, and P. Maciel, "Synchronization server infrastructure: A relationship between system downtime and deployment cost," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct 2017, pp. 1250–1255.

[7] I. Costa, J. Araujo, J. Dantas, E. Campos, F. A. Silva, and P. Maciel, "Availability evaluation and sensitivity analysis of a mobile backend-as-a-service platform," *Journal Quality and Reliability Engineering International*, 2015.

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

[9] A. Avižienis, J. Laprie, B. Randell, and U. of Newcastle upon Tyne. Computing Science, *Fundamental Concepts of Dependability*, ser. Technical report series. University of Newcastle upon Tyne, Computing Science, 2001. [Online]. Available: https://books.google.com.br/books?id=cDkmGwAACAAJ

[10] M. Malhotra and K. Trivedi, "Power-hierarchy of dependability-model types," *Reliability, IEEE Transactions on*, vol. 43, no. 3, pp. 493–502, Sep 1994.

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

[12] I. Software, "Reliability block diagram," http://www.reliabilityeducation.com/rbd.pdf, 2007, [Online; accessed 26-September-2015].

[13] S. Distefano and L. Xing, "A new approach to modeling the system reliability: dynamic reliability block diagrams," in *RAMS '06. Annual Reliability and Maintainability Symposium, 2006.*, Jan 2006, pp. 189–195.

[14] P. M. Frank, *Introduction to Sensitivity Analysis*. Academic, 1978.

[15] R. Matos, J. Araujo, D. Oliveira, P. Maciel, and K. Trivedi, "Sensitivity analysis of a hierarchical model of mobile cloud computing," *Elsevier Journal Simulation Modelling Practice and Theory*, vol. 50, pp. 151–164, Jan. 2015.

[16] Microsoft, "Azure blockchain service," https://azuremarketplace.microsoft.com/en-us/marketplace/apps/microsoft-azure-blockchain.azure-blockchain-service?tab=Overview, 2017, [Online; accessed 19-December-2017].

[17] H. Cello, "Setup cello platform," https://github.com/hyperledger/cello/blob/master/docs/setup.md, 2017, [Online; accessed 19-December-2017].

[18] Docker, "What is a container," https://www.docker.com/what-container, 2017, [Online; accessed 19-December-2017].

[19] MongoDB, "What is mongodb?" https://www.mongodb.com/what-is-mongodb, 2017, [Online; accessed 19-December-2017].

[20] D. Oliveira, A. Brinkmann, and P. Maciel, "Advanced stochastic petri net modeling with the mercury scripting language," in *ValueTools 2017, 11th EAI International Conference on Performance Evaluation Methodologies and Tools*, December 2017.